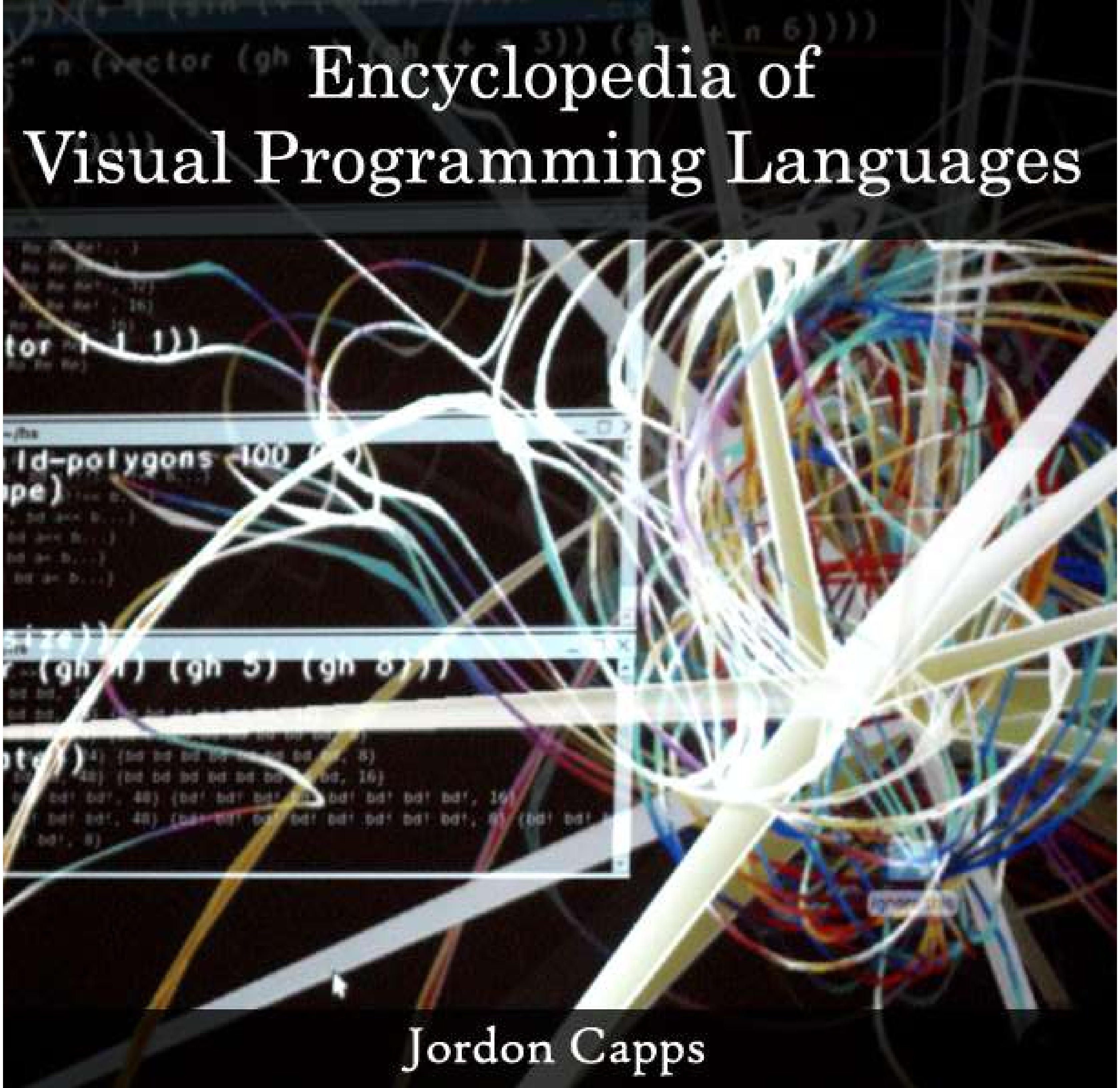
Encyclopedia of Visual Programming Languages



Jordon Capps

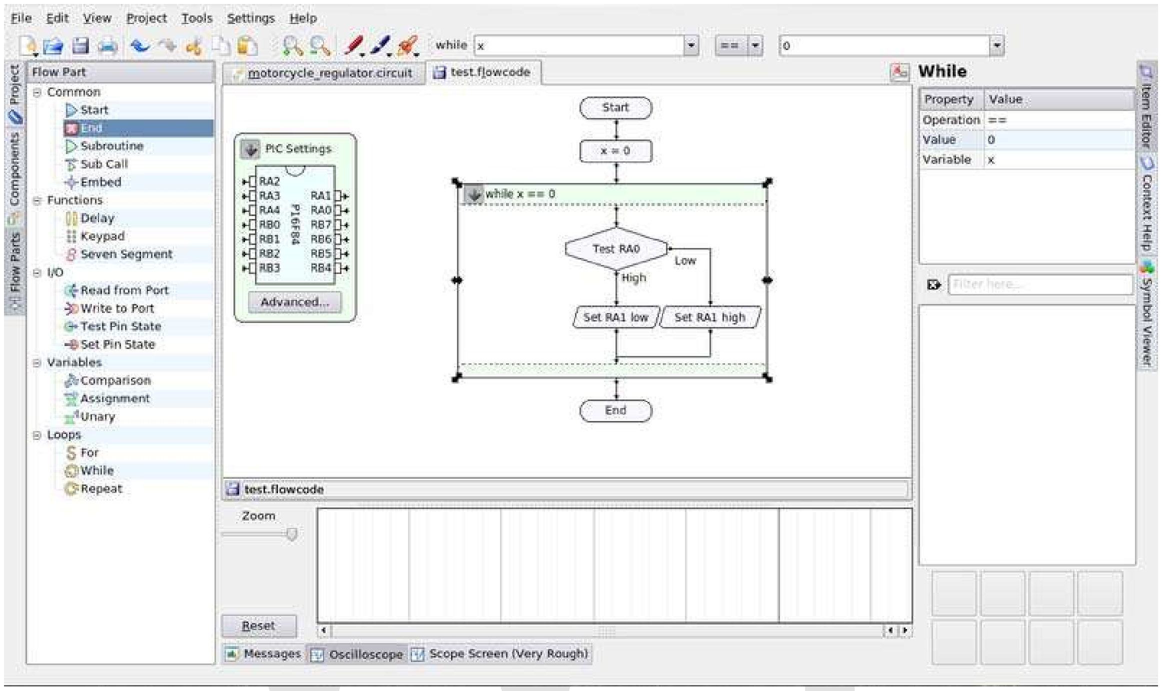
First Edition, 2012

ISBN 978-81-323-1726-5

Published by:
Learning Press
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Introduction



While

Property	Value
Operation	==
Value	0
Variable	x

KTechlab,uses flowchart to program microcontrollers graphically.

A **visual programming language** (**VPL**) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually (also known as **dataflow** or **diagrammatic programming**). A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols, used either as elements of syntax or secondary notation. Many VPLs are based on the idea of "boxes and arrows," where boxes or other screen objects are treated as entities, connected by arrows, lines or arcs which represent relations.

VPLs may be further classified, according to the type and extent of visual expression used, into icon-based languages, form-based languages, and diagram languages. Visual programming environments provide graphical or iconic elements which can be manipulated by users in an interactive way according to some specific spatial grammar for program construction.

A visually transformed language is a non-visual language with a superimposed visual representation. Naturally visual languages have an inherent visual expression for which there is no obvious textual equivalent.

Current developments try to integrate the visual programming approach with dataflow programming languages to either have immediate access to the program state resulting in online debugging or automatic program generation and documentation (i.e. visual paradigm). Dataflow languages also allow automatic parallelization, which is likely to become one of the greatest programming challenges of the future.

Visual languages and interfaces

- A-Flow, software general-purpose rapid development tool that doesn't require writing code
- AgentSheets, easy to use game authoring and computational science authoring tool
- Alice
- Analytica
- AppWare, also known as MicroBrew, icon based programming for Mac OS and Microsoft Windows
- AudioMulch, an audio signal flow based sound and music creation environment
- Automate, a workflow building tool with out writing any lines of code by hand
- Macromedia Authorware
- Automator
- Aviary Peacock, browser based visual laboratory
- Baltie
- Befunge, an esoteric text-based programming language in which commands are laid out graphically in a text file
- BT Rules, a web-based graphical programming tool with specialised tilesets to define behaviours for different devices
- Cisco Unified Application Designer, a visual programming environment for building Unified Communications applications, formerly known as the Metreos Visual Designer
- CODE
- DRAKON, a language designed for developing the Soviet Buran spacecraft
- EICASLAB, a software suite including a graphical language for supporting the design of control architectures
- Executable UML, a profile of the Universal Modeling Language specification defining an executable semantics for a subset of UML
- eXpecco, a graphical flow based language and IDE for test automation
- Flow a graphical integration language used in the webMethods platform
- Flow-based programming
- Flowcode, a language used by KTechlab (software), open source simulator and workbench for programming microcontrollers using flowcode
- Flowstone DSP Used for programming real time DSP and robotics applications
- Function block diagrams, used in programmable logic controllers
- FxEngine Framework - Open C++ dataflow programming for audio, video, signal, ...
- G, language based on data flow diagrams used in the LabVIEW development environment

- Game Maker, an easy to use game development software made by Mark Overmars
- Google App Inventor, a tool for creating applications for Google Android, based on OpenBlocks and Kawa
- GNU Radio Companion, a signal processing environment using visual blocks
- Grasshopper 3D, a generative modeling interface for Rhinoceros_3D
- Helix and Double Helix, a pioneering database management system for the Apple Macintosh platform, created in 1983
- Illumination Software Creator, a language and IDE for visually creating desktop and mobile software
- Kodu, a software designed to program games with a 3D Interface developed by Microsoft Research
- Kwikpoint, an isotype visual translator created by Alan Stillman
- LabVIEW, a graphical language designed for engineers and scientists
- Ladder logic, a language that simulates relay logic commonly used in Programmable logic controllers
- Lava
- Lily, browser based visual programming environment
- Limnor
- Mama (software) - a programming language and IDE for building 3D animations and games
- Marten commercial visual programming environment for Mac OS/X based on Prograph CPX
- Max (software), visual programming environment for building interactive, real-time music and multimedia applications
    - Max/MSP
    - Pure Data
    - jMax
    - nato.0+55+3d
- Microsoft Visual Programming Language, dataflow language for robotics programming that is a component of Microsoft Robotics Studio
- Mindscript - A simplistic data-flow programming language for multipurpose use
- MobileNation - a web-based tool for creating mobile applications
- Morphic (software), makes it easier to build and edit graphical objects by direct manipulation and from within programs; the whole Self (programming language) programming environment is built using Morphic
- MST Workshop, an interactive visual programming language for creating mathematical solutions, rapid prototyping, two-dimensional and three-dimensional graphic applications
- NXT-G, a visual programming language for the Lego Mindstorms NXT robotics kit
- OpenAlea.Visualea visual programming for plant modeling
- OpenBlocks an extendable framework for graphical block programming systems
- OpenDX scientific data visualization using a visual programming language and data flow model

- OpenMusic, a visual programming language for music composition (based on CLOS) applications, and mobile apllications
- OpenWire - adds visual dataflow programming capabilities to Delphi via VCL components and a graphical editor (homonymous binary protocol is unrelated)
- OutSystems language, a visual modeling language to develop and change all layers of business centric web applications
- Piet is an esoteric programming language designed by David Morgan-Mar, whose programs are bitmaps that look like abstract art.

- PointDragon, a visual programming language for cloud computing offered by GraphLogic.
- Prograph
- Ptolemy
- Programming Without Coding Technology (PWCT), specialist innovative technology where programmers need not write code but can visually specify every functional aspect of programs similar to flowcharts and algorithms; includes (Mahmoud programming language, RPWI environment, DoubleS [Super Server] programming paradigm); free open source; uses interaction by presenting a GUI between human and programming languages, so doing anything requires knowing procedure instead of being declarative
- PWGL, a language based on Common Lisp, CLOS and OpenGL
- Pypes, a Flow-based programming implementation by Eric Gaumer using Stackless Python
- Quartz Composer, a language for processing and rendering graphical data (Mac OS X)
- Quest3D, a 3D engine/platform for virtual reality based development (Win)
- Reaktor, a DSP and MIDI-processing language by Native Instruments
- Red-R, a cross platform Python/QT based visual programming framework for R-Project
- SCADE
- Scala Multimedia Authoring suite and complete multimedia system for AmigaOS and Windows
- Scicos A graphical language associated with the numerical analysis package ScicosLab (originally SciLab).
- Simulink
- Built on Squeak
    - Etoys scripting
- Scratch, a product of MIT designed for kids in K-12 and after school programs
- Sequential function chart, a Petri-net like programming language for programmable logic controllers
- Softimage ICE
- Stagecast Creator, formerly Apple's Cocoa: Internet Authoring for Kids
- SourceBinder, a node based visual development environment for Flash 10+
- Subtext
- SynthMaker, an audio programming tool using a visual programming language
- SynthEdit, a tool similar to SynthMaker

- tarpipe, a social media platform using a visual programming language to connect different Web services together
- Tersus, an open source platform for the development of rich web applications by visually defining user interface, client side behavior and server side processing
- TestShell, a graphical test automation environment
- ThingLab
- ToonTalk, programming system for children
- VEE
- VisSim, modeling and simulation language, allows making mathematical models quickly and executing them in real-time
- Virtools, a middleware used to create interactive 3D experiences
- VISION/HPC Python-based, drag-and-drop visual-programming environment for programming HPCs
- WireFusion, visual programming environment for creating interactive 3D web presentations
- Vsxu, music visual / real time 3D graphics generation (Windows, GNU/Linux, Mac Os X)
- vvvv, real time video synthesis
- XEE, a visual data processing language for ETL tasks
- X-Gen, a visual financial enterprise application integration for financial messaging and workflow

Note: Microsoft Visual Studio and the languages it encompasses (Visual Basic, Visual C#, Visual J#, etc.) are commonly confused to be but are not visual programming languages. All of these languages are textual and not graphical. MS Visual Studio is a visual programming environment, but not a visual programming language, hence the confusion.

Chapter 1

Tersus and Subtext (Programming Language)

Tersus

Tersus Visual Programming Platform is a general purpose software development platform that enables the development of applications, mainly rich web applications, by drawing flow diagrams instead of writing code. It's dual licensed as open-source and proprietary software.

The Tersus Modeling Language is a visual language for defining user interface, client side behavior and server side processing. The language shares many features of dataflow programming languages.

When used for web development, Tersus can be classified as Client Side + Server Side (using AJAX techniques). The modeled applications are executed by the Tersus Server.

Tersus Studio is an IDE, an extension of the Eclipse platform, used by developers (modelers) to graphically define the functionality of applications.

The platform also contains a visual debugging capability. The Tersus Server can record every step during the application's execution, and this recording ("trace") can then be played back in the Tersus Studio to view the flow of the application and the value of each data element.

Both the Studio and the Server are available on a variety of platforms, including Microsoft Windows, UNIX, Linux, and Mac OS X. The latest stable version of Tersus is version 1.3.59, released in June of 2010.

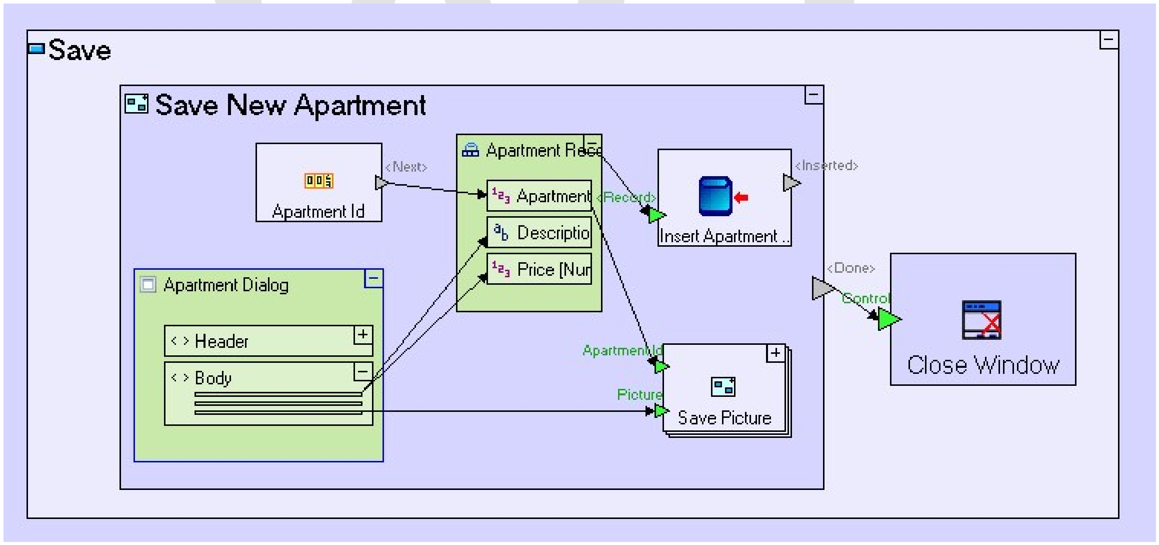
The *Tersus Visual Programming Platform* version is published under the GPL v2 license. There's also a *Tersus Enterprise Platform* version under a commercial license providing additional integration features and support.

Concept

An application is defined by a hierarchy of visual models, where high level models are composed of lower level components. The developer (modeler), employing an “infinite drawing board” that displays graphically the whole model hierarchy, starts at a top-level diagram representing the whole system, and then continues with an iterative top-down refinement process, drilling down from each model to specify its components. At the lowest level, a library of atomic building blocks is used, including, among others, data types, GUI elements, mathematical functions, database actions, and document handling actions.

Processes (and in certain cases also display elements) can receive and send out data through input "slots" ("triggers") and output slots ("exits"). The flow of data between processes, as well as the sequencing of processes, is governed by "flows" (visually represented as arrows connecting model elements).

When developing a web application, the high level models define the application's screen layout and GUI, using "display elements" (text displays, links, buttons, tables, images, etc.). Lower level models define the application's logic, using "data elements" and "process elements".



Architecture

The platform includes:

- Tersus Studio, the IDE used by modelers. It manages projects, each containing the models and resources of one application. The application models are saved as a set of XML files, each containing the details of all models in a certain package within the project.
- Model Libraries, containing building blocks for assembling applications.

- Tersus Server, which executes the modeled applications and performs the required database updates. It contains an embedded application server (Tomcat) and an embedded database server (HSQLDB), which allow for immediate testing of the modeled applications. External application servers and database servers can be used to deploy applications operationally.

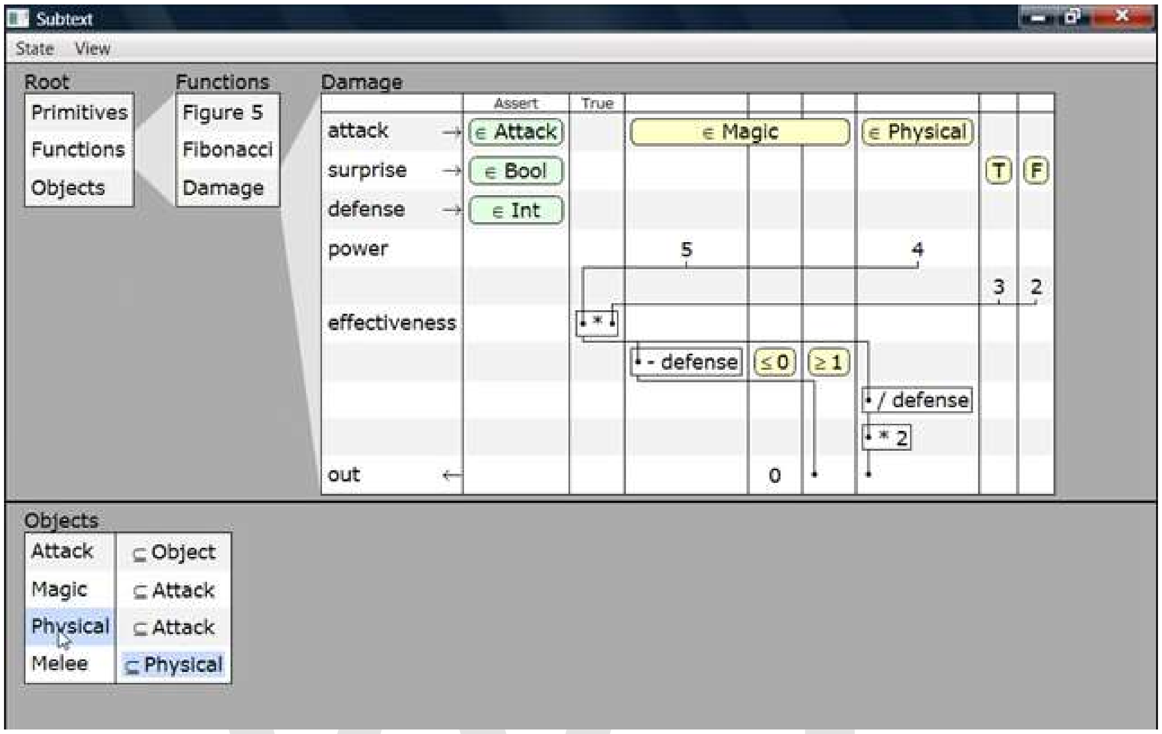
The Tersus Studio and Tersus Server are implemented in Java, while client side behavior is implemented by Javascript and HTML which are generated by the Tersus Server according to the model.

End-users invoke the applications from their browsers (for web applications), or directly from their mobile devices (e.g. for native iPhone applications).

Features

- Language independence (model names and GUI can be in any language)
- Model templates and model prototypes (templates with constraints)
- User-defined data types (data element with restricted content)
- Importing WSDL definitions of web services as Tersus building blocks
- Look and feel customization through CSS
- Visual debugging (tracing) by playback of application execution
- Automated testing through the definition of "test suites"

Subtext (programming language)

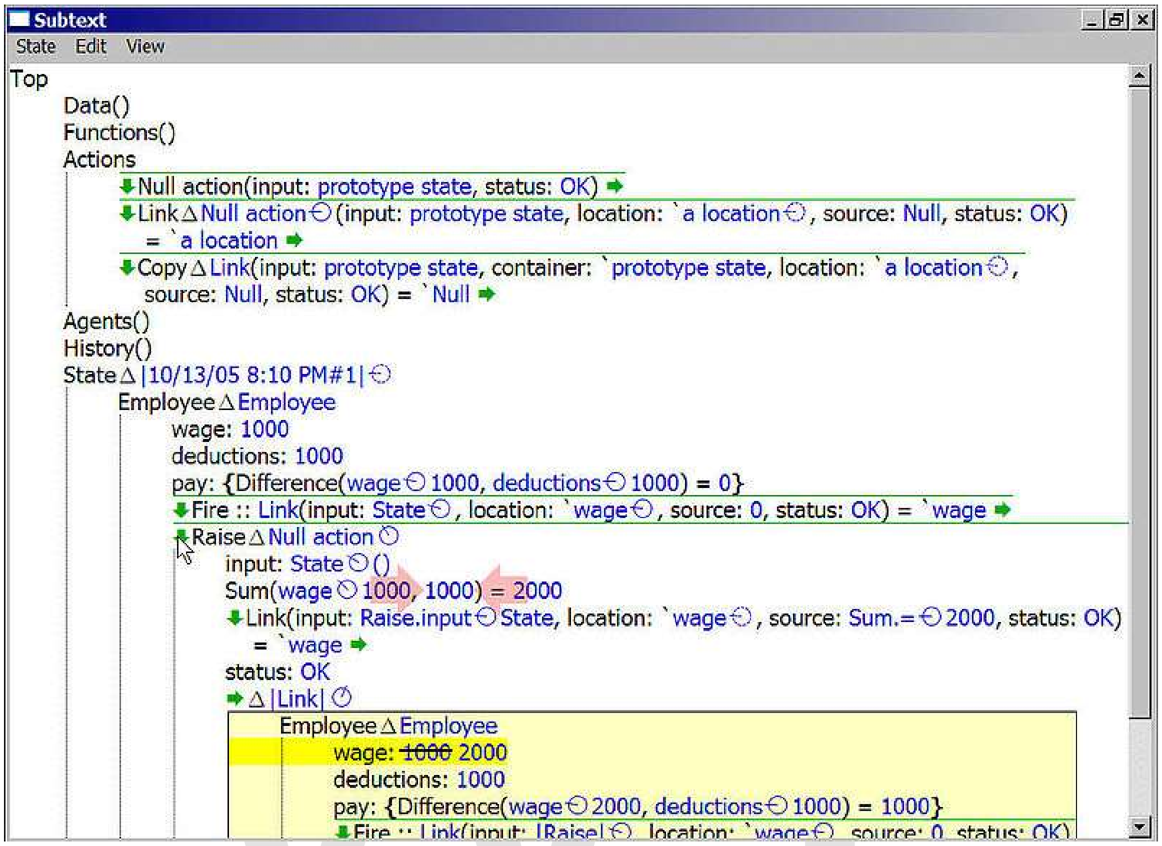


Schematic tables. An alpha build of the Subtext environment, which illustrates the unique "polymorphic conditionals" present in the IDE.

Subtext is a moderately visual programming language and environment, for writing application software. It is an experimental, research attempt to develop a new programming model, called Example Centric Programming, by treating copied blocks as first class prototypes, for program structure. It uses live text, similar to what occurs in spreadsheets as users update cells, for frequent feedback. It is intended to eventually be developed enough to become a practical language for daily use. It is planned to be open software; the license is not yet determined.

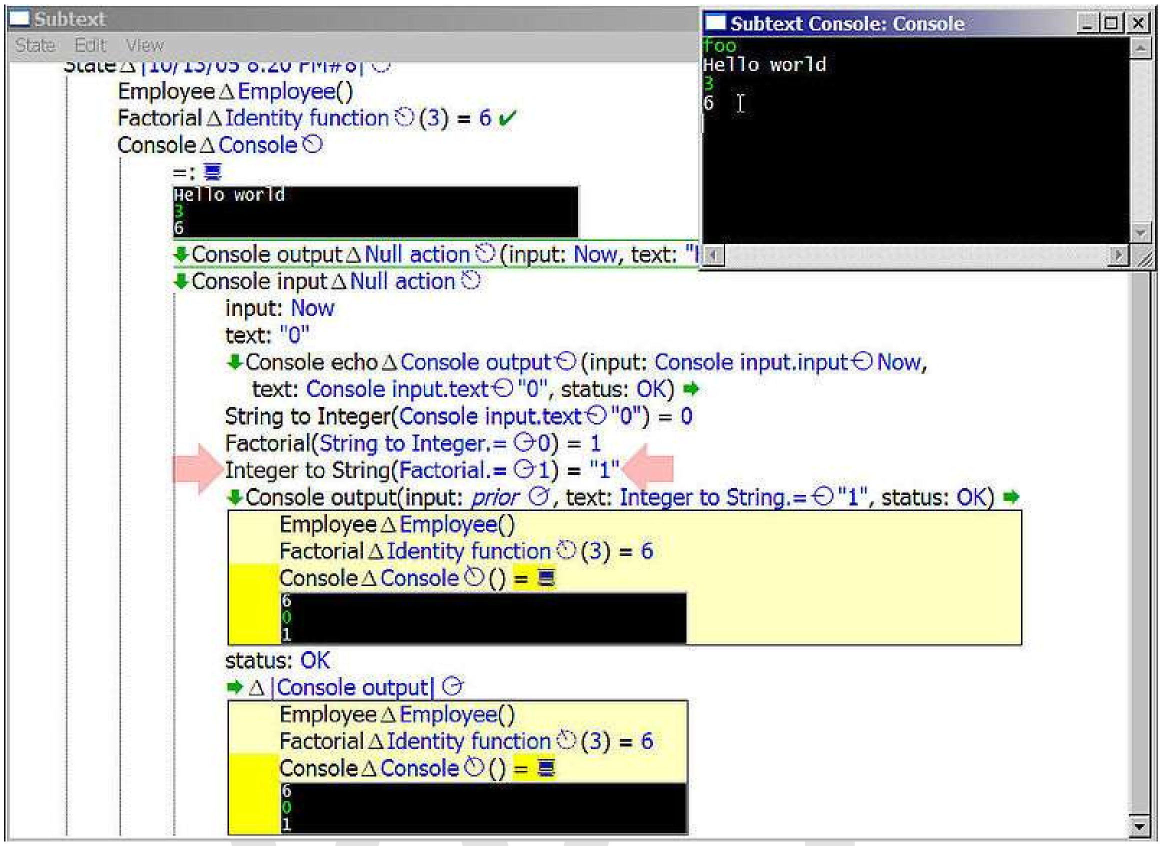
Subtext was created by Jonathan Edwards who submitted a paper on the language to OOPSLA. It was accepted as part of the 2005 conference.

Environment



```
Subtext
State   Edit   View
Top
        Data()
        Functions()
        Actions
                ⬇Null action(input: prototype state, status: OK) ➡
                ⬇Link∆Null action⊖(input: prototype state, location: `a location⊖, source: Null, status: OK)
                  = `a location ➡
                ⬇Copy∆Link(input: prototype state, container: `prototype state, location: `a location⊖,
                  source: Null, status: OK) = `Null ➡
        Agents()
        History()
        State∆|10/13/05 8:10 PM#1|⊖
                Employee∆Employee
                        wage: 1000
                        deductions: 1000
                        pay: {Difference(wage⊖1000, deductions⊖1000) = 0}
                        ⬇Fire :: Link(input: State⊖, location: `wage⊖, source: 0, status: OK) = `wage ➡
                        ⬇Raise∆Null action⊖
                                input: State⊖()
                                Sum(wage⊖1000, 1000) = 2000
                                ⬇Link(input: Raise.input⊖State, location: `wage⊖, source: Sum.=⊖2000, status: OK)
                                   = `wage ➡
                                status: OK
                                ➡∆|Link|⊖
                                        Employee∆Employee
                                                wage: ~~1000~~ 2000
                                                deductions: 1000
                                                pay: {Difference(wage⊖2000, deductions⊖1000) = 1000}
                                                ⬇Fire :: Link(input: |Raise|⊖, location: `wage⊖, source: 0, status: OK)
```

Early build of the Subtext environment with the program's current state visible.



```
foo
Hello world
3
6
```

```
State △ |10/13/05 8:20 PM#8|
    Employee △ Employee()
    Factorial △ Identity function (3) = 6 ✓
    Console △ Console
        =:
        Hello world
        3
        6
        Console output △ Null action (input: Now, text: "
        Console input △ Null action
            input: Now
            text: "0"
            Console echo △ Console output (input: Console input.input Now,
              text: Console input.text "0", status: OK)
            String to Integer(Console input.text "0") = 0
            Factorial(String to Integer.= 0) = 1
            Integer to String(Factorial.= 1) = "1"
            Console output(input: prior , text: Integer to String.= "1", status: OK)
                Employee △ Employee()
                Factorial △ Identity function (3) = 6
                Console △ Console () =
                6
                0
                1
            status: OK
            △ |Console output|
                Employee △ Employee()
                Factorial △ Identity function (3) = 6
                Console △ Console () =
                6
                0
                1
```

Early build of the Subtext environment with interactive console inputs.

Early video previews of the Subtext environment were released circa 2006, which demonstrated the semantics of Subtext programs, and the close integration with the Subtex environment and runtime.

Subtext programs are declared and manipulated (or mutated) by adding and linking elements of various types to a syntax tree, and entering in values or names as necessary, as opposed to typing out textual programs. Due to the design of the Subtext language and environment, there is no distinction between a program's representation and its execution. Like spreadsheets, Subtext programs are live executions within an environment and runtime, and programming is direct manipulation of these executions via a graphical environment. Unlike typical functional programming languages, Subtext has simple semantics and is easily applicable to reactive systems that require mutable state, I/O, and concurrency, under a model known as "Reactive Programming". Console input ("invocations") can be utilized via data flow within a Subtext program, allowing users to manipulate values interactively.

Coherence

A continuation and subset of the Subtext language using other principles, is **Coherence**, an experimental programming language and environment, which uses a new model of change-driven computation called "Coherent reaction", to coordinate the effects and side-effects of programs interactively as they are being developed. The language is specialized for interactive application software, and is being designed by the creator of Subtext, Jonathan Edwards, who reports upon its development by publishing white papers.

> “Side effects are both the essence and bane of imperative programming. The programmer must carefully coordinate actions to manage their side effects upon each other. Such coordination is complex, error-prone, and fragile. Coherent reaction is a new model of change-driven computation that coordinates effects automatically. Automatically coordinating actions lets the programmer express what to do, not when to do it.”
>
> —Jonathan Edwards, *Coherent Reaction*, MIT CSAIL

Coherence

Logo for the Coherence programming language

State changes trigger events called reactions, that in turn change other states. A coherent execution order is one in which each reaction executes before any others that are affected by its changes. A coherent order is discovered iteratively by detecting incoherencies as they occur and backtracking their effects. The fundamental building block of Coherence is the dynamically typed mutable tree. The fundamental abstraction mechanism is the virtual tree, whose value is lazily computed, and whose behavior is generated by coherent reactions.

Chapter 2

Stagecast Creator and Quartz Composer

Stagecast Creator

Stagecast Creator is a visual programming language intended for use in teaching programming to children. It is based on the programming by demonstration concept, where rules are created by giving examples of what actions should take place in a given situation. It can be used to construct simulations, animations and games, which run under Java on any suitable platform.

History

What is today known as Creator originally started as a project by Allen Cypher and David Canfield Smith in Apple's Advanced Technology Group (ATG) known as **KidSim**. As the name implies, it was intended to allow kids to construct their own simulations, reducing the programming task to something that anyone could handle. Programming in Creator uses graphical rewrite rules augmented with non-graphical tests and actions.

In 1994, Kurt Schmucker became the project manager, and under him, the project was renamed **Cocoa**, and expanded to include a Netscape plug-in. It was also repositioned as "Internet Authoring for Kids", as the Internet was becoming increasingly accessible. The project was officially announced on May 13, 1996. There were three releases:

- DR1 (Developer Release 1) on October 31, 1996
- DR2 in June, 1997
- DR3 in June, 1998

When Steve Jobs returned to Apple in 1997, he began dismantling a number of non-productive departments. One of these was the ATG. Larry Tesler, Cypher, and Smith, left to form Stagecast Software after retaining the rights to the Cocoa system.

Apple went on to reuse the Cocoa name for the entirely unrelated Cocoa application framework, originally OpenStep. Apparently due to time constraints—it was easier to reuse an already-registered name than to register a new term.

Description

Creator is based on the idea of independent *characters* that have a graphical appearance and non-graphical properties. Each character has a list of rules that determine how it behaves. The rules are created *by demonstrating* what the character does in a specific situation. Each rule is a *before / after* rule, stating that when the *before* conditions of the rule are met, the *after* actions of the rule are performed.

For a simple example, let's consider a simulation showing a character walking across a field, jumping over any rocks it encounters. Such a simulation would start with the construction of the playfield, in this case a line of icons representing the grass and a few rocks. A character is then placed on the playfield and double-clicked to open a rule editor. The rule editor will start by displaying the current conditions, that is, the character is standing on the grass. Below is an area to place the various "after" conditions, in this case the user drags open the default grid to two spaces, drags the character into the new grid cell, say to the right, and closes the rule editor.

If the simulation is started at this point, the character will start walking across the playfield to the right until it reaches the first rock. Since there is no rule showing what should happen when a rock is to the character's right, the character simply stops. At this point the rule editor is opened again, but now it shows the new condition that applies, the character is to the left of a rock. The actions in this case would be two steps, the first showing the character moving up and to the right, the next down and to the right. When the simulation is re-run, the character will walk to the right, and then "jump" over the rocks. In this case the character will now stop moving when it reaches the side of the screen, and a new rule could be added at that point to "wrap around" to the left side again.

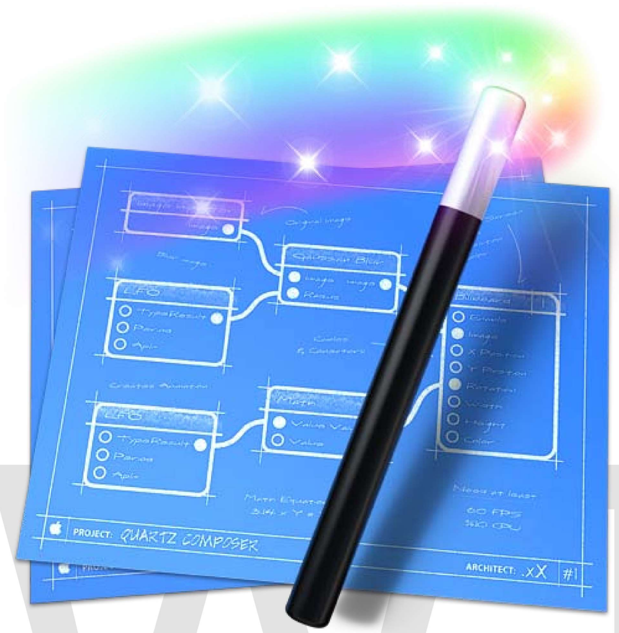
Additional rules can be added "on the fly" to flesh out the simulation. In this example additional rules would likely be added to allow the conditions to apply no matter which direction the character is walking, duplicating the existing set of two rules for movement to the left, and up and down. To make the simulation interactive, the "automatic movement" rules can be removed, and replaced by ones that move only when the cursor keys are held down, and jump only if the user presses space. Now the simulation becomes a simple game.

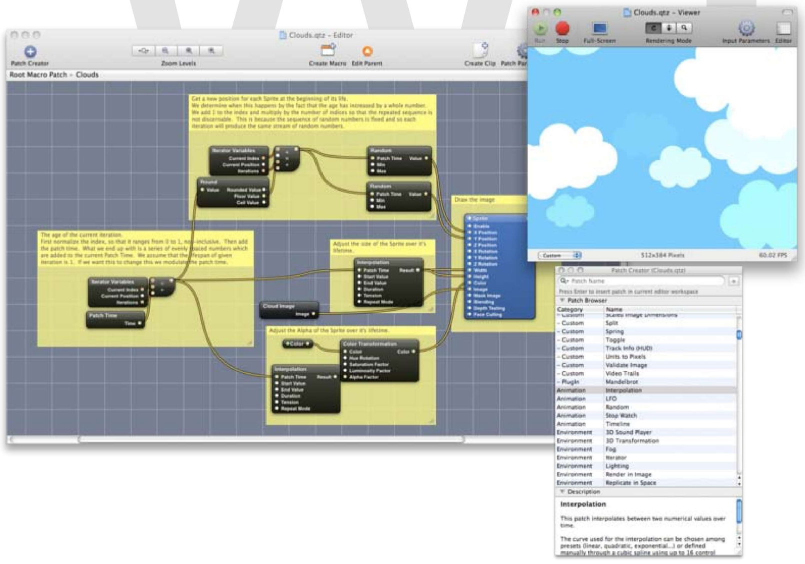
Many new features were added to the system during its evolution from KidSim/Cocoa to Creator. These include the introduction of 'jars' as a means of object classification, a new z-variable that allows Creator to simulate a 3-D space (as a stack of 2-D sheets), the ability to control more than one character at a time, and the option to redraw the screen only after all moves in a turn have been made.

In order to ensure Creator's cross-platform compatibility, the entire system was ported to the Java programming language. As Cocoa, the system was a Mac-only product and included an 'Autoplayer' functionality that allowed a Cocoa simulation to be run as a stand-alone program on any Mac. With the port to Java, Creator simulations can be posted on a Web page and run as an applet.

Quartz Composer

Quartz Composer





The Quartz Composer 4.0 interface and a composition

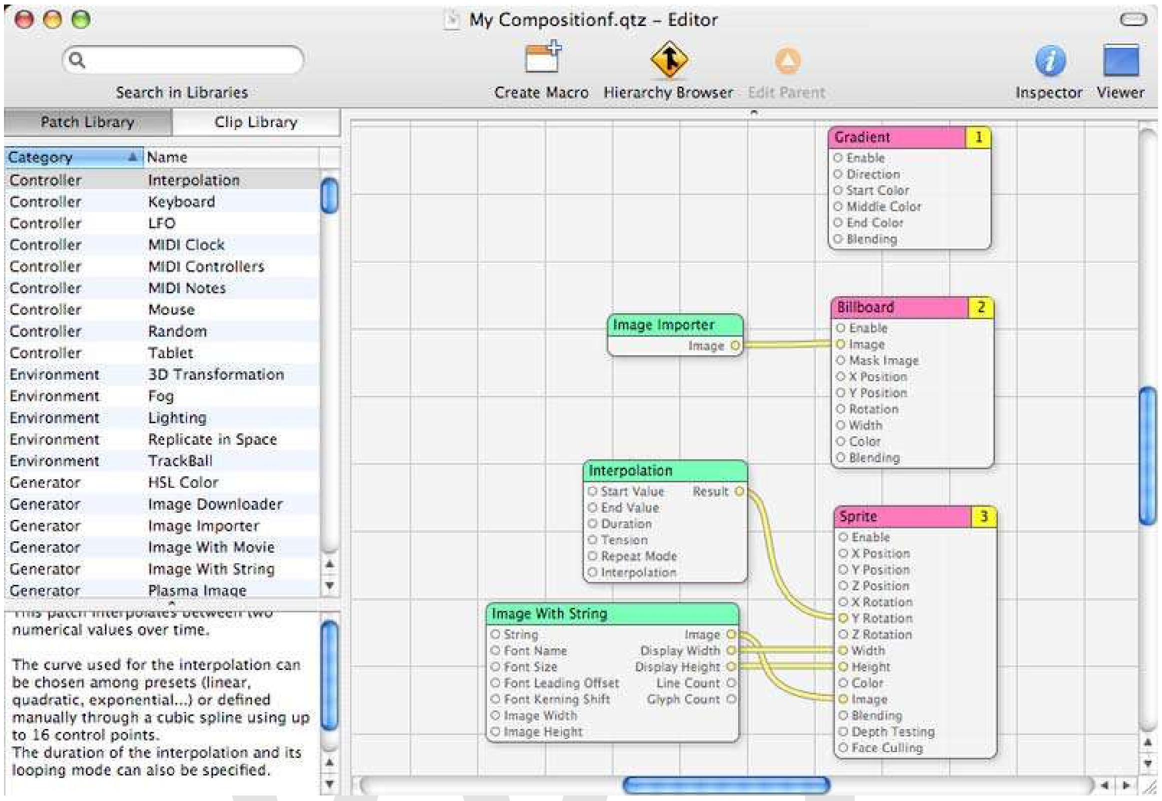
Developer(s)	Apple Computer
Stable release	4.0 (103.1) / 2009-08-28
Development status	Active

Operating system	Mac OS X v10.4 (Version 2.0) Mac OS X v10.5 (Version 3.0) Mac OS X v10.5 + iPhone SDK (Version 3.1) Mac OS X v10.6 (Version 4.0)
Available in	English
Type	Visual programming language/Software development tool
License	Proprietary

Quartz Composer is a node-based visual programming language provided as part of the Xcode development environment in Mac OS X for processing and rendering graphical data.

Quartz Composer uses OpenGL (including GLSL), OpenCL (only in Mac OS X 10.6 and later), Core Image, Core Video, JavaScript, and other technologies to build an API and a developer tool around a simple visual programming paradigm. Apple has embedded Quartz technologies deeply into the operating system. Compositions created in Quartz Composer can be played standalone in any QuickTime-aware application (although only on Mac OS X 10.4 and later), as a system Screen Saver, as an iTunes Visualizer, from inside the Quartz Composer application, or can be embedded into a Cocoa or Carbon application via supplied user interface widgets. Because Quartz Composer makes extensive use of hardware acceleration and pixel shaders, it is recommended to have a graphics card with at least 32MB of VRAM. While Quartz Composer is included with the iPhone SDK, there is currently no way of running Quartz Compositions on iOS devices.

Patches



Category	Name
Controller	Interpolation
Controller	Keyboard
Controller	LFO
Controller	MIDI Clock
Controller	MIDI Controllers
Controller	MIDI Notes
Controller	Mouse
Controller	Random
Controller	Tablet
Environment	3D Transformation
Environment	Fog
Environment	Lighting
Environment	Replicate in Space
Environment	TrackBall
Generator	HSL Color
Generator	Image Downloader
Generator	Image Importer
Generator	Image With Movie
Generator	Image With String
Generator	Plasma Image

This patch interpolates between two numerical values over time.

The curve used for the interpolation can be chosen among presets (linear, quadratic, exponential...) or defined manually through a cubic spline using up to 16 control points.
The duration of the interpolation and its looping mode can also be specified.

The Quartz Composer 2.x programming/editing interface.

Quartz programming through Quartz Composer works by implementing and connecting *patches*. Similar to routines in traditional programming languages, patches are base processing units. They execute and produce a result. For better performance, patch execution follows a lazy evaluation approach, meaning that patches are only executed when their output is needed. There are three types of patches: Consumers, Processors, and External Input patches that can receive and output mouse clicks, scrolls, and movements; MIDI and audio; keyboard; or other movements. A collection of patches can be melded into one, called a macro. Macros can be nested and their subroutines also edited.

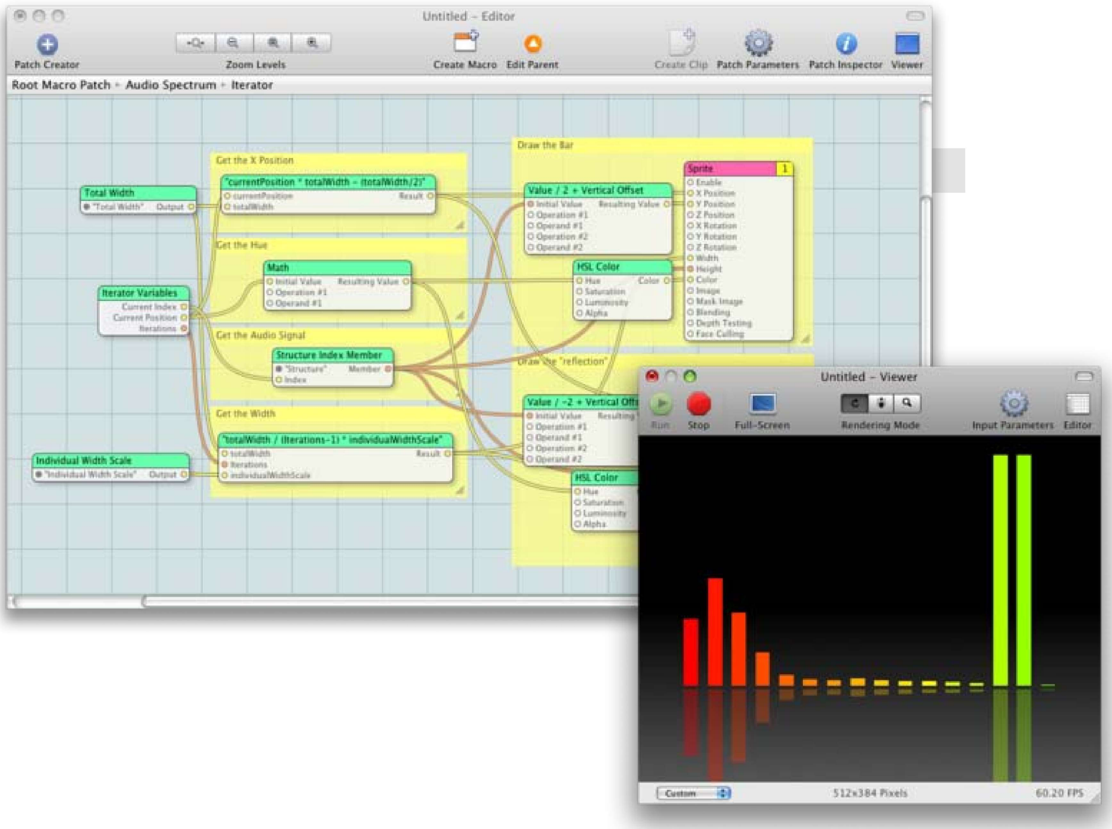
To control the order of rendering, each renderer is assigned a layer, indicated in its upper-right corner. Layers are rendered sequentially, lowest to highest. Renderers can be enabled or disabled, essentially turning on or off that particular layer. Turning off unused layers often results in better performance, since fewer upstream patches need to be evaluated.

Some patches can have subpatches, which allows for global parameter changes to just the included subpatches. This is useful for lighting, 3D transformation, and GLSL shaders, among other things. Subpatch support is indicated by square corners on a patch, rather than the typical rounded corners.

With Version 3.0, it became possible to turn compositions into Virtual Patches. These allow the user to reuse functionality without having to store duplicate copies in each composition. The Quartz Composer Editor allows the user to save a "flattened" copy (with the virtual patches fully expanded inside), for easy distribution. Version 4.0 extended this functionality even more, and automatically includes "flattened" copies of virtual patches for use as a fallback if the desired virtual patch isn't installed on the host system. This greatly simplifies composition distribution.

Network functionality was greatly improved with the release of Leopard. It became possible to transmit data and synchronize over a network interface, and it also added support for Open Sound Control transmission and reception.

Plugins



The Quartz Composer 3.0 interface.

Also new in Version 3.0 was the possibility to write custom patch plugins, using an Xcode template, and the notion of a "safe mode", where plugins and other unsafe patches fail to load. This prevents malicious compositions from performing dangerous or insecure operations. Custom patches using Apple's Xcode template are always considered unsafe.

It was possible to develop custom patch plugins for Version 2.0, but the API was undocumented and private, and was never supported by Apple. Eventually templates were released to simplify this procedure.

Hidden Options

In the Quartz Composer editor, holding the option key while selecting "Preferences..." from the menu adds 3 additional tabs of options for the user to configure. These options include System settings, Editor settings, and QuickTime integration settings. Notable options include expanded tooltips, software rendering, and uncapped framerate rendering. Multisampling was added as a hidden option on supported GPUs in version 4.0, allowing for antialiasing inside the QC Editor.

Native Datatypes

Data inside QC can be one of the following types:

- Boolean - a boolean value, 0 or 1
- Index - a positive integer between 0 and 2147483647
- Number - a double precision floating point number
- String - a unicode string
- Color - an RGBA or CMYK quartet, or a Grayscale value
- Image - a 2D image of arbitrary (possibly infinite) dimensions
- Structure - a named or ordered collection of objects, including nested structures
- Virtual - any of the above

Two additional types were introduced in version 4.0:

- Mesh - a collection of vertices, and per-vertex normals, texture coordinates, and colors in 3-space.
- Interaction - a valueless type used to associate user input with user-interactive elements of the composition.

Type Conversion

Data can usually be converted to other types transparently. In Quartz Composer 3.0, the connections between patches change color to indicate conversions that are taking place. Yellow connections mean no conversion is taking place, Orange indicates a possible loss of data from conversion (Number to Index), and Red indicates a severe conversion; Image to Boolean, for example.

Compositions

Quartz Composer documents are called *Compositions*. Compositions are Binary Property Lists (though XML versions are also supported) with a filename extension *.qtz*, and a *com.apple.quartz-composer-composition* UTI. Patches, their connections, and their input port states are saved in the composition file. Images can be stored inside a composition as well, making for self-contained compositions with embedded graphics. By dragging a movie file into the Quartz Composer editor, a reference to the movie file is created, providing a changing image that can be connected to a renderer.

Compositions also store metadata such as composition author, copyright, and description. The user can also add arbitrary metadata items, if desired.

A wide variety of image formats are supported, including JPEG, JPEG2000, GIF, PNG, TIFF, TGA, OpenEXR, BMP, ICO, PDF, PICT, ICNS, and some raw digital camera types. Images are maintained in their native form for as long as possible before rasterizing for display. This means that it will keep vector images as vectors when cropping, scaling, rotating, or translating. This allows it to work with very large logical image dimensions without consuming large amounts of memory or processing time. Such functionality is most apparent when working with text-based images, or PDFs.

Version 3.0 added the ability to add annotations to areas of the composition, called *notes*. These notes parallel comments in other programming languages. Notes can be yellow, red, green, blue, or gray, and can overlap other notes.

Composition Protocols

In Version 3.0, the concept of Composition Protocols was introduced. Protocols provide a template of required and optional inputs and outputs to qualify conforming compositions for various purposes. The following protocols are available by default:

- Graphic Animation - These don't have required inputs or outputs, but are required to render to the screen. Graphic Animations are useful for animated backgrounds in applications such as Keynote.
- Image Filter - Modifies an image using filters. No renderers are allowed in Image Filters, unless they are inside of a Render In Image environment.
- Graphic Transition - Generates a transition from a source image to a destination image over a fixed time interval.
- RSS Visualizer - Parses and Displays an RSS Feed.
- Screen Saver - Integrates with Finder for animated screen savers.
- Music Visualizer - Integrates with iTunes for audio visualization.

One new protocol was added in version 4.0:

- Mesh Filter - deforms an input mesh.

There is no officially supported way to add additional protocols to Quartz Composer. However, there are some undocumented methods that may make this possible in the future.

Composition Runtimes

In addition to protocols, compositions can also conform to different runtimes where Quartz Composer is available. In Leopard, there are runtimes for Tiger (32-bit), as well as 32-bit and 64-bit versions of the Leopard Quartz Composer runtime. The editor can also indicate used patches that are unsafe, or unavailable in Tiger to aid in making compatible compositions.

Composition Repository

A System-wide Composition Repository is available as of Version 3.0. This allows applications to share and make use of common compositions for effects and processing. It is also possible for applications to query the repository for compositions that match certain criteria, such as protocol conformance.

The Repository is spread across 3 file system locations:

- /System/Library/Compositions - core system compositions (the user typically doesn't modify these)
- /Library/Compositions - compositions available for all users
- /Users/username/Library/Compositions - compositions available for only this user

Adding compositions to the repository is as simple as adding the composition file to one of these locations.

Comparing Compositions

It became possible to compare compositions in Quartz Composer 3.0. This feature allows the user to compare inputs, rendered output, and graph appearance of any two compositions.

Related Software

Quartz Composer has many similarities to Max/MSP or Vvvv although its primary usage is for graphical rather than audio processing. The ability to construct interactive video compositions that react to audio or MIDI signals but which can be played from any QuickTime-aware application has caused a great deal of interest in Quartz Composer from VJs.

Quartz Composer Visualizer

A developer tool called Quartz Composer Visualizer was released with Quartz Composer 3.0 that allows compositions to be rendered across multiple screens on a single machine, or even spanned across several machines and displays.

Automator Support

Support for some Automator actions were added with the release of Leopard.

- Apply Quartz Composition Filter to Image Files
- Convert Quartz Compositions to QuickTime Movies
- Render Quartz Compositions to Image Files

History

Pierre-Olivier Latour originally developed the predecessor to Quartz Composer under the name PixelShox Studio.

Chapter 3

LabVIEW and Max (Software)

LabVIEW

LabVIEW

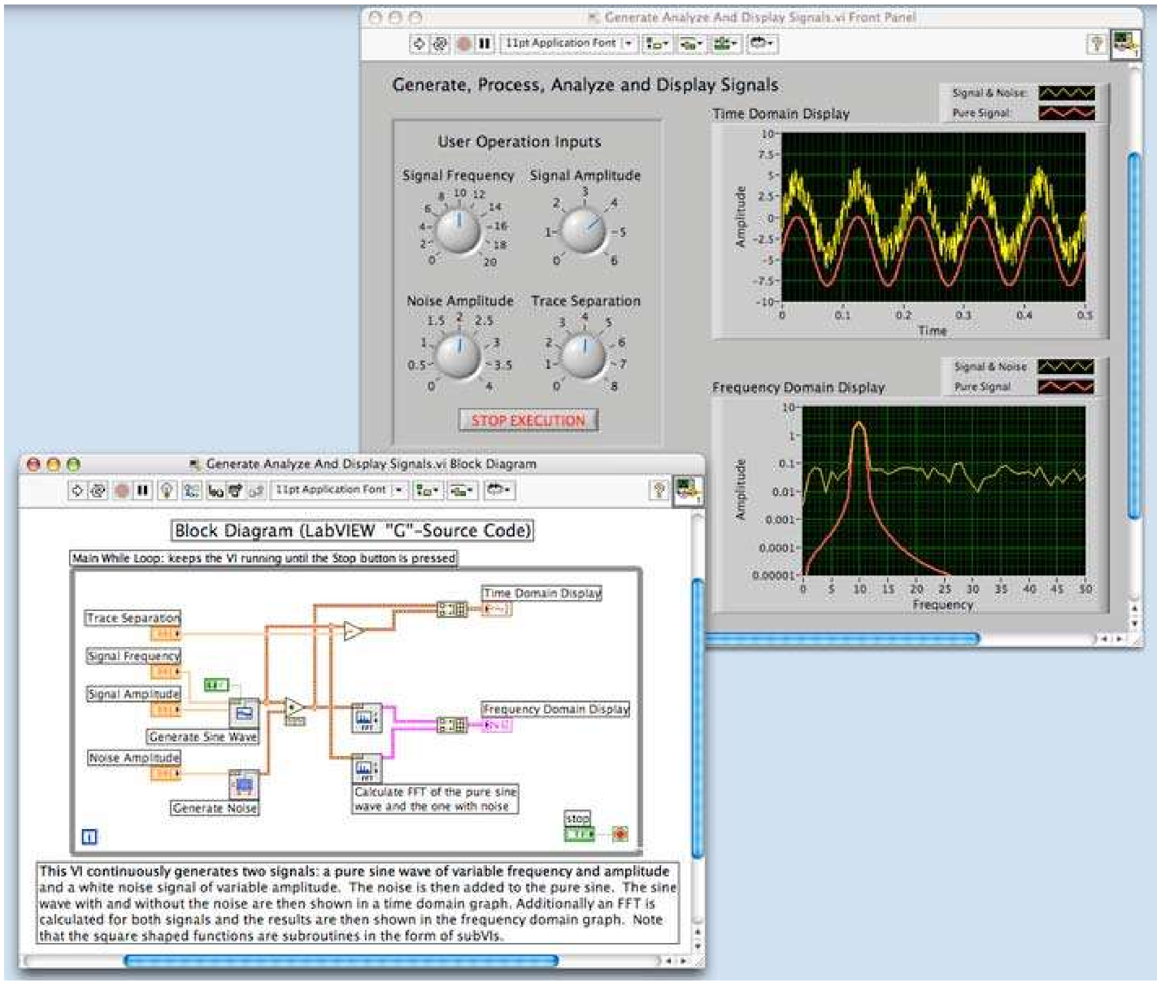
Developer(s)	National Instruments
Stable release	2010 / August 4, 2010; 4 months ago
Operating system	Cross-platform: Windows, Mac OS X, Linux
Type	Data Acquisition, Instrument Control, Test Automation, Analysis and Signal Processing, Industrial Control, Embedded Design
License	Proprietary

LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) is a platform and development environment for a visual programming language from National Instruments. The graphical language is named "G". Originally released for the Apple Macintosh in 1986, LabVIEW is commonly used for data acquisition, instrument control, and industrial automation on a variety of platforms including Microsoft Windows, various versions of UNIX, Linux, and Mac OS X. The latest version of LabVIEW is version LabVIEW 2010, released in August 2010.

Dataflow programming

The programming language used in LabVIEW, also referred to as G, is a dataflow programming language. Execution is determined by the structure of a graphical block diagram (the LV-source code) on which the programmer connects different function-nodes by drawing wires. These wires propagate variables and any node can execute as soon as all its input data become available. Since this might be the case for multiple nodes simultaneously, G is inherently capable of parallel execution. Multi-processing and multi-threading hardware is automatically exploited by the built-in scheduler, which multiplexes multiple OS threads over the nodes ready for execution.

Graphical programming



This VI continuously generates two signals: a pure sine wave of variable frequency and amplitude and a white noise signal of variable amplitude. The noise is then added to the pure sine. The sine wave with and without the noise are then shown in a time domain graph. Additionally an FFT is calculated for both signals and the results are then shown in the frequency domain graph. Note that the square shaped functions are subroutines in the form of subVIs.

Screenshot of a simple LabVIEW program

LabVIEW ties the creation of user interfaces (called front panels) into the development cycle. LabVIEW programs/subroutines are called virtual instruments (VIs). Each VI has three components: a block diagram, a front panel, and a connector panel. The last is used to represent the VI in the block diagrams of other, calling VIs. Controls and indicators on the front panel allow an operator to input data into or extract data from a running virtual instrument. However, the front panel can also serve as a programmatic interface. Thus a virtual instrument can either be run as a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the given node through the connector pane. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.

The graphical approach also allows non-programmers to build programs by dragging and dropping virtual representations of lab equipment with which they are already familiar. The LabVIEW programming environment, with the included examples and the documentation, makes it simple to create small applications. This is a benefit on one side, but there is also a certain danger of underestimating the expertise needed for good quality "G" programming. For complex algorithms or large-scale code, it is important that the programmer possess an extensive knowledge of the special LabVIEW syntax and the topology of its memory management. The most advanced LabVIEW development systems offer the possibility of building stand-alone applications. Furthermore, it is possible to create distributed applications, which communicate by a client/server scheme, and are therefore easier to implement due to the inherently parallel nature of *G*-code.

The image above is an illustration of a simple LabVIEW program showing the dataflow source code in the form of the block diagram in the lower left frame and the input and output variables as graphical objects in the upper right frame. The two are the essential components of a LabVIEW program referred to as a Virtual Instrument VI.

Benefits

One benefit of LabVIEW over other development environments is the extensive support for accessing instrumentation hardware. Drivers and abstraction layers for many different types of instruments and buses are included or are available for inclusion. These present themselves as graphical nodes. The abstraction layers offer standard software interfaces to communicate with hardware devices. The provided driver interfaces save program development time. The sales pitch of National Instruments is, therefore, that even people with limited coding experience can write programs and deploy test solutions in a reduced time frame when compared to more conventional or competing systems. A new hardware driver topology (DAQmxBase), which consists mainly of G-coded components with only a few register calls through NI Measurement Hardware DDK (Driver Development Kit) functions, provides platform independent hardware access to numerous data acquisition and instrumentation devices. The DAQmxBase driver is available for LabVIEW on Windows, Mac OS X and Linux platforms.

In terms of performance, LabVIEW includes a compiler that produces native code for the CPU platform. The graphical code is translated into executable machine code by interpreting the syntax and by compilation. The LabVIEW syntax is strictly enforced during the editing process and compiled into the executable machine code when requested to run or upon saving. In the latter case, the executable and the source code are merged into a single file. The executable runs with the help of the LabVIEW run-time engine, which contains some precompiled code to perform common tasks that are defined by the G language. The run-time engine reduces compile time and also provides a consistent interface to various operating systems, graphic systems, hardware components, etc. The run-time environment makes the code portable across platforms. Generally, LV code can be slower than equivalent compiled C code, although the differences often lie more with program optimization than inherent execution speed.

Many libraries with a large number of functions for data acquisition, signal generation, mathematics, statistics, signal conditioning, analysis, etc., along with numerous graphical interface elements are provided in several LabVIEW package options. The number of advanced mathematic blocks for functions such as integration, filters, and other specialized capabilities usually associated with data capture from hardware sensors is immense. In addition, LabVIEW includes a text-based programming component called MathScript with additional functionality for signal processing, analysis and mathematics. MathScript can be integrated with graphical programming using "script nodes" and uses a syntax that is generally compatible with MATLAB.

The fully modular character of LabVIEW code allows code reuse without modifications: as long as the data types of input and output are consistent, two sub VIs are interchangeable.

The LabVIEW Professional Development System allows creating stand-alone executables and the resultant executable can be distributed an unlimited number of times. The run-time engine and its libraries can be provided freely along with the executable.

A benefit of the LabVIEW environment is the platform independent nature of the G code, which is (with the exception of a few platform-specific functions) portable between the different LabVIEW systems for different operating systems (Windows, Mac OS X and Linux). National Instruments is increasingly focusing on the capability of deploying LabVIEW code onto an increasing number of targets including devices like Phar Lap or VxWorks OS based LabVIEW Real-Time controllers, FPGAs, PocketPCs, PDAs, and Wireless sensor network nodes.

There is a low cost LabVIEW Student Edition aimed at educational institutions for learning purposes. There is also an active community of LabVIEW users who communicate through several e-mail groups and Internet forums. sdfe

Criticism

LabVIEW is a proprietary product of National Instruments. Unlike common programming languages such as C or FORTRAN, LabVIEW is not managed or specified by a third party standards committee such as ANSI.

As of version 8, all LabVIEW installations on Windows computers require customers to contact National Instruments by Internet or phone to "activate" the product. Macintosh and Linux users are not subject to this requirement.

Building a stand-alone application with LabVIEW requires the Application Builder component which is included with the Professional Development System but requires a separate purchase if using the Base Package or Full Development System. Compiled executables produced by the Application Builder are not truly standalone in that they also require that the LabVIEW run-time engine be installed on any target computer on which users run the application. The use of standard controls requires a runtime library for any language and all major operating system suppliers supply the required libraries for common languages such as C. However, the runtime required for LabVIEW is not supplied with any operating system and is required to be specifically installed by the administrator or user. This requirement can cause problems if an application is distributed to a user who may be prepared to run the application but does not have the inclination or permission to install additional files on the host system prior to running the executable.

According to the National Instruments license agreement an executable built with LabVIEW should contain a note that the software is written in LabVIEW.

```
(1.) You include the following copyright notice "Copyright © [insert
year] National Instruments Corporation.
All Rights Reserved." in (a) the Authorized Application's About Box (if
applicable) and (b)(i) any
applicable written documentation or, (ii) if no such documentation
exists, in a "read me" or other .txt file
distributed with each copy of the Authorized Application (you may
include your own copyright notice
with the notice(s) required above);
```

There is some debate as to whether LabVIEW is really a general purpose programming language (or in some cases whether it is really a programming language at all) as opposed to an application-specific development environment for measurement and automation. Critics point to a lack of features, common in most other programming languages, such as, until version 2009, native recursion and, until version 8.20, native object oriented features.

Also, for an environment heavily targeted for test, LabVIEW includes no built-in functions for formally testing limits, reading a limits file, and conveniently tracking the passing or failing results. Companies tend to build their own proprietary functions for this basic feature if they choose not to use TestStand.

Timing System

LabVIEW uses the January 1, 1904 Epoch (reference date) as its "zero" time. Other programs that use the January 1, 1904 epoch are Apple Inc.'s Mac OS through version 9, Palm OS, MP4, and Microsoft Excel (optionally)

Release history

Starting with LabVIEW 8.0, a minor release is released around the first week of August to coincide with the National Instruments conference NI Week, followed by a bug-fix release in February. In 2009 National Instruments names the releases after the year they are released in. The bug-fix is called a Service Pack (for instance the 2009 service pack 1 is released in February 2010).

Name/Version	Build Number	Date
LabView 1.0 (for Macintosh)	??	1986
LabView 2.0	??	1990
LabView (for Sun & Windows)	??	1992
LabView (Multiplatform)	??	1993
LabView 4.0	??	1997
LabView 5.0	??	1998
LabView Real-Time	??	1999
LabView 6i	??	2000
LabView 7 Express	??	2003
LabView 8	??	2005
LabView 8.20	??	2006
LabView 8.2.1	8.2.1.4002	2/21/2007
LabView 8.5	8.5.0.4002	2/19/2008
LabVIEW 8.6	8.6.0.4001	7/24/2008
LabVIEW 2009 (32 and 64-bit)	9.0.0.4022	8/4/2009
LabVIEW 2010 (32 and 64-bit)	10.0.0.4032	8/4/2010

Repositories and libraries

OpenG, as well as LAVA Code Repository (LAVAcr), serve as repositories for a wide range of Open Source LabVIEW applications and libraries. SourceForge has LabVIEW listed as one of the possible languages which code can be written in.

VI Package Manager has become the standard package manager for LabVIEW libraries. It is very similar in purpose to Ruby's RubyGems and Perl's CPAN, although it provides a graphical user interface similar to the Synaptic Package Manager. VI Package Manager provides access to a repository of the OpenG (and other) libraries for LabVIEW.

Related software

National Instruments also offers a product called Measurement Studio, which offers many of the test, measurement and control capabilities of LabVIEW, as a set of classes for use with Microsoft Visual Studio. This allows developers to harness some of LabVIEW's strengths within the text-based .NET framework. National Instruments also offers LabWindows/CVI as an alternative for ANSI C programmers.

When applications require sequencing, users often use LabVIEW with TestStand test management software, also from National Instruments.

The TRIL Centre Ireland BioMobius platform and DSP Robotics' FlowStone DSP also use a form of graphical programming similar to LabVIEW, but are limited to the biomedical and robotics industries respectively.

Max (software)

Max

Developer(s)	Cycling '74
Stable release	5.1 / November 22, 2009; 12 months ago
Operating system	Windows XP, Mac OS X
Type	music and multimedia development

Max is a visual programming language for music and multimedia developed and maintained by San Francisco-based software company Cycling '74. During its 20-year history, it has been widely used by composers, performers, software designers, researchers, and artists for creating innovative recordings, performances, and installations.

The **Max** program itself is highly modular, with most routines existing in the form of shared libraries. An API allows third-party development of new routines (called "external objects"). As a result, Max has a large userbase of programmers not affiliated with Cycling '74 who enhance the software with commercial and non-commercial extensions to the program. Because of its extensible design and graphical interface (which in a novel way represents the program structure and the GUI as presented to the user simultaneously), Max is widely regarded as the lingua franca for developing interactive music performance software.

History

Max was originally written by Miller Puckette as the *Patcher* editor for the Macintosh at IRCAM in the mid-1980s to give composers access to an authoring system for interactive computer music. It was first used in a piano and computer piece called *Pluton* (written by Philippe Manoury in 1988), synchronizing the computer to the piano and controlling a Sogitec 4X, which performed the audio processing.

In 1989, IRCAM developed and maintained a concurrent version of Max ported to the IRCAM Signal Processing Workstation for the NeXT (and later SGI and Linux), called Max/FTS (FTS standing for "Faster Than Sound", and being analogous to a forerunner to MSP enhanced by a hardware DSP board on the computer).

In 1989, it was licensed by IRCAM to Opcode Systems, which sold a commercial version of the program in 1990 called Max (developed and extended by David Zicarelli). Never a perfect fit for Opcode Systems, the company ceased active development on the software in the mid-90s. The current commercial version of Max has since been distributed by Zicarelli's company, Cycling '74 (founded in 1997), since 1999.

Puckette released an entirely re-designed free software program in 1996 called Pd (short for "Pure Data"), which, despite a number of fundamental differences from the IRCAM original, is superficially very similar and remains an open-source alternative to Max/MSP.

Max has a number of extensions and incarnations; most notably, a set of audio extensions to the software appeared in 1997, derived in part from Puckette's subsequent work in Pure Data. Called **MSP** (short for either Max Signal Processing or the initials of Miller S. Puckette), this "add-on" package for Max allowed for the manipulation of digital audio signals in real-time, allowing users to create their own synthesizers and effects processors (Max had previously been designed to interface with hardware synthesizers, samplers, etc. as a "control" language using MIDI or some other protocol).

In 1998, a direct descendant of Max/FTS was developed in Java (jMax) and released as open-source.

1999 saw the release of nato.0+55, a suite of externals developed by Netochka Nezvanova that brought to Max extensive control of realtime video. Although nato became increasingly popular among multimedia artists, its development was dropped in 2001. SoftVNS, a third-party package for visual processing in Max was developed by Canadian media artist David Rokeby and released in 2002.

In the meantime, Cycling '74 developed their own set of extensions for video. A major package for Max/MSP called **Jitter** was released in 2003, providing real-time video, 3-D, and matrix processing capability.

In addition, a number of Max-like programs exist which share the same concept of visual programming in realtime such as Quartz Composer (by Apple) and vvvv which are both focusing on realtime video synthesis and processing.

A major update to Max/MSP/Jitter, Max 5, was released in 2008.

Language

Max is named after Max Mathews, and can be considered a descendant of MUSIC, though its graphical nature disguises that fact. As with most MUSIC-N languages, Max/MSP/Jitter distinguishes between two levels of time: that of an "event" scheduler, and that of the DSP (this corresponds to the distinction between k-rate and a-rate processes in Csound, and control rate vs. audio rate in SuperCollider).

The basic language of Max and its sibling programs is that of a data-flow system: Max programs (called "patches") are made by arranging and connecting building-blocks of "objects" within a "patcher", or visual canvas. These objects act as self-contained programs (in reality, they are dynamically-linked libraries), each of which may receive input (through one or more visual "inlets"), generate output (through visual "outlets"), or both. Objects pass messages from their outlets to the inlets of connected objects.

Max supports six basic atomic data types that can be transmitted as messages from object to object: int, float, list, symbol, bang, and signal (for MSP audio connections). A number of more complex data structures exist within the program for handling numeric arrays (*table* data), hash tables (*coll* data), and XML information (*pattr* data). An MSP data structure (*buffer~*) can hold digital audio information within program memory. In addition, the Jitter package adds a scalable, multi-dimensional data structure for handling large sets of numbers for storing video and other datasets (*matrix* data).

Max is typically learned through acquiring a vocabulary of objects and how they function within a patcher; for example, the *metro* object functions a simple metronome, and the *random* object generates random integers. Most objects are non-graphical, consisting only of an object's name and a number of arguments/attributes (in essence class properties) typed into an *object box*. Other objects are graphical, including sliders, number boxes, dials, table editors, pull-down menus, buttons, and other objects for running the program interactively. Max/MSP/Jitter comes with about 600 of these objects as the standard package; extensions to the program can be written by third-party developers as Max patchers (e.g. by encapsulating some of the functionality of a patcher into a sub-program that is itself a Max patch), or as objects written in C, C++, Java, or JavaScript.

The order-of-execution for messages traversing through the graph of objects is defined by the visual organization of the objects in the patcher itself. As a result of this organizing principle, Max is unusual in that the program logic and the interface as presented to the user are typically related, though newer versions of Max provide a number of technologies for more standard GUI design.

A large number of people use Max, even if they aren't aware of it. Max documents (called patchers) can be bundled into standalone applications and distributed free or sold commercially. In addition, Max can be used to author audio plugin software for major audio production systems.

With the increased integration of laptop computers into live music performance (in electronic music and elsewhere), Max/MSP and Max/Jitter have received quite a bit of attention as a development environment available to those serious about laptop music / laptop video performance.

Chapter 4

Limnor and Illumination Software Creator

Limnor

Limnor is a generic-purpose codeless and visual programming system. The aim is to enable users to create computer software without directly coding in a texture programming language. It can be extended by software developers.

Vision

In the future most people should be able to do computer programming. Visual and codeless programming can be one of possible ways to reach that goal.

When events were added to classes it enabled rapid application development, especially the visual development of graphic user interface. But event handling is still expressed via textual programming languages.

The idea of Limnor codeless programming is to add "Actions" to classes. Adding actions to classes eliminates the need of textual programming languages, making generic purpose codeless and visual programming possible without sacrificing programming power.

A class in Object-Oriented Programming is defined by properties, methods and events. Now it is enhanced by actions.

Visual programming by "properties, methods, events and actions" can be a fifth-generation programming language.

People expect 5GL to be easy to use, to be visual. There are many excellent visual programming languages and systems available now. One issue is that while each visual programming innovation has its unique advantages in some aspects it also has its limitations in other aspects. If different visual programming systems may work together then developers may take advantages of different systems and avoid the limitations. 5GL may solve the problem by being an abstraction layer for visual programming.

Following criteria may define an abstraction layer for visual programming:

- All visual programming systems can use the abstraction layer to represent their programming results.
- Given a programming task, if visual programming system A and visual programming system B may be used to accomplish all or parts of the task then A and B may use the same representation of the abstraction layer for the parts of the task they both can accomplish.

The first criterion requires the layer to have a wide coverage. Most programming languages, such as C/C++, Java, C#, VB, etc., meet this criterion. The first criterion does not restrict each visual programming system to have its own programming elements for unique visual representations.

The second criterion requires the layer to be an abstraction for visual programming and makes it possible for different visual programming systems to work together.

It can be deduced that when designing a 5GL to meet the criteria, such a 5GL should allow decorative expansions by individual visual programming systems. When the 5GL compiles/interprets a program, it ignores those decorative expansions.

Design

An action is defined by Action-Executer, Action-Data, and Action-Condition. An action has an owner which defines the scope of data available for the action.
There are 3 types of actions:

- Method-Invoke Action. It is the execution of a method of a class. The Action-Executer is the class defining the method. The Action-Data is the values provided to the method parameters. Method return value can be assigned to a value (a property or a variable).
- Property-Setting Action. It assigns a value to a property of a class. The Action-Executer is the class. The Action-Data is the value provided to the property.
- Event-Firing Action. It fires an event of a class being developed. The Action-Executer is the class. The Action-Data is the values provided to the event parameters.

Action data can be a constant, a variable, a method/event parameter, a property of a class/variable/parameter/property, or a math expression.
An Action Condition is a math expression evaluated to a Boolean value.
A math expression is a math formula with its variables linked to constants, properties, variables, parameters, and math expressions.
A math expression must be displayed and edited graphically for codeless and visual programming and for intuitive using of the programming tools. The math expression programming tool must be able to handle math elements and functions developed by third parties to enable unlimited expansions.

Implementation

Limnor Studio is a newer implementation of codeless visual programming by "properties, methods, events and actions".
Limnor Studio implements "properties, methods, events and actions" on Microsoft .Net types. Software is developed by developing classes, just as all object-oriented programming languages do, but it is done visually. Developers derive new classes from existing classes. The existing classes are from .Net libraries made by any companies or individuals, using any programming languages or programming systems, including Limnor Studio. Limnor Studio compiler generates C# source code from visual programming representations. It uses C# compiler to generate the programming results, EXE or DLL files.
A base class for math functions provides a framework for visually display the math expressions in original math expressions in programming instead of texture formation. For example, developers do programming using formula like
$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$ instead of writing something like Math.sqrt((x0-x1)*(x0-x1)+(y0-y1)*(y0-y1)). A math expression editor may handle math classes derived from the base math class, allowing third parties to develop new math functions to be used visually.
The developers use "action diagram" (flowchart) to form programming logics visually and intuitively.
Visual programming is better done by more than one way. Limnor Studio uses a plug-ins system to allow different visual programming systems to work together. UI designer, which represents classes in UI form and icons, is one visual programming system; object-explorer, which represents classes in tree-views, is another visual programming system. They are totally independent of each other even though they represent and work on the same class being developmented. Other independent visual programming systems can be developed and plugged in by implementing certain interfaces, for example, visual data-flow, visual control-flow, UML, etc.

Features

Limnor Studio Major Features:

- Visually creation of all 3 types of Actions.
- Action diagram (flowchart)
- Math Expression Editor
- Directly use of software libraries made by other .Net programming languages and programming systems
- Development work can be directly used by other .Net programming languages and programming systems
- Generates C# source code and Visual Studio project file from visual codeless programming representation
- Plug-ins system for independent 5-th generation programming language tools to work on "properties, methods, events and actions"

- Class derivation
- Method and property overriding
- Create properties, methods and events
- Attribute programming
- Polymorphism by interfaces
- High performance 2D drawing tools
- Database programming, Visual Query Builder, Data-binding
- Kiosk application
- Form Designer from Microsoft Visual Studio for GUI programming
- Object-Explorer for viewing and working on "properties, methods, events and actions" in tree-views
- Event-Path for viewing and working on action creation and events-actions mapping. It is a kind of control-flow programming
- Effortlessly creation of multi-threaded action executions
- Cloud computing

Illumination Software Creator

Illumination Software Creator

Developer(s)	Radical Breeze
Stable release	2.2.0 / 11 October 2010
Operating system	Linux, Microsoft Windows, Mac OS X
Type	Programming
License	Shareware

Illumination Software Creator (**Illumination**) is a tool for visually designing and developing software, and a corresponding Visual programming language that is available for Microsoft Windows, Linux and Mac OS X. Software developed with Illumination runs on Microsoft Windows, Linux, Mac OS X, Android (operating system) powered devices, Maemo powered devices and Adobe Flash powered websites. Illumination is developed and sold by Radical Breeze.

History

Illumination was created by Bryan Lunduke, and first released in May 2010. The earliest known public mention of Illumination was on the April 11, 2010 episode of The Linux Action Show!.

Features

Illumination works by arranging "building blocks" in order to visually describe the functionality of a software application. Like many other Visual programming languages, Illumination does not require any code to be written by hand in order to develop software.

Within Illumination each "Block" is a self contained piece of functionality. Blocks are tied together via "inputs" and "outputs" which pass no data, and only serve to structure the flow of the application. Illumination also contains a "Window Editor" to allow for the building of applications with simple user interfaces.

As of 2.0, Illumination Software Creator supports creating Python (programming language) (PyGTK) applications as two distinct targets: Desktop and Maemo Tablet. And also supports creating Adobe Flex based rich Internet applications that run on the Adobe Flash platform.

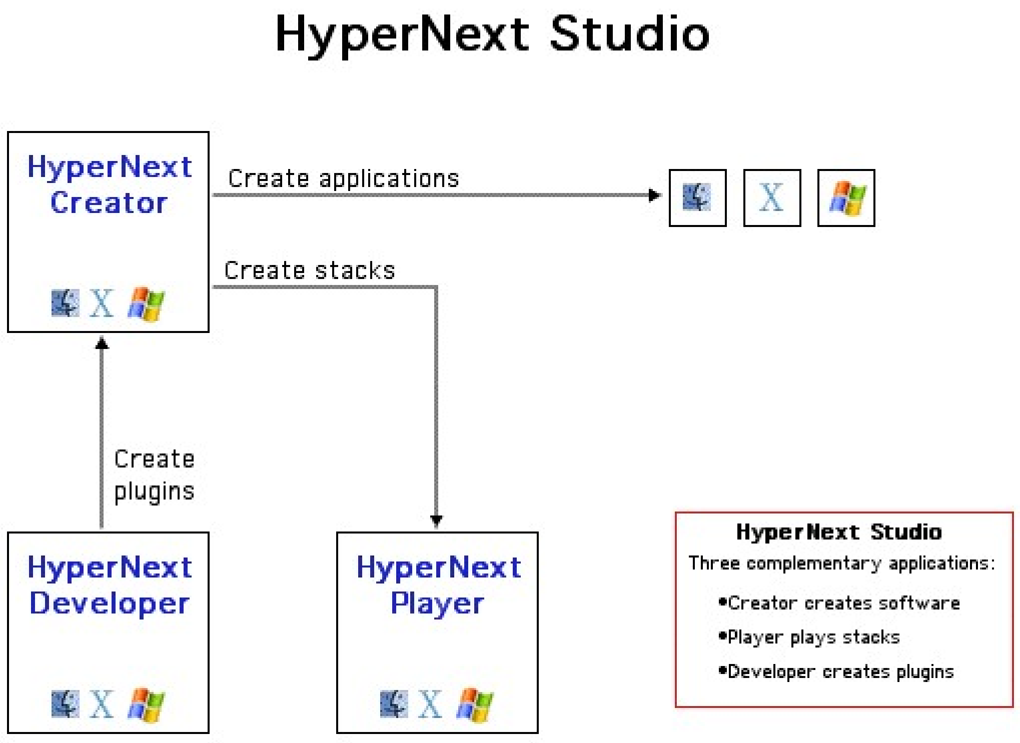
At version 2.1 support for building Android applications was added.

In version 2.2 support for building what are called "Custom Blocks" was added to allow developers to expand the functionality of their projects as needed using traditional programming languages (such as Java, Python and ActionScript).

Chapter 5

HyperNext and Kyma (Sound Design Language)

HyperNext



HyperNext Studio Family.

HyperNext is a freeware visual software development system aimed at beginner programmers that runs on Macintosh and Windows computers. It was inspired by HyperCard and includes a GUI having controls such as buttons and listboxes, and an interpreted English-like programming language. HyperNext also includes a high-level object-oriented compiled BASIC. The HyperNext Studio package comprises three complementary applications that can help users create and run software under Windows and Mac OS X and Mac OS 9 platforms:

(1) HyperNext Creator is similar to Hypercard and enables users to create their own software, both cross-platform standalone applications and stacks for the freeware HyperNext Player. The design interface of Creator has just a design window, one toolbar and a mode switcher. Its three modes of Design, Preview and Run enable rapid switching between the creation/editing and running of programs.

(2) HyperNext Developer builds plugins/libraries for HyperNext Creator and allows users to extend the functionality of Creator with their own or 3rd party plugins. HyperNext Developer has a very similar user interface to that of Hypernext Creator and places built plugins directly into the Creator's plugin folder. For faster plugin development is possible to have both Creator and Developer open simultaneously.

(3) HyperNext Player is a standalone application similar to Hypercard Player in that it runs stacks created by its creator application, HyperNext Creator. HyperNext Player is freeware and there are versions for Windows and Mac OS X and Mac OS 9 platforms.

Programming environment

HyperNext has a relatively simple design interface that allows programs to be quickly run and tested. Controls such as buttons can be selected on the toolbar and then added to the design window. Once placed, controls can have their properties changed and their scripts edited using the in-built script editor. HyperCard is based on the concept of a "stack" of virtual "cards." Cards hold data, just as they would in a rolodex but can also contain user-interface elements such as buttons.

HyperNext's main programming language is simply called HyperNext and is loosely based on Hypercard's HyperTalk language. HyperNext is an interpreted English-like language and has many features that allow creation of full applications and stacks. These features includes a large variety of keywords to affect and receive feedback from its many GUI control types and multi-media capabilities using Apple Quicktime. Midi notes and sounds can be played using Note Player and HyperNext's sprite surface has various uses including game development. The Macintosh versions of HyperNext include functions to run the AppleScript scripting language.

The HyperNext language can be extended with user-defined or third-party plugins developed using HyperNext Developer. HyperNext also has the ability to run RBscripts at runtime so allowing users to make their own programmable applications. There is a wide range of RBscript code and similar BASIC code already available at various sites on the internet. RBscript is a version of REALbasic, a modern object-oriented BASIC developed by REAL Software.

HyperNext has a framework for developing neural networks and comes with examples showing how to implement neural network training and testing. The example projects and stacks include analysis of patient heart data, DNA sequences and second hand car prices.

A recent introduction to HyperNext is the ability to control biofeedback devices such as the LightStone using USB HID communications. The LightStone device sends out data that can be processed to give heart rate and skin conductance values which can then be monitored by a program to aid meditation or enable biofeedback games to be played. HyperNext has an example project and stack that display graphically the data from a LightStone device.

Kyma (sound design language)

Kyma is a visual programming language for sound design used by musicians, researchers, and sound designers. In Kyma, a user programs a multiprocessor DSP by graphically connecting modules on the screen of a Macintosh or Windows computer.

Background

Kyma has characteristics of both object-oriented and functional programming languages. The basic unit in Kyma is the "Sound" object, not the "note" of traditional music notation. A Sound is defined as:

i) a Sound atom
ii) a unary transform T(s) where s is a Sound
iii) an n-ary transform $T(s_1, s_2,.., s_n)$, where $s_1,s_2,..s_n$ are Sounds

A Sound atom is a source of audio (like a microphone input or a noise generator), a unary transform modifies its argument (for example, a LowpassFilter might take a running average of its input), and an n-ary transform combines two or more Sounds (a Mixer, for example, is defined as the sum of its inputs).

History

The first version of Kyma, which computed digital audio samples on a Macintosh 512K was written in the Smalltalk programming language in 1986 by Carla Scaletti in Champaign, Illinois. In May 1987, Scaletti had partitioned Kyma into graphics and sound generation engines and ported the sound generation code to a digital signal processor called the Platypus designed by Lippold Haken and Kurt J. Hebel of the CERL Sound Group.

In 1987, Scaletti presented a paper on Kyma and demonstrated live digital sound generation on the Platypus at the International Computer Music Conference where it was identified by electronic synthesis pioneer Bob Moog as a technology to watch in his conference report for Keyboard Magazine:

One new language that acknowledges no distinction between sound synthesis and composition is Kyma, a music composition language for the Macintosh that views all elements in a piece of music, from the structure of a single sound to the structure of the entire composition, as objects to be composed.

When the University of Illinois at Urbana-Champaign eliminated the funding for the PLATO laboratory in 1989, Scaletti and Hebel formed Symbolic Sound Corporation in order to continue developing Kyma and digital audio signal processing hardware.

Chapter 6

Object Process Graph and Prograph

Object Process Graph

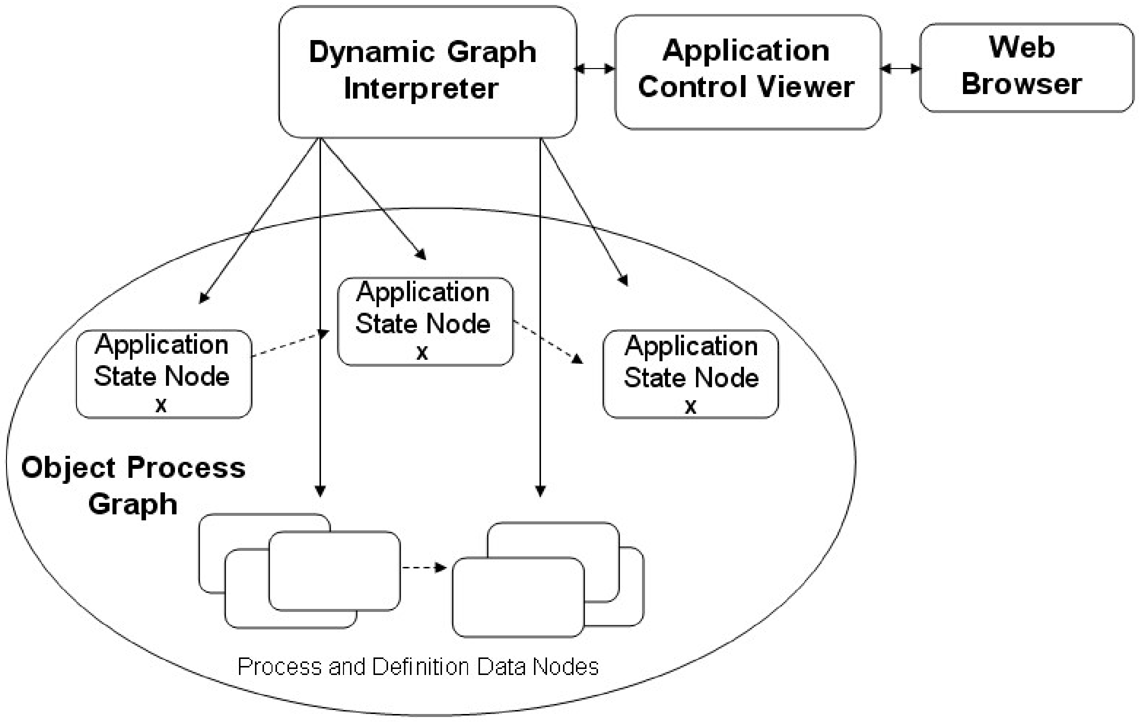
An **Object Process Graph (OPG)** is a general purpose executable graph that incorporates every aspect of an application, including process, user interface, and database. No programming language, tool, or database is required to handle any part of an application. A complete high-level visual programming environment is used to define an OPG. No code is generated; the graph is the code. Object Process Graphs interface with traditional programming languages and databases through industry standard protocols. The reference implementation of the OPG model is GraphLogic's PointDragon platform.

Overview of Object Process Graph Technology

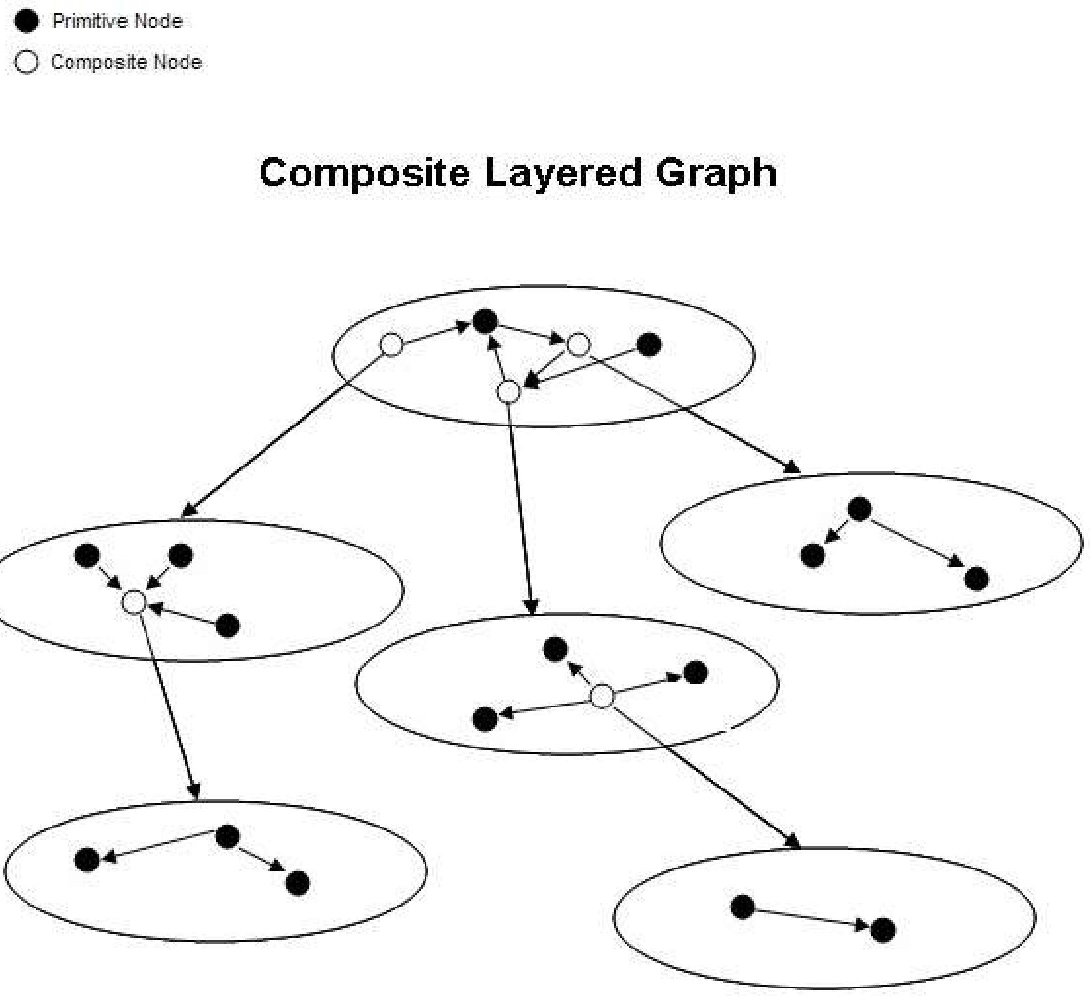
The major elements of Object Process Graph Technology are: the Object Process Graph (OPG), the Dynamic Graph Interpreter (DGI), the Application Controller Viewer (ACV), the Application Editor System (AES) and the Object Process Graph Application Program Interface (OPGAPI). Each of these elements are briefly described below.

Object Process Graph

An **Object Process Graph (OPG)** completely defines a computer application's persistent and transient data, its processing logic and data flow, and the display and validation characteristics of every application data item. Its graph-based structure incorporates the concept of a graph-oriented object database model, and by making it executable extends its capabilities to encompass all the requirements of the new software architecture. Formally the Object Process Graph is a Turing complete programming language. Fundamental to GraphLogic's invention was the conversion of an executable graph into a practical programming language, through the addition of specialized nodes and edges, properties and specific protocols. The OPG is interpreted as the program it defines is executed. OPGs are stored in both transient and persistent computer memory. They can hold any data structure, including but not limited to: relational tables, hierarchical tables, n-dimensional data arrays, spreadsheets, and both graphical 2-D and 3-D models. Large complex data structures are not stored as blobs as is common in relational database systems, but in OPG structures that reflect their original structure and internal relationships. Object Process Graph process and control structures provide complete control over the order and timing of persistent and transient data validation, transformation, and display within an application. Object Process Graph process and control structures can completely define mathematical formulas, regular expressions (in the case of textual data or mixed quantitative and textual data) and complete algorithms. Object Process Graph based applications are accessed, interpreted, modified, and run by the **Dynamic Graph Interpreter** (described below).



Dynamic Graph Interpreter (DGI) accessing an Object Process Graph (OPG) to execute an application through a web browser.



OPG is a Composite Layered Graph comprising primitive and composite nodes and related by edges

Dynamic Graph Interpreter

The **Dynamic Graph Interpreter (DGI)** runs an instance of an application by making transitions from one application state to another and displaying the application's state information via the Application Controller Viewer (described below) on an internet browser. This is equivalent to interpreting, executing, performing or running in the traditional sense. The DGI makes it possible to concurrently develop and modify an application while it is being run. Development and modification may apply to all subsequent runs of an application, a subset of subsequent runs, or to the current instance of the application being run. This feature of the OPG and DGI facilitates the rapid development and long-term maintenance of enterprise-wide application systems. User access to this dynamic development capability is tightly controlled in a production environment by the OPG-DGI multi-level security and access system. In addition to data entered or changed via terminals, the system also accepts input data to application processes in various digital formats, including industry standards such as XML. Note this is a graph interpreter of object process graphs as opposed to a dynamic analysis of Object Process Graphs offered by Jochen Quante and Rainer Koschke.

Application Controller Viewer

The **Application Controller Viewer** consists of an *Application Controller* and an *Application Viewer*. The Application Controller Viewer together with the OPG and DGI form the *Dynamic Model–view–controller* application structure. This dynamic application structure manifests running instances of OPG applications. The Application Controller controls the running of applications by: processing input data and instructions/selections from users or other systems; initiating the display or output of information via the Application Viewer, commanding the DGI to initiate application state transitions and controlling the import and export of OPG application descriptions. The Application Controller is the controller component of the Dynamic Model View Controller.

The Application Viewer is the view component of the Dynamic Model View Controller. It receives display/output commands from the Application Controller to render OPG application data on a display media. The rendered application data provides a means of interacting with the applications via selection and input controls, and a way to view and update their content.

The third element of the Dynamic Model View Controller, is the integrated OPG and DGI, which functions as the model component. In addition to providing application control and view capabilities, the Application Controller Viewer provides a WYSIWYG (What You See Is What You Get) editor for editing application data display windows. Changes made to an application's windows can affect all executions of an application, a single window within an application or a particular instance of running an application.

Application Editor System

The **Application Editor System (AES)** is a fully-featured editing environment for defining and updating OPG applications. It renders graphical representations of OPG applications that closely correlate to actual OPG application structures. The graphical representations include symbols and structures that developers can manipulate with editing commands entered via a keyboard, computer mouse or other input device. One example of an AES implementation is the Web-based client–server application, PointDragon.

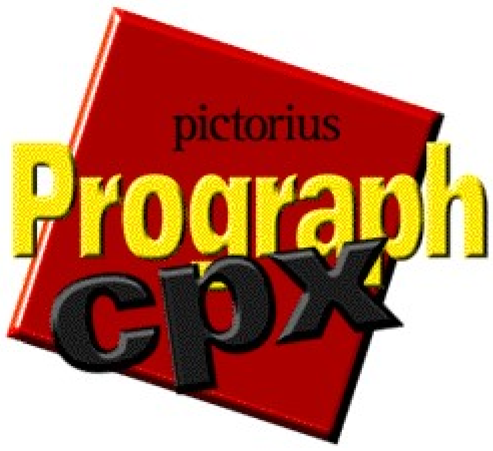
The Application Editor System is implemented with the Model View Controller structure. It interfaces with the OPG-DGI via the OPGAPI.

Object Process Graph Application Program Interface

The **Object Process Graph Application Program Interface (OPGAPI)** is a standard set of functions and associated parameters implemented by an OPG Translator. It is composed of OPG and DGI components that enable non-OPG applications, such as, the Application Editor System, to interact with an OPG application. It includes functions for starting, stopping, defining and modifying OPG applications. It also includes functions for inputting data to OPG applications and reporting their outputs.

Prograph

Prograph



Paradigm	multi-paradigm: object-oriented, visual, dataflow
Appeared in	1983
Designed by	Acadia University
Developer	Various
Major implementations	Prograph CPX, Marten
Influenced by	functional programming, dataflow diagrams
OS	Cross-platform: Classic MacOS, Microsoft Windows, Mac OS X

License	Proprietary

Prograph is a visual, object-oriented, dataflow, multiparadigm programming language that uses iconic symbols to represent actions to be taken on data. Commercial Prograph software development environments such as Prograph Classic and Prograph CPX were available for the Apple Macintosh and Windows platforms for many years but were eventually withdrawn from the market in the late 1990s. Support for the Prograph language on Mac OS X has recently reappeared with the release of the Marten software development environment.

History

Research on Prograph started at Acadia University in 1982 as a general investigation into dataflow languages, stimulated by a seminar on functional languages conducted by Michael Levin. Diagrams were used to clarify the discussion, leading to the insight: "since the diagrams are clearer than the code, why not make the diagrams themselves executable!" Thus Prograph - Programming in Graphics - was born as a visual dataflow language. This work was led by Dr. Tomasz Pietrzykowski, with Stan Matwin and Thomas Muldner co-authoring early papers. From 1983 to 1985, research prototypes were built on a Three Rivers PERQ graphics workstation (in Pascal, with the data visualized as fireballs moving down datalinks), and a VAX with a Tektronix terminal, and an experimental compiler was programmed in an IBM PC. This work was continued at Technical University of Nova Scotia by Pietrzykowski and Dr. Philip Cox, including a version done in Prolog.

In 1985, work began on a commercialisable prototype on the Macintosh, the only widely available, low-priced computer with high-level graphics support available at the time. In early 1986, this prototype was taken over by *The Gunakara Sun Systems* (later renamed to *TGS Systems*) for commercialisation, TGS formerly being a consulting firm formed by Pietrzykowski at Acadia University. Working with Pietrzykowski and Cox, Terry Kilshaw hired and managed the original development team, with Jim Laskey as the lead developer. In 1987 Mark Szpakowski suggested the merger of object-orientation with visual dataflow, creating an "objectflow" system. After almost four years of development, the first commercial release, v1.2, was introduced at the OOPSLA conference in New Orleans in October 1989. This product won the 1989 MacUser Editor's Choice Award for Best Development Tool. Version 2.0, released in July 1990, added a compiler to the system.

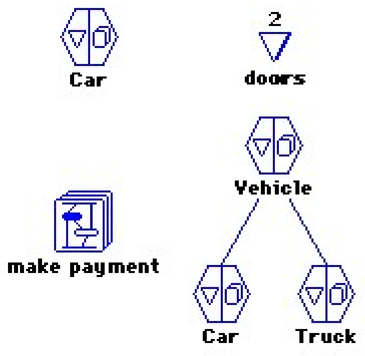
TGS changed its name to *Prograph International* (PI) in 1990. Although sales were slow, development of a new version, *Prograph CPX* (*Cross-Platform eXtensions*) was undertaken in 1992, that was intended to build fully cross-platform applications. This version was released in 1993, and was immediately followed by development of a client-server application framework. Despite increasing sales, the company was unable to sustain operating costs, and following a failed financing attempt in late 1994, went into receivership in early 1995.

As the receivership proceeded, the management and employees of PI formed a new company, *Pictorius*, which acquired the assets of PI. Shortly afterwards, development of a Windows version of Prograph CPX was begun. Although it was never formally released, versions of Windows Prograph were regularly made available to Prograph CPX customers, some of whom ported existing applications written in Macintosh Prograph, with varying degrees of success.

After management changes at the new company, emphasis shifted from tools development to custom programming and web application development. In April 2002 the web development part of the company was acquired by the Paragon Technology Group of Bermuda and renamed Paragon Canada. The Pictorius name and rights to the Prograph source code were retained by McLean Watson Capital, a Toronto-based investments firm which had heavily funded Pictorius. A reference to Pictorius appeared for a time on the former's Portfolio page, but has since disappeared. The Windows version of CPX was later released for free use, and was available for some time for download from the remnants of the Pictorius website (link below).

A group of Prograph users ("Prographers") calling themselves "The Open Prograph Initiative" (OPI) formed in the late 1990s with the goal of keeping Prograph viable in the face of OS advances by Apple and Microsoft. For a time the group also sought to create a new open-source visual programming language to serve as Prograph's successor, but with the advent of Andescotia's Marten visual programming environment, participation in the group essentially ceased.

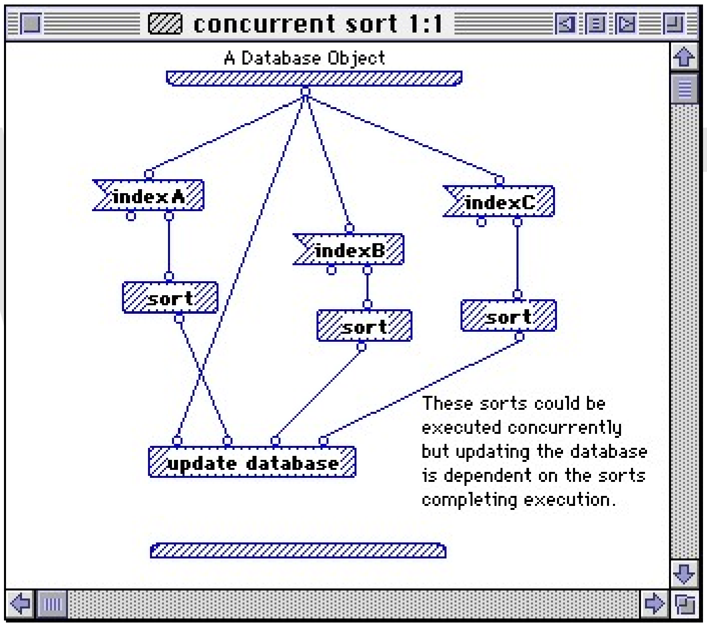
Description



Prograph objects. All images courtesy MacTech

During the 1970s program complexity was growing considerably, but the tools used to write programs were generally similar to those used in the 1960s. This led to problems when working on larger projects, which would become so complex that even simple changes could have side effects that are difficult to fully understand. Considerable research into the problem led many to feel that the problem was that existing programming systems focused on the logic of the program, while in reality the purpose of a program was to manipulate data. If the data being manipulated is the important aspect of the program, why isn't the data the "first class citizen" of the programming language? Working on that basis, a number of new programming systems evolved, including object-oriented programming and dataflow programming.

Prograph took these concept further, introducing a combination of object-oriented methodologies and a completely visual environment for programming. Objects are represented by hexagons with two sides, one containing the data fields, the other the methods that operate on them. Double-clicking on either side would open a window showing the details for that object; for instance, opening the variables side would show class variables at the top and instance variables below. Double-clicking the method side shows the methods implemented in this class, as well as those inherited from the superclass. When a method itself is double-clicked, it opens into another window displaying the logic.

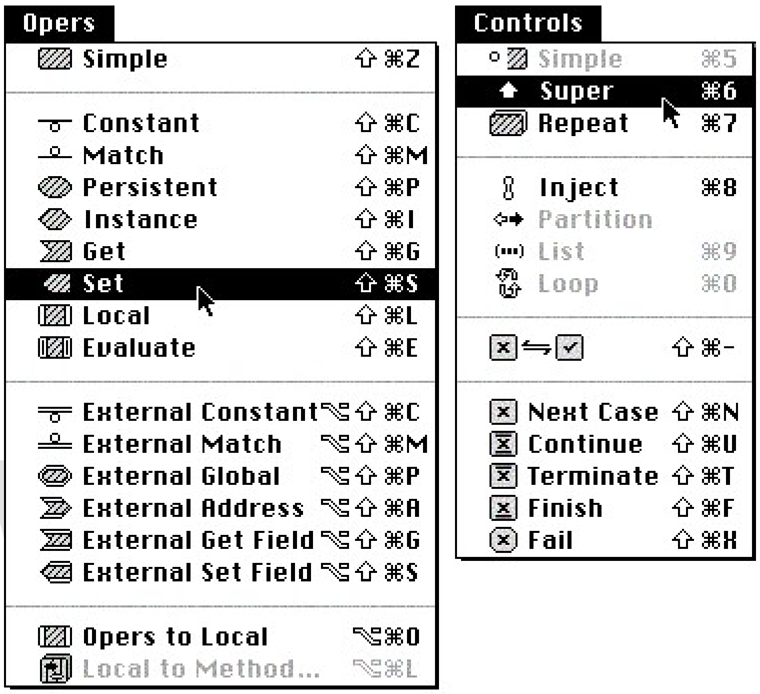


These sorts could be executed concurrently but updating the database is dependent on the sorts completing execution.

Prograph database operation. *Method implementation*

In Prograph a method is represented by a series of icons, each icon containing an instructions (or group of them). Within each method the flow of data is represented by lines in a directed graph. Data flows in the top of the diagram, passes through various instructions, and eventually flows back out the bottom (if there is any output).

Several features of the Prograph system are evident in this picture of a database sorting operation. The upper bar shows that this method, `concurrent sort`, is being passed in a single parameter, `A Database Object`. This object is then fed, via the lines, into several operations. Three of these extract a named index (`indexA` etc.) from the object using the `getter` operation (the unconnected getter output passes on the "whole" object), and then passes the extracted index to a sort operation. The output of these sort operations are then passed, along with a reference to the original database, to the final operation, `update` `database`. The bar at the bottom of the picture represents the outputs of this method, and in this case there are no connections to it and so this method does not return a value. Also note that although this is a method of some class, there is no `self`; if self is needed, it can be provided as an input or looked up.



Prograph operators and controls

In a dataflow language the operations can take place as soon as they have valid inputs for all of their connections. That means, in traditional terms, that each operation in this method could be carried out at the same time. In the database example, all of the sorts could take place at the same time if the computer were capable of supplying the data. Dataflow languages tend to be inherently concurrent, meaning they are capable of running on multiprocessor systems "naturally", one of the reasons that it garnered so much interest in the 1980s.

Loops and branches are constructed by modifying operations with annotations. For instance, a loop that calls the `doit` method on a list of input data is constructed by first dragging in the doit operator, then attaching the loop modifier and providing the list as the input to the loop. Another annotation, "injection", allows the method itself to be provided as an input, making Prograph a dynamic language to some degree.

Execution

The integrated Prograph development and execution environment also allowed for visual debugging. The usual breakpoint and single-step mechanisms were supported. Each operation in a data-flow diagram was visually highlighted as it executed. A tooltip-like mechanism displayed data values when the mouse was held over a data-link when stopped in debug mode. Visual display of the execution stack allowed for both roll-back and roll-forward execution. For many users the visual execution aspects of the language were as important as its edit-time graphical facilities.

The most important run-time debugging feature was the ability to change the code on the fly while debugging. This allowed for a truly unique development approach allowing bugs to be fixed while debugging without the need to recompile.

Critique

Several problems with the Prograph system are also evident in this method implementation.

Prograph code could be commented using labels. In initial versions, the majority of the included classes were unlabeled. It was often necessary to consult the documentation to determine the proper inputs to a method. This was largely addressed in subsequent versions, but the methods were never documented to the point that the comments explained how and why the methods worked.

Developers had to pay attention to routing of wiring, and to commenting inputs and outputs, to keep their diagrammatic code clean. In the Prograph Database Operation example above, two of the paths cross because one of the wires from the input bar must flow to a certain input on the update operation. This could be avoided by simply dragging and repositioning the 'indexA' and 'sort' icons to be inside the leftmost wire, but in general terms there was no way to avoid this sort of literal spaghetti code.

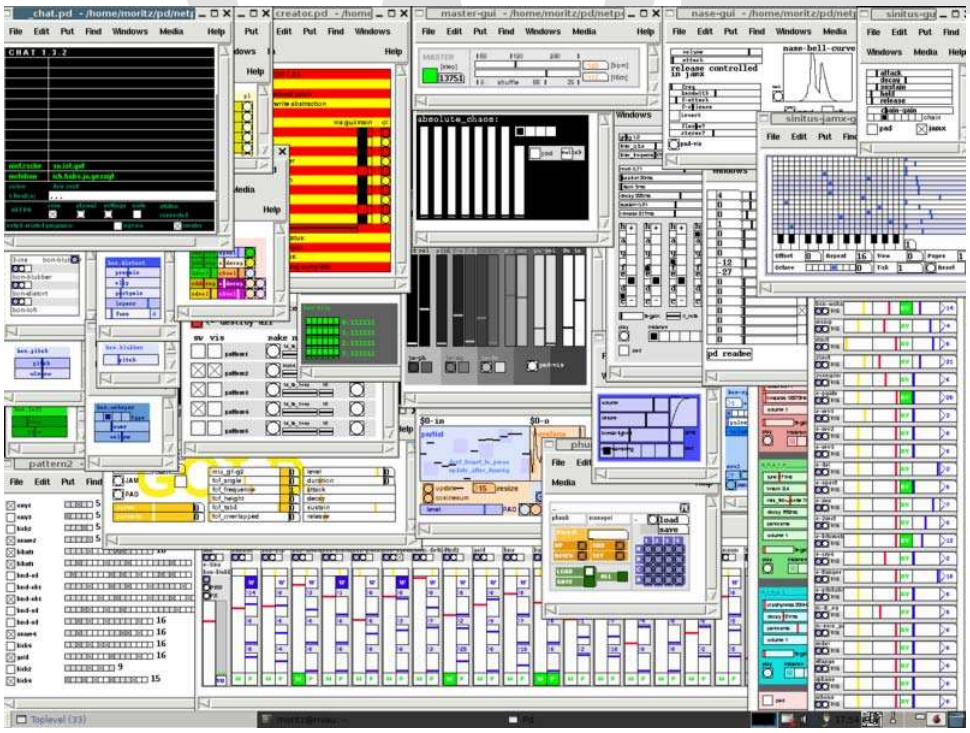
Another problem was a profusion of windows. When moving around the Prograph workspace, the IDE generally required a new window to be opened to see the contents of methods.

Chapter 7

Pure Data and Scratch (Programming Language)

Pure Data

Pure Data



Pure Data with many patches open (netpd project)

Original author(s)	Miller Puckette
Stable release	0.42.5 (Extended version)/0.42.6 (Vanilla version) / September 16, 2010; 2 months ago (Extended version)/August 30, 2010; 3

	months ago (Vanilla version)
Operating system	Cross-platform
Platform	Cross-platform
Available in	C
License	SIBSD

Pure Data

Paradigm	Dataflow
Appeared in	1996
Stable release	0.42 (May 13, 2009; 18 months ago)
Influenced by	Patcher
OS	Cross-platform
License	SIBSD

Pure Data (or **Pd**) is a visual programming language developed by Miller Puckette in the 1990s for the creation of interactive computer music and multimedia works. Though Puckette is the primary author of the software, Pd is an open source project and has a large developer base working on new extensions to the program. It is released under a license similar to the BSD license. It runs on GNU/Linux, Mac OS X, iPhoneOS, Android and Windows. There are older ports for FreeBSD and IRIX.

Pd is very similar in scope and design to Puckette's original Max program (developed while he was at IRCAM), and is to some degree interoperable with Max/MSP, the commercial successor to the Max language.

With the addition of the Graphics Environment for Multimedia (GEM) external, and externals designed to work with it (like Pure Data Packet / PiDiP (for Linux, OSX), framestein for Windows, GridFlow (as n-dimensional matrix processing, for Linux, OSX, Windows), it is possible to create and manipulate video, OpenGL graphics, images, etc. in realtime with seemingly endless possibilities for interactivity with audio, external sensors, etc.

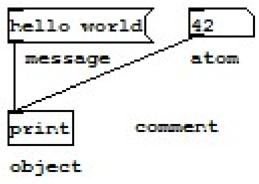
Additionally, Pd is natively designed to enable live collaboration across networks or the internet, allowing musicians connected via LAN or even in disparate parts of the globe to create music together in real time.

Similarities to Max

Both Pd and Max are arguably examples of Dataflow programming languages. In such languages, functions or "objects" are linked or "patched" together in a graphical environment which models the flow of the control and audio. Unlike the original version of Max, however, Pd was always designed to do control-rate and audio processing on the host CPU, rather than offloading the synthesis and signal processing to a DSP board (such as the Ariel ISPW which was used for Max/FTS). Pd code forms the basis of David Zicarelli's MSP extensions to the Max language to do software audio processing.

Like Max, Pd has a modular code base of *externals* or objects which are used as building blocks for programs written in the software. This makes the program arbitrarily extensible through a public API, and encourages developers to add their own control and audio routines, either in the C programming language or, with the help of other externals, in Python, Scheme, Lua, Tcl, and many other languages as well. However, Pd is a programming language in its own right. Modular, reusable units of code written natively in Pd, called "patches" or "abstractions", are used as standalone programs and freely shared among the Pd user community, and no other programming skill is required to use Pd effectively.

Language features



Pd's four text objects: message, atom, object, and comments.

Like Max, Pd is a data flow programming language. As with most DSP software, there are two primary rates at which data is passed: sample (audio) rate, usually at 44100 samples per second, and control rate, at 1 block per 64 samples. Control messages and audio signals generally flow from the top of the screen to the bottom between "objects" connected via inlets and outlets.

Pd supports 4 basic types of text entities: messages, objects, atoms, and comments. Atoms are the most basic unit of data in Pd, and they consist of either a float, a symbol, or a pointer to a datastructure. (In Pd, all numbers are stored as 32-bit floats). Messages are composed of one or more atoms and provide instructions to objects. A special type of message with null content is called a *bang* is used to initiate events and push data into flowing, much like pushing a button.

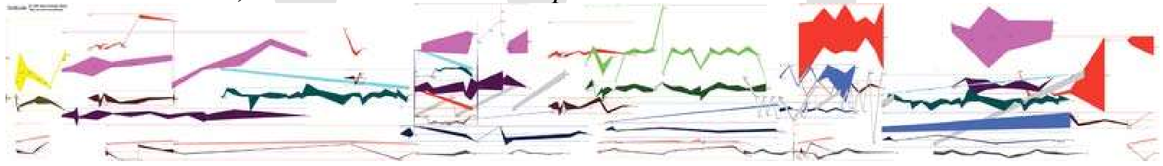
Pd's native objects range from the basic mathematical, logical, and bitwise operators found in every programming language, to general and specialized audio-rate DSP functions (designated by a tilde (~) symbol), such as wavetable oscillators, the fft~, and a range of standard filters. Data can be loaded from file, read in from an audio board, MIDI, via Open Sound Control (OSC) through a Firewire, USB, or network connection, or generated on the fly, and stored in tables, which can then be read back and used as audio signals or control data.

Data structures

One of the key innovations in Pd over its predecessors has been the introduction of graphical data structures, which can be used in a large variety of ways, from composing musical scores, sequencing events, to creating visuals to accompany Pd patches or even extending Pd's GUI.

Living up to Pd's name, data structures enable Pd users to create arbitrarily complex static as well as dynamic or animated graphical representations of musical data. Much like C structs, Pd's structs are composed of any combination of floats, symbols, and array data, which can be used as parameters to describe the visual appearance of the data structure or, conversely, to control messages and audio signals in a Pd patch. In Puckette's words:

Pd is designed to offer an extremely unstructured environment for describing data structures and their graphical appearance. The underlying idea is to allow the user to display any kind of data he or she wants to, associating it in any way with the display. To accomplish this Pd introduces a graphical data structure, somewhat like a data structure out of the C programming language, but with a facility for attaching shapes and colors to the data, so that the user can visualize and/or edit it. The data itself can be edited from scratch or can be imported from files, generated algorithmically, or derived from analyses of incoming sounds or other data streams.
—Miller Puckette, *Pd Documentation Chapter 2 — 2.9. Data structures*



Score for Hans-Christoph Steiner's *Solitude*, created using Pd's data structures.

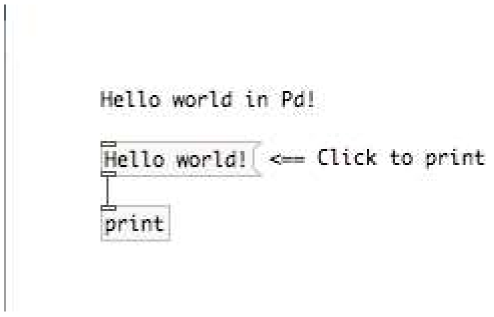
Projects that use Pd

Pd has been used as the basis of a number of projects, as a prototyping language and a sound engine. The table interface called reactable and the iPhone app rjdj both embed Pd as a sound engine.

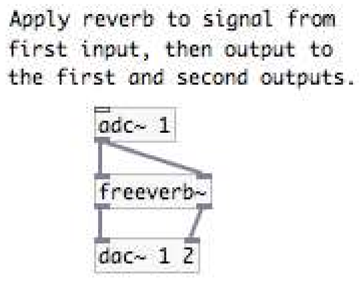
Pd has been used for prototyping audio for video games by a number of audio designers. For example, EAPd is the internal version of Pd that is used at Electronic Arts (EA). It has also been embedded into EA Spore

Pd has also been used for networked performance, in the NRCI (Networked Resources for Collaborative Improvisation) Library.

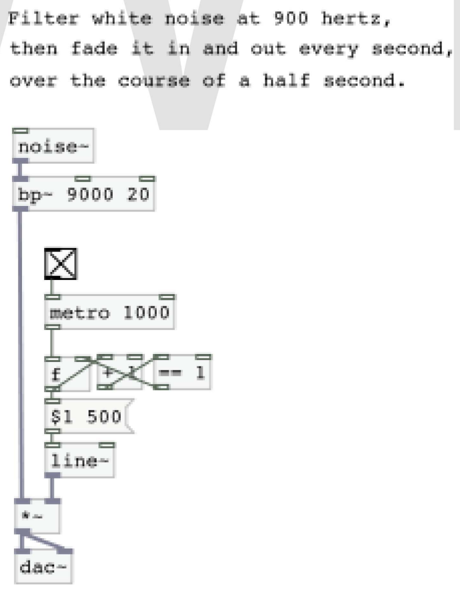
Code examples


Hello world in Pd!

Pd example 1


Apply reverb to signal from
first input, then output to
the first and second outputs.

Pd example 2


Filter white noise at 900 hertz,
then fade it in and out every second,
over the course of a half second.

Pd example 3

The first example is a hello world program in Pd.

The second patch applies reverb to the incoming signal from channel 1, then outputs it on channels 1 and 2.

The last, more complicated patch filters white noise at 9000 hertz (with a Q of 20), then fades it in and out each second over the course of a half second. As in all of Pd, time is measured in milliseconds, therefore the '1000' is one second and the '500' is a half second.

Scratch (programming language)

Scratch

Paradigm	object-oriented, educational
Appeared in	2007
Designed by	Mitchel Resnick
Developer	MIT Media Lab Lifelong Kindergarten Group
Stable release	1.4 (July 2, 2009)
Typing discipline	dynamic
Major implementations	Scratch
Influenced by	Logo, Smalltalk, HyperCard, StarLogo, AgentSheets, Etoys
Implementation language	Squeak
License	Proprietary (source code available for noncommercial reuse)
Usual file extensions	.sb

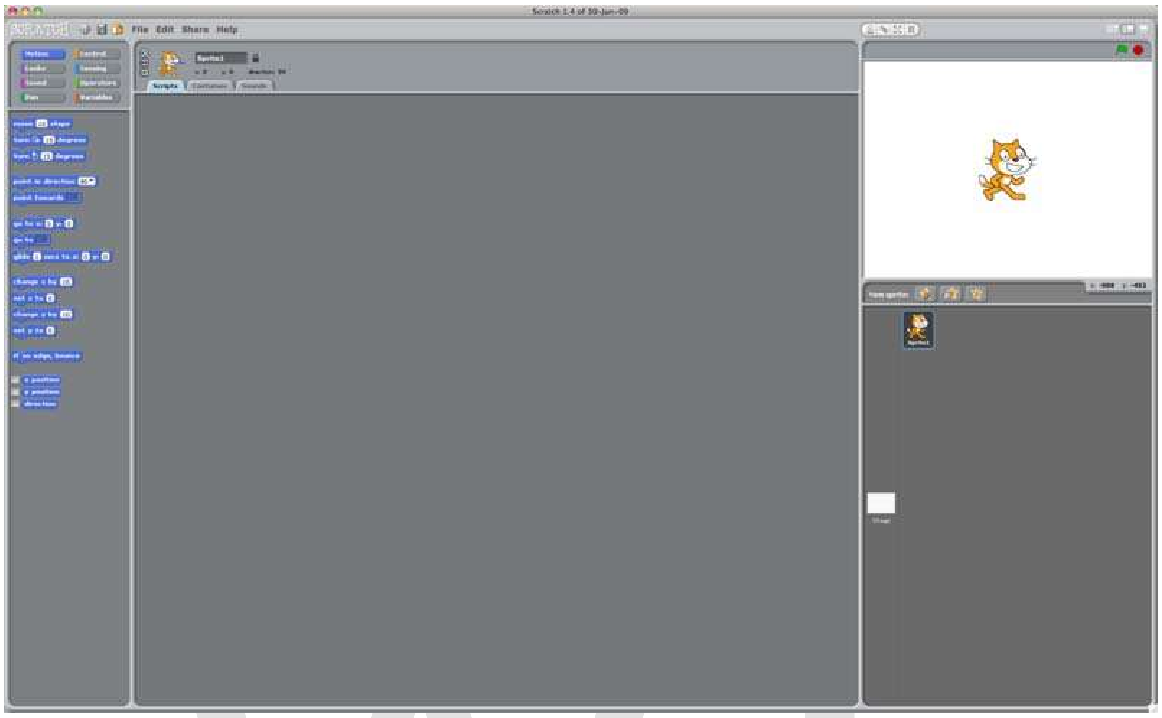
Scratch is an educational programming language that allows people of any experience background and age to experiment with the concepts of fully versatile computer programming by using an alterable GUI. It is developed by the Lifelong Kindergarten group at the MIT Media Lab by a team led by Mitchel Resnick and first appeared in the summer of 2007. Scratch can be installed and freely redistributed on any Windows, Mac OS X or Linux computer. The source code is made available under a license that allows modifications for non-commercial uses.

The name Scratch is derived from the turntablist technique of scratching, and refers to both the language and its implementation. The similarity to musical "scratching" is the easy reusability of pieces: in Scratch all the objects, graphics, sounds, and scripts can be easily imported to a new program and combined in new ways allowing beginners to get quick results and be motivated to try further.

Languages and environments

Scratch is used worldwide in many different settings: schools, museums, community centers, and homes. It is intended especially for 6- to 16-year-olds, but people of all ages have used Scratch. For example, younger children can create projects with their parents or older siblings, and college students use Scratch in some introductory computer science classes.

In designing the language, the creators' main priority was to make the language and development environment intuitive and easily learned by children who had no previous programming experience. There is a strong contrast between the powerful multimedia functions and multi-threaded programming style and the rather limited scope of the Scratch programming language.

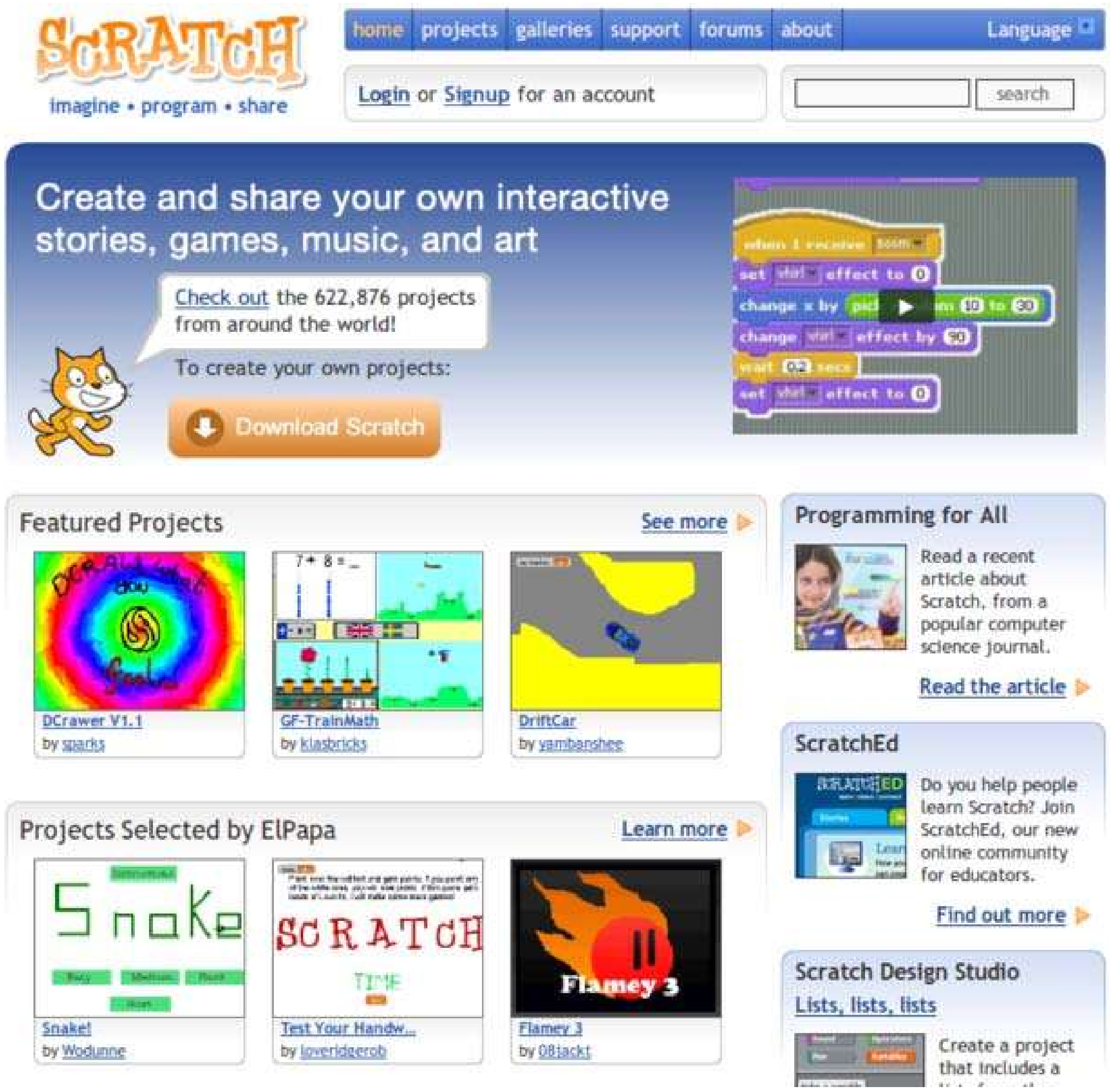


Screenshot of Scratch 1.4's development environment at startup (running on Apple Inc.'s Mac OS X Snow Leopard)

The user interface for the Scratch development environment divides the screen into several panes: on the left is the blocks palette, in the middle the current sprite info and scripts area, and on the right the stage and sprite list. The blocks palette has code fragments (called "blocks") that can be dragged onto the scripts area to make programs. To keep the palette from being too big, it is organized into 8 groups of blocks: movement, looks, sound, pen, control, sensing, operators, and variables. In versions 1.3.1 and lower, *operators* was named *numbers*.

Empirical studies were made of various features—those that interfered with intuitive learning were discarded, while those that encouraged beginners and made it easy for them to explore and learn were kept. Some of the results are surprising, making Scratch quite different from other teaching languages (such as BASIC, Logo, or Alice). For example, multi-threaded code with message passing is fundamental to Scratch, but it has no procedures or file Input/Output (I/O) and only supports one-dimensional arrays, known as Lists. Floating point scalars and strings are supported as of version 1.4, but with limited string manipulation capability.

Online community



Login or Signup for an account

Create and share your own interactive stories, games, music, and art

Check out the 622,876 projects from around the world!

To create your own projects:

Featured Projects — See more

DCrawer V1.1 by sparks

GF-TrainMath by klasbricks

DriftCar by yambanshee

Projects Selected by ElPapa — Learn more

Snake! by Wodunne

Test Your Handw... by loveridgerob

Flamey 3 by 08jackt

Programming for All

Read a recent article about Scratch, from a popular computer science journal.

Read the article

ScratchEd

Do you help people learn Scratch? Join ScratchEd, our new online community for educators.

Find out more

Scratch Design Studio

Lists, lists, lists

Create a project that includes a

Screenshot of the Scratch website

The Scratch slogan is "Imagine · Program · Share." The emphasis on sharing and the social aspects of creativity are an important part of the pedagogy for Scratch. Programs are not seen as black boxes, but as objects for remixing to make new projects. Scratch programs can be uploaded directly from the development environment to the Scratch website, where other members of the Scratch community can download them (including the full source code) for learning or for remixing into new projects. Members can also comment, tag, "love" others' projects and share ideas. The projects published in the Scratch website are licensed under a Creative Commons attribution and are played in a Java applet known as the Scratch Player. The Scratch Player allows Scratch programs to be run from almost any browser. As of December 15, 2009 the community had more than 408,227 registered members, 95,033 of them had uploaded projects, with a total of 796,359 projects shared by the whole community. The website receives close to 7 million page views per month. In 2009, the site reached a total of 500,000 scratch projects, and in 2010, 1,000,000. The website frequently establishes "Scratch Design Studio" challenges to encourage creation and sharing by providing users with a basic design concept. Examples have been creating projects that have an undersea theme or that play a song. There are also local Scratch websites in places such as Portugal and the United Arab Emirates. In 2008, the Scratch online community platform (named "ScratchR") received an honorary mention in the Ars Electronica Prix. There is also an online community for educators, called ScratchEd.

Scratch Mods

Recently, many derivations of Scratch have been created using the Source Code of version 1.4. These programs are a variation of Scratch that normally include a few extra 'blocks', or minor changes to the GUI (Graphic User Interface). Some of these are BYOB (by Jens), Panther (by the Panther Team), Streak (By Billyedward), Black Leopard (by midnightleopard), and Slash (by Billybob-Mario) (based on BYOB). There are also many more modifications.

Chapter 8

VisSim and Visual Basic

VisSim

VisSim

VisSim Viewer icon

Paradigm	Modular, Visual Programming, Simulation language
Appeared in	1989
Developer	Visual Solutions
Stable release	Version 7 (2008)
Influenced by	C, Laboratory Workbench, AVS (Advanced Visualization System)
OS	Windows, Linux
Usual file extensions	.VSM

VisSim is a visual block diagram language for simulation of dynamical systems and model based design of embedded systems. It is developed by Visual Solutions of Westford, Massachusetts.

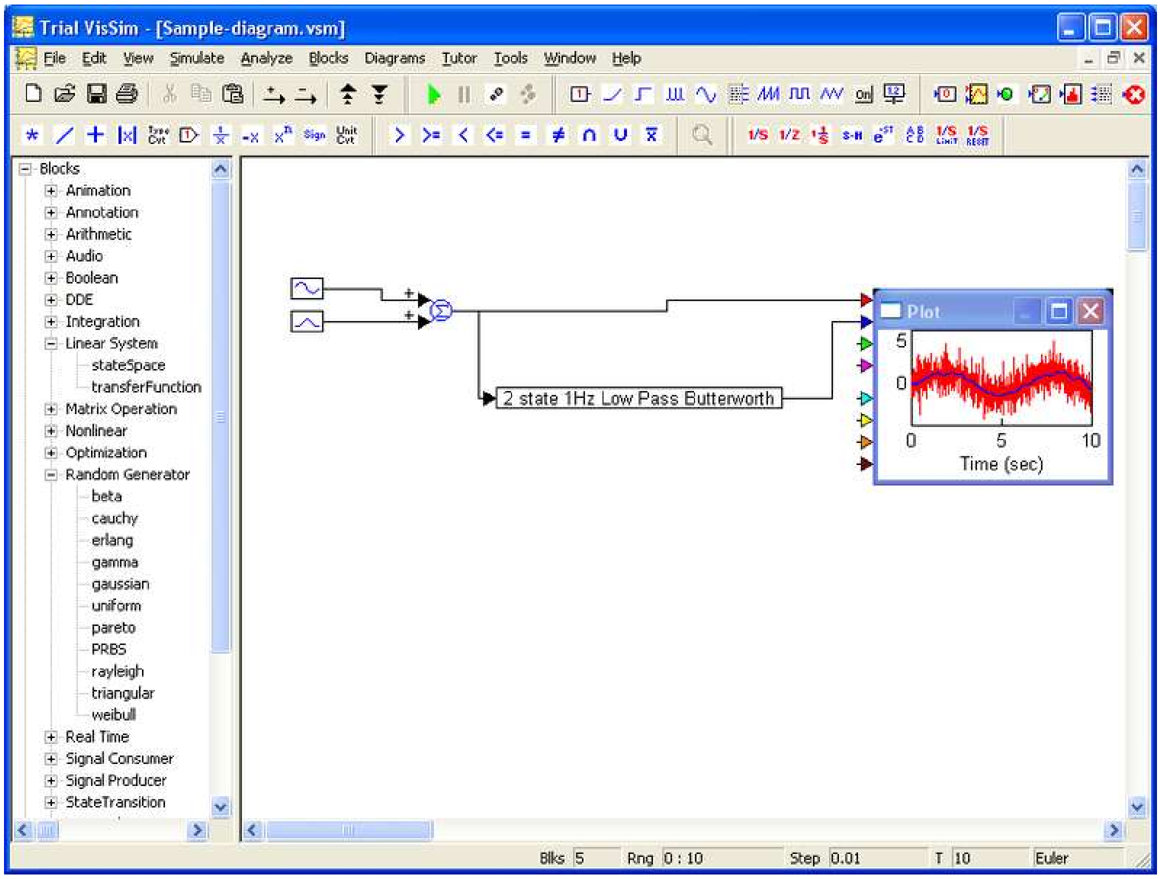
Applications

VisSim is widely used in control system design and digital signal processing for multidomain simulation and design. It includes blocks for arithmetic, Boolean, and transcendental functions, as well as digital filters, transfer functions, numerical integration and interactive plotting. The most commonly modeled systems are aeronautical, biological/medical, digital power, electric motor, electrical, hydraulic, mechanical, process, thermal/HVAC and econometric.

Academic use

Accredited educational institutions are allowed to site license VisSim v3.0 at no cost. The latest versions of VisSim and add-ons are also available to students and academic institutions at greatly reduced pricing.

Distributing VisSim models



VisSim viewer screenshot with sample model.

A read-only version of the software, VisSim Viewer is available free of charge and provides a way for people not licensed to use VisSim to run VisSim models. This program is intended to allow models to be more widely shared while preserving the model in its published form. The viewer will execute any VisSim model, and only allows changes to block and simulation parameters to illustrate different design scenarios. Sliders and buttons may be activated if included in the model.

Code generation

The "VisSim/C-Code" add-on generates ANSI C code for the model. This is most useful during development of embedded systems; when the logic of the model has been thoroughly tested, the C-code can be generated and 'flashed' to the target. The VisSim generated code is efficient and readable, making it easy to use as a development platform for embedded systems. VisSim's author served on the X3J11 ANSI C committee and wrote several C compilers, in addition to co-authoring a book on C. This deep understanding of ANSI C, and the nature of the resulting machine code when compiled, is the key to the code generator's efficiency. VisSim can target small 16-bit fixed point systems like the Texas Instruments MSP430, using only 740 bytes flash and 64 bytes of RAM for a small closed-loop Pulse-width modulation (PWM) actuated system, as well as allowing very high control sample rates over 500 kHz on larger 32-bit floating point processors like the Texas Instruments 150 MHz F28335.

Use of model-based development

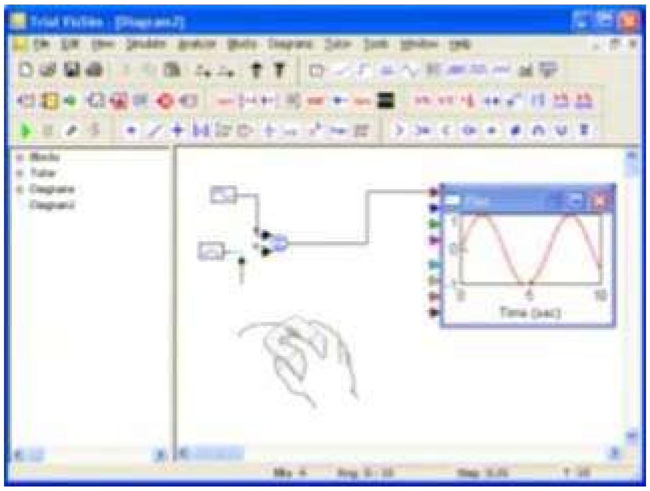
The technique of simulating system performance off-line, and then generating code from the simulation is known as "model-based development". Model-based development for embedded systems is becoming widely adopted for production systems because it shortens development cycles for hardware development in the same way that Model-driven architecture shortens production cycles for software development.

Model building is a visual way of describing a situation. In an engineering context, instead of writing and solving a system of equations, model building involves using visual "blocks" to solve the problem. The advantage of using models is that in some cases problems which appear difficult if expressed mathematically may be easier to understand when represented pictorially.

VisSim uses a hierarchical composition to create nested block diagrams. A typical model would consist of "virtual plants" composed of various VisSim "layers", combined if necessary with custom blocks written in C or FORTRAN. A virtual controller can be added and tuned to give desired overall system response. User interface elements such as sliders and buttons allow control of what-if analysis for operator training or controller tuning.

Although VisSim was originally designed for use by control engineers, it can be used for any type of mathematical model. An example is work done by Steve Keen, an economist.

VisSim optional features



Screenshots show the simulation of a sine function in VisSim. Noise is added to the model, then filtered out using a Butterworth filter. The signal traces of the sine function with noise and filtered noise are first shown together, and then shown in separate windows in the plot block.

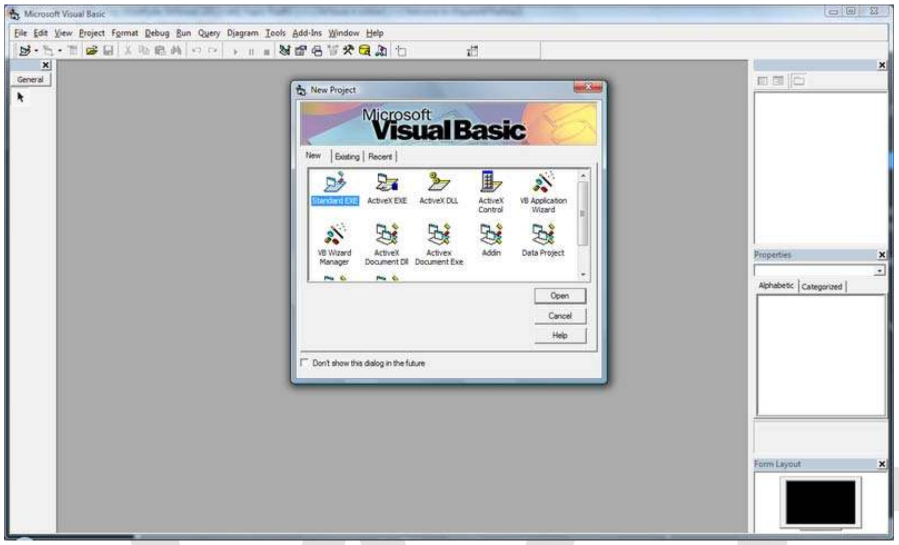
This video size: 50% (320x240 pixels)

Other size: 100% (640x480 pixels)

- Physical layer communication system simulation (modulators, encoders, PLLs, Costas Loop, BPSK, QPSK, DQPSK, QAM, Bit Error Rate (BER), Eye Diagram, Viterbi algorithm, Reed-Solomon, etc.)
- Frequency domain analysis (Bode plot, Root locus, Nyquist plot)
- CAN bus (Controller-area network) packet read and write
- Automatic C programming language code generation
- Electric motor simulation library for AC induction, Brushless DC, and Stepper motors
- Neural networks
- OPC (OLE for process control) client gives read and write of OPC tags for real-time simulation of SCADA/HMI virtual plants
- Global optimization of system parameters
- Real-time analog signal and digital I/O under Windows
- Fixed-point arithmetic blockset for bit-true simulation and code generation
- Embedded system targeting for Texas Instruments C2000 and MSP430 chips. Supports on-chip peripherals like serial ports, CAN, PWM, Quadrature Encoder Pulse (QEP), Event Capture, Serial Peripheral Interface Bus (SPI), I²C, Analog-to-digital converter (ADC), Digital-to-analog converter (DAC), and GPIO.

Visual Basic

Visual Basic



Paradigm	Object-based and Event-driven
Appeared in	1991
Developer	Microsoft
Stable release	VB6 (1998)
Typing discipline	Static, strong
Influenced by	QuickBASIC
Influenced	Visual Basic .NET, Gambas, REALbasic, Basic4ppc
OS	Microsoft Windows, MS-DOS

Visual Basic (**VB**) is the third-generation event-driven programming language and integrated development environment (IDE) from Microsoft for its COM programming model. Visual Basic is relatively easy to learn and use.

Visual Basic was derived from BASIC and enables the rapid application development (RAD) of graphical user interface (GUI) applications, access to databases using Data Access Objects, Remote Data Objects, or ActiveX Data Objects, and creation of ActiveX controls and objects. Scripting languages such as VBA and VBScript are syntactically similar to Visual Basic, but perform differently.

A programmer can put together an application using the components provided with Visual Basic itself. Programs written in Visual Basic can also use the Windows API, but doing so requires external function declarations.

The final release was version 6 in 1998. Microsoft's extended support ended in March 2008 and the designated successor was Visual Basic .NET (now known simply as Visual Basic).

Language features

Like the BASIC programming language, Visual Basic was designed to be easily learned and used by beginner programmers. The language not only allows programmers to create simple GUI applications, but can also develop complex applications. Programming in VB is a combination of visually arranging components or controls on a form, specifying attributes and actions of those components, and writing additional lines of code for more functionality. Since default attributes and actions are defined for the components, a simple program can be created without the programmer having to write many lines of code. Performance problems were experienced by earlier versions, but with faster computers and native code compilation this has become less of an issue.

Although programs can be compiled into native code executables from version 5 onwards, they still require the presence of runtime libraries of approximately 1 MB in size. This runtime is included by default in Windows 2000 and later, but for earlier versions of Windows like 95/98/NT it must be distributed together with the executable.

Forms are created using drag-and-drop techniques. A tool is used to place controls (e.g., text boxes, buttons, etc.) on the form (window). Controls have attributes and event handlers associated with them. Default values are provided when the control is created, but may be changed by the programmer. Many attribute values can be modified during run time based on user actions or changes in the environment, providing a dynamic application. For example, code can be inserted into the form resize event handler to reposition a control so that it remains centered on the form, expands to fill up the form, etc. By inserting code into the event handler for a keypress in a text box, the program can automatically translate the case of the text being entered, or even prevent certain characters from being inserted.

Visual Basic can create executables (EXE Files), ActiveX controls, or DLL files, but is primarily used to develop Windows applications and to interface database systems. Dialog boxes with less functionality can be used to provide pop-up capabilities. Controls provide the basic functionality of the application, while programmers can insert additional logic within the appropriate event handlers. For example, a drop-down combination box will automatically display its list and allow the user to select any element. An event handler is called when an item is selected, which can then execute additional code created by the programmer to perform some action based on which element was selected, such as populating a related list.

Alternatively, a Visual Basic component can have no user interface, and instead provide ActiveX objects to other programs via Component Object Model (COM). This allows for server-side processing or an add-in module.

The language is garbage collected using reference counting, has a large library of utility objects, and has basic object oriented support. Since the more common components are included in the default project template, the programmer seldom needs to specify additional libraries. Unlike many other programming languages, Visual Basic is generally not case sensitive, although it will transform keywords into a standard case configuration and force the case of variable names to conform to the case of the entry within the symbol table. String comparisons are case sensitive by default, but can be made case insensitive if so desired.

The Visual Basic compiler is shared with other Visual Studio languages (C, C++), but restrictions in the IDE do not allow the creation of some targets (Windows model DLLs) and threading models.

Characteristics

Visual Basic has the following traits which differ from C-derived languages:

- Multiple assignment available in C language is not possible. A = B = C does not imply that the values of A, B and C are equal. The boolean result of "Is B = C?" is stored in A. The result stored in A would therefore be either false or true.
- Boolean constant `True` has numeric value −1. This is because the Boolean data type is stored as a 16-bit signed integer. In this construct −1 evaluates to 16 binary 1s (the Boolean value `True`), and 0 as 16 0s (the Boolean value `False`). This is apparent when performing a `Not` operation on a 16 bit signed integer value 0 which will return the integer value −1, in other words `True = Not False`. This inherent functionality becomes especially useful when performing logical operations on the individual bits of an integer such as `And`, `Or`, `Xor` and `Not`. This definition of `True` is also consistent with BASIC since the early 1970s Microsoft BASIC implementation and is also related to the characteristics of CPU instructions at the time.
- Logical and bitwise operators are unified. This is unlike some C-derived languages (such as Perl), which have separate logical and bitwise operators. This again is a traditional feature of BASIC.
- Variable array base. Arrays are declared by specifying the upper and lower bounds in a way similar to Pascal and Fortran. It is also possible to use the Option Base statement to set the default lower bound. Use of the Option Base statement can lead to confusion when reading Visual Basic code and is best avoided by always explicitly specifying the lower bound of the array. This lower bound is not limited to 0 or 1, because it can also be set by declaration. In this way, both the lower and upper bounds are programmable. In more subscript-limited languages, the lower bound of the array is not variable. This uncommon trait does exist in Visual Basic .NET but not in VBScript.

`OPTION BASE` was introduced by ANSI, with the standard for ANSI Minimal BASIC in the late 1970s.

- Relatively strong integration with the Windows operating system and the Component Object Model. The native types for strings and arrays are the dedicated COM types, BSTR and SAFEARRAY.
- Banker's rounding as the default behavior when converting real numbers to integers with the `Round` function. `? Round(2.5, 0)` gives 2, `? Round(3.5, 0)` gives 4.
- Integers are automatically promoted to reals in expressions involving the normal division operator (`/`) so that division of one integer by another produces the intuitively correct result. There is a specific integer divide operator (`\`) which does truncate.
- By default, if a variable has not been declared or if no type declaration character is specified, the variable is of type `Variant`. However this can be changed with Deftype statements such as `DefInt`, `DefBool`, `DefVar`, `DefObj`, `DefStr`. There are 12 `Deftype` statements in total offered by Visual Basic 6.0. The default type may be overridden for a specific declaration by using a special suffix character on the variable name (`#` for Double, `!` for Single, `&` for Long, `%` for Integer, `$` for String, and `@` for Currency) or using the key phrase `As (type)`. VB can also be set in a mode that only explicitly declared variables can be used with the command `Option Explicit`.

History

VB 1.0 was introduced in 1991. The drag and drop design for creating the user interface is derived from a prototype form generator developed by Alan Cooper and his company called *Tripod*. Microsoft contracted with Cooper and his associates to develop Tripod into a programmable form system for Windows 3.0, under the code name *Ruby* (no relation to the Ruby programming language).

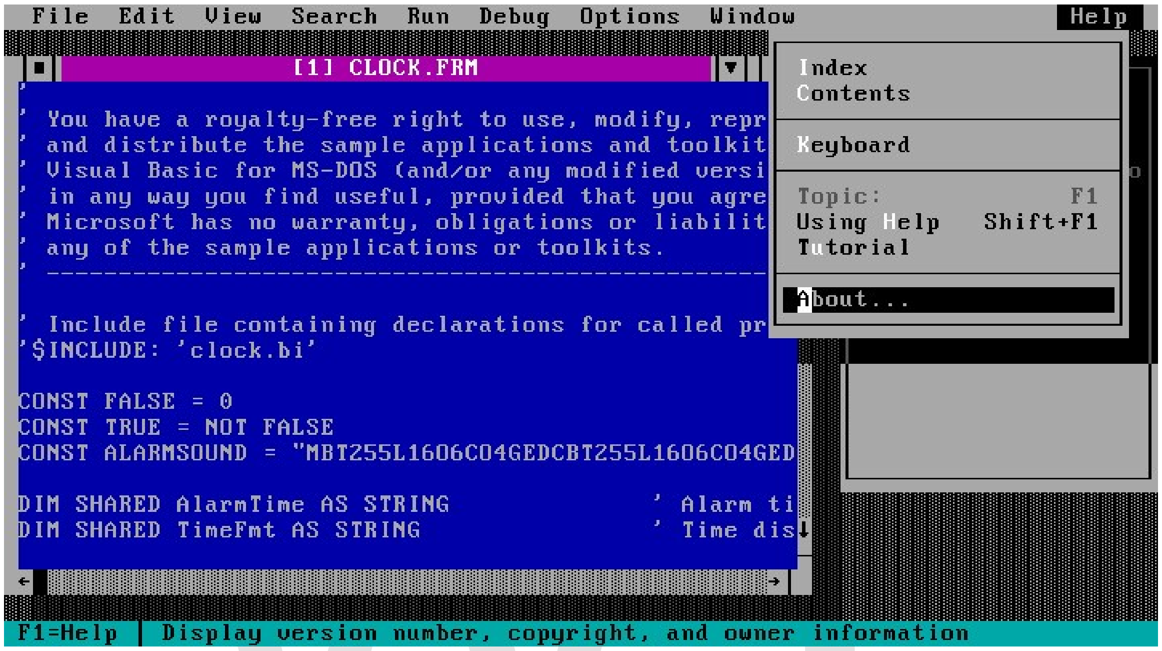
Tripod did not include a programming language at all. Microsoft decided to combine Ruby with the Basic language to create Visual Basic.

The Ruby interface generator provided the "visual" part of Visual Basic and this was combined with the "EB" Embedded BASIC engine designed for Microsoft's abandoned "Omega" database system. Ruby also provided the ability to load dynamic link libraries containing additional controls (then called "gizmos"), which later became the VBX interface.

Timeline

- Project 'Thunder' was initiated
- Visual Basic 1.0 (May 1991) was released for Windows at the Comdex/Windows World trade show in Atlanta, Georgia.

- Visual Basic 1.0 for DOS was released in September 1992. The language itself was not quite compatible with Visual Basic for Windows, as it was actually the next version of Microsoft's DOS-based BASIC compilers, QuickBASIC and BASIC Professional Development System. The interface used a Text user interface, using extended ASCII characters to simulate the appearance of a GUI.



```
'
' You have a royalty-free right to use, modify, repr
' and distribute the sample applications and toolkit
' Visual Basic for MS-DOS (and/or any modified versi
' in any way you find useful, provided that you agre
' Microsoft has no warranty, obligations or liabilit
' any of the sample applications or toolkits.
'  -----------------------------------------------
'
' Include file containing declarations for called pr
'$INCLUDE: 'clock.bi'

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST ALARMSOUND = "MBT255L1606C04GEDCBT255L1606C04GED

DIM SHARED AlarmTime AS STRING              ' Alarm ti
DIM SHARED TimeFmt AS STRING                ' Time dis
```

```
F1=Help | Display version number, copyright, and owner information
```

Visual Basic for MS-DOS

- Visual Basic 2.0 was released in November 1992. The programming environment was easier to use, and its speed was improved. Notably, forms became instantiable objects, thus laying the foundational concepts of class modules as were later offered in VB4.
- Visual Basic 3.0 was released in the summer of 1993 and came in Standard and Professional versions. VB3 included version 1.1 of the Microsoft Jet Database Engine that could read and write Jet (or Access) 1.x databases.
- Visual Basic 4.0 (August 1995) was the first version that could create 32-bit as well as 16-bit Windows programs. It also introduced the ability to write non-GUI classes in Visual Basic. Incompatibilities between different releases of VB4 caused installation and operation problems. While previous versions of Visual Basic had used VBX controls, Visual Basic now used OLE controls (with files names ending in .OCX) instead. These were later to be named ActiveX controls.
- With version 5.0 (February 1997), Microsoft released Visual Basic exclusively for 32-bit versions of Windows. Programmers who preferred to write 16-bit programs were able to import programs written in Visual Basic 4.0 to Visual Basic 5.0, and Visual Basic 5.0 programs can easily be converted with Visual Basic 4.0. Visual Basic 5.0 also introduced the ability to create custom user controls, as well as the ability to compile to native Windows executable code, speeding up calculation-intensive code execution. A free, downloadable Control Creation Edition was also

released for creation of ActiveX controls. It was also used as an introductory form of Visual Basic: a regular .exe project could be created and run in the IDE, but not compiled.

- Visual Basic 6.0 (Mid 1998) improved in a number of areas including the ability to create web-based applications. VB6 has entered Microsoft's "non-supported phase" as of March 2008. Although the Visual Basic 6.0 development environment is no longer supported, the runtime is supported on Windows Vista, Windows Server 2008 and Windows 7.
- Mainstream Support for Microsoft Visual Basic 6.0 ended on March 31, 2005. Extended support ended in March 2008. In response, the Visual Basic user community expressed its grave concern and lobbied users to sign a petition to keep the product alive. Microsoft has so far refused to change their position on the matter. Ironically, around this time (2005), it was exposed that Microsoft's new anti-spyware offering, Microsoft AntiSpyware (part of the GIANT Company Software purchase), was coded in Visual Basic 6.0. Its replacement, Windows Defender, was rewritten as C++ code.

VB DOS Logo

Derivative languages

Microsoft has developed derivatives of Visual Basic for use in scripting. Visual Basic itself is derived heavily from BASIC, and subsequently has been replaced with a .NET platform version.

Some of the derived languages are:

- Visual Basic for Applications (VBA) is included in many Microsoft applications (Microsoft Office), and also in many third-party products like SolidWorks, AutoCAD, WordPerfect Office 2002, ArcGIS, Sage Accpac ERP, and Business Objects Desktop Intelligence. There are small inconsistencies in the way VBA is implemented in different applications, but it is largely the same language as VB6 and uses the same runtime library.
- VBScript is the default language for Active Server Pages. It can be used in Windows scripting and client-side web page scripting. Although it resembles VB in syntax, it is a separate language and it is executed by vbscript.dll as opposed to the VB runtime. ASP and VBScript should not be confused with ASP.NET which uses the .NET Framework for compiled web pages.

- Visual Basic .NET is Microsoft's designated successor to Visual Basic 6.0, and is part of Microsoft's .NET platform. Visual Basic.Net compiles and runs using the .NET Framework. It is not backwards compatible with VB6. An automated conversion tool exists, but fully automated conversion for most projects is impossible.
- StarOffice Basic is a Visual Basic compatible interpreter included in StarOffice suite, developed by Sun Microsystems.
- Gambas is a Visual Basic inspired free software programming language. It is not a clone of Visual Basic, but it does have the ability to convert Visual Basic programs to Gambas.

Performance and other issues

Earlier counterparts of Visual Basic (prior to version 5) compiled the code to P-Code only. The P-Code is interpreted by the language runtime, also known as a virtual machine. The benefits of P-Code include portability and smaller binary file sizes, but it usually slows down the execution, since having a runtime adds an additional layer of interpretation. However, small amounts of code and algorithms can be constructed to run faster than compiled native code.

Visual Basic applications require Microsoft Visual Basic runtime MSVBVMxx.DLL, where xx is the relevant version number, either 50 or 60. MSVBVM60.dll comes as standard with Windows in all editions after Windows 98 while MSVBVM50.dll comes with all editions after Windows 95. A Windows 95 machine would however require inclusion with the installer of whichever dll was needed by the program.

Visual Basic 5 and 6 can compile code to either native or P-Code.

Criticisms levelled at Visual Basic editions prior to VB.NET include:

- Versioning problems associated with various runtime DLLs, known as DLL hell
- Poor support for object-oriented programming
- Inability to create multi-threaded applications, without resorting to Windows API calls
- Inability to create Windows services
- Variant types have a greater performance and storage overhead than strongly typed programming languages
- Dependency on complex and fragile COM Registry entries
- The development environment is no longer supported by Microsoft.

Legacy development and support

All versions of the Visual Basic development environment from 1.0 to 6.0 have been retired and are now unsupported by Microsoft. The associated runtime environments are unsupported too, with the exception of the Visual Basic 6 core runtime environment, which will be officially supported by Microsoft for the lifetime of Windows 7. Third party components that shipped with Visual Studio 6.0 are not included in this support statement. Some legacy Visual Basic components may still work on newer platforms, despite being unsupported by Microsoft and other vendors.

Development and maintenance development for Visual Basic 6 is possible on legacy Windows XP, Windows Vista and Windows 2003 using Visual Studio 6.0 platforms, but is unsupported. Documentation for Visual Basic 6.0, its application programming interface and tools is best covered in the last MSDN release before Visual Studio.NET 2002. Later releases of MSDN focused on .NET development and had significant parts of the Visual Basic 6.0 programming documentation removed. The Visual Basic IDE can be installed and used on Windows Vista, where it exhibits some minor incompatibilities which do not hinder normal software development and maintenance. As of August 2008, both Visual Studio 6.0 and the MSDN documentation mentioned above are available for download by MSDN subscribers.

Example code

Here is an example of the language: Code snippet that displays a message box "Hello, World!" as the window *Form* loads:

```
Private Sub Form_Load()
    ' Execute a simple message box that will say "Hello, World!"
    MsgBox "Hello, World!"
End Sub
```

Chapter 9

IIf, Visual Basic .NET and Visual DialogScript

IIf

In computing, **IIf** (an abbreviation for **Immediate if**) is a function in several editions of the Visual Basic programming language, related languages such as ColdFusion Markup Language (CFML), and on spreadsheets that returns one of its two parameters based on the evaluation of an expression. It is an example of a conditional expression, which is similar to a conditional statement.

Syntax

The syntax of the IIf function is as follows:

```
IIf(expr, truepart, falsepart)
```

All three parameters are required:

- *expr* is the expression that is to be evaluated.
- *truepart* defines what the IIf function returns if the evaluation of *expr* returns true.
- *falsepart* defines what the IIf function returns if the evaluation of *expr* returns false.

Many languages have operators to accomplish the same purpose, generally referred to as ternary operators; the best known is ?:, as used in C, C++, and related languages. Some of the problems with the IIf function, as discussed later, do not exist with ternary operators, because the language is free to examine the type and delay evaluation of the operands, as opposed to simply passing them to a library function.

Examples

These examples evaluate mathematical expressions and return one of two strings depending on the outcome.

```
 result = IIf(5 < 10, "Yes it is", "No it isn't")      ' Returns "Yes it
is"
 result = IIf(2 + 2 = 5, "Correct", "Wrong")           ' Returns "Wrong"
```

Criticisms

Efficiency

Because `IIf` is a library function, it will always require the overhead of a function call, whereas a conditional operator will more likely produce inline code.

Furthermore, the data type of its arguments is `Variant`. If the function is called with arguments of other types (variables or literals), there will be additional overhead to convert these to `Variant`. There may also be additional overhead to check the argument types and convert one of them if they do not have the same type.

Side Effects

Another issue with `IIf` arises because it is a library function: unlike the C-derived conditional operator, both *truepart* and the *falsepart* will be evaluated regardless of which one is actually returned. Consider the following example:

```
value = 10
result = IIf(value = 10, TrueFunction, FalseFunction)
```

Although *TrueFunction* is the function intended to be called, `IIf` will cause both *TrueFunction* and *FalseFunction* to be executed.

Also consider this one:

```
a = 10
b = 0
result = IIf(b <> 0, a / b, 0)
```

While the programmer intends to avoid raising an error by performing a division by zero, whenever b is zero the error will actually happen. This is because the code in the snippet is to be read as

```
a = 10
b = 0
_temp1 = a / b ' Error if b = 0
_temp2 = 0
_temp3 = b <> 0
result = IIf(_temp3, _temp1 , _temp2)
```

This issue makes the IIf() call less useful than the ternary operator. To solve this issue, Microsoft developers had considered converting `IIf` to an intrinsic function; had this happened, the compiler would have been able to perform type inference and short-circuiting by replacing the function call with inline code.

Alternatives to IIf

In Visual Basic, IIf is not the sole way to evaluate and perform actions based on whether an expression is true or false.

The following example uses IIf:

```
result = IIf(x = y, value1, value2)
```

It could also be written in the following way, using standard conditional statements:

```
If x = y Then
  result = value1
Else
  result = value2
End If
```

The above example would also eliminate the problem of IIf evaluating both its *truepart* and *falsepart* parameters.

Visual Basic 2008 (VB 9.0) introduced a true ternary operator, called simply "If", which also eliminates this problem. Its syntax is similar to the IIf function's syntax:

```
result = If(x = y, value1, value2)
```

IIf in other programming languages

`$iif()` is also present in mIRC script, with similar syntax.

```
 alias testiif {
  %testiif = 0
  echo -a $iif(1,$testiif2,$testiif2) %testiif execution(s)
  unset %testiif
 }
 alias testiif2 { inc %testiif | return testing $!iif: }
```

calling `/testiif` will print out "testing $iif: 1 execution(s). mIRC's `$iif` acts more like C's `?:` than `IIf()` in VB since it won't pre-evaluate both.

`IIF()` is a function in dBase and xBase. (1992 and before.)

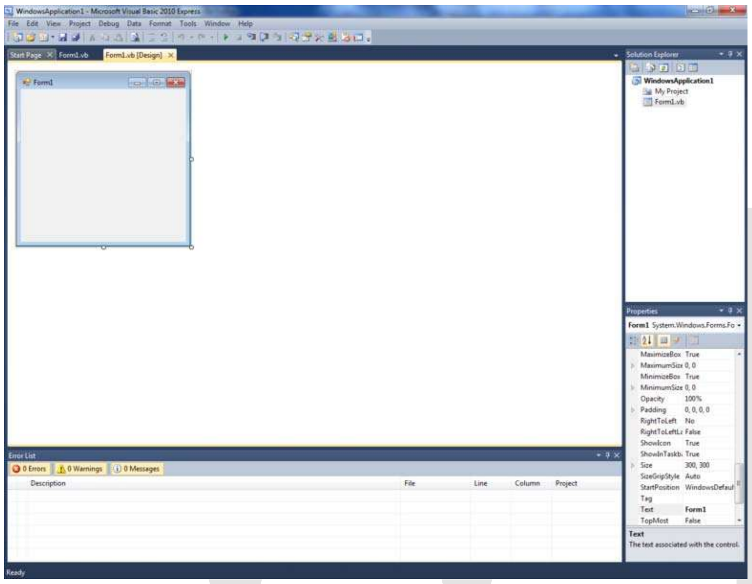
`iif()` is also a compiler magic function of Oxygene. It is not a real function and is at compile time unrolled to conditional statements.

```
var someString :=  iif(someInt > 35 , 'Large', 'Small');
```

In this example a new strong type string named "someString" is created (using Type inference) and the `iif` function will fill it depending on the outcome of the boolean expression.

Visual Basic .NET

Visual Basic .NET



Properties

Form1 System.Windows.Forms.Fo

MaximizeBox	True
MaximumSize	0, 0
MinimizeBox	True
MinimumSize	0, 0
Opacity	100%
Padding	0, 0, 0, 0
RightToLeft	No
RightToLeftL	False
ShowIcon	True
ShowInTaskb	True
Size	300, 300
SizeGripStyle	Auto
StartPosition	WindowsDefaul
Tag	
Text	Form1
TopMost	False

Text
The text associated with the control.

Paradigm	Structured, imperative, object-oriented and declarative
Appeared in	2001
Designed by	Microsoft
Developer	Microsoft
Stable release	2010 (10.0.30319.1) (12 April 2010; 8 months ago)
Typing discipline	static, strong, both safe and unsafe, nominative
Major implementations	Microsoft Visual Studio, Microsoft Visual Studio Express, .NET Framework SDK and Mono
Dialects	Microsoft Visual Studio .NET, .NET 2003, 2005, 2008, 2010
Influenced by	.NET Framework

Platform	CLR
OS	Microsoft Windows
License	Proprietary software (Commercial software or freeware)
Usual file extensions	`.vb` and `.vbs`

Visual Basic .NET (**VB.NET**) is an object-oriented computer programming language that can be viewed as an evolution of the classic Visual Basic (VB) which is implemented on the .NET Framework. Microsoft currently supplies two major implementations of Visual Basic: Microsoft Visual Studio, which is commercial software and Microsoft Visual Studio Express, which is free of charge.

Versions of Visual Basic .NET

There are four versions and five releases of Visual Basic .NET implemented by the Visual Basic Team.

Visual Basic .NET (VB 7)

The original Visual Basic .NET was released alongside Visual C# and ASP.NET in 2002. Significant changes broke backward compatibility with older versions and caused a rift within the developer community.

Visual Basic .NET 2003 (VB 7.1)

Visual Basic .NET 2003 was released with *version 1.1* of the .NET Framework. New features included support for the .NET Compact Framework and a better VB upgrade wizard. Improvements were also made to the performance and reliability of the .NET IDE (particularly the background compiler) and runtime. In addition, Visual Basic .NET 2003 was available in the *Visual Studio .NET Academic Edition* (VS03AE). VS03AE is distributed to a certain number of scholars from each country without cost.

Visual Basic 2005 (VB 8.0)

Visual Basic 2005 is the name used to refer to the update to Visual Basic .NET, Microsoft having decided to drop the .NET portion of the title.

For this release, Microsoft added many features, including:

- *Edit and Continue*
- Design-time expression evaluation.

- The *My* pseudo-namespace (overview, details), which provides:
    - easy access to certain areas of the .NET Framework that otherwise require significant code to access
    - dynamically-generated classes (notably *My.Forms*)
- Improvements to the VB-to-VB.NET converter
- The *Using* keyword, simplifying the use of objects that require the Dispose pattern to free resources
- *Just My Code*, which when debugging hides (steps over) boilerplate code written by the Visual Studio .NET IDE and system library code
- Data Source binding, easing database client/server development

The above functions (particularly *My*) are intended to reinforce Visual Basic .NET's focus as a rapid application development platform and further differentiate it from C#.

Visual Basic 2005 introduced features meant to fill in the gaps between itself and other "more powerful" .NET languages, adding:

- .NET 2.0 languages features such as:
    - generics
    - Partial classes, a method of defining some parts of a class in one file and then adding more definitions later; particularly useful for integrating user code with auto-generated code
    - Nullable Types
- XML comments that can be processed by tools like NDoc to produce "automatic" documentation
- Operator overloading
- Support for unsigned integer data types commonly used in other languages

'IsNot' operator patented

One other feature of Visual Basic 2005 is the `IsNot` operator that makes `'If X IsNot Y'` equivalent to `'If Not X Is Y'`, which gained notoriety when it was found to be the subject of a Microsoft patent application.

Visual Basic 2005 Express

Part of the Visual Studio product range, Microsoft created a set of free development environments for hobbyists and novices, the Visual Studio 2005 Express series. One edition in the series is Visual Basic 2005 Express Edition, which was succeeded by Visual Basic 2008 Express Edition in the 2008 edition of Visual Studio Express.

The Express Editions are targeted specifically for people learning a language. They have a streamlined version of the user interface, and lack more advanced features of the standard versions. On the other hand, Visual Basic 2005 Express Edition *does* contain the Visual Basic 6.0 converter, so it is a way to evaluate feasibility of conversion from older versions of Visual Basic.

Visual Basic 2008 (VB 9.0)

Visual Basic 9.0 was released together with the Microsoft .NET Framework 3.5 on 19 November 2007.

For this release, Microsoft added many features, including:

- A true conditional operator, "If(boolean, value, value)", to replace the "IIf" function.
- Anonymous types
- Support for LINQ
- Lambda expressions
- XML Literals
- Type Inference
- Extension methods

Visual Basic 2010 (VB 10.0)

In April 2010, Microsoft released Visual Basic 2010. Microsoft had planned to use the Dynamic Language Runtime (DLR) for that release but shifted to a co-evolution strategy between Visual Basic and sister language C# to bring both languages into closer parity with one another. Visual Basic's innate ability to interact dynamically with CLR and COM objects has been enhanced to work with dynamic languages built on the DLR such as IronPython and IronRuby. The Visual Basic compiler was improved to infer line continuation in a set of common contexts, in many cases removing the need for the " _" line continuation character. Also, existing support of inline Functions was complemented with support for inline Subs as well as multi-line versions of both Sub and Function lambdas. Another thing that is new is that, unlike in the previous versions, source code of a paused application in debug mode can no longer be edited.

Relation to older versions of Visual Basic (VB6 and previous)

Whether Visual Basic .NET should be considered as just another version of Visual Basic or a completely different language is a topic of debate. This is not obvious, as once the methods that have been moved around and that can be automatically converted are accounted for, the basic syntax of the language has not seen many "breaking" changes, just additions to support new features like structured exception handling and short-circuited expressions. Two important data type changes occurred with the move to VB.NET. Compared to VB6, the `Integer` data type has been doubled in length from 16 bits to 32 bits, and the `Long` data type has been doubled in length from 32 bits to 64 bits. This is true for all versions of VB.NET. A 16-bit integer in all versions of VB.NET is now known as a `Short`. Similarly, the Windows Forms GUI editor is very similar in style and function to the Visual Basic form editor.

The version numbers used for the new Visual Basic (7, 7.1, 8, 9, ...) clearly imply that it is viewed by Microsoft as still essentially the same product as the old Visual Basic.

The things that *have* changed significantly are the semantics—from those of an object-based programming language running on a deterministic, reference-counted engine based on COM to a fully object-oriented language backed by the .NET Framework, which consists of a combination of the Common Language Runtime (a virtual machine using generational garbage collection and a just-in-time compilation engine) and a far larger class library. The increased breadth of the latter is also a problem that VB developers have to deal with when coming to the language, although this is somewhat addressed by the *My* feature in Visual Studio 2005.

The changes have altered many underlying assumptions about the "right" thing to do with respect to performance and maintainability. Some functions and libraries no longer exist; others are available, but not as efficient as the "native" .NET alternatives. Even if they compile, most converted VB6 applications will require some level of refactoring to take full advantage of the new language. Documentation is available to cover changes in the syntax, debugging applications, deployment and terminology.

Comparative samples

The following simple example demonstrates similarity in syntax between VB and VB.NET. Both examples pop up a message box saying "Hello, World" with an OK button.

```
Private Sub Command1_Click()
    MsgBox "Hello, World"
End Sub
```

A VB.NET example, MsgBox or the MessageBox class can be used:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Msgbox("Hello, World")
    End Sub
End Class
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        MessageBox.Show("Hello, World")
    End Sub
End Class
```

- Both Visual Basic 6 and Visual Basic .NET will automatically generate the `Sub` and `End Sub` statements when the corresponding button is clicked in design view. Visual Basic .NET will also generate the necessary `Class` and `End Class` statements. The developer need only add the statement to display the "Hello, World" message box.
- Note that all procedure calls must be made with parentheses in VB.NET, whereas in VB6 there were different conventions for functions (parentheses required) and subs (no parentheses allowed, unless called using the keyword `Call`).

- Also note that the names `Command1` and `Button1` are not obligatory. However, these are default names for a command button in VB6 and VB.NET respectively.
- In VB.NET, the `Handles` keyword is used to make the sub `Button1_Click` a handler for the `Click` event of the object `Button1`. In VB6, event handler subs must have a specific name consisting of the object's name ("Command1"), an underscore ("_"), and the event's name ("Click", hence "Command1_Click").
- There is a function called `MsgBox` in the `Microsoft.VisualBasic` namespace which can be used similarly to the corresponding function in VB6. There is a controversy about which function to use as a best practice (not only restricted to showing message boxes but also regarding other features of the `Microsoft.VisualBasic` namespace). Some programmers prefer to do things "the .NET way", since the Framework classes have more features and are less language-specific. Others argue that using language-specific features makes code more readable (for example, using `int` (C#) or `Integer` (VB.NET) instead of `System.Int32`).
- In VB 2008, the inclusion of `ByVal sender as Object, ByVal e as EventArgs` has become optional.

The following example demonstrates a difference between VB6 and VB.NET. Both examples close the active window.

Classic VB Example:

```
Sub cmdClose_Click()
    Unload Me
End Sub
```

A VB.NET example:

```
Sub btnClose_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles btnClose.Click
    Me.Close()
End Sub
```

Note the 'cmd' prefix being replaced with the 'btn' prefix, conforming to the new convention previously mentioned.

Visual Basic 6 did not provide common operator shortcuts. The following are equivalent:

VB6 Example:

```
Sub Timer1_Timer()
    Me.Height = Me.Height - 1
End Sub
```

VB.NET example:

```
Sub Timer1_Tick(ByVal sender As Object, ByVal e As EventArgs) Handles
Timer1.Tick
    Me.Height -= 1
End Sub
```

Criticism

Long-time Visual Basic users have complained about Visual Basic .NET because initial versions dropped a large number of language constructs and user interface features that were available in VB6 (which is no longer sold by Microsoft), and changed the semantics of those that remained; for example, in VB.NET parameters are (by default) passed by value, not by reference. Detractors refer pejoratively to VB.NET as *Visual Fred* or *DOTNOT*. On 8 March 2005, a petition was set up in response to Microsoft's refusal to extend its mainstream support for VB6.

VB.NET's supporters state that the new language is in most respects more powerful than the original, incorporating modern object oriented programming paradigms in a more natural, coherent and complete manner than was possible with earlier versions. Opponents tend to respond that although VB6 has flaws in its object model, the cost in terms of redevelopment effort is too high for any benefits that might be gained by converting to VB.NET.

It is simpler to decompile languages that target Common Intermediate Language (CIL), including VB.NET, compared to languages that compile to machine code. Tools such as .NET Reflector can provide a close approximation to the original code due to the large amount of metadata provided in CIL.

Microsoft supplies an automated VB6-to-VB.NET converter with Visual Studio .NET, which has improved over time, but it cannot convert all code, and almost all non-trivial programs will need some manual effort to compile. Most will need a significant level of code refactoring to work optimally. Visual Basic programs that are mainly algorithmic in nature can be migrated with few difficulties; those that rely heavily on such features as database support, graphics, unmanaged operations or on implementation details are more troublesome.

In addition, the required runtime libraries for VB6 programs are provided with Windows 98 SE and above, while VB.NET programs require the installation of the significantly larger .NET Framework. The framework is included with Windows 7, Windows Vista, Windows XP Media Center Edition, Windows XP Tablet PC Edition, Windows Server 2008 and Windows Server 2003. For other supported operating systems such as Windows 2000 or Windows XP (Home or Professional Editions), it must be separately installed.

Microsoft's response to developer dissatisfaction has focused around making it easier to move new development and shift existing codebases from VB6 to VB.NET. Their latest offering is the VBRun website, which offers code samples and articles for:

- Using VB.NET to complete tasks that were common in VB6, like creating a print preview
- Integrating VB6 and VB.NET solutions (dubbed *VB Fusion*)

Cross-platform and open-source development

The creation of open-source tools for VB.NET development have been slow compared to C#, although the Mono development platform provides an implementation of VB.NET-specific libraries and a VB.NET 8.0 compatible compiler written in VB.NET, as well as standard framework libraries such as Windows Forms GUI library.

SharpDevelop and MonoDevelop are open-source alternative IDEs.

Examples

The following is a very simple VB.NET program, a version of the classic "Hello world" example created as a console application:

```
Module Module1

    Sub Main()
        Console.WriteLine("Hello, world!")
    End Sub

End Module
```

The effect is to write the text *Hello, world!* to the command line. Each line serves a specific purpose, as follows:

```
Module Module1
```

This is a module definition, a division of code similar to a class, although modules can contain classes. Modules serve as containers of code that can be referenced from other parts of a program.
It is common practice for a module and the code file, which contains it, to have the same name; however, this is not required, as a single code file may contain more than one module and/or class definition.

```
Sub Main()
```

This is the entry point where the program begins execution. *Sub* is an abbreviation of "subroutine."

```
Console.WriteLine("Hello, world!")
```

This line performs the actual task of writing the output. *Console* is a system object, representing a command-line interface and granting programmatic access to the operating system's standard streams. The program calls the *Console* method *WriteLine,* which causes the string passed to it to be displayed on the console. Another common method is using MsgBox (a *Message Box*).

This piece of code is a solution to Floyd's Triangle:

```
Module Module1
    Sub Main()
        Dim Rows As Integer
        Dim DisplayNumber As Integer = 1
        Dim check1 As Integer = 0
        Dim check2 As Integer = 0

        Do Until check1 = 1 : Try
                Console.Write("Enter a value for how many rows to be
displayed: ")
                Rows = Convert.ToInt32(Console.ReadLine)
                If Rows < 1 Then
Retry:              Console.WriteLine("Invalid, please try again")
                    check1 = 0
                Else : check1 = 1
                End If : Catch
                GoTo Retry
            End Try
        Loop : check1 = 1 : Do Until check2 = Rows
            If DisplayNumber = check1 * (check1 + 1) / 2 Then
                Console.WriteLine(DisplayNumber)
                Console.ReadLine()
                check1 += 1 : check2 += 1 : DisplayNumber += 1
            ElseIf DisplayNumber <> check1 * (check1 ^ 2 + 1) / 2 Then
                Console.Write(DisplayNumber & " ")
                DisplayNumber += 1
            End If
        Loop
    End Sub
End Module
```

Visual DialogScript

Visual DialogScript (VDS) is an interpreted programming language for Microsoft Windows. It can be used to create small, fast programs. VDS has a large number of dialog and graphical elements available to create professional looking programs. VDS programs have access to the Windows API; therefore, it is possible to write applications that can perform the same advanced tasks as other programming languages such as Visual Basic, C++, or Delphi.

Language

Unlike other programming languages, the syntax of VDS is very simple. Each command occupies one line, and has a plain English name that clearly describes its purpose. Variables are typeless, and can hold many kinds of information, for example, numbers or text. Functions are clearly distinguishable with names that start with '@', just like a spreadsheet.

The DialogScript language has a simple syntax not unlike MS-DOS batch language. It is designed for ease of use and efficiency when being interpreted by the run-time engine. There are 10 system variables, %0 to %9, which initially have the script file name in %0 and command line parameters in %1 through %9, just as in a batch file. There are also a further 26 user variables, %A to %Z. The contents of all variables (including system ones) can be changed once the script is running. There are now also 4032 global variables. These variables begin with %%, a letter, then alphanumerics plus underscores (e.g. %%my_variable_1.) There is no limit on the length of these user-defined variable names.

Syntax Examples

Comments:

- `# This is a single line comment`
- `REM This is a single line comment`

Simple Information Message Box:

- `info "This is the information text"`

Simple Warning Message Box:

- `warn "This is the warning text"`

Create a custom dialog box:

- `dialog create,<name>,<top pixel position>,<left pixel position>,<width in pixels>,<height in pixels>`

Write to the Windows Registry:

- `registry write,<root key>,<key>,<subkey>,<data>`

Display an input prompt dialog box, storing the result in the variable %A:

- `%A = @input("Please enter a value:")`

History

Visual DialogScript was originally created by Julian Moss of JM-Tech. Eventually, S.A.D.E. s.a.r.l., a French company, took over ownership and development of VDS, altering and improving upon its syntax. Currently, VDS is owned and developed by the British company Commercial Research Ltd.

Several versions of VDS have been released over time:

- Visual DialogScript 2.0
- Visual DialogScript 2.5
- Visual DialogScript 3.0
- Visual DialogScript 3.5
- Visual DialogScript 4
- Visual DialogScript 4.5
- Visual DialogScript 5
- Visual DialogScript 5.01
- Visual DialogScript 5.02
- Visual DialogScript 6

Currently Available Versions

There are several versions available for download:

1. **Personal Visual DialogScript (PVDS) 4:** This freeware version is intended for students and home PC users. The package includes a short tutorial and full online help which includes many example scripts. The software is not licensable for commercial use. This version is incapable of producing compiled executable files; however, compiling files is not necessary, as a script file can be executed directly by opening it from Windows Explorer on any system that has PVDS installed on it.
2. **Visual DialogScript 2.5 (16-Bit Edition):** This version marked the last release for Windows 3.1+ (16-bit).
3. **Visual DialogScript 5:** This version is for power users, business users, and professional developers who use—or are developing scripts for—Windows 95/98/ME or Windows NT/2000/XP. This version can create compiled executable files, and includes a royalty-free run-time license (once registered). Additional features include an icon editor and support for many add-on extensions. As a 32-bit program, it supports long filenames, task bar tray icons, unlimited length strings and string lists and the Windows Registry. This legacy version is now available for download for registered users and is not available for purchase.
4. **Visual DialogScript 6:** The newest version of Visual DialogScript improves upon Visual DialogScript 5 and adds full support for Windows Vista. Additionally, the registered version can now create standalone compiled executable files that do not require an external runtime file.