# Core 4.0

## “Multi-Technology” Compiler “All-Purpose”

### Introduction

Core 4.0 is a compiler open to the definition of new languages. It has three main characteristics. The first is to be a compiler or a computer program that translates a language (source code) into another language (target language). The second feature is that of being All-Purpose because it is possible to define new languages and whose instruction code can have the potential to easily perform any task, using well-known programming languages. The third is to be Multi-Technology because the target language can be of any (and more than one) language (technology) used today. Core 4.0 was also designed to translate high-level languages into low-level languages such as assembly languages. And for this reason it is suitable as a compiler for special processors.

The existence of this innovative technology cannot fail to have an effect in the possible change in the way of programming and change in the software analysis and development paradigms.

Philosophically we can affirm that the current programming languages and their related libraries have the purpose of providing the widest programming possibility at the expense of the complexity and difficulty of programming. This is due to the fact that the language provider provides a product obviously not knowing the needs and development level solutions that the programmer intends to use.

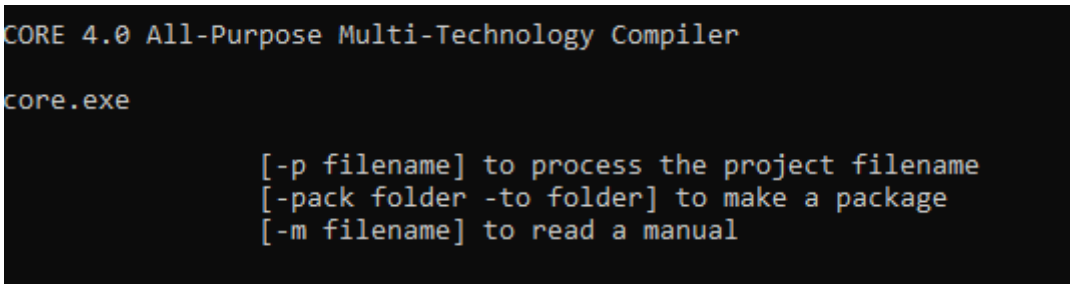
So any programmer can get what he wants but with difficulty due to the complexity that languages and their libraries have assumed. And when he finds himself having to re-implement his own solutions even tens of hundreds of times, perhaps having to do a little customization, he has to put his hand to long code, in terms of lines of code, and complex, as well as many technologies.

With Core 4.0 we can work in terms of language engineering to provide developers with ad-hoc languages for their solutions that can clearly be customizable. The result is that of engineering simple languages, which in a few statements can be translated into complex implementation solutions. So that the problem or problems we discussed earlier can be said to be solved.

It is clear that we who offer advice on the adoption of Core 4.0 must in a first phase of a few months collaborate with the company that wants to engineer a language. But once a framework for Core 4.0 has been created that realizes the language and translates it into solutions, the increase in developer productivity is remarkable. On your part, there must therefore be the will to undertake a period of research and development with us.

## Overview

We can start introducing Core 4.0 with an overview, showing what it means to work (program) the compiler. And we will do it through a very simple example, so simple that it cannot be considered a case study, like the one presented in the frameworks page. At first we simply show what is displayed by launching the Core.exe file.




The image shows the console, and three options, of which the first is the most important, that is the possibility of processing a project written in an ad-hoc engineered language for what we have to do. The creation of packages is equivalent to the encryption of files to make them unchangeable, in the case of collaboration with third-party companies, while the third allows you to always read the user manuals of the languages on the console.

## PARSER

The first step in programming a language is to set up the parser. That is, write what types of statements it must recognize, how and how these are structured. In this simple example we are dealing with a student booklet, so there are three, student, exams, and blank booklet.

```
BEGIN
student&: student|(frw) serial|(str) number|.serial|(frw) {|.%exams|(**) ;|(**)|(opt) }
exam&: exam|.exam|(frw) credits|.credits|(frw) vote|.vote|(frw)
empty&: empty
END
```

The parser has been configured very simply, and furthermore the example deals with a simple configuration of it. I think the code of all these examples is intuitive and does not require annotations.

## SOURCE

Now we can already syntactically express ourselves with the language we wanted. Here is what the drafting of a student booklet looks like:

```
start {{

        student nome e cognome serial number 100234 {
                exam Computer Science credits 12 vote 30 ;
                exam Elettronics credits 12 vote 28 ;
                exam Elettronics Calculators credits 12 vote 28 ;
                exam Theory of Signals credits 6 vote 28 ;
                exam Elettrotecnics credits 12 vote 26 ;
                exam Automatics credits 6 vote 26 ;
                exam Software Engineering credits 12 vote 30
                }

}}
```

## RULES

So let's start the programming work, in the Core 4.0 Compiler Language, starting to write the composition rules of the statements. That is how statements can be composed for the purpose of correct semantics.

```
public function student.rules ( null ) {{

        set ::cont_type = 0

}} ;

public function exam.rules ( null ) {{

        case ( ::__CALLER ! type student ) {{ rule error exam must be a statement of student }} ;
        if ( ::__CALLER.cont_type == 1 ) {{ rule error declaretion of exam in a empty freshman }} ;

        set ::cr = true int ::credits ;
        set ::vt = true int ::vote ;

        if ( ( ::cr < 1 ) || ( ::cr > 24 ) ) {{ rule error invalid credits }} ;
        if ( ( ::vt < 18 ) || ( ::vt > 32 ) ) {{ rule error invalid vote }} ;

        ::__CALLER.cont_type = 2

}} ;

public function empty.rules ( null ) {{

                case ( ::__CALLER ! type student ) {{ rule error empty must be a statement of student }} ;

                if ( ::__CALLER.cont_type == 1 ) {{ rule error redeclaretion of empty freshman }} ;
                if ( ::__CALLER.cont_type == 2 ) {{ rule error empty declaration on not empty freshman }} ;

                ::__CALLER.cont_type = 1
}} ;
```

In this example, which I do not want to note, it is verified that the exams or the empty statement belong to a booklet, and that a booklet can also be declared empty at most once, and that if it is empty it cannot contain exams and vice versa. It is also verified that the credits and votes are whole numbers belonging to a certain interval.

## TRANSLATION CODE

Now we show the code needed to translate the booklet written by the new language into an html table. The weighted average of the votes rounded to the second decimal place is also calculated and written in the html file.

```
public function student.autoinit ( null ) {{

        set ::html = "\n<section>\+\n<b>" + ( text ::student ) + "\sn. exams:\s" + ( text size ::exams ) + "</b>\n<table>\+"
                   + "\n<tr>\+\n<td>n.</td>\n<td><b>credits</b></td>\n<td><b>exam</b></td>\n<td><b>vote</b></td>\-\n</tr>" ;

        set ::i = 0 ;

        set ::vote_sum_weighted = 0 ;
        set ::weight_sum = 0

}} ;

public function exam.autoinit ( null ) {{

        set ::html = "\n<tr>\+\n<td>" + ( text ::__CALLER.i + 1 ) + "</td>"
                   + "\n<td>" + ( text ::credits ) + "</td>"
                   + "\n<td>" + ( text ::exam ) + "</td>"
                   + "\n<td>" + ( text ::vote ) + "</td>\-\n</tr>" ;

        ::__CALLER.i = ::__CALLER.i + 1 ;

        ::__CALLER.vote_sum_weighted = ::__CALLER.vote_sum_weighted + ( ::cr * ::vt ) ;
        ::__CALLER.weight_sum = ::__CALLER.weight_sum + ::cr

}} ;


public function exam.autoexec ( null ) {{ ::__CALLER.html = ::__CALLER.html + ::html }} ;

public function student.autoexec ( null ) {{

         set .av = ( ::vote_sum_weighted * 100 ) / ::weight_sum ;
         set .average_string = ( text .av / 100 ) + "." + ( text .av % 100 ) ;

         ::html = ::html + "\n\-\n</table>\n Average:" + .average_string +  "\-\n</section>" ;

         debug {{
                 file clean "test.html" ;
                 print ::html ;
                 file write "test.html" , ::html
         }}
}}
```

A manual on programming using the Core 4.0 compiler language is not in the public domain, because this is our job. We engineer languages. Writing down the code would therefore be an end in itself. However, what you can guess from the example are the basic operating principles.

## COMPILATION

Here is the translation, the compilation as we call it, of the language in HTML code.

```
<section>
        <b>nome e cognome n. exams: 7</b>
        <table>
                <tr>
                        <td>n.</td>
                        <td><b>credits</b></td>
                        <td><b>exam</b></td>
                        <td><b>vote</b></td>
                </tr>
                <tr>
                        <td>1</td>
                        <td>12</td>
                        <td>Computer Science</td>
                        <td>30</td>
                </tr>
                <tr>
                        <td>2</td>
                        <td>12</td>
                        <td>Elettronics</td>
                        <td>28</td>
                </tr>
                <tr>
                        <td>3</td>
                        <td>12</td>
                        <td>Elettronics Calculators</td>
                        <td>28</td>
                </tr>
```

```
                <tr>
                        <td>4</td>
                        <td>6</td>
                        <td>Theory of Signals</td>
                        <td>28</td>
                </tr>
                <tr>
                        <td>5</td>
                        <td>12</td>
                        <td>Elettrotecnics</td>
                        <td>26</td>
                </tr>
                <tr>
                        <td>6</td>
                        <td>6</td>
                        <td>Automatics</td>
                        <td>26</td>
                </tr>
                <tr>
                        <td>7</td>
                        <td>12</td>
                        <td>Software Engineering</td>
                        <td>30</td>
                </tr>

        </table>
         Average:28.16
</section>
```

The page, or section, can now be viewed via the browser.

## EXECUTION

**nome e cognome n. exams: 7**

| n. | credits | exam | vote |
|---|---|---|---|
| 1 | 12 | Computer Science | 30 |
| 2 | 12 | Elettronics | 28 |
| 3 | 12 | Elettronics Calculators | 28 |
| 4 | 6 | Theory of Signals | 28 |
| 5 | 12 | Elettrotecnics | 26 |
| 6 | 6 | Automatics | 26 |
| 7 | 12 | Software Engineering | 30 |

Average:28.16

## SINTAX ERRORS

Interesting to you that you will be working with your language and knowing that syntactic and semantic errors are displayed for the purpose of a correction.

```
student nome e cognome serial number 100234 {
        exm Computer Science credits 12 vote 30 ;
        exam Elettronics credits 12 vote 28 ;
        exam Elettronics Calculators credits 12 vote 28 ;
        exam Theory of Signals credits 6 vote 28 ;
```

```
student nome e cognome serial number 100234 {
sintax error
```

## DISCUSSION

Those who have read and started to understand what has been said so far can widely imagine the great advantages that the adoption of this technology can bring in terms of simplicity in application development and increased developer productivity. Thanks to being able to translate a few lines of code into large amounts of code belonging to all the technologies necessary for a purpose.

To those who had already felt the need for a tool like Core 4.0 being grappling with all the problems that software development puts on the shoulders of developers every day, I can only congratulate you because the solution that they intuitively asked for has been conceived and implemented. An effort by the latter would be enough to push their companies to adopt Core 4.0.

The adoption of the compiler is strongly recommended for companies that develop according to product lines. It is ideal, for example, for those who produce information systems, but not only. Anyone with a web-based or app-based product line or both can benefit from all the benefits of compiler adoption.

On the other hand, it is not advisable to use the compiler for engineering companies which, having to cope with the most diverse customer requests, are forced from time to time to study to create the required solution. In this case, due to the lack of reusable code, it would be inconvenient to engineer a programming language that will be used only once and not repeated times.

## FRAMEWORKS

The implementation of a framework for the Core 4.0 all-purpose compiler is actually an implementation work using the compiler language for the realization of a programming language that is translated into code of other languages. By code we mean instructions and data written even on multiple files and in multiple languages but with a single semantics that of the language implemented.

Core 4.0 was born to give the possibility to define new languages, you know this, and it has inside all the instructions for compiling in multiple files and types of languages. So reading, writing, modifying and saving, of the object code.
The work of engineering a framework and then implementing it is our job. But having to implement your solutions, we cannot carry out the work other than the collaboration with the client company. We are open to collaboration with any type of company, but companies that already have a research and development department are clearly at an advantage, as they can broaden their goals and inherit our already done research and development work that has cost us 5 years. research and development.

Below we will show some concepts illustrated by images, on the framework fully

implemented by us bsa 1.0. Bsa stands for (Basic Applications) and is a framework made for the Core 2.0 compiler for demonstration and testing purposes only.

The bsa 1.0 framework for the Core 2.0 compiler deals with the possibility of automating full stack programming for web-based applications. It was therefore decided to create a language whose statements are translated both into code on the front-end side and the corresponding code on the back-end side.

It was decided to write a language that, starting from the definition of the database (with its creation scripts), would allow, through a few lines of code, to insert components such as smart-tables and multi-step forms that can be easily programmed for various objectives such as user registration, visualization and modification of data by displaying in tables. The work did not leave us out of dealing with the validation that is linked to the custom types in the definition of the database fields, and also with security.

In addition to the implementation of the language, it was necessary to implement versatile libraries for server-client communication. This work of building execution support libraries has been called support engineering by us. You will understand that these are concepts that already distort the way of thinking in software analysis. Creating an ad-hoc language, translating it into your own solutions, and developing execution support libraries, are all concepts that propel you into the world of advanced software engineering.

## SUPPORT

We can write just two lines on the support of the bsa 1.0 framework saying that it obviously deals with the management of all possible client server communication errors and communications, therefore viewing of pages, messages through dialogs, change of status and information, then modification of the components made as simple as everything using a string.

## TECHNOLOGIES

The technologies in which the language is translated are: Html, Javascript, JQuery, Boostrap, Ajax, Json, PHP, MYSQL

## MODULES

The modules of the framework are:

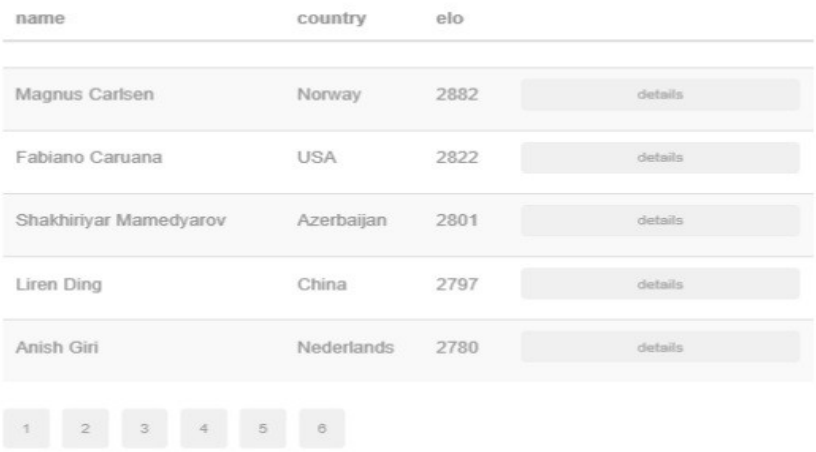
- framework :( Main structure of framework )
- database : ( Definition and custom types of databes fields)
- web : ( Language and how to translate the front-end )
- scripts : ( Language and how to translate the back-end )

- modules : ( Back-end code of modules to customize )
- components : ( Component definition structure and components smart-tables and multi-step forms )

## RUNTIME

```
Core 2.0 All-Purpose Multi-Technology Compiler


database module initialization .. initialized.
scripts module initialization .. initialized.
web module initialization .. initialized.
source modules generation loading .. done.
dinamic components loading .. done.
framework initialized.
resources processing..
resources processed.
```




## SOURCE

Wanting to take a look at what a web page code looks like, we can show this example:

```
path application\web\ ;

generate { module index.js :: javascript support ( * ) } ;

page index : mode web init ( dir htdocs\ , filesave index.html , location /index.html ) {
        src scn\web\header.html ; @Js index.js ;

        internalize t WEB ( html ) from champions table ;
        externalize t WEB ( js , ready ) from champions table to index.js system support ;

        externalize actions ( js ) from champions table to index.js system support ;

        src scn\web\footer.html
} ;
```

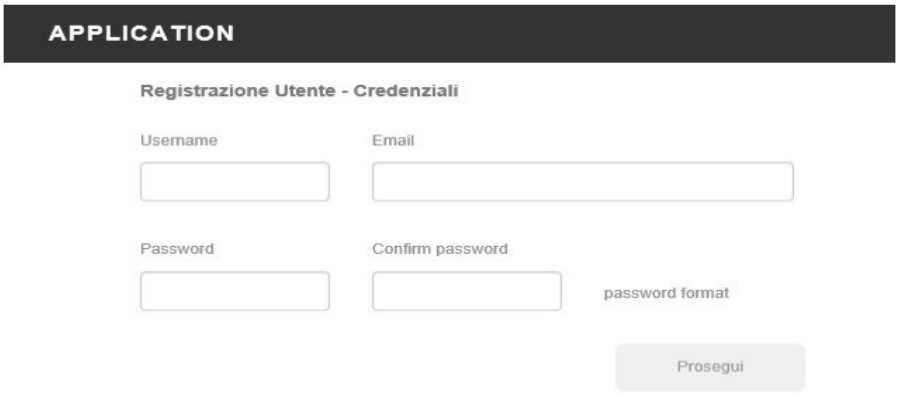
I would also like to emphasize the beauty and elegance of how the code looks. Here the page foresees the loading of sections, but first of all the relative javascript file is generated with the support system and the code for the smart table (previously defined) is pasted both in the web page, and exported in the javascript page through the support system. By

means of these very simple lines of code, the table is ready for use. It is a table linked to the database with actions applicable on individual records and works according to the server-client model or if you want it is an example of front-end, back-end programming.

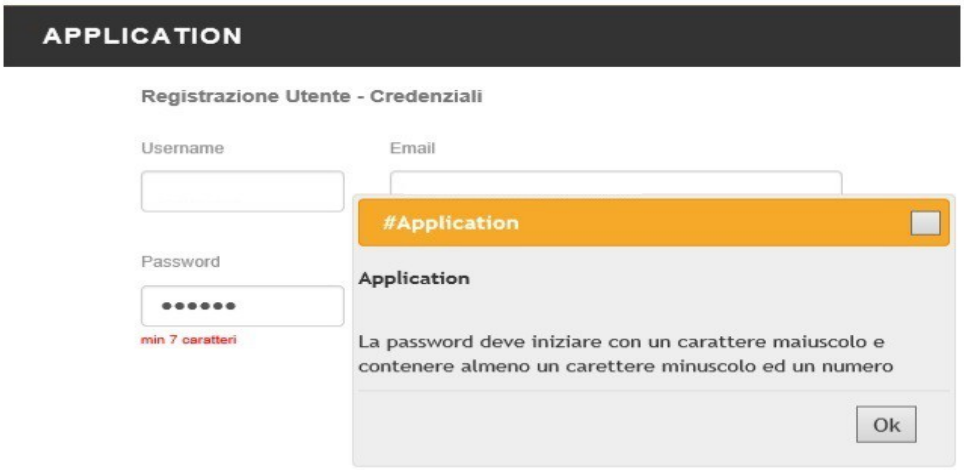
## COMPONENTS

The automation of graphics, logos, drawings, graphics goes beyond the types of automation possible with Core 4.0. The bsa 1.0 framework is aimed at automating logical and data processing processes such as data validation and entry into the database, recovery of these, treatment, then visualization and modification. And the preliminary analysis study that we have done has led to the result that the great part of the work is done by the components, such as smart-tables, multi-step forms, and dialog boxes. Having said that, the programming of these components has been automated, which appear as in the following images.

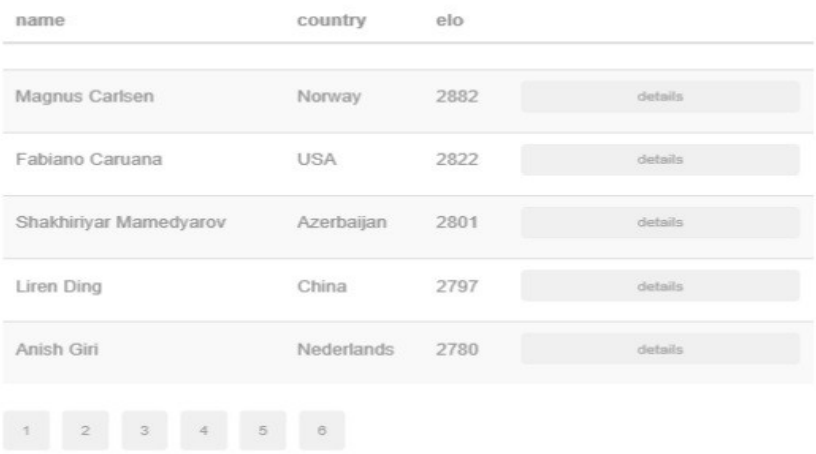
## FORMS




## DIALOGS



## TABLES



| name | country | elo | |
|---|---|---|---|
| Magnus Carlsen | Norway | 2882 | details |
| Fabiano Caruana | USA | 2822 | details |
| Shakhiriyar Mamedyarov | Azerbaijan | 2801 | details |
| Liren Ding | China | 2797 | details |
| Anish Giri | Nederlands | 2780 | details |

1 2 3 4 5 6

## DISCUSSION

Making a language is strongly recommended by us. Entire applications can be created simply with 200, 500, 1000 lines of code and by a single developer who works, not on multiple files and using multiple languages, but on a single file using a single language.

The use of a language makes it possible to have considerable flexibility. In fact, if one day you decide to add components, for example, it would simply be a language upgrade and this applies to all the implementation solutions that you want to automate.

The product code, in particular cases, being absolutely readable and modifiable by the developers, can be manually refined after compilation.

However, there are technologies that do not lend themselves to creating frameworks. I'm talking about those that allow you to program through graphic and non-textual tools. It is also more difficult, but obviously not impossible, to build a framework on top of another existing commercial framework. Also in this case it is not recommended. Or rather, it is advisable to switch to other technologies.

Core 4.0 is also clearly capable of writing ASM code. Therefore, if you want to define a high-level language for those special processors that are currently programmed only in their own assembly language, with Core 4.0 it is possible. It is possible to define primitive and derived types, functions, function calls, any kind of translation of expressions, also based on accumulator and stack architecture. As well as any type of instructions for configuring the processor based on use. High-level languages can be somewhat compliant with known ones, therefore having cycles etc., but not only.

## Core 4.0 Compiler for Custom Processors

Core 4.0 is an All-Purpose compiler that can be used as a compiler for custom processors and special processors. This is because it allows you to define and translate high-level languages into the assembly languages specific to these special types of processors. We will show below the principles of how this is possible which will serve as proof that Core 4.0 is an “All-Purpose” compiler.

Let's start the discussion by imagining we are dealing with a processor whose architecture is based on accumulator and stack. The example deals with the definition of primitive types, and types derived through the "struct" construct. And the offsets for the variables on the stack will be calculated, based on the type of program we are going to write.

The language provides these types of statements:

```
primary&: primary|(str) type|.type|(frw) size|.size|(frw)
var&: var|.var|(frw) type|.type|(frw)
struct&: struct|.name|(frw) {|.%statements|(**) ;|(**)|(opt) }
statement&: statement|.name|(frw) {|.%statements|(**) ;|(**)|(opt) }
```

and a possible source code could be the following:

```
primary type int32 size 32 ;
primary type int64 size 64 ;

struct s {
        var a type int32 ;
        var b type int64
        } ;

struct r {
        var c type s ;
        var d type int32 ;
        var e type s
        } ;


statement function {

       var a type s ;
       var b type int64 ;

       statement for {
              var c type r ;
              var d type s
       }

}
```

The calculation of the possible instances of the variables and their offset on the stack, if we carry out the first steps of the compilation, is as follows:

```
statement for{
        c.c.a   32      160     0
        c.c.b   64      192     0
        c.d     32      256     0
        c.e.a   32      288     0
        c.e.b   64      320     0
        d.a     32      384     0
        d.b     64      416     0

}
statement function{
        a.a     32      0       0
        a.b     64      32      0
        b       64      96      0

}
```

The compiler architecture clearly provides for the treatment of expressions of which I give a brief explanation.
Expressions are made from unary and binary variables and operators. For example the expression:

```
.a + .b + 1 ;
```

It comes to be pre-processed and we print it to see how it looks before being processed and then translated into the processor's own language:

```
v|.a o|+ v|.b o|+ n|1
```

Pre-processing also involves the resolution of the priority of the operators in fact for the expression:

```
.a + .b * 2 ;
```

we have :

```
v|.a o|+ r|( v|.b o|* n|2 r|)
```

As for vectors and expressions that return the index, these are treated as nested expressions that can be processed recursively:

```
.a[ .b + 2 ] ;
```

```
v|.a o|[] {{ r|( v|.b o|+ n|2 r|)}}
```

The nested statements complete the discussion, such as those for the function call:

```
.a + exec f ( .b ) ;
```

```
v|.a o|+ function f parameter v|.b
```

Recursive logic applies to both expressions and nested statements, in the sense that nested expressions can contain nested statements and vice versa.

A possible accumulator-based translation of the expression:

```
.a + .b * 2 ;
```

is the following:

```
mov eax , [ a ]
push eax
mov eax , [ b ]
mul eax , 0x2
add [ esp ] , eax
pop eax
```

Core 4.0 was created to define programming languages other than those used. If you want to create a compiler for a well-known language you have to contact compiler engineers. Conversely, for languages that must have a certain originality, the ideal product is Core 4.0.

## Frequently Asked Questions

How can I make my own programming language ?

The basic question is whether you want to embrace the all-purpose compiler philosophy. That is, if you want to create a language that can be translated with code from other well-known and used programming languages. Or you want to create a programming language in the classic sense. The other question is whether this language should be distributed, or whether it is for personal or internal company use.

- I want to create a language that facilitates programming by embracing this new compiler philosophy and for internal corporate purposes.

If you want to do this, you simply need to make a framework for the Core 4.0 compiler that implements the desired programming language.

- I want to create a programming language in the classical sense.

To do this, there are tools to study and use. If so, you need to be well aware that you are embarking on the path of programming language engineering. This road is difficult, long and requires a lot of knowledge and a lot of work. Given and considering that history has reported cases of languages that have not been successful, the advice I would like to give is to have a prototype of the programming language created, and in a short time, using Core 4.0.

Why can't I create a language in the classical sense with Core 4.0 ?

Core 4.0 is a tool that makes programming much easier. But it's not a perfect system. In this sense. Typically a framework leads to languages with the number of statement types exceeding one hundred. Predicting all the possible syntactically and semantically correct combinations is a huge job, and one that I don't do. Core 4.0 is a system that, if it receives a correct program as input, produces correct output. For programs that are incorrect in the sense of semantics, it can produce incorrect code. But the basic consideration is that it is not convenient to predict and manage all the possible errors of a language with such a large number of statement types. And since the rule of pragmatism applies to me, I prefer to concentrate on constructive work and do not encounter problems of language perfection which lead to a large amount of work, but without practical benefits.

Are there other reasons why the created languages are not suitable for distribution ?

Surely. Core 4.0 produces code belonging to various technologies. Typically those who use a programming language expect to execute the code directly. Instead, Core 4.0 produces code belonging to different technologies that clearly require those who have the experience and awareness to configure the environments and manage the generated code. Which will subsequently be executable following these processes.

Who can program with Core 4.0 ?

Anyone can program with Core 4.0. From beginner to expert. Core 4.0 was also created to allow beginners or those who do not have a thorough knowledge of the technologies behind programming to program. As long as there are reference figures within the company who are experts in order to be able to support beginners in difficulties, explaining to them how the language should be used and the exceptions. The expert par excellence is the one or are those who have actively participated in the process of creating the framework for the language.

Is Core 4.0 a compiler belonging to a well-known typology of compilers ?

The answer is no. The term "All-Purpose Multi-Technology Compiler" is used to identify this product. There is currently no scientific branch that deals with compilers that translate their programming languages into other languages and technologies used. Core 4.0 was born from an engineering-technological idea, from the need that those who program would have to easily define a language that implements their solutions. Perhaps this need has not been paid attention to by the scientific world, which is very focused on great steps forward, in great technologies for progress, and which has perhaps overlooked something which fortunately was nevertheless achieved within the world of simple engineering.

Is Core 4.0 a source-to-source compiler or a transpiler ?

Core 4.0 does not translate a well-known programming language into another well-known programming language, but allows you to quickly and easily define your own language according to your implementation needs and translate it into code of other well-known languages. For this reason it belongs and does not belong to the category of source-to-source compilers or transpilers, but brings together the concepts of both with new concepts.

So why is it "all-purpose" ?

For reasons related to the fact that it is possible to do almost anything. But for reasons that date back to the history of computing and in particular to the BASIC language. BASIC is the acronym for Beginner's All-Purpose Symbolic Instruction Code. And it is a language through which you can easily do anything. In our imagination, even languages created with Core 4.0 can have this feature. However the difference is that it is not for beginners. So we want to remove the "for beginners" from the acronym BASIC.

So why is it "multi-technology" ?

It is also multi-technology because nowadays to create something you have to use from five to twenty technologies put together.

Is it difficult to program the compiler, then make a framework ?

No it's not difficult. In fact it's something very nice to do. The Core 4.0 programming language has been developed with all the features to quickly and easily translate your language into other languages and technologies.

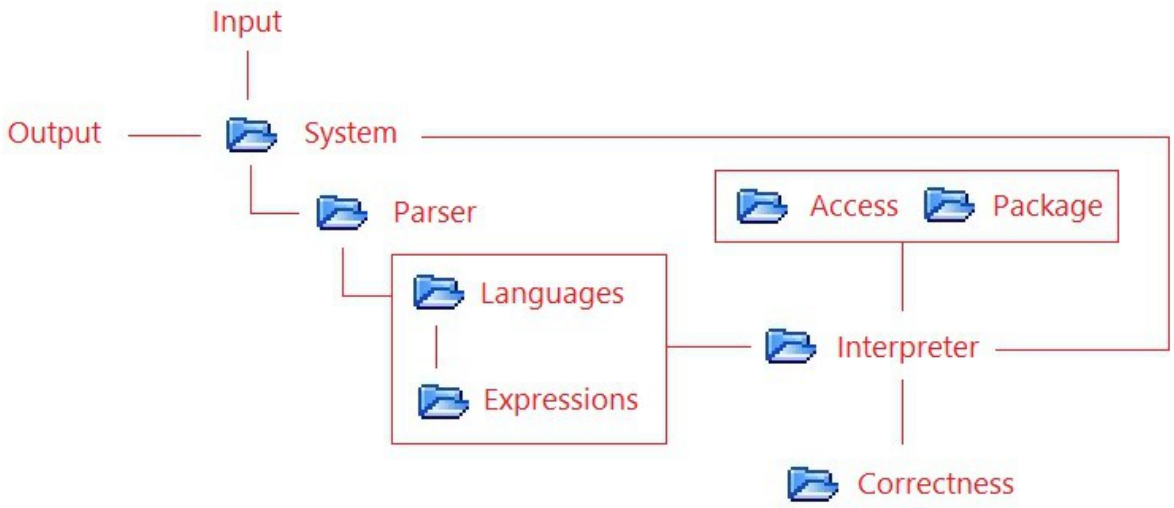
Is Core 4.0 an interpreter or a compiler ?

Core 4.0 is a compiler with the Core 4.0 compiler language inside it which is an interpreted language. So we made an interpreter to compile.

How does the Core 4.0 compiler work ?

The Core 4.0 compiler is software built for modules. Which can be explained in a simple way through a simple diagram that must be noted down. The theory around its structure and functioning is complex and has required years of work. But something in terms of explanation can be provided, indeed it is necessary to provide it.

Here is the Compiler module scheme.




Let's give a brief explanation of the modules.

System : This module takes care of checking what the compilation options are, and manages all the system entities. Such as files, reading and writing files. Input from command prompts and more.

Parser : The Parser is responsible for recognizing the statements of the various languages and verifying their syntactic correctness.

Language : The Languages module deals with associating semantics with programs written in the various languages that have been defined.

Expressions : The Expressions module deals with expressions intended as operations between variables. This module is a separate module, because it requires particular attention.

Interpreter : The interpreter interprets the code associated with the semantics of languages and expressions. And it verifies the package access rules and also the semantic correctness of the program.

Access & Package : The Access module and the Package module are used to verify correct access to the public and private environment.

Correctness : The Correctness module is used to verify the correctness of all operations performed by the interpreter and provide error messages.

What is Core 4.0 Compiler Language ?

It is the language that is interpreted by the interpreter and whose programs written with it are associated with the semantics of every other defined user language. It therefore serves to translate user languages into the languages of various technologies.
What are the features of the Core 4.0 Compiler Language ?

Primitive Types
Vectors
Objects
Structures
Scope Rules
Functions
Parameters
Reference & Values
Public and Private Envirorment
Function Calls
Return Values
Cycles
Condictions
Operators & Priority
Calculation of Expressions
Conversion Operators
System Calls
and more

Are there possible improvements to the system ?

The answer is no. There are branches of science whose theory has a beginning and an end. And others whose theory has a beginning but not an end, because it makes use of every scientific discovery inherent to it that is made. In our imagination, he thought of creating a

product whose theory has a beginning and an end. By knowing everything there is to know about the theory, you are an expert at the highest level. This is the non-trivial advantage of having made a finished product in theory and in practice.

## Consulting

The consulting service is based on your willingness to undertake a research and development period directly with the creator of the Core 4.0 Compiler. He will work in contact with people from your company, implementing the solution you asked for.

He offers : A complete consultancy aimed at improving software development times and methods through the adoption of the all-purpose Core 4.0 compiler in its specialization for the creation of a programming language that is the one you have commissioned.

He Asks : A Contract and Collaboration in the development and engineering process of the programming language. It is also required that collaborators must have a strong problem solving ability, as well as strong motivation in the success of the work.

He Gives : An executable file (result of the implementation work) that represents the compiler's specialization for engineered language. He does not provide the Core 4.0 compiler. He does not provide the compiler open to building frameworks. He does not even provide the framework open to possible modifications. He provides the language that is translated into business solutions as it is engineered.

For any changes to the language and the framework, there must be a collaboration, for skills and professionalism. This is desiderable for top notch results.

Thank you