# Circle Language Spec Products

***-***

*Author: JJ van Zon*

*Location: Oosterhout, The Netherlands*

*Date: February 2008 – February 2020*

## Contents

Contents 2

Purpose of this Document 3

Topics Roughly 3

Topics To Cover 3

Fundamental Principles 3

Topic List Detailed 3

TODO 3

Coding Essentials 3

Coding Concepts 4

Other Requirements 4

Automatic Diagram Organization 4

Diagrams 4

Evaluate 5

Input Output 5

Coding Concepts 5

Postponed 6

Overview 6

Coding Concepts 6

Identifiers 6

Conversion 6

Conditions 6

Uncategorized Coding Concepts 7

New Computer Language Summary 7

Other Expression Topics 7

Diagrams 7

Fundamental Principles 7

Exchangability Principles 7

Extensibility Principles 7

Achievability Principles 7

Done 8

Coding Concepts 8

Identifiers 8

Objects 8

Classes 8

Relations 8

System Objects 8

Commands 9

Parameters 11

Globality 11

Execution Control 11

Black Box 12

Interfaces 12

Events 13

Inheritance 14

Diagrams 15

Fundamental Principles 15

Exchangability Principles 15

Extensibility Principles 16

Achievability Principles 16

## Purpose of this Document

This document is a raw list of the products to produce within the project *New Computer Language*. The goal and approach of the project were already described in the document *New Computer Language Strategy.*

## Topics Roughly

### Topics To Cover

The only things desirable to work out for the new computer language are:

- Fundamental Principles

- Coding Concepts

- Diagrams

### Fundamental Principles

The *Fundamental Principles* are more or less the *demands* for the language.

They are very abstract. The main groups of topics are:

- Achievability

- Exchangability

- Extensibility

They also explain how everything might work in harmony to achieve the goals.

Fundamental Principles also covers the basics of how you will be expressing yourself in the new language:

- Diagram & Text Code Expression

## Topic List Detailed

A sub-project may consist of documenting one or more concepts of a certain topic.

The list is not ordered by reading order, but rather by writing order, which is roughly the same as being ordered by level of difficulty.

The reason this list is quite big, is because of the split up into tiny little topics.

### TODO

#### Coding Essentials

After the phase described below, the coding concepts would be about as complete as any OO language.

##### Coding Concepts

###### Object Resolution

- Object Resolution

- Overriding

- Overloading

- Shadowing

- Melding

~ Hiding

- Interface Resolution

- Default Member

- Cascading Overriding

- Ambiguity

- Implicit Conversions

- With Blocks

###### Type Control

###### Classes, Relations & Static

- Static

Static could be worked out.

Brainstorm: There sem to be 2 competing ideas about static in this project: expressing an original language's intent with the word 'static' (e.g. C#) and, on the other hand, a more generalized concept of static, that anything about a class that remains unchangeable during or after instantiation, is *static*. (Including method definitions, static variables.) What happens when you use dashed lines to express anything 'static'? This while dashed lines conceptually also stand for 'class'. Do the concepts of static and class merge naturally, or how would that look in practice? This is fruit for thought for updating the descriptions of Static and Classes. 'Static' might not even have a separate description.

- Classes (update)

Classes articles might need an update considering that the notation(s) for static might change things.

Also vaguely in memory: the readability of the Classes articles was questionable, which also might be a reason for updating it.

- The explanation about class commands kind of sucks. It is really difficult.

- Merge conceptual explanation and diagram notation explanation.

- The term Target Class may have to be split up in two definitions.

- Relations (update)

- Relation Direction

- Relations to Pointers

Also: relations articles might need an updateconsidering that the notation(s) for static might change things.

- Merge conceptual explanation and diagram notation.

- Change the notation for a bidirectional relation.

- Relations does not make sense anymore when related classes are not necessarly fixed by the class.

- Relations to pointers (skip it if you find it too hard.)

###### Redo Themes

- Object articles update

Maybe only merge conceptual explanation with diagram notation explanation.

- System Objects articles update

- Commands articles update

- Parameters articles update

- Globality articles update

Not sure if this will have to change. Maybe change the term to 'Modules' instead, for less alienation.

- Execution articles update

Maybe only merge conceptual explanation with diagram notation explanation.

The split up into a conceptual explanation and then separately an article for the diagram notation, might have been easy for writing the docs, but could be merged into a single article again for readability.

The readability of these articles might be questioned. See 'Redo Easy Themes' in the document *Circle Language Spec Strategy*.

##### Other Requirements

Data Concepts:

- Enums

- Object Order

Objects:

- Multiplicity

> You might want to be able to introduce new kinds of lists and express and use them as any other list.

> You might want to be able to introduce new basic data structures and give them the nonagon symbol, and have different kinds of possible indexers, etcetera.

#### Automatic Diagram Organization

At this point the coding concepts may be complete, but without the diagrams being organized automatically, the effectivity of the language might be considerably limited.

##### Diagrams

- Automatic Containment

- Artificial Promotion, Esthetic Reference

- Diagram Metrics

- Spiraling

- Force-Based Ordering

- Object Order

##### Brainstorm

Part of it could be documenting ideas about diagram metrics that are only in my head and ones derived from a prototype the of diagram drawing engine I made.

An option to ignore fixed logical residence would be welcome.

In that case the fixed logical residence might only be pointed out with an empty reference to the imaginary residence. That way automatic containment will work, but publically-accessible classes are still available with just the namespace qualifier. (🡪 Lower Contents)

Elements of different assemblies seem to intermix too much and parts of one assembly are shown as an intrinsic part of the an assembly that uses it. Things might be better off as better visible as being externally defined.

It is a problem that everything is displayed as composite aggregation, because in some places it works completely counter-intuitive (where you would expect objects to be sibblings, but they are nested instead?).

The need to display large amounts of items really comes to light now plus that something should be done about it.

Also the need for line bundling starts becoming more apparent.

#### Evaluate

This may be a crossroads at which you migh switch to releasing it into the world.

The language could do without anything put here below.

An advantage of releasing it, is that from then on you might never be lonely working on the new computer language again.

#### Input Output

This phase would introduce a ‘revolutionary’ way to handle the input/output concept and concurrency.

##### Coding Concepts

###### Input Output

- In, Out & Thru

- Auto In, Out & Thru

- Command IO

- Compared IO

- Accessing parameters’ sub-objects

- Specific data unknown

- Parameters & IO

- Sub-commands’ IO

- Pre- & Postconditions

- Conditions

- User Commands

- Commands & Classes Loosely Coupled

- Automatic Execution Order

- Parameters of calls directly tied together

- Parameters tied together

- Parameters tied to objects

- Outcome dependency

- Compared IO

- Legacy Parameter IO

(from old parameter passing types)

- Parameters

- Parameter Types

- In Parameter

- Out Parameter

- Thru Parameter

- By Value

- By Reference

- Value In

- Value Out

- Value Thru

- Reference In

- Reference Out

- Reference Thru

- Object Out

- New Object Out

- Existing Object Out

- Three Parameter Passing Elements

- Parameters of calls directly tied together

- Strict about parameter passings

- The class of a parameter

- Sub-commands are never output objects

- In, Out, Thru Parameters

- Downput Parameter

- Data Direction

- Indirect Value Transmission

- Input / output not always values

###### Concurrency

…

### Postponed

These topics are not essential or they are already adequately, but not fully described.

#### Overview

Nice To Have:

- Namespaces, Aliases

- Coding Concepts Misc: Conversion, Conditions

- Diagram Topics Misc

- Expression Misc

- Fundamental Principles Already Worked Out

- Uncategorized Coding Concepts

- Language Summary

#### Coding Concepts

- Introduction

##### Identifiers

- Namespaces

- Aliases

##### Conversion

- Conversions

##### Conditions

- Conditions

- On Commands

- On Object Values

##### Uncategorized Coding Concepts

- Special Access

- Global Access

- Clause Access

- Interface Access

- Multiple Language Layers

- This

- Redirection

Where did these go?

- Value Direction

- Clone

##### New Computer Language Summary

#### Other Expression Topics

- Expression Mixing

- Specialized Expressions

#### Diagrams

- Coloring

- Abusing Diagram Expression

- Abstract Diagram Expression

#### Fundamental Principles

The list of fundamental principles might be extended in a future project, but not written straight away then.

The idea is: use the ideas in the idea box, but do not write all the articles, but do add a description to the Contents page.

- Introduction

##### Exchangability Principles

- Introduction

- Diagram & Text Code Expression

- Attributes Are Objects

- Command = Executable Object

- Hand Signs

- Hand Writing

##### Extensibility Principles

- Introduction

##### Achievability Principles

- Introduction

### Done

This section may actually mention postponed work, that is not done.

#### Coding Concepts

##### Identifiers

- Identifiers

- Omitted Identifiers

##### Objects

- Objects

- Sub-Objects

- Object Reference

- Related Objects

- Nothing

- Values

- Multiplicity

- Attributes

- Target Objects

##### Classes

- Classes

- Class Commands

- Class Reference

- Target Classes

##### Relations

- Referrers

- Class Referrers

- Related Classes

- Relations

- Dual & Unary

­

­­- Relations Between Objects

- Relation to a Pointer

- Relation Synchronization

##### System Objects

- System Objects

- System Aspects

- Object-Bound & Reference Bound Aspects

- System Commands

- System Commands for the Referene Aspect

- System Commands for the Object Aspect

- System Commands for the Class Aspect

- The Extra Commands & Overloads

- System Commands for the Value Aspect

- System Commands for the Clone Aspect

- System Commands for the Name Aspect

- System Commands for the Data Aspect

- System Commands for the Execute Aspect

- System Commands for the Existence Aspect

- System Commands for the List Aspect

- System Interfaces

- System Interface of an Object

- System Interface of a Related Item

- System Interface of a Related Item

- System Interface of a Related List

- System Interface of a Related List Item

- Use-Command Gets Another Aspect

- Aspect-In-A-Triangle

- System Interfaces of Objects & References

- Preliminariness of System Interface Notation

- Assignment

- Pointer Assignments

- Assignment With Pointer Source

- Cross-Aspect Assignments

- System Command Call Notations

- System Interface Comm Call Notation

- Simplified System Comm Call Notation

- System Comm Call with Argument Notation

- Explicit Get & Set Notation

- Explicit Get & Set Arguments Notation

- Completely Explicit Display

- Connectors

- Connections

- System Command Calls by User

- Objects Floating Around

- System Command Extension

- Parameters For Objects

- Ancestry Terms

- Assignment in Text Code

The System Objects article group needs extensions in the future.

It needs to be revisited to make complete the set of system aspects and system commands and cross out remaining ideas and topics.

##### Commands

- Commands

- Commands Implementation (not finished)

- Execute Once

- Executable Command

- Inactive Command

- Command Definition

- Command Call

- Command Reference

- Clause

- Inactive Clause

- Active Clause

- Start & Stop

- Start & Stop Implementation (not finished)

- Command Anywhere

- Changing Inactive to Executable

- Sub-Commands

- Command References Inside Commands

- Executables & Executions

- Procedure

- Resolution When Not Allowed For Commands

- Parent Controls Its Sub-Executions

- Sub-Commands Are Never Referenced

- Beware of Active Command References

- Sub-Commands Not Manually Started

- Public Inactive Clause = Command Out Parameter

- Creation Behavior of Calls

- Creation Behavior of Clauses

- Creation Behavior of ‘Inactive Calls’

- No Overhead Of Command Creation

- No Circular Command Creation

- No Private Contents in a Call in a Definition

- A Call in a Call Shows Privates When Running

- Active Command in Inactive Command

- Reading & Writing Parameters

- Comparison to CPU-Like Calls (not finished)

- Example Diagrams (folder)

- Command Calls in a Command Definition in a Diagram

- Command Calls in a Command Call in a Diagram

- Command Calls in a Command Reference in a Diagram

- Parameters in a Command Definition in a Diagram

- Parameters in a Command Call in a Diagram

- Parameters in a Command Reference in a Diagram

- Private Objects in a Command Definition in a Diagram

- Private Objects in a Command Call in a Diagram

- Private Objects in a Command Reference in a Diagram

- Command References in a Command Definition in a Diagram

- Command References in a Command Call in a Diagram

- Command References in a Command Reference in a Diagram

- Clauses in Clauses in a Diagram

- Inactive Clauses in a Command Definition in a Diagram

- Inactive Clauses in a Command Call in a Diagram

- Inactive Clauses in a Command Reference in a Diagram

- Active Clauses in a Command Definition in a Diagram

- Active Clauses in a Command Call in a Diagram

- Active Clauses in a Command Reference in a Diagram

- Command Calls in an Inactive Clause in a Diagram

- Command Calls in an Active Clause in a Diagram

- Parameters in an Inactive Clause in a Diagram

- Parameters in an Active Clause in a Diagram

- Private Objects in an Inactive Clause in a Diagram

- Private Objects in an Active Clause in a Diagram

- Command References in an Inactive Clause in a Diagram

- Command References in an Active Clause in a Diagram

- Inactive Command Object Redirection

- Inactive Command Class Redirection

- Executable Command Class Redirection

- Executable Command Object Redirection

- Target Command Object

- Target Command Definition

- Recursion

- Implementation of System Commands

- Command Object Referrers (not finished)

- Command Definition Referrers (not finished)

##### Parameters

- Relations Between Commands & Objects

- Parameter Passing

- Joint Display of Access Connectors & Object Relations

- Legacy Parameter Concepts

- Required & Optional

- Variable Amount of Parameters

- Return Values

- Parameter Order

- Miscellaneous Parameter Topics

- Parameters & Arguments

- Parameters For The Add Command

The article group might need a redo in the future.

##### Globality

- Globality

##### Execution Control

- Execution Control

- Conditional Execution

- If

- Else If

- Select Case

- Select Case (exact value)

- Select Case (split formula)

- Loops

- For

- For (range)

- For (conditional)

- For Each

- While

- Until

- Jumps

- Normal Execution Order

- Label & Goto

- Call & Return

- Exit Command

- Loop-Related Jumps:

- Continue

- Exit Loop

##### Black Box

- Introduction

- Black Box Main Concepts

- Public & Private

- Friend Declaration

- Notations of Private

- Friend Declaration in Instances

- Public & Friend Connections

- Access Controlling Aspects

- Object-Bound, Reference-Bound & Access Control

- Public & Private Connectors

- Public & Private Connections

- Public & Private Assignment

- Assignment between Friend Members

- Friend = Wavy Line

- Miscellaneous Issues

- Deeper Exclusion

- Protected

- Internal

- Objects Take Over Class Access Control

- Side-Issues

- Private Names

- Inclusion

- Black Boxing and User Access Control

- Programmers and Users

- Details to Cover Last

- Details, May Not Cover

##### Interfaces

- Introduction

- Interfaces Main Concepts

The general notation

Interface = Publics

Interface & implementation

Interface = Objects Melting Together

Interface = Contract

Mutliple Interfaces

Interface Type Controls

Explicit Interfaces

Implicit Interfaces

Polymorphism

Interface Object Resolution

Inheritance

Main Usages of Contracts:

a contract = a guarantee

a contract = a contract of usage

a contract = a contract of participation

- Interface Aspect

- Interface Commands for the Interface Aspect

- Interface Commmands for the Interface Merging Aspect

- Interface Aspect in System Interface

- Interface Assignment Types

- Interface System Command Calls

- Interface Connectors

- Interface Connectors

- New Command with Interface Parameter

- Command Interfaces

- Interface Referencing and Redirectioning (examples)

- Member Grouping

- Relations Through Interfaces

- Interface Referrers

- Reliability of Interfaces

- Group By Source

- Imaginary Backward Relations

- Target Interfaces

- Mutual Commands, Mutual Interfaces (Unfinished)

- Extensive Classification with Interfaces (Unfinished)

##### Events

Basic Concepts:

*- Events Introduction*

*- Prime Event Example: Button Clicked*

*- Events Main Concepts*

Main Notation of Events

Event Procedure / Event Implementation

Event Connector

Event Raising

Call to Parent

Event Implementation & Connector

*- Event Situations*

Multi-Cast Events

Multi-Cast in Same Parent

Event with Parameters

Event from Deeper Object

*- Explicit Implementation of Event Object*

*- Explicit Interface of Event Object*

For the sender

For the receiver

*- Event Interface Reference*

*- System Events*

Changed Events

Changing Events

Get Events

Use Events

List Events

Execution & Existence Events

Pointer to Pointer Events

*- System Events in Normal Notation*

*- Simplified System Event Notation*

*- System Event Parameters*

*- System Event Design Choices*

Misc topics:

*- Qualified Event Names*

*- Interaction Events*

*- Black Boxing Events*

*- Event Alternatives*

##### Inheritance

- Inheritance Introduction

- Inheritance Main Concepts

- Class Inheritance

- Object Inheritance

- List Inheritance

- System Inheritance

- Interface Inheritance

- Specialization

- Specialization & Data Replacement

- Altering the Member Set

- Member Addition

- Member Exclusion

- Member Inclusion

- Detouring Members

- Shadowing

- Overriding

- Altering Command Implementations

- Command Extension

- System Command Extension

- System Command Overriding

- System Command Shadowing

- Sys Comm Exten By Shadowing

- Sys Comm Exten By Overr (Questionnable)

- Sys Comm Overr By Extension

- Destructive & Non-Destr Spec Methods

- Misc Inheritance Situations

Command Inheritance

Backward Relation to Derived Classes

Multiple Objects Sharing one Base Object

Inheriting Multiply from the Same Class

- Enforcing & Preventing Specialization

Protected

Overriding

Optional / Required

Requirements for Other Side of Connection

Gut Feeling

Applied to Concepts from Other Languages

Protected

Virtual

Abstract Member

Abstract Class

Sealed / Final Class

Interface / Interface Member

Non-Overridable

More Combinations

Courtesy

Enforcing & Preventing Other Specialization Methods

Preventing Pre-Extension with Cancellation

Preventing & Enforcing Data Replacement

- Deeper Specialization

Deeper Exclusion

Deeper Inclusion

Deeper Member Addition

Even Deeper Member Addition

Deeper Shadowing

Deeper Overriding

Deeper Command Extension

Deeper System Command Extension

Deeper System Command Overriding

Deeper System Command Shadowing

The Deeper Specializations Structure

- Alternate Version Through Inheritance

- Extending System Objects (Older)

(This older article was left in tact to save time.)

#### Diagrams

- Basic Diagram Elements

#### Fundamental Principles

##### Exchangability Principles

- Data = Code

- Programming Language = Database

- Designtime = Runtime

- User = Programmer

- Clear Cut Coding Principles

- Hyperlinks = Referential Structure

- User Interface Not Procedure Oriented

- Symbol = Creator

##### Extensibility Principles

- Reflection

- Module Integration

##### Achievability Principles

- Generic, No Generators

- Small Code Base

- Computer Language Programmed Within Itself

- Everything Only ( Lack Of Choice = Guarantees )

- C++