Command Based Interpreter and Developer's Library

Prepared By

Elias Bachaalany

Outline

Introduction & Background

- Command Based Language Library for Software developers
- Teaching and academic tool for students and beginner programmers
- A replacement for DOS batch file scripting

Problem Statement

- Lack of command line interfaces in emergent GUI programs
- Redundancy of work and no easy way to batch-in repeated commands / tasks
- Too complicated: Modern programming languages are too complicated for beginners
- Lack of simple and integrated programming tools

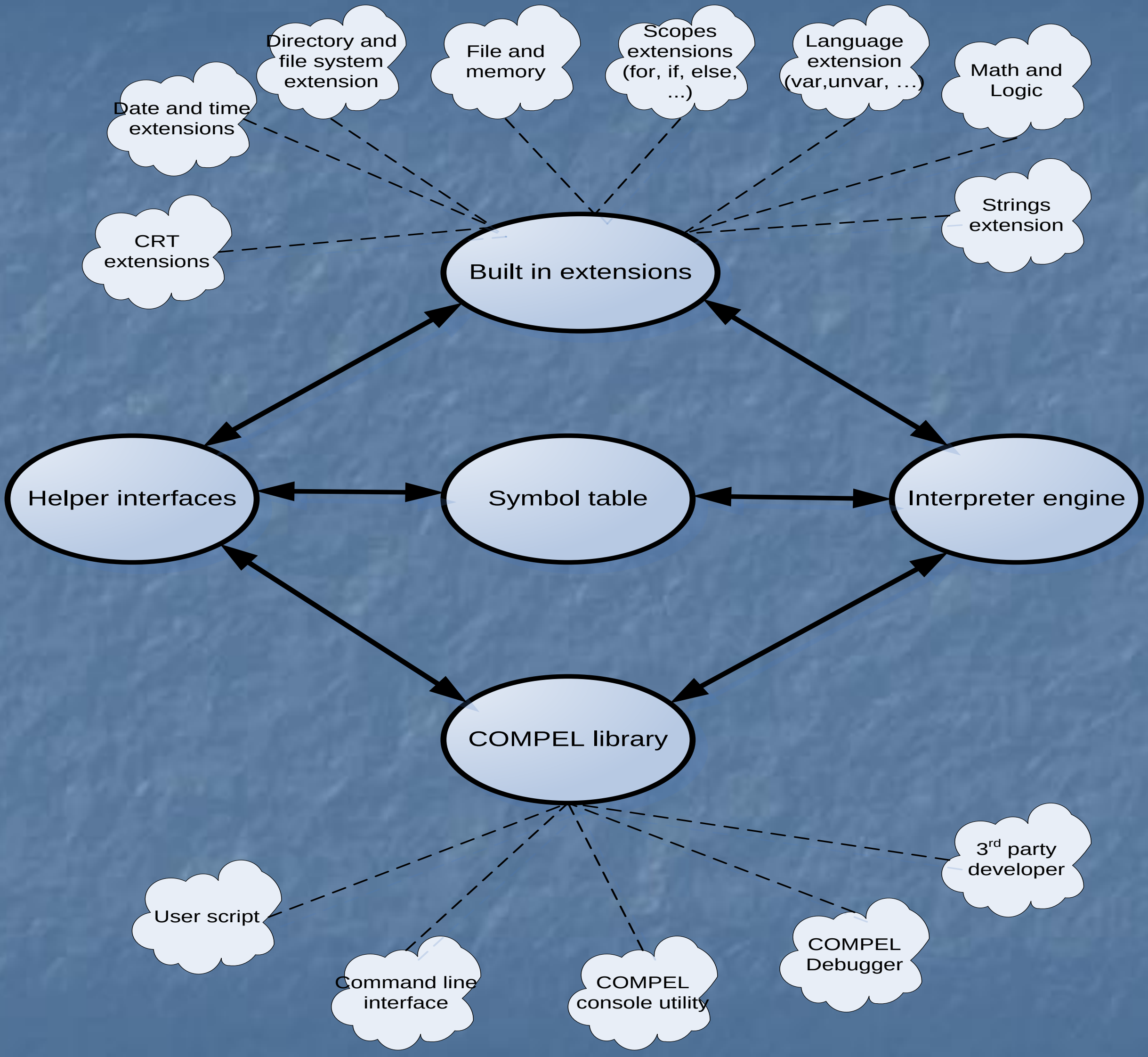
Purpose of Project

- Give developers a solution to easily integrate command line programming into their application with little work
- Provide teachers and academic institutions a simple tool to teach programming concepts
- A powerful tool and replacement of DOS batch files

Possible Solutions

- Flexible/easily pluggable developer library that require no effort from the part of the implementer
- Comprehensive tool covering most of academic and teaching purposes needs
- General purpose scripting tool

Design Specifications – Engine overview



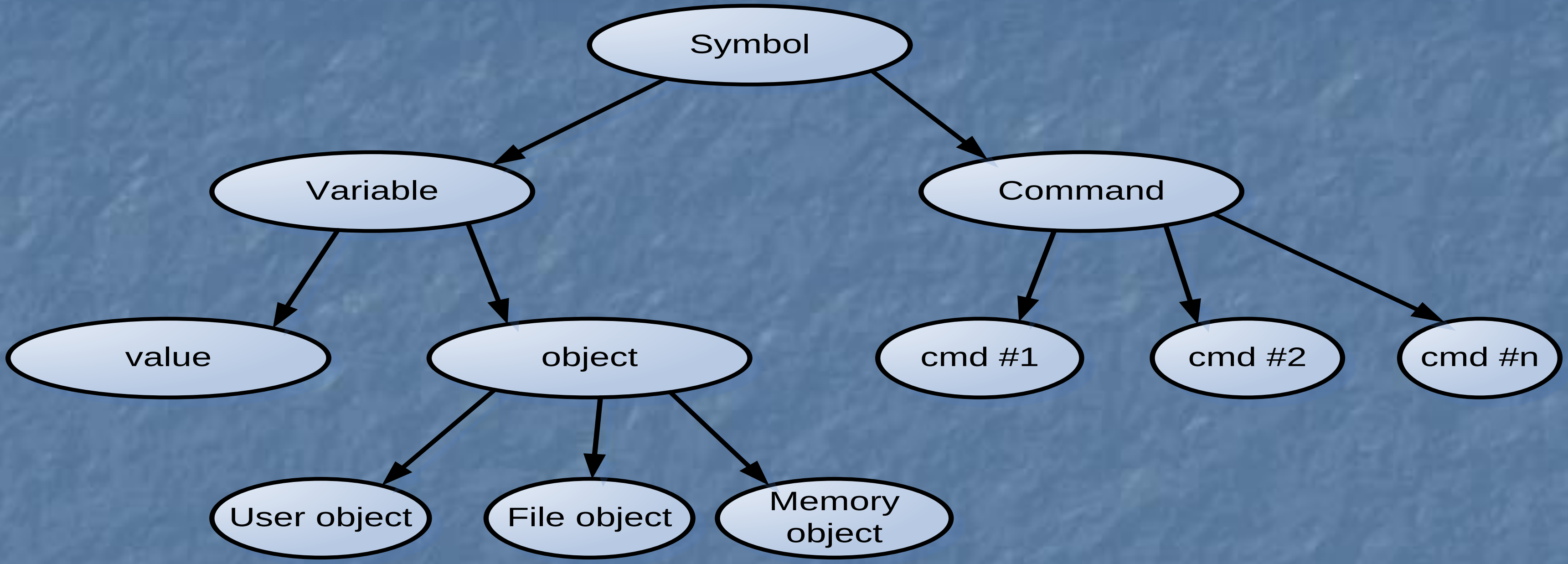
Design Specifications – Engine overview

Engine design

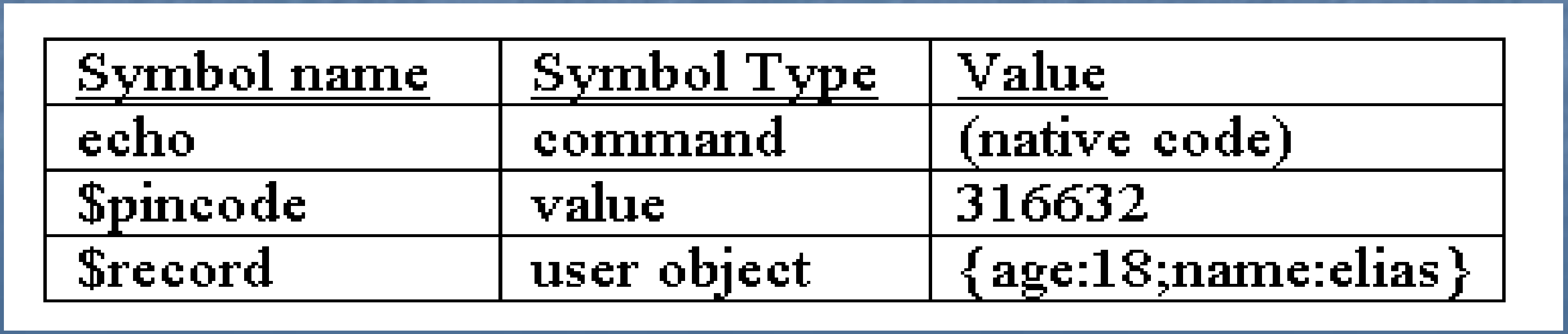
- **Modular: The engine is modular as seen in the previous graph**
- **Extensible: Allows easy extension and modification of code**
- **Portability: It is designed with high level of abstraction thus allowing portability**
- **Contribution: Extensions can be created by third party users and plugged into the system**
- **Language features: A wide range built-in extensions giving COMPEL a powerful stance among other programming languages**

Design Specifications – Symbol table overview

Symbol table hierarchical view:



Symbol table tabular view (sample):



Symbol name	Symbol Type	Value
echo	command	(native code)
$pincode	value	316632
$record	user object	{age:18;name:elias}

Symbol table design

- **Hierarchical: Symbols derive from basic type to different advanced typed.**
- **Inherent relationship: Different symbols can be inherently transformed into other symbol types**
- **Commands/Functions: Even commands and functions are considered as symbols and are managed by the symbol table manager**

Implementation Languages

- C++ language used for the engine development
- Borland Delphi was used for the COMPEL IDE tool
- .NET and other languages used to build demo application and COMPEL extensions

Why use C++ for the COMPEL engine?

- C++ is a portable language
- C++ allows natural expression of objects and OOP programming principles
- C++ generates fast and tight code thus the speed of execution
- C++ compilers are widely available and mostly for free

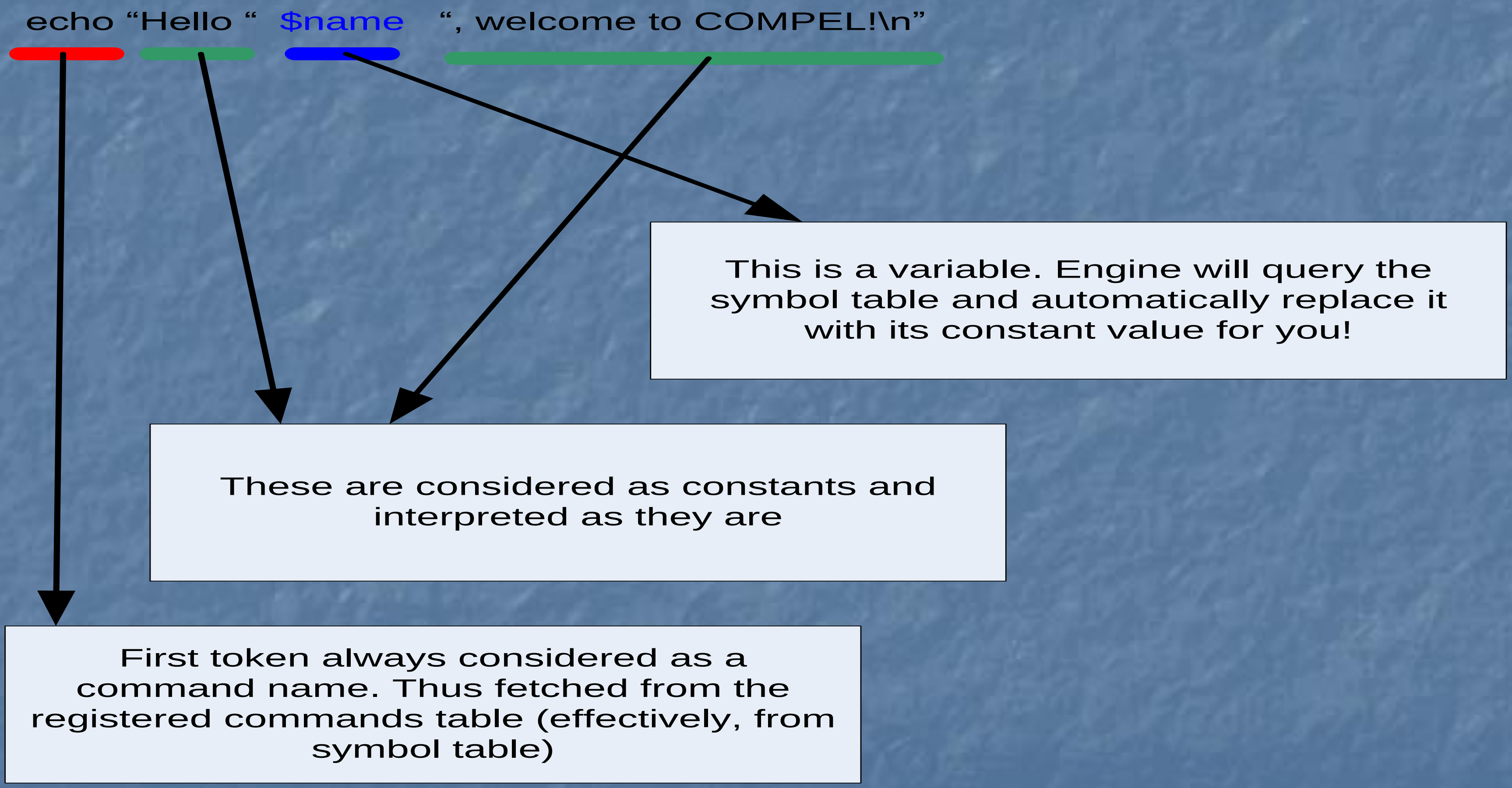
Why use Delphi for the COMPEL IDE tool?

- Borland Delphi / VCL is the best choice for RAD (rapid application development) and GUI programming
- Delphi code can be ported to Linux using the Kylix tool
- Borland Provides a free personal non-commercial use of their Delphi tool

Design Constraints

- Unicode scripts are not supported
- IDE does not support multiple scripts per workspace
- Scripts cannot be compiled into p-code
- IDE is not symbol aware
- Language is too simple thus advanced data structures cannot be easily described

Description of System Operation



```
echo "Hello " $name ", welcome to COMPEL!\n"
```

This is a variable. Engine will query the symbol table and automatically replace it with its constant value for you!

These are considered as constants and interpreted as they are

First token always considered as a command name. Thus fetched from the registered commands table (effectively, from symbol table)

Equipment Configuration

Developer (COMPEL Library):

- **Windows Operating system**
- **Windows development tool such as: C++, Delphi, Visual Basic or .NET**

End user (COMPEL interpreter):

- **Windows Operating system**
- **A command prompt tool (such as cmd.exe)**
- **32MB of RAM**
- **2MB disk space**

Testing & Verification

- Interpreter detects pre-parse / run-time script errors
- Correct symbol evaluation and command registration
- IDE / Debugger can debug and edit scripts

Testing Results

- Interpreter runs large script with a reasonable amount of time
- Accurate, relevant, complete, and concise information about variables and program state when using the debugger
- Simple and tight code can yield a handy application

Conclusions & Future Work

Conclusions:

- COMPEL language is simple and user friendly
- COMPEL IDE assists as a visual tool to develop and debug programs
- COMPEL developer library is well documented and developer-friendly

Future work:

- Complete re-design of the COMPEL grammar to allow dynamic script grammar
- Allow the compilation of a script into p-code
- Enhance the debugging engine
- Enhance the developer library and allow more control to the COMPEL engine internals for developers