C# Fact Extractor Phase 2

BY

Muhammad Ahmad

Institute of Information Technology

Quaid-i-Azam University

Islamabad, Pakistan

DEDICATION

I would like to dedicate this thesis to my loving Parents and Teachers, with their support and encouragement, most important their trust and inspiring personality of my supervisor motivate me at each difficult stage.

Project Brief

Project Title:	C# FactExtractor Phase 2
Internal Supervisor:	Mr. Abdul Qudus Abbasi
Developed By:	Muhammad Ahmad
Development Tools:	Microsoft Visual Studio 2006 and 2012
Operating System:	Windows 7
Submission Date:	23rd, June, 2014

Abstract

Software systems are growing day by day and becoming more complex due to quickly changing market requirements and upcoming new technologies. Maintenance of software system is an expensive and time taking activity but mostly work of software development is done by maintenance instead of new development. Program comprehension and architectural understanding is very important for maintenance. There is great need to understand the relationships and dependencies between different entities in source code of a software system. The relationships between software entities, their dependencies and organization help understand the nature of the system to better re-establishment of software system. To comprehend an unfamiliar system there is need to understand many things about it. In this regard we demonstrate a reverse engineering approach and carry out detailed study of object oriented relationships to extract relationships from software systems.

We have developed FactExtractor that is tool for reverse engineering, maintenance software system. FactExtractor analyzes the software system at source code level. Firstly in first phase my system manages input files of C# project to analyze the source code and will hold all the files and with path and file names and store them in vector for futher processing. Second phase of fact extractor takes input of from previous phase and do lexical analysis of stored files and generate tokens of these files and store these tokens in vector for more processing of next phase. Third phase of fact extractor is to parse these token files and store this information in desired data structure for further processing to the next phase. Fourth phase is build relationships and find out the dependencies. Overall system will identify the source code characteristics, relationships and present the extracted information in XML based formatted structure and text file format as well as. FactExtractor accept C# projects as input from user and generate an abstract representation of information so a user can understand object oriented relationships between software entities. I have developed two phases completed and third phase is half has been done and I am doing this project with my best efforts by the supervision of my respected teacher “sir Abdul Qudus”. InshaA ALLAH I will complete this project. Now this system is underdeveloped by me.

Acknowledgment

All praises are due to Almighty Allah who had given blessing, knowledge and strength for establishing me to complete this project, and enlightening the right path for me. I am deeply indebted to my parent for all their unconditional love, generous support and throughout guidance for setting up highest target.

I would acknowledge my sincere, thanks to my supervisor Mr. Abdul Qudus Abbasi for his patience, enthusiasm, and immense knowledge. His guidance, motivation and invaluable support during my project lifted me up in my spirits.

I would like to express the deepest appreciation to all my class mates who struggled together to help finish my studies during the course of 2 years. I will forever be grateful for their happiness, support and suggestions during my hard and happy times.

Last but not the least I would like to offer my special thanks to my friends, seniors and juniors for their love, trust and best wishes for my project.

Muhammad Ahmad

Contents

1.1 Introduction

Software is a collection of computer programs, data and documentation. The computer programs perform desired tasks on data and documentation provides the useful information about software. Computer programs are set of instruction written in computer programming language carry out a specific task. There are different types of programming languages available with some specific rules to develop software. FactExtractor and interpreter are used to convert the language syntax into understandable form of computer.

Computer software is a conceptual entity. It can't be presented in physical form. Software exists in the logical form of code and this code is recorded on some medium. When there is a change request in software then corresponding code is also needed to be changed. As the software grows, relationship between the entities of software becomes complex. Due to complexity it becomes difficult to understand and to maintain or modify software. To increase the understandability of software and minimize the complexity there are different tools and methodologies developed and still there is space for new ones. The complexity of software also increased due to the increased number of platforms and availble languages. The number of hardware and software platforms is significantly increased. The developer must be now aware of environment or platform of software before development.

The object oriented approach introduced in 1960 but its good use to develop software systems started in 1990s. [1]. In early object oriented system like the structured system, the architecture design phase and architecture design documentation were not given importance. Now a days object oriented languages facilitates a huge software development community to meet their deadlines effectively by improving quality and understandability of code. Object oriented helps the development and maintenance of complex systems and promotes software reusability. Object oriented programming supports the development of applications by configuring the existing software systems.

1.2 Problem Definition

There are mostly products already developed for same or slightly different problem. Software products for telecommunication, educational and commercial use are same or similar but build again and again. So the same functionality is developed at different places.

Another problem in software development is software systems rapidly growing and becomes complex due to changing market requirements, advancement and continuously changing technologies. Now software is not preferred to develop from scratch, but using integration and modification of already developed systems. To use already developed systems there is need to understand many things about it. So to develop software using reuse methods or to modify or maintain an existing system there is need to comprehenced the system and all the elements of system. One has to completely understand the system by reading line of codes and extracting facts from code. As manual fact extraction is a time taking process, so we thought a solution for fact extraction of C# source code.

1.3 Purpose of project

The purpose of this project is to develop a tool that would extract the meaningful information from C# source code. Extracting meaningful information means uncovering the facts, entities, modules and relationships between modules of a software system. This is an application of software reverse engineering.

"The process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. Reverse engineering is the process of design recovery". [2]

The project outcome will be a tool that would first separate the source code elements into tokens. Secondly the source code elements would be represented using intermediate data structures and finally the facts would be displayed using matrices and xml formatted structure.

The main objective of developing this tool is to support programmers and maintainers to understand the code as efficiently and effectively as possible. The tool will enable its users to extract as many facts as possible (from the code). Programmers can easily extract all relationships between classes, members of any class and function calls etc. Also it will save time and effort needed to maintain code.

1.4 Scope

Source code is reliable source of information about software system. Due to the complexity of software in various languages, it is difficult for a developer to understand the source code to make change. Because in this case developer must have to study targeted system modules and then extract their dependencies and relationships to implement desired change. The reason is that most software is large and complex, and it is difficult and time consuming for a developer to understand the code complexity. To overcome this complexity there should be a mechanism that provides a solution to the developer in terms of fact extraction. We need automated support to extract information about source code entities and their relationships.

Since C# programming language took great achievement over other programming languages due to its extra features and functionality. Microsoft introduced the C# language to provide platform independent systems up to some extent. C# is enjoying great edge over other languages in small duration. So our fact extractor will base on C# language. We will build a fact extractor for all C#

developers to save their time while working on a system new to them by providing basic information about classes used in that system. With the use of C# fact extractor the developer can get basic information about C# source code in few minutes without wasting their time.

Existing Fact Extractor are also serving this purpose and provides extracted information in matrix based view. Our Fact Extractor will provide some extra relationship information than existing fact extractors. Our Fact Extractor will also display extracted information in matrix based view as well as string view.

1.5 Resource Identification

Software Resources

Operating System	Windows7
Programming Language	Visual C++
IDE	Visual Studio 2006 and 2012

Hardware Resources

The project is being developed under following Hardware resources.

Processor	Corei5
RAM	4GB
Hard Disk	640GB

1.6 Requirement Analysis

Introduction

Requirements are a specification of what should be implemented. Requirements are descriptions of what should be implemented. Systematic requirements analysis is also known as requirements engineering. It is also referred as requirements gathering, requirements capture, or requirements specification. The deliverable of this phase is called software requirement specification document (SRS).

Functional requirements

Functional requirements specify what functions or features the system is required to provide. Functional requirements are what and how user interacts with system and specific task is performed:

- ❖ Create new project

User wants to create a new project when there is need to parse a project for facts.

- ❖ Open project

User wants to open an existing project. If selected files are not C# code files then error message would be displayed in dialog box.

- ❖ Save project

Save information on xml file format and store in any data structure.

- ❖ Tokenize the file

User can tokenize the file after selecting a specific file. If file contain any kind of error then dialog will display error message.

- ❖ Generate relationship

Relationships will be generated.

- ❖ View tokenize file

User can view any specific tokenized file.

- ❖ View relationships

Relationships can be displayed as list view.

- ❖ Parse project

User wants to parse the opened project to generate token files. The information about folder, file, class, function and attribute is extracted at parsing time.

- ❖ Generate output

User will generate desired format of resulted files of project. E.g. xml.

2 Study of relationships

Files and Folders

Source code organization is an important aspect of software development, therefore files and folders can be considered as source code elements (Riva, 2000). Most of times source code is organized into different files for better code management and these are source files containing source code elements (Kazman, O'Brien, 2003). The relationship between source code elements through files and folders is based on two facts:

The occurrence of source code of programming elements in same file. If source code of two elements occurs in same file then there are chances that they are closely related and more dependent on each other.

One file includes the other one. This relationship shows that one program element is dependent on the other. In computer based software development, source files related to component are mostly organized in same folder. However if component is large a folder hierarchy can be maintained. This folder hierarchy can give significant clues about composition of components (A Qudus, 2008). [1]

Object Oriented Based Relationships

Different relationships according to object oriented domain are discussed below.

2.1 Inheritance Based Relationships

Implementation inheritance means that a type derives all the member fields and functions of base type. Implementation inheritance is useful when requirements are to add functionality to an existing type, or where a number of related types share a significant amount of common functionality. [1]

Implementation inheritance depth

This relationship provides information whether a class is direct child of other class or lies down in the hierarchy. Consider a class MyDerivedClass which inherits its functionality from some base class MyBaseclass. There is another class MyDerivedClass1 which inherits its functionality from MyDerivedClass.

Example is illustrated with the help of code as

```
MyBaseClass {

// functions and data members

};

Public class MyDerivedClass: MyBaseClass {

//Junctions and data members

};
```

Public class MyDrivedClass1: MyDerivedClass {

//Junctions and data members

};

The relationship field will contain value “1” between MyBaseClassand MyDrivedClass.In the case of MyBaseClass and MyDrivedClass1 it has value “2”.If two classes have not above mentioned relationship then the relationship field has value “0”. [1]

Sibling

If two classes share a base class then these classes will said to be have sibling relationship between them.

Public class MyDerivedClass: MyBaseClass {

//Junctions and data members

};

Public class MyDrivedClass1: MyBaseClass {

//Functions and data members

};

If above mentioned relationship is true then field value of this relationship will be “1” and “0” otherwise. [1]

Override virtual method

Inheritance facilitates us to customize a inherited function according to our requirements by declaring a method as a virtual method in base class. Then this virtual function can be override in derived class to fulfil our requirements. When the overridden method is called, the appropriate method for the type of object is invoked. Methods are not virtual by default. We have to explicitly declare them virtual by using the keyword virtual. C# differs from other languages syntax because it requires you to use override keyword while overriding a virtual function in derived class.

```
Class MyBaseClass {
Public virtual string VirtualMethod (); {
Return “This method is virtual and defined in MyBaseClass”;
}
};
Public class MyDrivedClassl: MyBaseClass {
Public override string VirtualMethod (); {
Return “This method is an override defined in MyDerivedClass"
};
};
```

The relationship field will contain value “0” if one class is derived from other class and does not override any method. If class overrides virtual method of base class then field value of this relationship contains “1”. If there exists no inheritance relationship between given classes then the field value of relationship will contain “-1”. [1]

Child Interface

Interfaces can inherit from one another as the classes do. We will illustrate this concept by defining two interfaces, IBankAccount and ITransferBankAccourt. IBankAccount has two methods. ITransferBankAccount has the same features as IBankAccount but also defines a method to transfer money directly to a different account:

```
Namespace myNameSpace

{

Public interface IBankAccount

{

Void setTitle (string title); void setAccountNo (string accountNo);

}


Public interface ITransferBankAccount: IBankAccount

{

BoolTransferTo (IBankAccount destination, decimal amount);

}

}
```

If given two interfaces have same relationship as IBankAccount and ITransferBankAccount then field value for this relationship will contain "1" and "0" otherwise. [1]

Implemented Interface

If a class inherits interface then it has to implement all the methods declared in that interface. If a class inherits child interface then it must have to implement all the methods declared in the child.

Interface as well as in the parent interface, otherwise compilation error occurs. A class can inherit more than one interface.

```
MylmpiementedClass: InterfaceOne, Interface Two {

// Class Implementation

//InterfaceOne Implementation

//InterfaceTwo Implementation

};
```

This relationship field value will contain “1” if a class inherits interface and “0” otherwise. [1]

Interface inheritance depth

This relationship informs whether an interface is inherited directly from other interface or indirect inheritance exists between them, consider an interface Myderivedinterface which inherits its functionality signatures from some base interface MyBaseinterface. There is another interface MyDerivedinterface1 which inherits its functionality signatures from MyDerivedinterface. Example is illustrated with the help of code as

```
MyBaseinterface {

//Only method signatures

};

Public interface MyDerivedlnterface: MyBaseinterface {

// only method signatures

};


Public interface MyDrivedlnterfacel: MyDerivedInterface {

// only method signatures
};
```

The relationship field will contain value “1” between MyBaseinterface and MyDrivedlnterface. In the case of MyBaseinterface and MyDrivedlnterface1 it has value “2”.If two interfaces have not above mentioned relationship then the relationship field has value “0”. [1]

2.2 Containment

In containment one class is added as a member of another class. It is also referred as composition. By using containment we can avoid declaring fields again and defining their related functions. Code clarity is also achieved by containment.

Containment as an object

Consider a class Contained having basic fields and methods related to that fields. There is another class Container which contains some extra fields than class Contained. So to avoid redundancy and for clarity we can use class Contained as a data member in class Container instead of declaring basic fields again and defining their related methods.

```
Class Contained {

// Basic info

};

Class Container {

Contained basiclnfo;

//Enhanced info

};
```

This relationship field value will contain “1” if Container class contains Contained and “0” otherwise. [1]

Two classes using same class

Consider a class Contained having basic field and methods related to that field. There are two other classes Container1 and Container2 which contains some extra fields than class Contained. So to avoid redundancy and for clarity we can use class Contained as a data member in class Container1 as well as in Container2 instead of declaring basic fields again and defining their related methods.

```
 Class Contained {

// Basic info

};

Class Container 1 {

Contained basiclnfo;

// enhanced info

};

Class Container2 {

Contained basiclnfo;

//enhanced info

};
```

This relationship field value between Container1 and Container2 will contain “1” if both Container1 and Container2 contain Contained and “0” otherwise. [1]

Containment at Method Parameter Level

Consider a class Contained having basic field and methods related to that field. There is another class Container which contains a method that accepts a parameter of type Contained,

Class Contained {

// Basic info

};

Class Container {

// Container class implementation

Void acceptContained (Contained containedParam);

};

This relationship field value will contain “1” if some method of Container class accepts parameter of type Contained and “0” otherwise. [1]

Containment at Method level declaration

Consider a class Contained having basic fields and methods related to that fields. There is another class Container which contains a method that declares object of type Contained,

Class Contained

{

Study of Object Oriented Relationships

// Basic info

};

Class Container {

// Container class implementation

Void declareContainedObj ( )

```
{
Contained containedObj;
}
};
```

This relationship field value will contain “1” if some method of Container class declares object of type Contained and “0” otherwise. [1]

Two Classes Using Containment at Method Parameter Level

Consider a class Contained having basic field and methods related to that field. There exist two classes Container1 and Container2. Both classes contain a method that accepts a parameter of type Contained,

```
Class Contained {
// Basic info
};
Class Container1 {
//Container class implementation
Void acceptContained (Contained containedParam);
};
Class Container2 {
// Container class implementation
Void acceptContained (Contained containedParam);
};
```

This relationship field between Container and Contained will contain “1” if some method of both classes accepts parameter of type Contained and “0” otherwise. [1]

Two Classes Using Containment at Method Level Declaration:

Consider a class Contained having basic fields and methods related to that fields. There exist two classes Container1 and Container2. Both classes contain a method that declares object of type Contained,

```
Class Contained {

// Basic info

};

Class Container1 {

// Container class implementation

Void declareContained ()

{

Contained containedObj;

}

};

Class Container2 {

// Container class implementation

Void declareContained ()

{

Contained containedObj;
}

};
```

This relationship field between Container 1 and Container2 will contain “1” if some method of both declare object of type Contained and “0” otherwise. [1]

2.3 Files and folder based relationships

Folders contain files and files contain data. In our scenario files data is C# source code. There could be one are more than one classes in a source file. Similarly there may be one or more than one files in folder. Possible relationships based on files and folders in object oriented domain are described below.

Classes from same file

If both given classes share same file then field value of this relationship will be true. Source code example of given two classes say MyClassA and MyClassB is as follows.

```
Public class MyClassA {

//data and function members....

};

Public class MyClassB {

//data and function members

};
```

The filed value of this relationship will hold “1” in that case and “0” otherwise and then we check the condition folder based relationships as below paragraph. [1]

Classes from same folder

Classes sharing same file are obvious to share same folder. But there is a possibility that classes belong to different files but they may be share same folders. The filed value of this relationship will hold “1” in both cases and “0” otherwise. [1]

Class and Implemented Interface are from same file

If given class and implemented interface share same file then field value of this relationship will be true. Source code example of given class say MyClass and given interface IMylnterface is as follows.

```
Public interface lMylnterface {

// methods signatures

};

Public class MyClass: IMyInterface {

// Implementation of IMylnterface

// other class members

};
```

Class and Implemented Interface are from same folder

Given interface and class share same file then they are obvious to share same folder. But there is a case that given classes from different file and implemented interface exists in some other file but they share same folder. The filed value of this relationship will hold “1” in both cases and “0” otherwise. [1]

Summary

In this chapter, we defined a set of Object Oriented Relationships which will be considered during process of fact extraction. These relations are defined in context of inheritance, containment, files and folders.

3 List of tools for static code analysis [4]

3.1 Multi-language

ConQAT - Supports C#, C#, C++, C#Script, and many other languages. ConQAT implements algorithms for detecting redundancy, architecture analysis. Furthermore, it integrates established tools, like **FindBugs**, **FxCop** etc. using processors that read their output formats. Although ConQAT supports different output formats (e.g. XML), usually generated HTML files are used to present the analysis results. Visualizations like different types of diagrams, treemaps, architecture diagrams etc.

Coverity SAVE – A static code analysis tool for C, C++, C# and C# source code. Coverity commercialized a research tool for finding bugs through static analysis, the Stanford Checker, which used abstract interpretation to identify defects in source code. Under a United States Department of Homeland Security contract, the tool was used to examine over 150 open source applications for bugs, 6000 bugs found by the scan were fixed, across 53 projects.

GrammaTech CodeSonar – Defect detection (buffer overruns, memory leaks, etc.), concurrency and security checks, architecture visualization and software metrics for C, C++, and

C# source code. CodeSonar is a source code and binary code analysis tool that performs a whole-program, interprocedural analysis on C, C++, C#, and binary executables. It identifies programming bugs and security vulnerabilities in software. CodeSonar is used in the Defense/Aerospace, Medical, Industrial Control, Electronic, Telecom/Datacom and Transportation industries.

Imagix 4D – Imagix 4D helps software developers comprehend complex or legacy C, C++ and C# source code. By using Imagix 4D to reverse engineer and analyze your code, you're able to speed your development, enhancement, reuse, and testing. Eliminate bugs due to faulty understanding. A comprehensive source code analysis tool, Imagix 4D enables you to rapidly check or systematically study your software on any level -- from its high level architecture to the details of its build, class and function dependencies.

Parasoft – Provides static analysis (pattern-based, flow-based, in-line, metrics) for C, C++, C#, .NET (C#, VB.NET, etc.), JSP, XML, and other languages. Through a Development Testing Platform, static code analysis functionality is integrated with unit testing, peer code review, runtime error detection and traceability.

Parasoft develops automated defect prevention technologies that support the Automated Defect Prevention methodology developed by Adam Kolawa. These technologies automate a number of defect prevention practices for C#, C and C++, and .NET. The static code analysis practice identifies coding issues that lead to security, reliability, performance, and maintainability issues . Parasoft also develops memory error detection technology that finds run-time errors in C and C++ programs.

PMD is a static rule-set based C# source code analyzer that identifies potential problems like: Possible bugs—Empty try/catch/finally/switch blocks. Dead code—unused local variables, parameters and private methods. Empty if/while statements. Overcomplicated expressions—Unnecessary if statements, for loops that could be while loops. Suboptimal code—

Wasteful String/StringBuffer usage. Classes with high Cyclomatic Complexity measurements. Duplicate code—Copied/pasted code can mean copied/pasted bugs, and decreases maintainability. While PMD does not officially stand for anything, it has several unofficial names, the most appropriate probably being Programming Mistake Detector.

Polyspace –.Polyspace examines the source code to determine where potential run-time errors such as arithmetic overflow, buffer overrun, division by zero, and others could occur. Software developers and quality assurance managers use this information to identify which parts of the code are faulty or proven to be reliable.

IBM Rational AppScan Source Edition – Analyzes source code to identify security vulnerabilities while integrating security testing with software development processes and systems. Supports C/C++, .NET, C#, JSP, ColdFusion, Classic ASP, PHP, Perl, VisualBasic 6, PL/SQL, T-SQL, and COBOL

MALPAS – A software static analysis toolset for a variety of languages including Ada, C, Pascal and Assembler (Intel, PowerPC and Motorola). Used primarily for safety critical applications in Nuclear and Aerospace industries.

SonarQube –

Supports 25+
languages: C#, C/C++, C#, PHP, Flex, Groovy, C#Script, Python, PL/SQL, COBOL, etc. (note that some of them are commercial)

Can also be used in Android development.

Integrates with the Eclipse development environment

Veracode – Finds security flaws in application binaries and bytecode without requiring source. Supported languages include C, C++, .NET (C#, C++/CLI, VB.NET, and ASP.NET), C#, JSP, ColdFusion, PHP, Ruby on Rails, and Objective-C, including mobile applications on the Windows Mobile, BlackBerry, Android, and iOS platforms.

Axivion Bauhaus Suite – A tool for Ada, C, C++, C#, and C# code that performs various analyses such as architecture checking, interface analyses.

DMS Software Reengineering Toolkit – Supports custom analysis of C, C++, C#, C#, COBOL, PHP, Visual Basic and many other languages.

Black Duck Suite – Analyzes the composition of software source code, searches for reusable code, manages open source.

Visual Studio Team System – Analyzes C++, C# source codes. Only available in team suite and development edition.

3.2 Doxygen

Generate documentation from source code

Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, C#, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other UNIX flavors as well. Furthermore, executables for Windows are available.

Features [3]

Requires very little overhead from the writer of the documentation. Plain text will do, Markdown is support, and for more fancy or structured output HTML tags and/or some of doxygen's special commands can be used.

Generates structured XML output for parsed sources, which can be used by external tools.

Supports C/C++, C#, (Corba and Microsoft) C#, Python, VHDL, PHP IDL, C#, Fortran, TCL, Objective-C 2.0, and to some extent D sources.

Supports documentation of files, namespaces, packages, classes, structs, templates, variables, functions, typedefs, enums and defines.

C#Doc (1.1), qdoc3 (partially), and ECMA-334 (C# spec.) compatible.

Allows grouping of entities in modules and creating a hierarchy of modules.

Generates a list of all members of a class (including any inherited members) along with their protection level. Can cope with large projects easily.

3.3 Imageix 4D

Imagix 4D helps software developers comprehend complex or legacy C, C++ and C# source code. By using Imagix 4D to reverse engineer and analyze your code, you're able to speed your development, enhancement, reuse, and testing. Eliminate bugs due to faulty understanding. Get new hires on board faster. Spend time engineering, not reading through code.

A comprehensive source code analysis tool, Imagix 4D enables you to rapidly check or systematically study your software on any level -- from its high level architecture to the details of its build, class and function dependencies. You can visually explore a wide range of aspects about your software - control structures, data usage, and inheritance. All based on Imagix 4D's precise static analysis of your source code.

You're able to find and focus on the relevant portions of your source code through Imagix 4D's querying capabilities. Automated analysis, database lookups, and graphical querying all sift through the mountains of data inherent in your source code so you examine the structural and dependency info you're interested in. Quickly and accurately.

Relationship Types [5]

A relationship is a directed, binary dependency between two symbols. The Imagix 4D database recognizes the following relationship types. Unless otherwise noted, the relationship data is collected and updated by analyzing your source code.

Aggregates

class1 => class2: Represents class2 being used to declare members of classl.

Base Class Of

Class 1 => class2: Represents class2 inheriting from classl.

Calls

Function => function, function => macro: Represents macro being expanded when function is pre-processed.

Contains

directory => directory or file file => symbol class => member function => local_variable

Declares

file => symbol class => member

3.4 SolidFX - Fact Extractor for C/C++

The Fact Extractor (SolidFX) is a standalone, non-intrusive framework for static analysis of industry-size projects written in the C and C++ programming languages. SolidFX uses proprietary technology to analyze even the most complex C/C++ code bases efficiently and robustly. SourceFX offers predefined analysis scenarios for:

reverse engineering C/C++ code;

detecting potential threats and bugs;

Measuring code quality, maintainability and modularity.

Feature

SolidFX supports a wide range of tasks. SolidFX supports most C and C++ dialects and platforms and is able to quickly analyze multi-million line projects, even if the code is incorrect and/or incomplete. The extracted information can be queried at levels ranging from detail information over each line of code to automatic reverse-engineered, full system architectural diagrams. SolidFX comes with several visualization plug-ins showing diagrams, call-graphs, and tens of quality metrics ranging from individual lines of code to entire subsystems. New plug-ins can be easily added to compute custom quality checks and to interface with third-party tools. [6]

3.5 SolidSX

The Software Explorer (SolidSX) is a standalone Windows application that gives insight in large software systems. SolidSX creates information-rich visualizations that show structure, metrics and dependencies between different types of source code elements (files, classes, methods, fields, etc.). By using hardware-accelerated graphics, SolidSX is able to display large amounts of information in a clear and concise manner and provides fast and easy exploration of large source codes.

Features

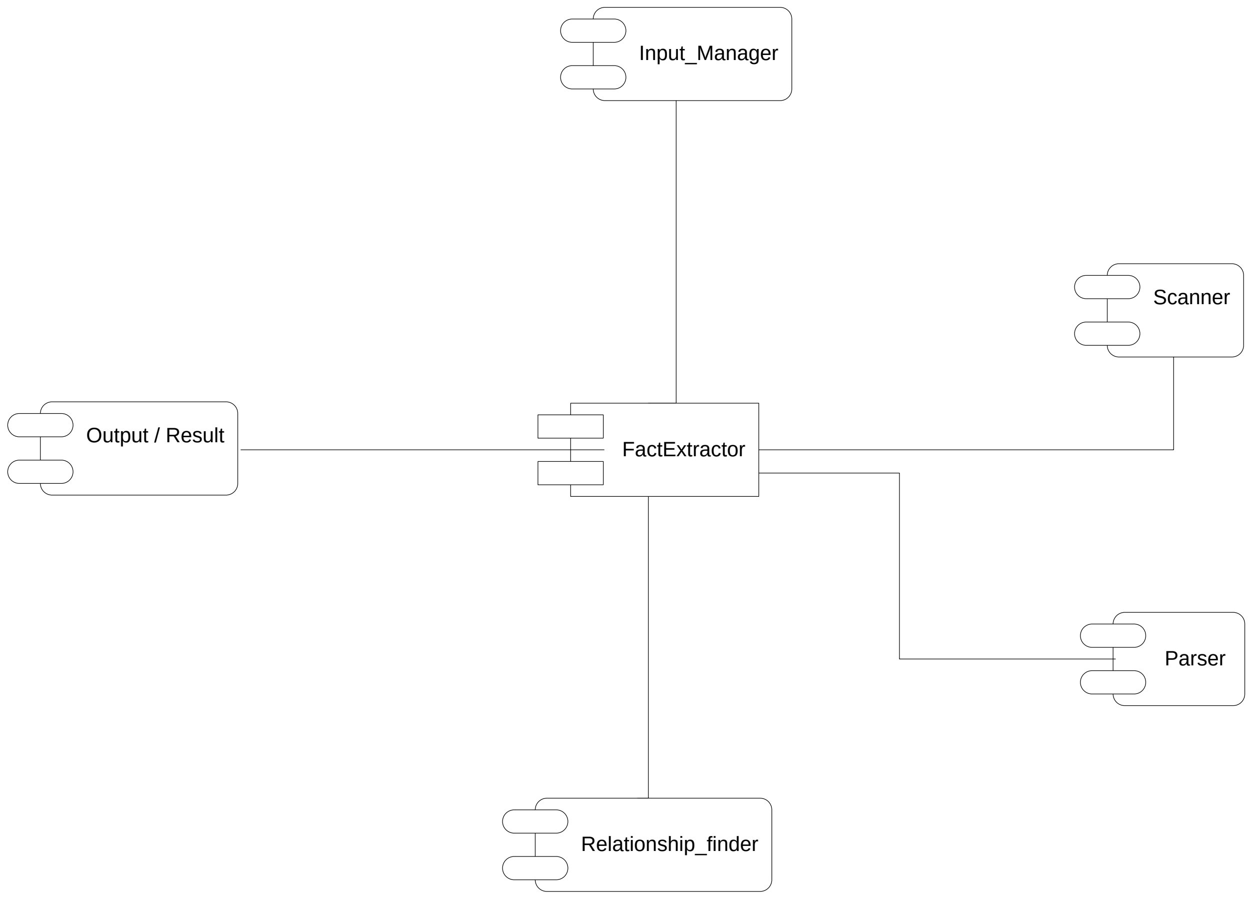
Extracts and visualizes call graphs, class inheritance, type usage, field accesses and various code metrics, such as method complexity, lines of code, and number of casts (some plug-ins might not support all of these metrics). Plug-ins are available for analysing software written in Visual Basic, C#, C++ and C.The source code is not needed to analyze .NET assemblies, so SolidSX can also be used to inspect third-party assemblies (similar to .NET Reflector).Open input data format. Easy to integrate with third-party tools. SolidSX is not restricted to visualize software, but can also be used as a generic dependency viewer see the User (Manual for more information). Explore large databases of tens of thousands of elements in real time, from packages to individual variables. Store all settings of a view, such as its visualization parameters, filter settings, aggregation settings, etc. so that you can later reopen them later on the same or other datasets.Hardware-accelerated graphics for high quality presentation and smooth navigation.Extremely easy to use: point and click in a simple interface. Ready to use in less than 5 minutes (including installing). [7]

4 What is Software Architecture?

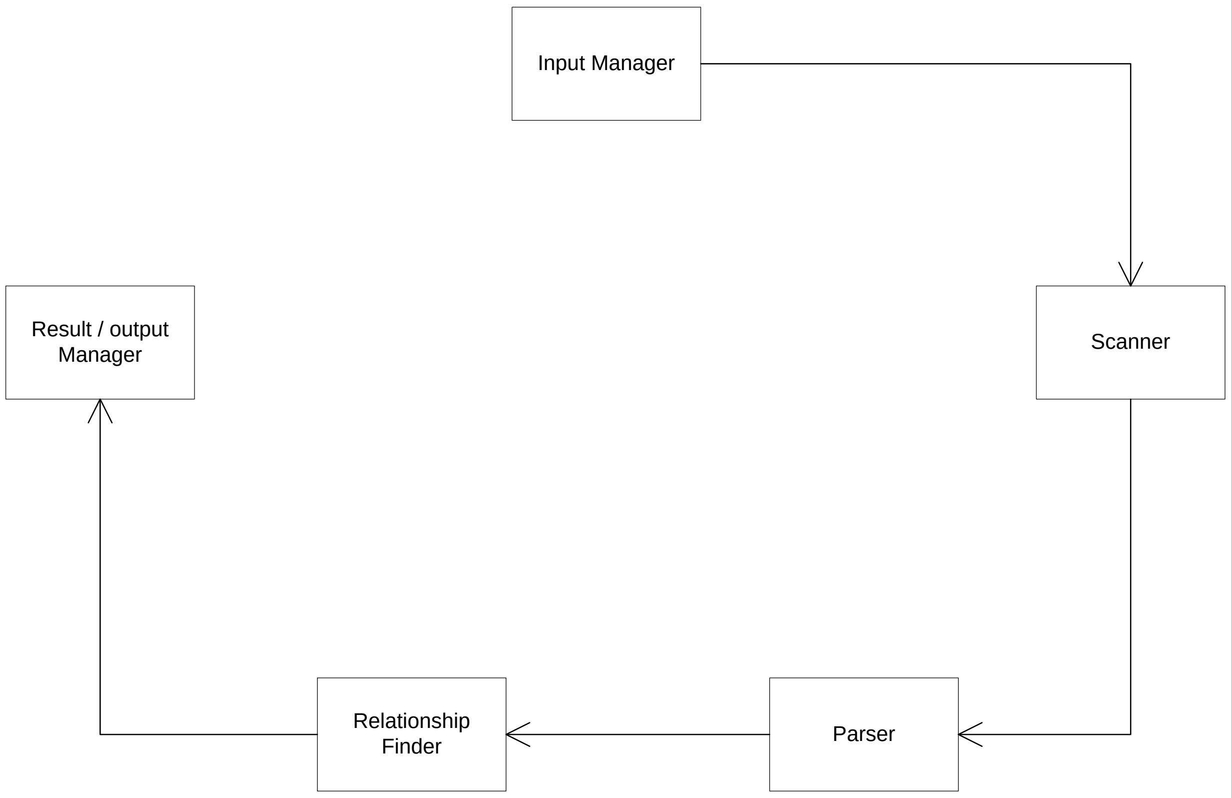
Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

Nelu Suciu (Solution Architect, ISDC, Cluj, Romania): The core element of any software system is data. Along with data any software system provides the following capabilities: collection, processing, storage, presentation, and distribution of data. Software architecture has a two-fold purpose: first it has to design the balanced mix of capabilities to support the realization of business goals, and second it needs to use an optimal mix of implementation technologies regarding hardware, software, and operations. A good software architecture can be recognized when the business goals are fulfilled by the functionality delivered, within the estimated time and cost. [8]

4.1 Architecture Diagram



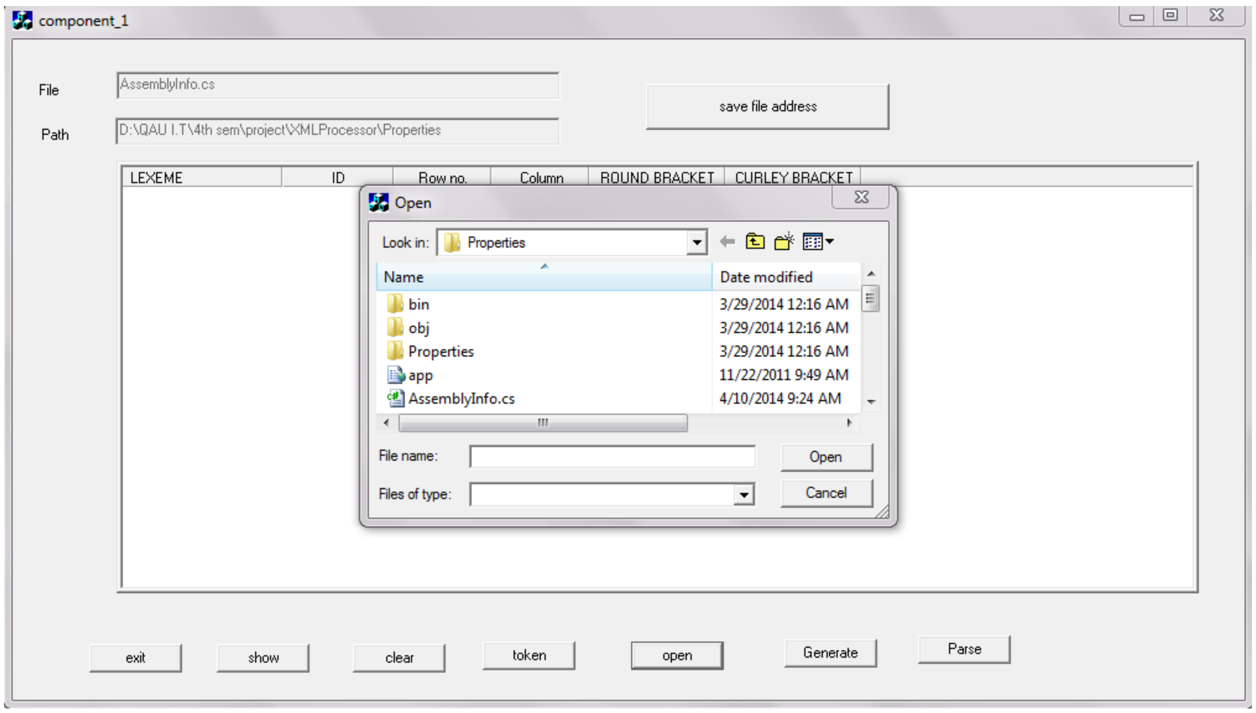
4.2 Component Interaction Diagram



4.3 Component Description:

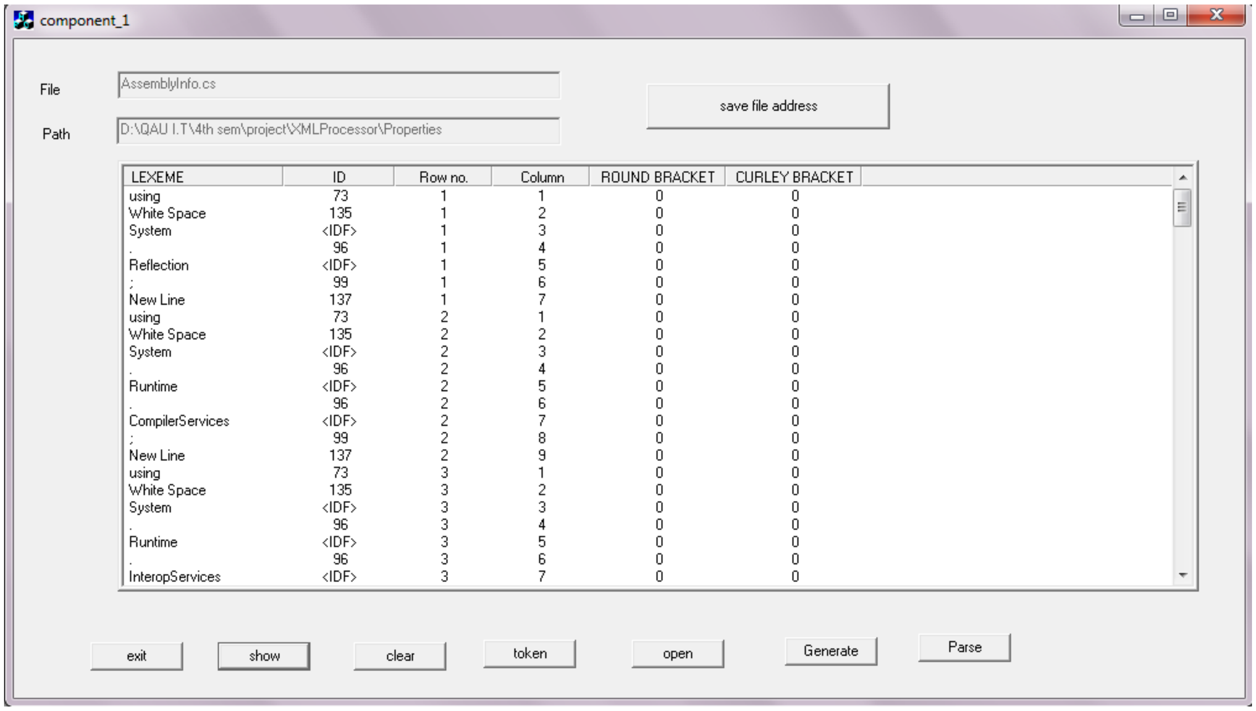
Input Manager:

Input manager is component of Fact Extractor that is used to locate and getting input different C# project files. Using input manager user can select and add project files into system. Input manager is used to add .cs source files into system that are to be analysed. Input manager consists of different classes that help to locate and add source files into system while traversing through the folder hierarchy. Input manager communicates with main component "FactExtractor". The information that is browsed and added by input manager in the form of list that is used by other components. Input manager also provides a standard interface to user for browsing files.



Scanner:

Scanner is another major component that is connected with the main component name "FactExtractor". Scanner consists of number of classes that performs lexical analysis and store the analysed information into a data structure. The source files collected by input manager are then passed to scanner component for scanning. This component takes source files as an input and produces tokens of source code. The sequence of token produced by lexical analyzer is used by syntax analyzer as an input. Lexical analyzer breaks the sequence of characters into tokens. Lexical Analyzer recognizes the particular instances of tokens called lexemes. Output of this component is in the form of list of tokens.



File: AssemblyInfo.cs

Path: D:\QAU I.T\4th sem\project\XMLProcessor\Properties

LEXEME	ID	Row no.	Column	ROUND BRACKET	CURLEY BRACKET
using	73	1	1	0	0
White Space	135	1	2	0	0
System	<IDF>	1	3	0	0
.	96	1	4	0	0
Reflection	<IDF>	1	5	0	0
;	99	1	6	0	0
New Line	137	1	7	0	0
using	73	2	1	0	0
White Space	135	2	2	0	0
System	<IDF>	2	3	0	0
.	96	2	4	0	0
Runtime	<IDF>	2	5	0	0
.	96	2	6	0	0
CompilerServices	<IDF>	2	7	0	0
;	99	2	8	0	0
New Line	137	2	9	0	0
using	73	3	1	0	0
White Space	135	3	2	0	0
System	<IDF>	3	3	0	0
.	96	3	4	0	0
Runtime	<IDF>	3	5	0	0
.	96	3	6	0	0
InteropServices	<IDF>	3	7	0	0

Parser:

Parser is another important component of my Fact Extractor. Parser uses the output produced by scanner component. The list of tokens produced by scanner component is used by parser component to analyze the syntax and extract the useful information from list of tokens according to the rules of language. Parser component reads the tokens and represents the elements of language into a data structure.

Relationship Finder:

Relationship finder is also connected with FactExtractor named main component of my Fact Extractor. The purpose of relationship finder is to find the relationships from the information generated by parser component. Relationship finder component uses the data structures that contain the information about individual source code elements. Relationship finder find the dependencies and relationships of source code elements.

Output Handler:

This component is also connected with main component named “FactExtractor”. Once the relationships are discovered, specified output files are generated. This component will generate output as xml files as well as user can view resultant files in List View control in the form of matric and strings.

4.4 COMPONENT INTERACTION:

Each component will provide external interaction through the main component named "FACTEXTRACTOR”.

Component Relationships:

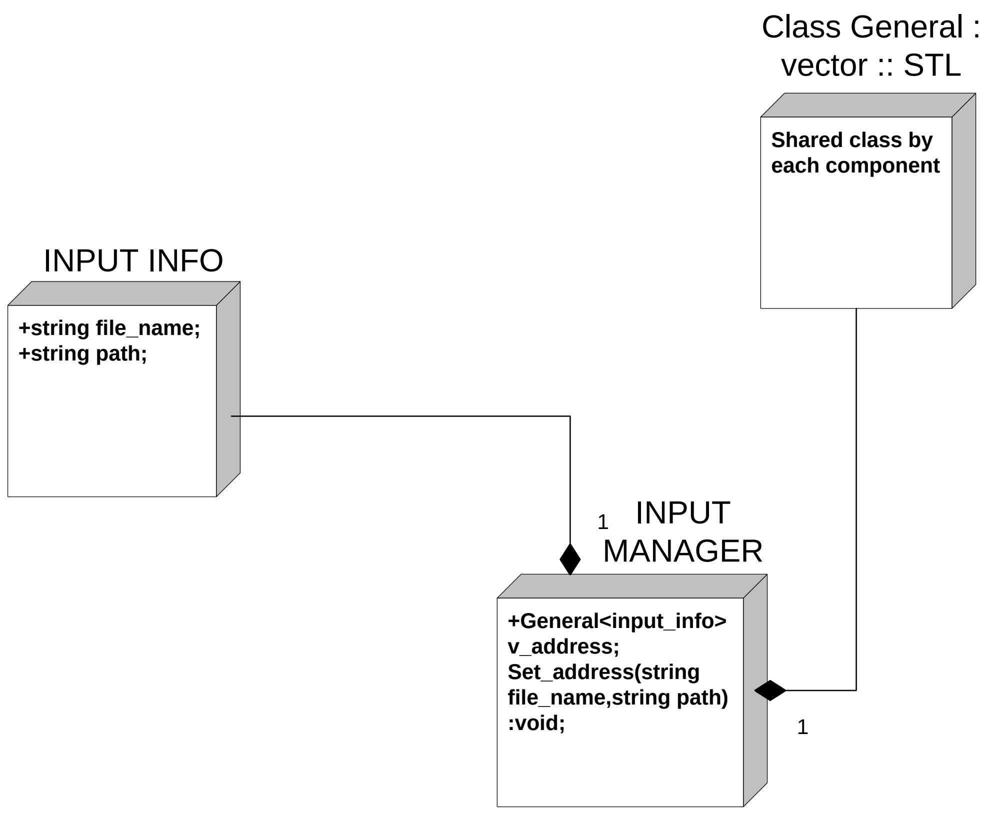
1. Input Manager will get input complete C# project from user saves files path with titles this all information will be useful to the Scanner component .

2. Scanner component manipulate all data which is taken by input manager component and tokenize all files and then will communicate with the Parser component.

3. Parser component takes all tokenize data and extract the information about classes, methods, variables etc. and then will communicate with the Relationship Finder component.

4. Relationship Finder component will take extracted information from parser and then will generate relationships and communicate with the Output Manager component that provides the results in our desired format.

Input manager class diagram


Class General :
vector :: STL
Shared class by each component
INPUT INFO
+string file_name;
+string path;
1
INPUT MANAGER
+General<input_info> v_address;
Set_address(string file_name,string path) :void;
1

Input info

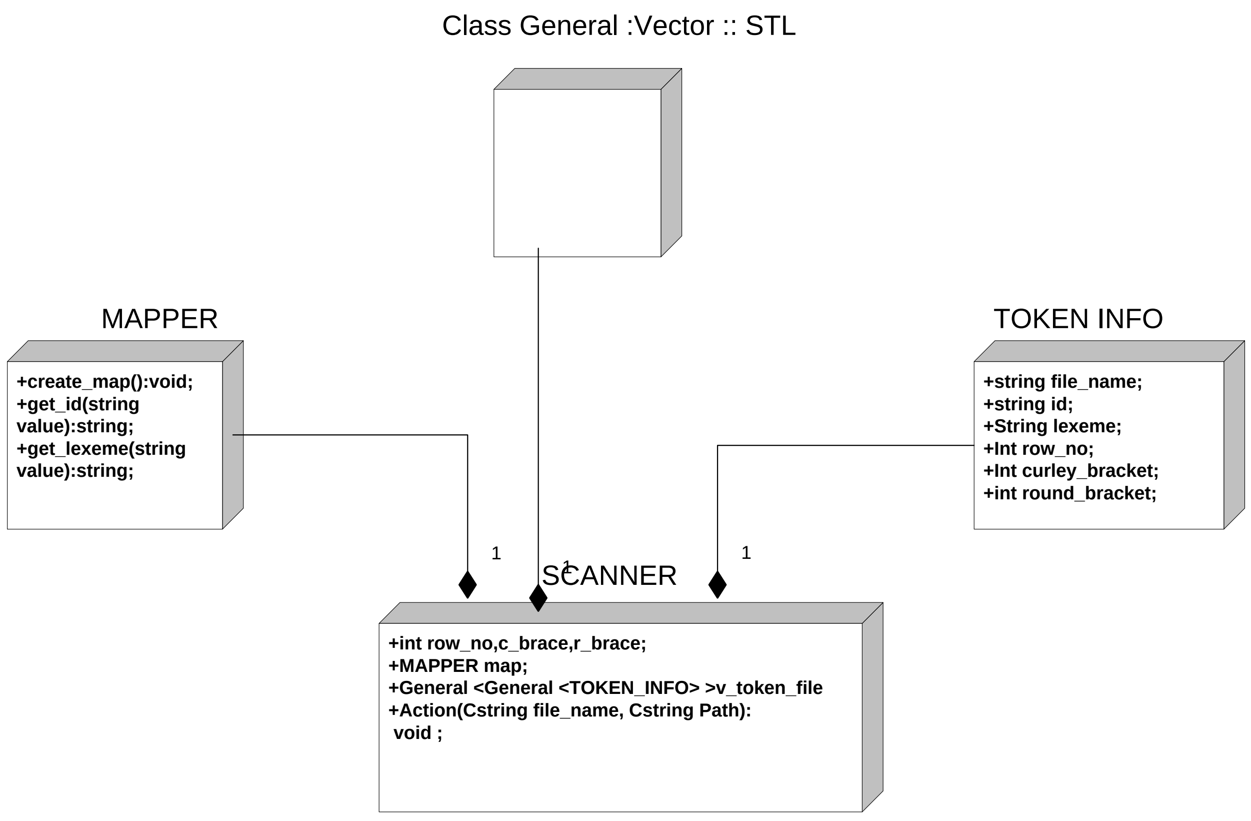
This class maintains the struct of input. Which will be used by input manager class. To use vector of STL we have derived a General class from vector which is used in input manager class.

Input manager

Through this class file name and path of input file will be save in vector for futher use.

Set address method will be used to maintain the vector that would save file name and path.

Scanner class diagram


Class General :Vector :: STL
MAPPER
+create_map():void;
+get_id(string value):string;
+get_lexeme(string value):string;
TOKEN INFO
+string file_name;
+string id;
+String lexeme;
+Int row_no;
+Int curley_bracket;
+int round_bracket;
1
1
1
SCANNER
+int row_no,c_brace,r_brace;
+MAPPER map;
+General <General <TOKEN_INFO> >v_token_file
+Action(Cstring file_name, Cstring Path):
void ;

Mapper

Mapper class is responsible for mapping all tokens to their specified ids. Initially all keywords and operators are assigned specific Ids and loaded to a STL container map. Map is a STL generic container in this scenario it is implemented for a string value and numeric id. When a token is found it is first looked into map to find out its id. Using this id and token value, object of class Token is created and added to vector of tokens.

Token Info

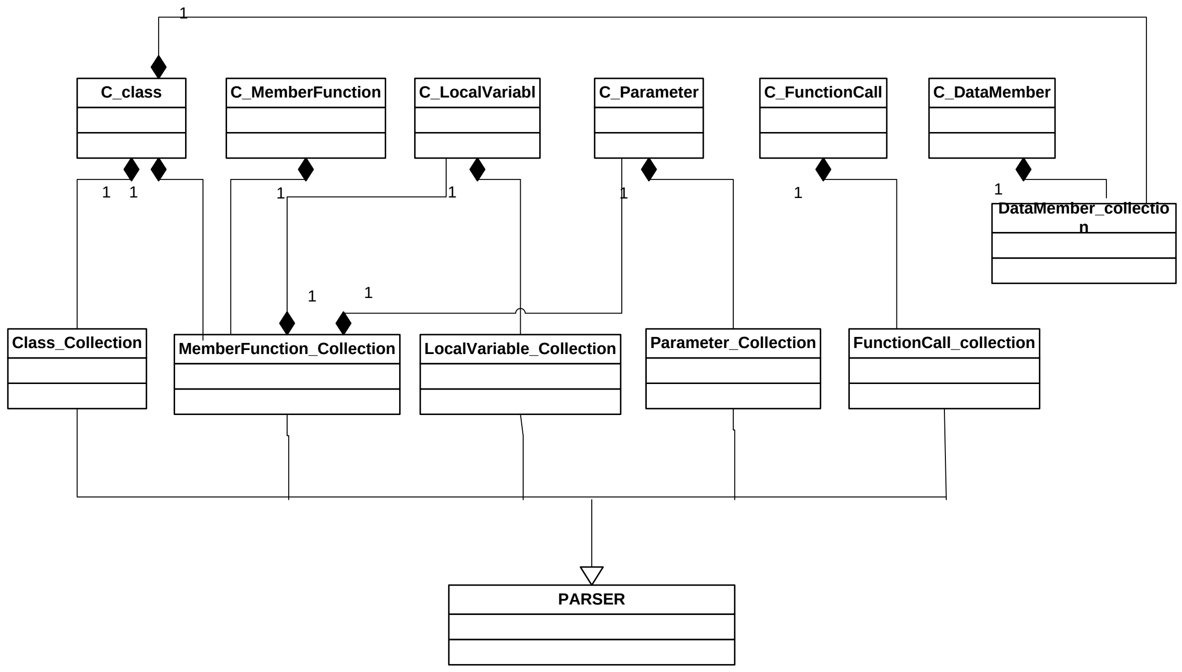
Token info class holds the attributes that will provide all the information about the token. E.g token i.d,

Lexeme, row number in which token found, brackets level in which token occurring.

Scanner

Scanner acts as manager class for all classes used for scanning. Scanner class interacts with all classes; provide necessary information or data required by these classes. Scanner class is responsible for opening a file and then using a parser method extracting tokens. When a token is extracted from file its ID is retrieved from Mapper class. After getting required information about token, it is added to a vector of tokens.

Parser class Diagram



C_Parameter

C_Parameter class contains information about parameters of a method. Parameter type can be user define or primitive type. A parameter can be passed in two states by reference or by value. C_Paramter class contains all these characteristics about parameter.

C_LocalVariable

C_LocalVariable class contains the information about local variables declared in any method of C# class. C_LocalVariable class maintains the information of a variable like name, variable type and name of file in which variable is found.

C_MemberFunction

C_MemberFunction class contains the information about functions or methods of a C# class. C_MemberFunction maintains the information about function name, function type and the name of file in which function is declared. C_MemberFunction class contains object of C_Parameter and C_LocalVariable to maintain information about local variables and parameters of a function.

C_DataMember

C_DataMember class contains the information about data members of a C# class. C_DataMember class maintains the information about data member name, type, class and the name of file.

C_Class

C_Class contains information about C# classes. C# class can contain methods and data members. C_Class contains C_DataMember class objects to maintain information about data members.

C_Class contains C_MemberFunction class objects to maintain information about methods of a C# class.

C_FunctionCall

C_FunctionCall class maintains the information about function call with file details and class details in which that particular function is called.

ParameterCollection

ParameterCollection class consists of collection of all parameters of methods of C# classes. ParameterCollection class contains a vector of type C_Parameter.

FunctionCollection

FunctionCollection class consists of collection of all methods C# classes contain. FunctionCollection class contains the collection of methods in a vector of type Function.

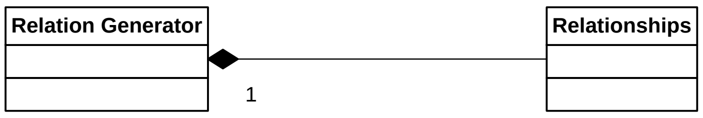
LocalVariableCollection

LocalVariableCollection class consists of collection of all local variables declared in methods of C# classes. LocalVariableCollection contains all C_LocalVariable class objects in STL vector.

DataMemberCollection

DataMemberCollection class is also a collection class which contains collection of data members of C# classes.

Relationship Finder Class Diagram



Relationships

Relationship class define the all possible relationships that are to be discovered in project

Relationship Generator

Relationship Generator class contains the information about relationships of different entities of system.

Output Handler

This phase of my proposed system will generate the out of our system in the form of xml. This xml will be contained on the following tags: <MAINTAINER> , <Class> , <Class_Name> , <Data_Members>

<DM> , <Functions> , <FM> , <Parameter> , <Local_Variable> , <Func_Call>

Proper format of generated xml will be as :

```
<?xml version="1.0" encoding="utf-8"?>

<MAINTAINER>

<Class>

        <Class_Name>CComboOwnDraw</Class_Name>

        <Data_Members>

        <DM name = "pTreeView" type = "CView" access_specifier = "public" attrib =
"pointer"></DM>

                <DM name = "m_BackDisableFlag" type = "BOOL" access_specifier = "public" attrib =
"none"></DM>

                <DM name = "m_cmbFont" type = "CFont" access_specifier = "public" attrib =
"none"></DM>

                <DM name = "pEdit" type = "CEdit" access_specifier = "public" attrib = "none"></DM>

                <DM name = "nSlectedItemIndex" type = "int" access_specifier = "public" attrib =
"none"></DM>

        </Data_Members>
```

```
	<Functions>

		<FM name = "CComboOwnDraw" return_type = "no">

			<Parameters>no</Parameters>

			<Local_Variables>no</Local_Variables>

			<FuncCall>no</FuncCall>

		</FM>

		<FM name = "DrawItem" return_type = "void">

			<Parameters>

				<PM name ="lpDrawItemStruct" type = "LPDRAWITEMSTRUCT"
attrib = "none" ></PM>

			</Parameters>

			<Local_Variables>

				<LV name = "style" type = "UNIT"></LV>

				<LV name = "dc" type = "CDC"></LV>

			</Local_Variables>

			<FuncCall>

				<FC name = "Attach" calling_obj = "CDC"></FC>

				<FC name = "GetUpperBound" calling_obj = "ComboEntry"></FC>

				<FC name = "CBrush" calling_obj = "CBrush"></FC>

			</FuncCall>

		</FM>

		<FM name = "FillComboBox" return_type = "void">

			<Parameters>

				<PM name ="nSelIndex" type = "int" attrib = "none" ></PM>

			</Parameters>

			<Local_Variables>

				<LV name = "Index" type = "int"></LV>
```

```
                <LV name = "i" type = "int"></LV>

                <LV name = "rect" type = "CRect"></LV>

            </Local_Variables>

            <FuncCall>

                <FC name = "GetSystemMetrics" calling_obj = "int"></FC>

                <FC name = "GetRect" calling_obj = "CEdit"></FC>

                <FC name = "GetDC" calling_obj = "CDC"></FC>

            </FuncCall>

        </FM>

    </Functions>

</Class>

<MAINTAINER>
```

5 Testing and Evaluation

5.1 Introduction

To evaluate the performance and functionality of developed system test systems are conduct. To evaluate our system we proposed two test systems. These test systems are developed in C# language. Our system is intended to extract the facts from source code so these test systems are carefully chosen to evaluate the developed system. These systems are compliable with any C# standard FactExtractor and error free. If this project that is used to test our proposed system will not error free then wrong results will be produced.

5.2 Test system

Blood inventory management system project is useful for blood banks and hospitals where blood in collected and stored. Blood management and donation is one of the important and tough job, blood banks plays important role in collecting blood from donors and test sample and store them. There is need to manage available blood of each group every time for knowing and collecting blood and always maintain in stock. In existing system all this data is managed through manual process which is time taking and in efficient process and retrieving old records is not easy. In order to solve these problems we implement a software application called blood inventory management system where data is stored in database and data can be viewed on web.

File names of this test system are given below with some important details:

Files

File name	Extension	size	LineOfCode	ClassName
CrystalReport1	.cs	5248	154	CrystalReport1
				CachedCrystalReprt1
CrystalReport2	.cs	5248	154	CrystalReport2
				CachedCrystalReport2
CrystalReport3	.cs	5248	154	CrystalReport3
				CachedCrystalReport3
CrystalReport4	.cs	5248	154	CrystalReport4
				CachedCrystalReport4
CrystalReport5	.cs	5248	154	CrystalReport5
				CachedCrystalReport5
CrystalReport6	.cs	5248	154	CrystalReport6
				CachedCrystalReport6
Form1	.cs	357	20	Form1

Form1.Designer	.cs	1000	37	Form1
frmAbout	.cs	1096	37	frmAbout
frmAbout.Designer	.cs	6977	146	frmAbout
frmBank	.cs	7937	207	frmBank
frmBank.Designer	.cs	29058	62	frmBank
frmDonor	.cs	8989	228	frmDonor
frmDonor.Designer	.cs	29976	62	frmDonor
frmLogin	.cs	2073	71	frmLogin
frmLogin.Designer	.cs	6690	149	frmLogin
frmMembers	.cs	9691	237	frmMembers
frmMembersDesigner	.cs	31897	573	frmMembers
frmNewUsers	.cs	1859	62	frmNewUser
frmNewUsers.Designer	.cs	5712	133	frmNewUser
frmPass	.cs	1924	61	frmPass
frmPass.Designer	.cs	5727	133	frmPass
frmPatient	.cs	9083	228	frmPatients
frmPatient.Designer	.cs	29641	536	frmPatients
frmReport	.cs	3501	91	frmReport
frmReport.Designer	.cs	15025	272	frmReport
frmSales	.cs	7750	216	frmSales
frmSales.Designer	.cs	27221	484	frmSales

frmPlash	.cs	1245	54	frmSplash
frmSplash.Designer	.cs	4725	104	frmSplash
frmTechnician	.cs	9421	234	frmTechnician
frmTechnician.Designer	.cs	30557	538	frmTechnician
home	.cs	355	20	home
home.Designer	.cs	5185	131	home
Main	.cs	355	20	Main
Main.Designer	.cs	6262	150	Main
MDIMain	.cs	5439	174	MDIMain
MDIMain.Designer	.cs	16390	273	MDIMain
Parent	.cs	2632	89	bbank
				bank
				Parent
Program	.cs	503	22	Program

REFERENCES

1: Thesis by Sir Innam Chathha.

2: Roger S.Pressman, “Software Engineering”.

3: http://www.stack.nl/~dimitri/doxygen/

4: http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#Multi-language.

5: from user guide of imageix 4D.

6: http://www.solidsourceit.com/products/SolidFX-static-code-analysis.html#features.

7: from user guide of solidSX.

8: http://www.sei.cmu.edu/architecture/start/glossary/community.cfm.

9: Thesis by Sir Usman Saeed.

- (Natural language parsing for fact extraction (**jens nilson, Welf Lowe, Johan Hall** Vaxjo University, School of mathematics and system engineering**,** IEEE).
- Teach yourself visual c++ 6 in 21 days.
- (university of tampere, Department of computer sciences, M.sc thesis, supervised by **Eleni Berki** October 2008)
- (**Yaun Lin, Richard c. Holt, Andrew J. Malton**.. School of computer science university of waterloo, 2003 , IEEE)
- Imageix 4D user guide
- Light weight lexical source model extraction (**David Notkin** university of Washington)
- Compiler principles, techniques and tools (**Alfred V.Aho, Ravi sethi , Jeffry D.Ullman**)