|  |
| --- |
| Circle Language Spec |

## Overview

This documentation attempts to describe a programming language called Circle. It is mostly about a diagram notation for visualizing computer code.

It makes an effort to explain how basic coding concepts are implemented in Circle language: concepts such as objects, classes, relations, interfaces, events, commands and parameters.

Related to this, several experimental versions of programming languages have been developed, but nothing ever entirely finished.

The documentation tries to cover the following topics:

### Overview

This overview, which attempts to briefly describe each topic.

*Introduction*

Might gently introduce what this Circle language looks like.

*Basic Diagram Elements*

Tries to give a more complete overview over the basic elements that the diagrams are composed of, such as circles, triangles, crosses, squares and lines. This documentation explains what they mean in diagrams. Later in the documentation more details about the diagram notation will be covered.

### Objects

This documentation aims to describe what objects are in computer technology and basic of concepts about objects. It also tries to shed light on drawing out objects using the Circle diagram notation.

### Classes

This documentation attempts to explain the concept of *classes*. A class is sort of like a special object, that describes the characteristics and behavior of other objects. The other objects might select the special object as their class and the idea is that their behavior is then bound to the rules set by that class. The foundation may be explained, but the concept of *static* is not really worked out yet.

### Relations

When an object connects to another object, it is like a relation between the two objects is established. This documentation tries to explain the concept of relations. Even though objects might relate to an arbitrary set of other objects, the concept can also be about how *classes* set the rules for the kinds of relations objects can have. Relations between classes form a sort of *model* of relations to which objects conform.

### System Objects

One possible application of how this language might run in practice, is one where systems are composed of *system objects*. System objects could be the *actual* objects, that a system runs on. This might be a very specific use of the language, but explaining it, seems to make generally usable things fall out. The main purpose of system objects might be, to manage *relations* between objects. System objects can also control *system aspects*, such as **Class**, **Interface**, **Execution** and **Data**. Aspects would be controlled through system commands. Then the story goes on with assignment commands, which would also be considered system commands. An assignment command copies an aspect from one object to another. The *System Objects* documentation alsotries to cover connectors, connections and different notational forms to control these aspects.

### Commands

Commands are considered executable objects. Other synonyms for commands are *methods*, *procedures*, *routines*, *subs* and *functions*. In the Circle Language Spec all those executable objects are called *commands*. In some programming languages, commands are not objects, but completely separate constructs, that you might not really exchange with objects. But in Circle an attempt is made to see them as objects, that just so happen to be executable. Commands do seem to have special (creation) behavior. This documentation tries to evaluate this special behavior and their rules.

### Parameters

Parameters are like instructions passed along with a command that make the command behave differently. This documentation tries to explain parameters as a relationshipbetween commands and objects. It entertains an idea of how *command parameters* and *class methods* are interchangeable: one and the same concept. It is not sure, if this idea will still be in there in the future. Also the diagram notation of commands and parameters is covered.

### Globality

(The made-up word 'globality' is based on the concept of 'global' from Visual Basic. But in the future the term *Modules* might used instead.)

A module is like a little world in which objects and commands live. A module can be a site, a computer program, a library, a name space or any other kind of module. A module groups together functionality or data. A module can be thought of as an object with a special property. This is the suggested description of that property: Anything directly inside the module can be directly referenced from anywhere within that module.

### Execution Control

The idea of execution control statements is that they would control the execution flow of a program: the order in which things are executed. Several execution control statements are proposed. Key examples of them are: **If** and **For**. An attempt is made to cover a complete set of execution control statements, to explore what would be needed from the diagram notation in that area.

### Black Boxes

It could be said, that *Black Boxes* have to do with *Public and Private*. *Private* would make something only accessible inside an object. *Public* would make something accessible outside an object as well. That way, the inner workings of an object may be hidden away and this would make it possible to only work with the input and output.

The notation might be changed compared to what is described here now. The current version tends to focus around something, that could be called the *friend* notation. This is because the current design of the notation was made with a different view on objects and commands. Commands were considered more independent and not necessarily considered part of an object. Which commands can access the private contents of which objects would be less obvious, when a command is not really part of a specific object. The effect seemed to be, that no command gets access to the private contents of an object, unless that command is declared a *friend* bythe object. Friend declaration would create a tighter bond between the command and the object. Thinking of commands as no more than object, seems to have affected the design of black boxing.

In the future, a simpler black box method might be proposed: more like how it works in other programming languages. The friend notation might still be preserved for specific use-cases.

### Interfaces

This chapter attempts to cover the basic concept of interfaces along with their diagram notation. Up till now the idea of interfaces was left out of the stories in some cases. This chapter tries to catch up with that. There are also efforts to explain: interface assignment, interface reference and relations that form through the use of interfaces. As interfaces can be applied to *objects*, interfaces can also be applied to *commands*.

### Events

Events might be thought of as notification calls. Call receivers would subscribe to an event. The call sender would call the receivers when it would see fit. Another variation of the events concept, that the world seems to adopts, is called *‘the observer pattern’*. The sender would decide what the message looks like. The receiver would supply a command, that should have an interface, specified by the sender.

The concept of events can be seen as a combination of constructs: an array of command references, a command interface, then supporting the command interface, registering the command inside the array of command references and then calling all the commands in the array at specific times.

Even though it could be established by just a combination of other constructs, it seems to be such a usable concept, that Circle explicitly attempts to define it as a separate construct, like some other programming languages seem to do as well.

The documentation also tries to show, how events look in the Circle diagram notation.

### Inheritance

This part of the documentation attempts to covers the concept of inheritance. This concept seems to be intricately part of some oriented programming languages for a while now. But the new computer language takes it a little bit further. In other languages one class can inherit from another class, taking over all its characteristics before the derived class gets extended or altered. Inheritance goes further than that in the new computer language. Next to *class inheritance* (the traditional form of inheritance), there is also *object inheritance*. This means, that one object takes over the characteristics of a specific other object. In fact, the other object melts together with the new object. When you change base members of the new object, the base object’s data also changes. This is callled *object inheritance*. There are even more forms of inheritance. A collection can be inherited from, so that a whole existing table of items can be extended with new columns. This is called *list inheritance*. A whole *system* can also inherit from another system, taking over all of its characteristics, and the derived system can be altered, augmented, specialized and extended. This is called *system inheritance*. If the original system gets changed, this may also change the derived systems. So basically: everything can be inherited from and this can render specialized versions of any class, object, collection or module.

Such specialization can be also be applied to *commands*. The inheritancedocumentation also covers otherconcepts in the area of *specialization* and *extension*. Inheritance is very closely related to the concept of *interfaces*.

Those were the *finished* documentation folders. Now follow the documentation folders, that are not finished yet. The descriptions below, however, *are* finished. The list begins with the easiest ones to finish. The tougher ones to finish are mentioned last.

### Type Control

Objects can relate to any arbitrary set of other objects. However, when you set the *class* of an object, the relations are bound to the rules of the class. This is a form of type control. Another form of type control is when you fix the interface of an object reference. This limits the types of objects you can assign to the object reference. Only objects with a class that supports the interface, can then be assigned.

Binding an object to more rules is also called *explicit* declaration. Not binding an object or object reference to rules is called *generic* declaration. Both concepts are *type control*.

Type control is so present inside programming languages, that it seems programming languages can not do without type control. But actually, it is easier to make a programming language without any type control in it, than to program type control into it. You would think, that when a programming language has to support classes and interfaces, it actually requires the programming language to have all sorts of extra possibilities. But in reality, all the possibilities are already present.

At first there is no type control at all. Anything can reference anything and this results in all sorts of possibilities, that things will go wrong in a program. Type control only enforces restrictions. When type controls is implemented inside the new computer language, it is like nothing extra is offered, but only the ability to impose more restrictions to the possiblities already offered.

This documentation folder contains only one document, that contains a stack-up of ideas yet to be turned into good documentation. It also contains type control as originally worked out in an older design of a diagram notation language.

### Object Resolution

Object resolution takes place when a reference to one object is blocked out by another object, shadowing or overriding the original object. (It also applies to commands or methods.) A reference is set to one object, but instead the reference is sort of inconsentiously gravitated towards another object. There are multiple situations in which object resolution takes place, such as: overriding, overloading, shadowing, ambiguity, implicit conversions and default members. At first it was thought, that these kinds of situations were due to ambiguity and the textual nature of code, but in the new computer language when you draw out the situation in a diagram, they turn out to be more like automatically detoured object references.

This documentation is not finished up yet. It contains a lot of ideas, that are not entirely described yet, that are yet to be turned into readable documentation.

### Conditions

You can set rules for what kind of values can be filled in into an object and conditions for starting a procedure. You can do this by adding code to a procedure or to a Set procedure for the assignment of a value. That code then makes sure, that when certain conditions are not met, the execution of the rest of the procedure is refused. This is the low-level way to specify conditions. However, conditions are such an important concept in programming, that they are defined as a separate construct inside the new computer language. Every value object and every command can define a set of conditions, that have to be met or the value will not be assigned or the command will not be executed. The documentation is not worked out yet, and only contains a couple of loose ideas.

*Object Order*

By default, objects inside a collection are not sorted. The *Object Order* concept should allow sorting a list, and also should provide a proper diagram notation for it. This documentation folder is just a small throw-together of ideas, yet to be turned into good documentation. *Object Order* was not implemented yet inside versions 0.9 and 2.0 of the new computer language, because it was not needed at the time yet.

### Uncategorized

This folder contains some articles, for which a proper place was not found yet. There are not many articles inside this documentation folder.

*Other Diagram Topics*

The diagram should be fully automatically drawn out. The metrics and positioning of the shapes and lines are automatically determined. This folder also covers the different kinds of ways you can express systematics. The basic ideas about a diagram notation were already covered at the beginning of the new computer language documentation, because it was important to quickly introduce those. The diagram expression is also explained throughout the code concepts documentation. This part of the documentation will cover the remaining topics about the diagram expression.

The idea behind expression in general is that a program’s systematics are not described by text code, but text code as well as the diagram notation are a mere expression of systematics, stored in a binary way as interlinked objects. Much of this documentation folder is a mere throw-together of ideas or pieces of text, that came out of previous descriptions of new computer programming languages.

Text code and diagram expression can be complemented by any other form of expression. Displaying data in tabular form is another form of expression. Expressing data and commands in a graphical windows user interface is also an alternative form of expression of systematics. Furthermore, certain objects have specialized expressions. For instance a sound object may be expressed by playing its sound, but not all objects can be expressed by playing it as sound. All expressions are considered alternative expressions of systematics, just like text code and diagram code.

### Summaries of Other Languages

Contains summaries of other languages, mainly C and C++, in the form of bulleted lists. They were made around the year 2004 during an analysis of the grammar of C and C++ to compare them to the new computer language.

*Previous Versions*

This folder contains previous versions of descriptions of a new computer programming language.

**Ideas**

*The texts below are loose ideas, yet to be turned into good documentation.*

*Other Programming Languages*

Looking at the 'Subtext' programming language

- 'Subtext' implies that this is always possible, but ofcourse underlying procedures could be

- 'Subtext' code when it grows is really bound to become unoverviewable.

- His assumption that humans are not good at abstract thinking is just bullshit. Humans do nothing else.

- If he would give his functions better identifiers, you would already be better off.

J Code

Analyse the features of F# and find a place for it in my system.

http://pro.tweakers.net/nieuws/49986/microsoft-stopt-functionele-taal-f-in-visual-studio.html

JJ

Caché bestrijkt vele gebieden al, waar Code een oplossing voor biedt. Het relationeel en object georienteerd unifyen, dan zit daar eigenlijk al bijna compleet in.

Afgeleide containment relaties niet, diagrammen niet en applicatie feature generatie niet, en aspecten niet (ik zal naast standaard aspecten van JJ, zorgen dan je customer concepten kunt introduceren.) extended inheritance niet, kiezen tussen geheugen en schrijf niet, speciale security niet.

Dat soort concepten niet, maar het hele idee relationeel en object georienteerd gelijk trekken wel. Behalve echt 1 taal maken van OO en SQL

Relaties ingaan in SQL statements in Cache, is precies de manier waarop ik het van plan was. But I don't think it takes relations and classes as the base of the system. It's either specify tables with foreign keys, or declare objects with sub objects in them. I don't know if the two counterparts of a relation are synchronized either.

Cache integreert wel al met bestaande grote standaarden, zoals ODBC, XML, SOAP. And it provides in importing data into it from other database systems.

Wat nog steeds wel origineel is, ook ten opzichte van Cache, is de visie om OO ondergeschikt te maken aan het relationele model. In OO - Relational oplossingen wordt in object-relational mapping, OO bovenop relationeel geplaatst. Chaché zet de OO benadering ook bovenaan, en de relationele benadering als handig alternatief.

*OO Versus Relational Database*

Ik zie ook beweringen op het internet, dat een OO data benadering geavanceerder is dan relationeel. Maar er zijn voordelen aan relationeel, waar dan niet meer over gesproken wordt.

Ik denk dat het ook een zaak is twee-kampen. Ik zie in beide methodes het licht.

Anderen vinden OO bijvoorbeeld het beste, en voegen hier relationeel aan toe, en andere mensen vinden relationeel het handigste, en voegen hier OO aan toe. Don’t want to consider

Je moet dus echt aansporen dingen in de objectstructuur te embedded en daar met recursie op te lossen, in plaats van EEN moeilijke procedure erlangs schrijven

*Other People’s Software*

Windows doesn't allow shortcuts with the same name in a folder even when the short cuts have different target types.

Oracle 8.0i

Zoek op in HTML for Dummies hoe je naar een 'bookmark' springt.

Ideas,

Ik heb het vermoeden, dat als we alle gegevensverwerkingen via webservices zouden laten lopen, dat de boel niet vooruit te branden is.

JJ

*Other*

Computer Language,

Referenties naar een copy functie wil je

ook niet in de in de copy command definitie zelf bijhouden.

Maar je zou wel de mogelijkheid willen hebben om te querien

welke kopieeracties er binnen een bepaald systeem zijn.

Je kunt altijd een ruwe sequentiele zoek-query uitvoeren op

een subsysteem.

Maar je wilt het misschien ook centraal bijhouden. Dan

zou je een filter index moeten kunnen maken,

maar een filter index gezet op een elders gedefinieerde

method of class.

Ik heb er toch best moeite mee, dat je

in een stuk diagram niet ziet wat er allemaal naar

een bepaald object verwijst, maar alleen waarnaar

de objecten in de diagram verwijzen.

O, wacht, dat gebeurt voor objecten wel, omdat

de gerelateerde objecten als sub objecten worden getoond.

Heen en weer relaties tussen objecten in principe gelijkwaardig.

Maar bij methods is het anders. Die hebben altijd een richting,

en de relatie terug is echt de backwards verwijzing.

Het is zeg maar een kwestie van 'belachelijk om allemaal bij te houden'.

Alleen soms wil je voor een definitie, die zijn referrers niet bijhoudt,

toch referrers bijhouden.

Eigenlijk moet dan een systeem de referrers naar een definitie van een

ander systeem bij kunnen houden.

Je maakt bij methods eigenlijk ook relaties tussen method definitions aan.

Die zouden dan ook referrers bij kunnen houden, en een gesynchroniseerde

relatie aan kunnen gaan.

JJ

Software System, General

An application of this new model, that is my software system, would be

for instance, that someone comes up with something that makes any lengthy

process pauzable, items in it skippable, etcetera. Just a handy concept

for handling lenghty processes.

A handy user could inherit an existing system and extend it with a new

concept, that was not applicable to the system yet.

I think in order for a system to be adaptable to new concepts, there

must be a set of standards to be upheld. Otherwise a program might be just

one big method, and that can not adopt new concepts.

So you have to impose design rules in order to make a program

adaptable to new concepts.

JJ

Software System

- Crap first

- Objects second

- Methods third

Not:

- Methods first

- Object second

- Crap third

JJ

Computer Language,

Windows Workflow raakvlakken met diagrammen.

JJ

Computer Language,

I saw something call speficying the structure of a program, rather than what happens step-by-step is called declarative programming, as opposed to imperative programming.

JJ

Computer Language,

I don't have ... for user program flow, for instance when using

multiple windows seemingly arbitrarily, but not...

And also not for workflow.

I don't know yet how workflow fits into the system.

Probably as internet threads.

JJ

Computer Language,

Ik wil gewoon dat workflow en methods die elkaar aanroepen gewoon hetzelfde concept zijn. Method stappen kunnen parallel lopen en seriele punten hebben en vertragingen en gezette tijden hebben.

JJ

Stereotyping

Perhaps another typing can be assigned to a relation, instead of containment. For instance: *ownership* or *usage*. Perhaps a few standard ones, and it may be possible to define your own typing by specifying a String.

JJ

Computer Language,

Dead links.

JJ

Collection,

In Collection kan je ook vanalles van verschillende modules van verschillende sites combineren, tot 1 home-page. Je hoeft dus niet te kiezen tussen sites, waar je een home-page op kunt maken. Je kunt zelf alles dat ze aanbieden met elkaar combineren, zonder al te veel heisa. Ook een leuk argument om het Collection te hoemen.

JJ

Computer Language,

Important statement to use: objects first, procedures second.

JJ

Computer Language,

People aren't realizing that a computer program IS a domain-specific language.

JJ

Computer Language,

Met diagrams in computer language wordt de systematiek echt zichtbaar.

Op het moment programmeren we eigenlijk allemaal blind en op de tast.

JJ

Computer Language,

The idea of object oriented is: objects first, procedures second.

JJ

Computer Language,

The diagram expression also makes it possible to

use hand signs in the air to draw out relations between objects,

in a technical story you try to explain.

JJ 2008-06-03

Computer Language,

De kracht van programmeertalen nu, zit hem niet in de taal zelf,

maar in het framework.

What makes programming languages today powerful,

is not the language itself, but the framework.

JJ

Computer Language,

2008-08-15

Other programming languages and diagram expressions:

http://www.obsolete.com/dug/sorcery/oop.htm

JJ

Computer Language,

Other programming languages and diagram expressions:

Leuke zoekterm om op te googlen:

"my own programming language"

JJ

Computer Language,

2008-08-15

Other programming languages and diagram expressions:

http://jolt-lang.org/

JJ

Single paradigm for all digital objects,

See the physical disk as an object.

See the IO on it,

and what source does IO on it.

Be able to see which reference has the most activity.

Be able to navigate through the system,

so that you can trace the source of the activity.

That way you not only fly through the internet,

and the applications, but you can also navigate through

the internal workings of your computer.

I guess I do want to see magnitude of activity

in the diagrams.

JJ

Computer Language,

2008-09-02

Some people seem to like isolated memory space.

But this is just for some protection that it gives.

Memory leaks only live inside a process.

When it is stopped, then the memory leaks

are released.

When something crashes, it is just that process, that crashes.

The rest keeps running.

But in my own system, the whole internet is really just one process.

So what needs to be done, is analyse the advantages

people think they can only get from isolated processes,

(advantages, that are probably only precautions for problems)

and see what solutions for those problems will be implemented

in my own system.

JJ

Computer Language,

2008-09-02

Google using a separate process for each tab in its

Google Chrome browser, is just a practical

solution, relatively easy to implement.

That's why process isolation offers a solution

to this problem. It is not, that with more effort,

a not-yet existing solution to this problem,

that does not use the isolation of processes,

would not offer a better or equally practical

solution, would products like that be finished

today and free to use.

JJ

Computer Language,

Ik ben me er wel van bewust,

dat een nieuwe programmeertaal,

of nieuwe manier van besturen van je computer,

een bijdrage is, niet een vervanging van alle

andere software. Daar zijn software giganten

veel te innovatief voor geworden.

JJ

Computer Language,

2008-08-23

The way it is right now, the system can not change the system as much as a user can change the system.

JJ

Computer Language Functional Design,

2008-08-31

Sometimes it is just clearer to have an article,

with diagrams in it straight away, without

any article with just textual explanations.

In the future, the whole form of the documentation

might change as such and have diagram expression

be more present in the conceptual explanations.

JJ

2008-11

The roles that symbols get when connecting them with lines.

JJ