

DEVELOPMENT OF A SEARCH TOOL FOR CARDIAC DEVICE TEST
LIBRARIES AND EXTERNAL INSTRUMENT INTERFACE
SPECIFICATION

A Project
Presented
to the Faculty of
California State University, Chico

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
© Udaykumar Shah 2007
Fall 2007

DEVELOPMENT OF A SEARCH TOOL FOR CARDIAC DEVICE TEST
LIBRARIES AND EXTERNAL INSTRUMENT INTERFACE
SPECIFICATION

A Project

by

Udaykumar Shah

Fall 2007

APPROVED BY THE DEAN OF THE SCHOOL OF
GRADUATE, INTERNATIONAL, AND INTERDISCIPLINARY STUDIES:

Susan E. Place, Ph.D.

APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

Benjoe A. Juliano, Ph.D., Chair

Seung-Bae Im, Ph.D.

PUBLICATION RIGHTS

No portion of this project may be reprinted or reproduced in any manner unacceptable to the usual copyright restrictions without the written permission of the author.

ACKNOWLEDGMENTS

I would like to dedicate this project to my lovely parents, wife and other family members. It was impossible to achieve this goal without their continuous support and blessings.

I would also like to express deepest appreciation to my committee chair, Dr. Ben A. Juliano, and my committee member, Dr. Seung-Bae Im, for their continuous guidance and encouragement.

Special appreciation goes to Polly Huntsinger and Donna Wang who have been supportive all along my project. They have provided me with the ideas and insights in the Cardiac Rhythm Management (CRM) devices libraries and External Instrument Interface Specification.

TABLE OF CONTENTS

	PAGE
Publication Rights	iii
Acknowledgments	iv
List of Figures.....	vii
Abstract.....	viii
 CHAPTER	
I. Software Verification and Validation.....	1
Software Testing Difficulty	1
Importance of Automation in Software Testing.....	3
II. Background of the Project	4
Relative Research	5
Project Deliverables.....	10
Plan of Action.....	11
III. Implantable Cardiac Device Firmware Testing.....	13
Midlevel.....	13
The Clinical Layer.....	13
Structure of the Clinlayer Libraries	14
Cardiac Device Parameter Categories	15
IV. Search Engine Design and Implementation	16
Parser	16
Search Tool.....	16
Graphical User Interface.....	16
Reporting	16
System Interactions	17

CHAPTER	PAGE
Technology and Data Structure Selection	18
Clinlayer	19
Programming	23
Interrogation	26
Clinlayer Functions	28
 V. User Guide	 32
Clinlayer User Interface	32
Interrogation User Interface	36
Programming User Interface	37
Clinlayer Function User Interface	39
 VI. Summary and Conclusion.....	 42
Benefits from the Project Implementation	43
Measured Impact	44
Limitations and Future Scope.....	44
 References	 46
 Appendices	
A. Definitions/Acronyms	49
B. Clinlayer Function Report	52
C. Interrogation Report	54
D. Programming Report	56

LIST OF FIGURES

FIGURE	PAGE
1. CRM Parameters and Test Libraries.....	5
2. Verification and Validation Dependencies	9
3. System Block Diagram	17
4. Midlevel Clinlayer Class Diagram.....	20
5. Clinlayer Class	21
6. Program Class	24
7. Interrogation Class	27
8. Clinlayer Function Class.....	29
9. Clinlayer Midlevel Application initial User Interface.....	33
10. Clinlayer Midlevel Application Cardiac Device Parameter User Interface	34
11. Interrogation User Interface	36
12. Programming User Interface	38
13. Clinlayer Function User Interface.....	40

ABSTRACT

DEVELOPMENT OF A SEARCH TOOL FOR CARDIAC DEVICE TEST LIBRARIES AND EXTERNAL INSTRUMENT INTERFACE SPECIFICATION

by

© Udaykumar Shah 2007

Master of Science in Computer Science

California State University, Chico

Fall 2007

There are hundreds of Cardiac Rhythm Management (CRM) device parameters in the External Instrument Interface Specification (EIIS) documents. Also there are many implementation files in CRM device test libraries to implement encoding and decoding algorithms for the CRM device parameters defined in EIIS. It has become harder for developers to remember and search these device parameters without an automated search tool. While writing test scripts, it is difficult for developers to find which clinical parameters are available and how to program those using their objects implemented in CRM test libraries.

When developers want to look at the implementation of any of these CRM device parameters, they need to search through all the EIIS documents and test library

implementation files, which is a time consuming and inefficient process. Sometimes, they are referred to incorrect files, which results in software errors. These problems impact the overall software development process and sometimes make it impossible to finish the development work on time.

In this manuscript, the author presents the need for developing an automated search engine and parser that helps developers quickly find any CRM device parameters and their objects implementation in test libraries.

CHAPTER I

SOFTWARE VERIFICATION AND VALIDATION

Software validation is the process of ensuring that requirements and designs solve the customer's problem [4]. The Food and Drug Administration (FDA) considers software validation to be “confirmed by examination and provision of evidence that software system specifications conform to user needs and intended uses, and that the particular requirements [13] implemented through software can be consistently fulfilled” [6]. It is required by the Food and Drug Administration to have 100% of code coverage during the CRM device testing.

Software Testing Difficulty

Verification and validation of software is the most complex task in the software development cycle. Software takes longer to test than it does to develop the code. The issues found during the software system testing phase add more time onto the coding/development phase, resulting in delays in the product's release, and so this vicious cycle goes [11].

The main purpose of software verification and validation is to discover software bugs and failures. As bugs are found, they can be fixed. However, software verification and validation can not guarantee that the software product is problem free.

Software verification finds many issues and bugs, but even the most extensive verification and validation processes cannot fix bugs that are not uncovered [11].

Sometimes it is impossible to map the problems that arise during the software testing cycle to any single point. Often, the problem is a design error, but this is not considered a software testing error. Design errors need to be fixed at the system requirements [13] or design level. It is also a big challenge for project managers to decide on when to stop software testing, and is often referred to as the optimal software release problem [3].

Software verification and validation is such a lengthy and frustrating process because there are many factors involved. One such factor is the complexity of the software being developed. The more complex the development is, the more there is to understand and develop the test plans. Also, complex software projects like Pacemaker and ICD development usually involve multiple development teams across multiple sites, including those working for the company directly and those working as contractors [11].

Software verification and validation often gets hit by human factors like poorly documented code and employee turnover. If the code is not documented properly by the developers and if there is a resource switch, the problem quickly compounds [11].

Another software verification and validation difficulty arises when the person developing the code and testing the code is the same person. It has been highly recommended to have a separate testing team to catch the uncovered errors and issues [11].

Regardless of the difficulties and challenges involved with software testing, one fact remains. Software failures and bugs discovered early on cost far less to fix than

ones that occur in later stages of the software development cycle; worse, after the software product has been released to the end customers [11].

Importance of Automation in Software Testing

Software professionals face the biggest challenge meeting shrinking deadlines with minimal resources. In order to overcome these challenges, software organizations want to verify and validate software products adequately, but as thoroughly and quickly as possible. To respond to the given challenges, software companies are going towards automated testing [1].

Since manual testing is labor intensive and error prone, the quality of manual software testing could not achieve the level of software testing supported by automated testing. The introduction of automated test tools can easily replace mundane manual verification and validation activities with a more sophisticated, efficient and repeatable automated environment. Automated software testing has been shown to help improve test engineer morale and retention [1].

Clearly, the software industry is widely accepting the automated software testing paradigm to improve software quality, to respond to ever shrinking schedules, and to improve productivity and to facilitate repeatability.

CHAPTER II

BACKGROUND OF THE PROJECT

The Firmware Verification team in the cardiac device companies ensures that the features of the pacemaker and defibrillator function properly under different conditions. The cardiac device verification and validation team performs white box testing to test the device firmware implementation.

The unavailability to have a single click search for cardiac device parameters, their attributes and verification test library functions implementing these parameters cause errors and prolongs development of test cases for testing Implantable Cardiac Device firmware implementations. Manual searching of these data was lengthy, frustrating and prone to errors.

Due to the large number of device parameters and implementation functions (see Figure 1) in test libraries, it became harder for developers to remember these programmable parameters without an automated search tool which can bring information from different interfaces. While writing test scripts, it becomes difficult for developers to find which clinical parameters are available and how to program those using their objects implemented in device test libraries.

When developers wanted to look at implementations of any of these device parameters, they needed to search through all these interface documents and files, which was a time consuming process. Sometimes, they referred to the wrong files, which

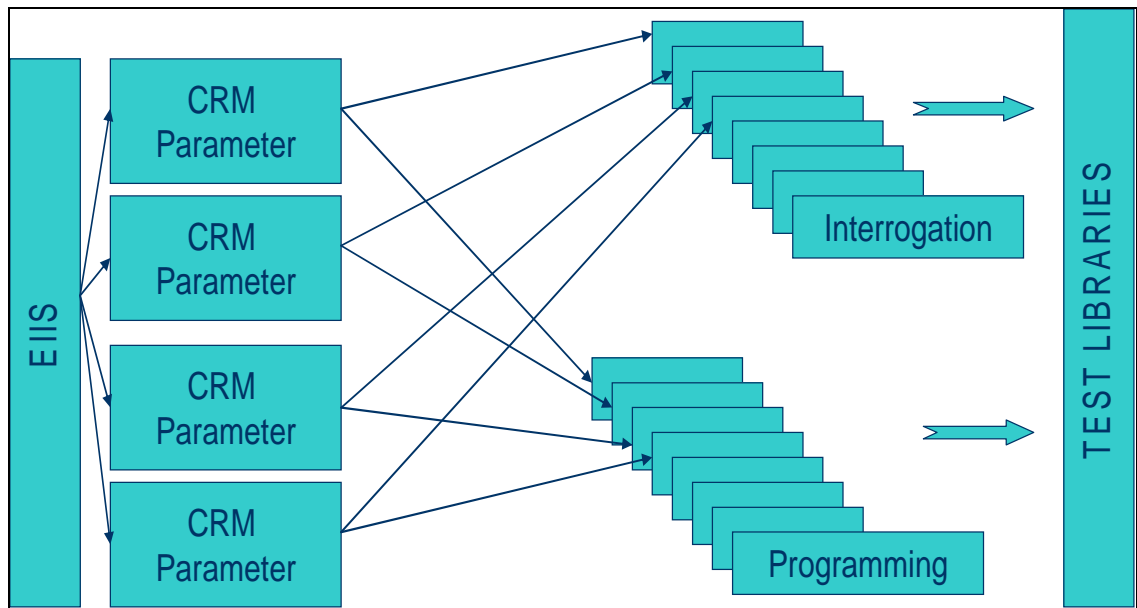


Fig. 1. CRM parameters and test libraries.

resulted in software errors and failures. These errors impacted the overall software development process and sometimes made it impossible to finish the development work on time.

While writing the test cases, engineers had to get the information from different sources like EIS and verification libraries. It was almost impossible to have a compound summary with all these data on a single report.

To overcome all the problems, the author took the opportunity to develop an automated search engine and parser that helps developers quickly find any CRM device parameters and their objects implementation.

Relative Research

The IEEE standard glossary defines “testability as the degree to which a system or component facilitates the establishment of test criteria and performance of tests

to determine whether those criteria have been met” [10]. ISO defines it in a similar way: “attributes of software that bear on the effort needed to validate the software product” [18].

It is harder to implement reliability and software quality improvements based on advances in the software testing area. Multidisciplinary approaches are needed to test the software systems to achieve higher quality and reliability. Along with quality and reliability, there has been pressure to consume fewer resources in terms of time and cost [20]. Author studied different testing methods, their advantages and disadvantages for the methods below [20]:

- One-factor-at-a-time testing
- Exhaustive testing
- Random Testing method
- Orthogonal arrays
- Design of Experiments (DOE) designs

Software testability is a result of six high-level factors: (1) the software process, (2) the test support environment, (3) the test suite, (4) built-in test capabilities, (5) characteristics of the implementation, and (6) characteristics of the representation, which include specification and requirements [5].

Software testing is one of the most expensive activities in the software development process. In order to plan the software testing activities and to allocate the resources, it is very important to measure and access the testability of software [5]. Measuring the software testability early during the software requirements, analysis and

design phases could help a lot as design refactoring could be used to improve the testability.

Software verification and validation is all about raising the confidence that the software is error-free. The cost involved might go up to 50% of the overall software development costs [3]. During V & V phase, testing resources should be systemically allocated to maximize the quality and reliability of the product, and to minimize potential operational failures [3]. The time involved in optimal testing of the software and releasing is determined based on two main aspects: reliability requirements defined in the product's system requirements and the total cost in terms of the software life cycle [3]. Also testing all possible input combinations to software could be tedious, time consuming and intractable. Author studied the papers to search for significant input and output combinations to a software control system [19].

The day-to-day software testing activities are quite manual, monotonous, laborious and something which really need the automation [7]. Author's vision here to automate the interfaces such that information from different sources could be automatically parsed and retrieved to develop the test cases.

Based on the project background mentioned earlier, the author strongly recommended improving the existing cardiac device firmware verification and validation process by the following factors:

- Provide verification engineers a tool to increase efficiency and effectiveness of their testing process.
- Save 30-60 minutes of time per verification engineer/day while developing the test cases involving such EHS parameters.

- Reduce complexity
- Reduce errors and failures
- Help continue focusing on the core verification components
- One click search functionality for the EIIS interface
- Automated tool to define common interface between EIIS and cardiac device

test libraries

In order to achieve the requirements [13] mentioned above, different studies and surveys were performed to find the main source of activities responsible for taking the longest time during the verification test development cycle, the activities during which the maximum errors were found and the areas of maximum complexity.

In order to identify the areas of major complexity, areas causing more failures and areas responsible for lengthy activities, the author went through the technical discussion with the subject matter experts (SME) from verification and validation team. Author also went through error logs from the past three verification projects and reviewed wide ranges of papers related to the External Instrument Interface Specification and cardiac devices test libraries. From these sources, External Instrument Interface Specification, CRM device test libraries stood as ideal candidates for improvement (see Figure 2).

It is required by Food and Drug Administration to have the 100% of code coverage during the CRM device testing. Considering the nature of the development processes involved in the medical industry, strict Food and Drug Administration (FDA) requirements [6] and so much customization, there are no off the shelf automation tool available to achieve the improvements mentioned above. The automation [16] had to be

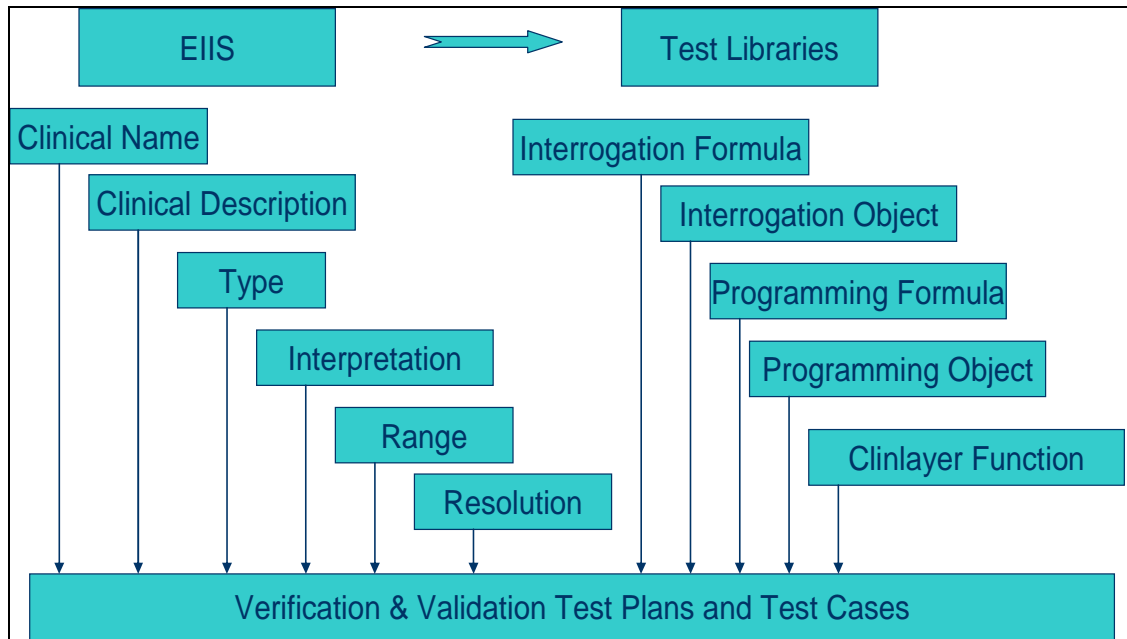


Fig. 2. Verification and validation dependencies.

developed in-house to meet certain Food and Drug Administration (FDA) requirements and to follow the departmental work instructions at the Cardiac Rhythm Management Division (CRMD) software organization.

The next phase of the investigation was to decide on the implementation technology [1]. Existing CRM device test libraries were implemented in Borland C++. The External Instrument Interface Specification documents were written in Microsoft Word.

It was decided to choose a technology that can easily interface and parse Borland C++ test library implementation files, and can easily parse External Instrument Interface Specification documents. In order to meet these requirements and for future maintainability, author recommended to implement an automated search tool using Visual Basic 6.0 [14] as a main implementation language to develop a user friendly GUI

and to implement all the necessary functionality to parse the data from EIIS and device test libraries.

With the help of this automated search engine, developers will be able to search for CRM device parameters. As a result they will get all the test library files implementing the chosen device parameters, along with the programming and interrogation algorithms.

Project Deliverables

In order to track the progress and success of the project, the following deliverables were defined:

- A common interface will be defined to search through EIIS and verification libraries with a single click search capability.
- A tool will be implemented that brings data from EIIS and Verification libraries on to a single report for reviewing the EIIS data and their implementation in verification libraries.
- While developing test scripts, this tool will help developers to search through available CRM device parameters
 - It will help find the programming objects easily without traversing through many implementation files.
 - It will greatly help in troubleshooting the programs when they fail due to different errors like clinlayer errors, unmatched conditions and incorrect use of terms.
 - It will make the development process easier, faster and mostly automated with respect to the existing clinlayer function based on the latest EIIS release.

- The developers will not need to depend on the test library team regarding the Clinlayer functions. For example,

1. What are those Clinlayer functions?
2. Where they are located,
3. What are the programming formulas and
4. What is on the interrogation side?

With help of this automated tool, developers will be able to search the desired programming parameters. They will have access to all the test library implementation files and all the programming and interrogation formulas.

Plan of Action

The project will be divided into three phases due to its magnitude. The first phase consists of reviewing papers on the subject of EIIS and verification test library interfaces. To this end, wide ranges of papers were gathered related to the cardiac devices testing libraries and External Instrument Interface Specification (EIIS).

Phase two is the design of the graphical user interface (GUI) of the search engine and implementation of the search tool. The search engine will be developed using Visual Basic 6.0 [14] as a main implementation language to develop a user friendly GUI [2] and to implement all necessary functionalities. The GUI will communicate with other Borland C++ Clinlayer files. With the help of this automated tool, developers will search for the needed programming parameters, and have access to the Clinlayer test library files, along with the programming and interrogation formulas.

The final phase involves documenting the process to train the engineers from the software organization on this automated search engine tool. Last but not the least, the tool will be installed on each of the test benches in the verification test labs to make it available to all software engineers involved in developing the test scripts to test the cardiac devices.

CHAPTER III
IMPLANTABLE CARDIAC DEVICE
FIRMWARE TESTING

Midlevel

Midlevel is a general term for three function libraries: Test runner, Midlevel and Clinlayer. They provide the testing conditions in an Engineering Mode Testing (EMT) simulation.

Test Runner

The test runner function library provides the testing mechanisms, document testing conditions and results; maintains the links with code, breadboard, BAR, Digital input (DI), Digital Output (DO), Heart Simulator (HS) and other parts of the EMT.

Midlevel

The midlevel function library consists of about 170 midlevel functions. The purpose of these functions is to ease the scripting process and provide testing conditions.

The Clinical Layer

The Clinical Layer is a software version of the EIIS. There are two halves to the Clinical Layer Code:

1. Clinical Parameters
2. RAM Variables

The Clinical Layer code is directly based on two separate sections of the EIIS document: clinparm.doc and rammapp.doc. Revisions to these two sections of the EIIS need to be reflected in the ClinLayer code.

The ClinLayer function library is used to simulate programmer to the EMT environment. Given an external instrument, the device code has to work within a given range, and under certain programmed settings. Firmware testing is a requirement based testing. If code can work beyond the range, it needs to be tested beyond the range.

Structure of the Clinlayer Libraries

The Clinlayer parameter is represented in two classes, one class will be on the interrogation side and the other on the programming side. The interrogation class structure will allow us to take the value from that address in the device and apply a resolution or an algorithm to the value. The programming class structure will allow us to give a value to that address in the device.

Since the programming class has to point to the instantiation of the interrogation class, most of the Clinlayer parameters have two classes. Both Interrogate and Program are virtual functions in the base class called Device in the class structure.

If this parameter can be both interrogated and programmed, it is represented in two classes in scout.h and scout.cpp

All parameter classes related to interrogation is located in the file with a heading of "SC."

All parameter classes related to programming is located in the file with a heading of "RV."

Cardiac Device Parameter Categories

Most of the parameters are divided into three big categories:

Brady Parameters

Brady parameters are the cardiac parameters which are used in treating the Brady Arrhythmias which is being caused due to the heart rate going under the normal heart rate.

All Brady parameters usually have BP0_ heading in front of the parameter name. The class declaration is located in files either called `scbrady` or `rvbrady`.

Tachy Parameters

Tachy parameters are the cardiac parameters which are used in treating the Tachy Arrhythmias which is being caused due to the heart rate above the normal heart rate.

All Tachy parameters usually have TP0_ heading in front of the parameter name. The class declaration is located in files either called `sctachy` or `rvtachy`.

General Parameters

All parameters not part of Brady or Tachy become part of the general parameter set. They are used in treating Atrial Fibrillation and other types of general Arrhythmias related to the cardiac function.

All General parameters usually have GP0_ heading in front of the parameter name. The class declaration is located in files either called `scgnr1` or `rvgparm`.

BP0 , TP0 and GP0 became the triggers to define a parser to parse the device parameters from the External Instrument interface Specification documents.

CHAPTER IV

SEARCH ENGINE DESIGN AND IMPLEMENTATION

In order to resolve the issues mentioned in Chapter II, the following components need to be designed, implemented and integrated in the final system.

Parser

It is required to parse the CRM device parameters information from External Instrument Interface Specification.

Search Tool

It is needed to perform fast search through the parsed CRM parameters information. It is also needed to search through the test library implementation files.

Graphical User Interface

User friendly graphical user interface needs to be developed to provide a common interface to access EIIS and test libraries data.

Reporting

It is required to provide a composite summary of data from different interfaces.

System Interactions

The interactions between CRM parameters and their implementation in test libraries are shown below (see Figure 3). It was very important to understand the overall interactions among the different components of the system.

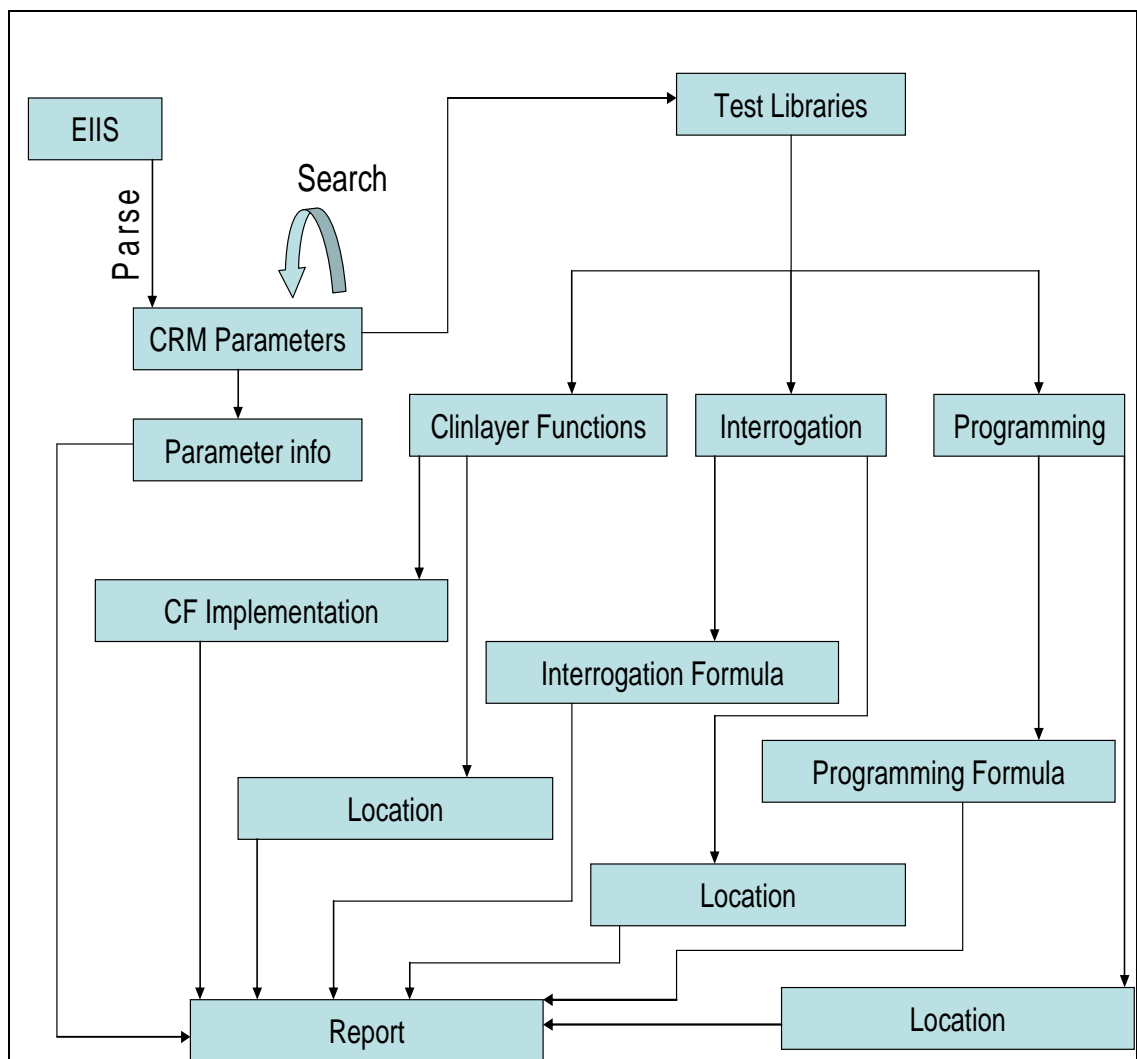


Fig. 3. System block diagram.

Technology and Data Structure Selection

Technology Selection

The following important factors were critical in deciding the technology for implementing this project:

- A technology that can easily parse and store the information from test library implementation files (Borland C++) and External Instrument Interface Specification (MS Word)
- A technology that could quickly return the search results.
- A technology to implement the user friendly graphical user interface.
- A technology using which a composite report could be produced to combine the data from different interfaces.
- A technology which is easy to implement and maintain in the future.
- Flexibility and ability of the technology to interact with different interfaces of the system.

Visual Basic 6.0 was selected as the implementation language since it has capabilities to easily support all of the factors mentioned above.

Data Structure Selection

Dynamic Arrays were selected to implement the searching and parsing functionalities. The reason behind choosing this particular data structure was:

- They allow to directly accessing each of its elements.
- Faster indexing [O(1)].
- Fast insertion at the end [O(1)].

- Faster sequential access.
- No extra storage for the references like linked list structures.
- Flexible and easier to use.
- Compactness and random access.
- Dynamic arrays are very important since the sizes need to be determined at run-time.

- Easier to implement.
- The class diagram given in Figure 4 represents the four main classes of the search tool application:

1. Clinlayer
2. Programming
3. Interrogation
4. Clinlayer function

Clinlayer

The ClinLayer class is the main interface class (see Figure 5). Interrogation and Programming instances get populated upon cardiac device parameter selection in this class. The ClinLayer class has all the functions to go through the external instrument interface documents, parse all the cardiac device parameters, and provides all the information for the chosen parameter like name, description, range, resolution, and parameter type.

The ClinLayer class provides search functionalities where users could search within the parameter name, description, range or resolution. The ClinLayer class also

Fig 4 oversized

Figure 5 oversized

provides an interface to go to the interrogation and programming formulas, as well as an interface to clinlayer functions. A user can also print all the information from different fields to a single output report for the chosen device parameter.

Important Functionalities Provided by Clinlayer Class

Some of the important functionalities provided by this class are:

- It searches the input string through the entire External Instrument Interface Specification and returns the array of parameters where the name, description or range matches the keyword string.
- It provides the search functionality within the parsed cardiac device parameters.
- It enables/disables programming and interrogation commands based on the parameter type selection and algorithms implementation availability in test library implementation.
- It provides interface to the clinlayer functions.
- It initializes all the fields on the clinlayer form to default settings.
- It restores all the parameters after removing the search filter.
- It defines parameter parsing rules.
- It parses the External Instrument Interface Specification documents.
- It prepares an array of programmable, diagnostics, status and internal cardiac device parameters.
- It parses and stores description, range, and resolution and parameter type information.

- Based on parameter selection, it updates description, range and resolution fields.
- It helps accessing the interrogation formula to obtain the implementation files where the interrogation objects are implemented.
- It helps accessing the programming formula to obtain the implementation files where the programming objects are implemented.
- Upon removal of the filter on the search criteria, it loads up `cboParam` array with all types of device parameters.
- It erases the `InterArray` and `ProgArray` to clear the interrogation and programming objects before exiting the main Clinlayer application.
- It parses implementation files and searches for user selected parameters.
- It also looks for the interrogation and/or programming object if defined for the selected parameter. Once the interrogation and/or programming object is found, it fills up the object arrays to be used by the other classes.

Programming

Programming code provide the interface from the External Instrument monitor being used by doctors; mainly, cardiologists accessing the Device firmware.

The Programming class (see Figure 6) provides the information related to the programming object, the path to the implementation of programming algorithms and loads the actual programming code to the programming form. It also provides users all the programming objects so users can pick the one they want to implement.

Figure 6 oversized

The Programming class also provides users the ability to print the information related to the parameter that was chosen and all other information related to the programming code.

Important Functionalities Provided by Programming Class

Some of the important functionalities provided by this class are:

- It provides a function to search the Programming object in the implementation files and returns the path of files where the asked object was implemented.
- It checks if any of the required implementation files have the search object in them.
- Once the programming search object is found, it updates the text in the Programming Formula with the implemented programming code.
- It prompts the users if they really want to print the report of the selected programming function.
- It provides users the ability to print all the information related to the parameter that was chosen and the information related to the programming code.
- It goes through the implementation files and returns the list of files implementing that programming object.
- It provides the information related to the programming object, path to the implementation of programming algorithms and loads the actual programming code on the Programming form.
- It provides users access to all the programming objects so users can pick the one they want to implement.

- It creates the report text file in the implementation directory. It formats the output in the report file.
- It prints the report to the selected printer and deletes the report once printing is complete (Appendix D).
- It loads up a Programming form with the default settings.

Interrogation

Interrogation code provides the interface from Device firmware to the External Instrument monitor being used by the doctors, mainly cardiologists.

The Interrogation class (see Figure 7) provides all the information related to the interrogation object, path to the implementation of interrogation algorithms and loads the actual interrogation code on the Interrogation form. It also provides users all the interrogation objects so users can pick the one they want to implement.

The Interrogation class provides users the ability to print the information related to the parameter that was chosen and all other information related to the interrogation code.

Important Functionalities Provided by Interrogation Class

Some of the important functionalities provided by this class are:

- It provides the information related to the interrogation object, path to the implementation of interrogation code and loads the actual interrogation code on the Interrogation form.
- It provides a function to search the Interrogation object in the implementation files and returns the path of files where the asked object was implemented.

Figure 7 oversized

- It checks if any of the required implementation files have the search object in them.
- Once the interrogation search object is found, it updates the text in the Interrogation Formula with the implemented interrogation algorithms.
- It provides users the ability to print all the information related to the parameter that was chosen and the information related to the interrogation algorithms.
- It creates the report text file in the implementation directory. It formats the output in the report file (Appendix C).
- It prints the report to the selected printer and deletes the report once printing is complete.
- It prompts the users if they really want to print the report of the selected interrogation function.
- It provides users access to all the interrogation objects so users can pick the one they want to implement.
- It loads up Interrogation form with the default settings.
- It goes through the implementation files and returns the list of files implementing that interrogation object.

Clinlayer Functions

The ClinLayer Functions class (see Figure 8) provides the interface to the implemented clinlayer functions in the CRM device test libraries, and provides the name and path to the implementation files in which the user selected functions reside.

Figure 8 oversized

The ClinLayer Functions class also provides the implementation detail and code of the selected clinlayer (see Figure 8). It provides the capability to print the detailed report with all the data grabbed from different implementation files (Appendix B).

Important Functionalities Provided by Clinlayer Functions Class

Some of the important functionalities provided by this class are:

- It enables the Find command (button) upon selecting clinlayer function name from the drop down list.
- It prompts the users if they really want to print the report of the selected clinlayer function.
- It provides the interface to the implemented clinlayer functions, and the name and path to the implementation files in which the user selected functions reside.
- It provides the implementation detail and code of the selected clinlayer function on to the `Clin_function` form.
- It goes through the implementation files and returns the array of files matching with the `SearchStr` function argument.
- It creates the report text file in the implementation directory. It formats the output in the report file. Then prints the report to the selected printer and deletes the report once printing is complete (Appendix B).
- It provides the capability to print the detailed report with the implementation code parsed from different implementation files.
- It allows users to check out more clinical parameters by going back to the main application form.

- It goes through the clinlayer implementation files and returns the array of files containing the implementation of user Selected Clinlayer function.
- It goes through the header files and finds all the clinlayer functions implemented in clinlayer libraries. They get filled in to `FunctionArray` array.

CHAPTER V

USER GUIDE

This chapter covers the technical user guide [17] intended to provide detailed instructions to the software engineers to perform the functionalities offered by the search engine tool. The Clinlayer Midlevel application consists of four main forms:

- Clinlayer
- Interrogation
- Program
- Clinlayer Function

Clinlayer User Interface

Clinlayer is the main application form which loads up first once user opens the application. Using this form, users can mainly perform the following tasks:

❑ Parse and access the cardiac device parameters from external instrument interface specification (see Figure 9). Open up the clinlayer executable and click on the Read EIIS button. This will go to the External Instrument Interface Specification documents which contains all types of cardiac device parameters.

It will parse through entire EIIS document and fill up different arrays with parameter names, their description, range, and resolution and type information.

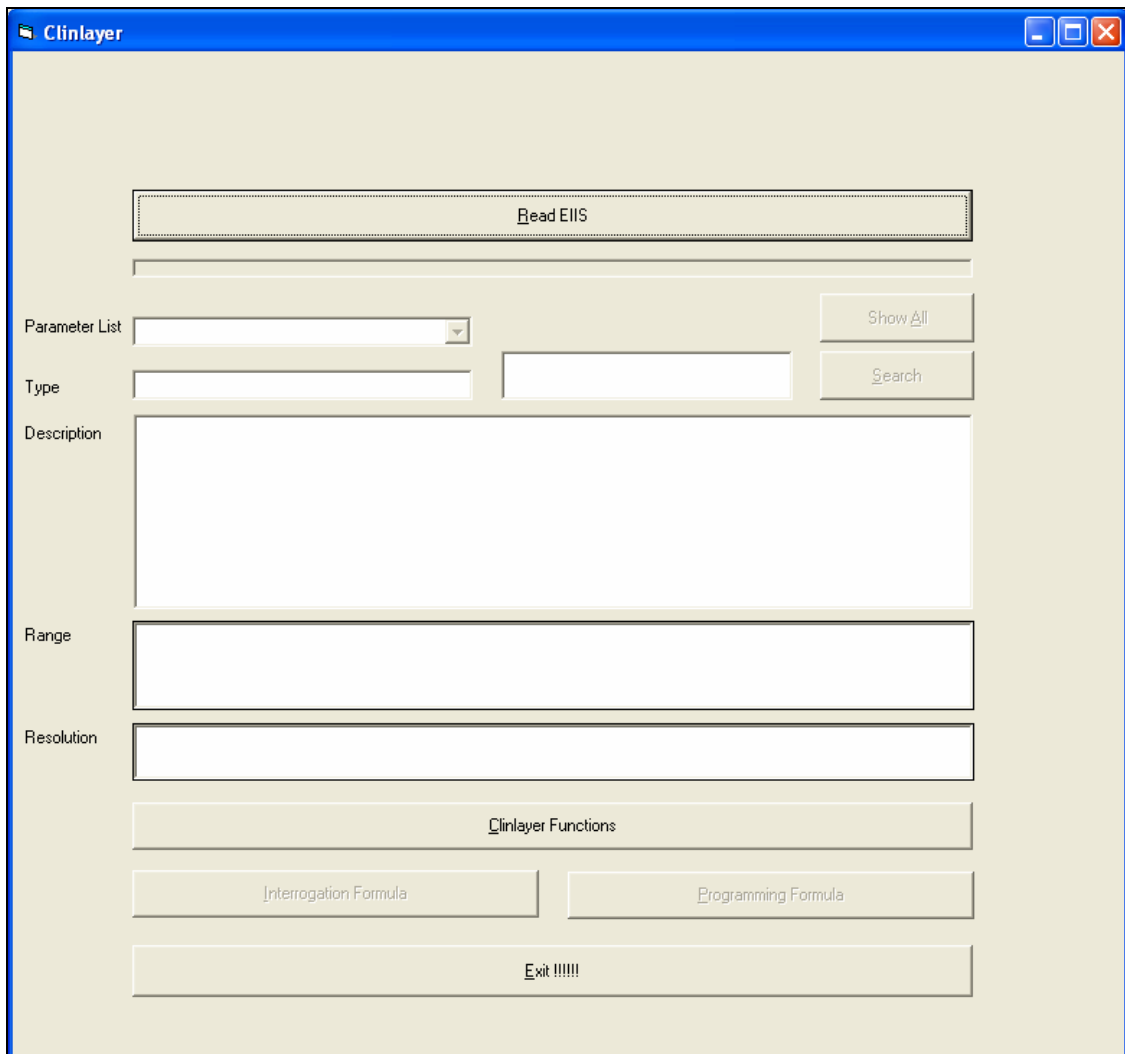


Fig. 9. Clinlayer midlevel application initial user interface.

Once parsing is done, cardiac device parameters are accessible from the Parameter List drop down list.

❑ Search the cardiac device parameter based on keyword search from name, description, range or resolution fields (see Figure 10). Once “Parameter List” is filled up with all available cardiac device parameters, they could be filtered by searching with a keyword which searches through parameter name, description, range or resolution arrays.

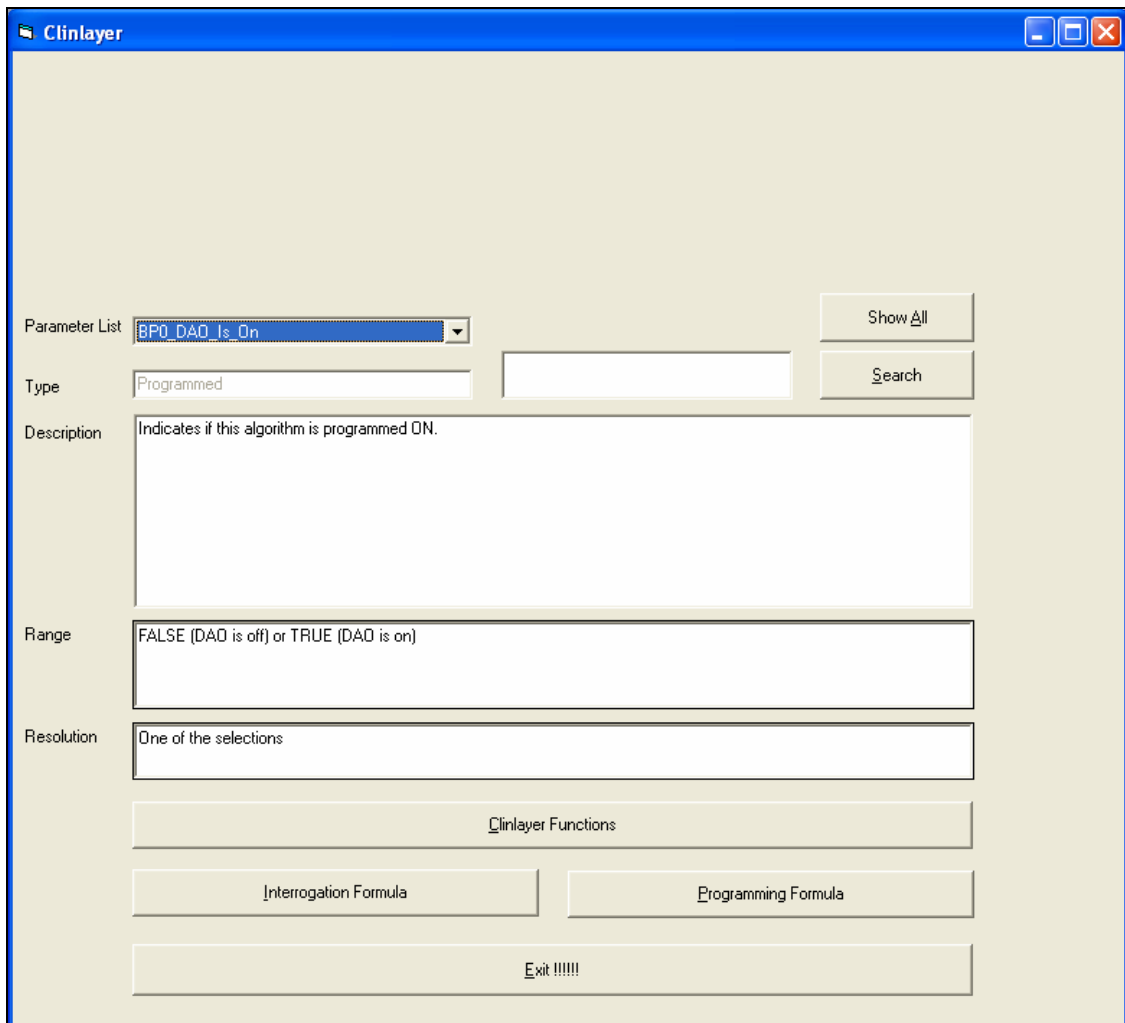


Fig. 10. Clinlayer midlevel application cardiac device parameter user interface.

Enter the search keyword in to the textbox right before the Search button. After hitting Enter or Search button, Parameter List gets updated with the list of parameters matching with the search criteria.

☐ Restore all available cardiac device parameters. Press Show All button in order to remove the filter and to restore Parameter List with all available Cardiac Device parameters.

- ❑ Get specific information about the user selected cardiac device parameter.

Select the parameter from the Parameter List one is interested in and it will update type, description, range and resolution information for the user selected cardiac device parameter.

- ❑ Interrogation formula. Based on the selected parameter, Clinlayer application will search through the library implementation files and will come back with the results if Interrogation algorithm was implemented. If the interrogation algorithm was found implemented, it will enable Interrogation Formula command button.

To go to the interrogation formula, click on the Interrogation Formula button and it will open up Interrogation form. At this time, it hides the main application form.

- ❑ Programming formula. Based on the selected parameter, Clinlayer application will search through the library implementation files and will come back with the results if Programming algorithm was implemented. If the Programming algorithm was found implemented, it will enable Programming Formula button.

To go to the programming formula, click on the Programming Formula button and it will open up Programming form. At this time, it hides the main application form.

- ❑ Access the clinlayer functions. Users can directly jump on to access the clinlayer functions without initial parsing of the External Instrument Interface Specification. Once user clicks on the Clinlayer Functions buttons, it opens up Clinlayer Function form and hides main application form.

- ❑ Exit out from the application. User can anytime exit from the Clinlayer Midlevel application by clicking on the Exit button.

Interrogation User Interface

Interrogation is the interface from Cardiac Device Firmware to the External Instrument monitor used by the cardiologists in the clinics. It translated the device implementation data to the format that could be understood by the clinicians.

Interrogation form (see Figure 11) loads up upon clicking interrogation Formula button from the Clinlayer User Interface. Using this form, users can mainly perform the following tasks:

The screenshot shows a Windows-style application window titled "Interrogation". The interface is divided into several sections:

- Select Programmable Object here:** A dropdown menu with "DAOMode" selected.
- Interrogation File:** A text box containing two file paths:


```
C:\School\Chico State\Ms Project_Clinlayer_2006\Final_Submission\Clinlayer_App_Epicll\clinlayr\rvbrady1.cpp
C:\School\Chico State\Ms Project_Clinlayer_2006\Final_Submission\Clinlayer_App_Epicll\clinlayr\scbrady.cpp
```
- Interrogation Formula:** A large text area containing the following C++ code:


```
void DAQ_Mode::Interrogate(Device* Dev)
{
    // Location: DAQ_Is_On
    if(!(*Dev)[Location] == 0x00)
        CurrentValue = "OFF";
    else
        CurrentValue = "ON";
}
```
- Buttons:** Three buttons are located at the bottom: "Back To Main Page", "Print", and "Exit".

Fig. 11. Interrogation user interface.

□ Interpret Interrogation formula. It becomes very easy for users to interpret the interrogation formula when they find the correct object and its implementation. In the

example provided in the UI screen shot below, DAO_Mode is the interrogation object where OFF maps to 0x00 and ON maps to 0x01.

- ❑ Print Interrogation report. Interrogation report could be printed by clicking on the Print button. It will bring parameter name, type, description, range, resolution, interrogation object, path to the implementation file and interrogation algorithm all on a single page report (Appendix C).

- ❑ Select Interrogate object to use in the test case for the cardiac device verification and validation. Select the Interrogation object from the drop down list on the top part of the Interrogation form. Based on the selected object, the tool will look for this object implementation in the verification library files and would return the interrogation algorithm associated with the selected object.

It also returns the path to the implementation file where the interrogation object was implemented.

- ❑ Go back to the main Clinlayer application UI. Click Back to The Main Page to go back to the main Clinlayer application UI.

- ❑ Exit from the Clinlayer and Interrogation User Interface and application. Click on Exit button anytime to exit out from the Interrogation and main Clinlayer application.

Programming User Interface

Programming is the interface from the External Instrument monitor used by the cardiologists in the clinics to Cardiac Device Firmware. It translated the format that could be understood by the clinicians to the device implementation data.

Programming form (see Figure 12) loads up upon clicking Programming Formula button from the Clinlayer User Interface. Using this form, users can mainly perform the following tasks:

The screenshot shows a window titled "Program" with the following elements:

- Select Internal Object here:** A dropdown menu showing "RV_DAO_Mode".
- Programming File:** A text box containing the path: "C:\School\Chico State\MS Project_Clinlayer_2006\Final_Submission_Clinlayer_App_Epicll\clinlay\rvbrady1.cpp".
- Programming Formula:** A large text area containing the following C++ code:


```
void DAO_ModeFlag::Program(Device* Dev2)
{
    Scout* Dev = (Scout*)Dev2;
    unsigned char Temp;

    // Location:    BP_DAO_Is_On

    if(Dev->DAOMode == "ON")
        Temp = 0x01; // temp[0] = 1
    else
        Temp = 0x00; // Temp[0] = 0

    (*Dev)[RAMLoc] = Temp;
}
```
- Buttons:** Three buttons are located at the bottom: "Back To Main Page", "Print", and "Exit".

Fig. 12. Programming user interface.

- Print Programming report. Programming report could be printed by clicking on the Print button. It will bring parameter name, type, description, range, resolution, programming object, path to the implementation file and programming algorithm all on a single page report (Appendix D).

❑ Select Programming object to use in the test case for the cardiac device verification and validation. Select the Programming object from the drop down list on the top part of the Programming form. Based on the selected object, the tool will look for this object implementation in the verification library files and would return the programming algorithm associated with the selected object.

It also returns the path to the implementation file where the programming object was implemented.

❑ Interpret Programming formula. It becomes easier for users to interpret the programming formula when they find the correct object and its implementation. In the example provided in the UI screen shot below, DAO_ModeFlag is the programming object where OFF maps to 0x00 and ON maps to 0x01.

❑ Go back to the main Clinlayer application UI. Click Back to The Main Page to go back to the main Clinlayer application UI.

❑ Exit from the Clinlayer and Programming User Interface and application. Click on the Exit button to exit from the Programming and Clinlayer application.

Clinlayer Function User Interface

Cardiac devices need to be programmed to certain configuration before they get implanted in to the patient's body. All different available configurations are defined and implemented in the clinlayer functions.

Clinlayer Function form (see Figure 13) loads up upon clicking Clinlayer Functions button from the Clinlayer User Interface. Using this form, users can mainly perform the following tasks:

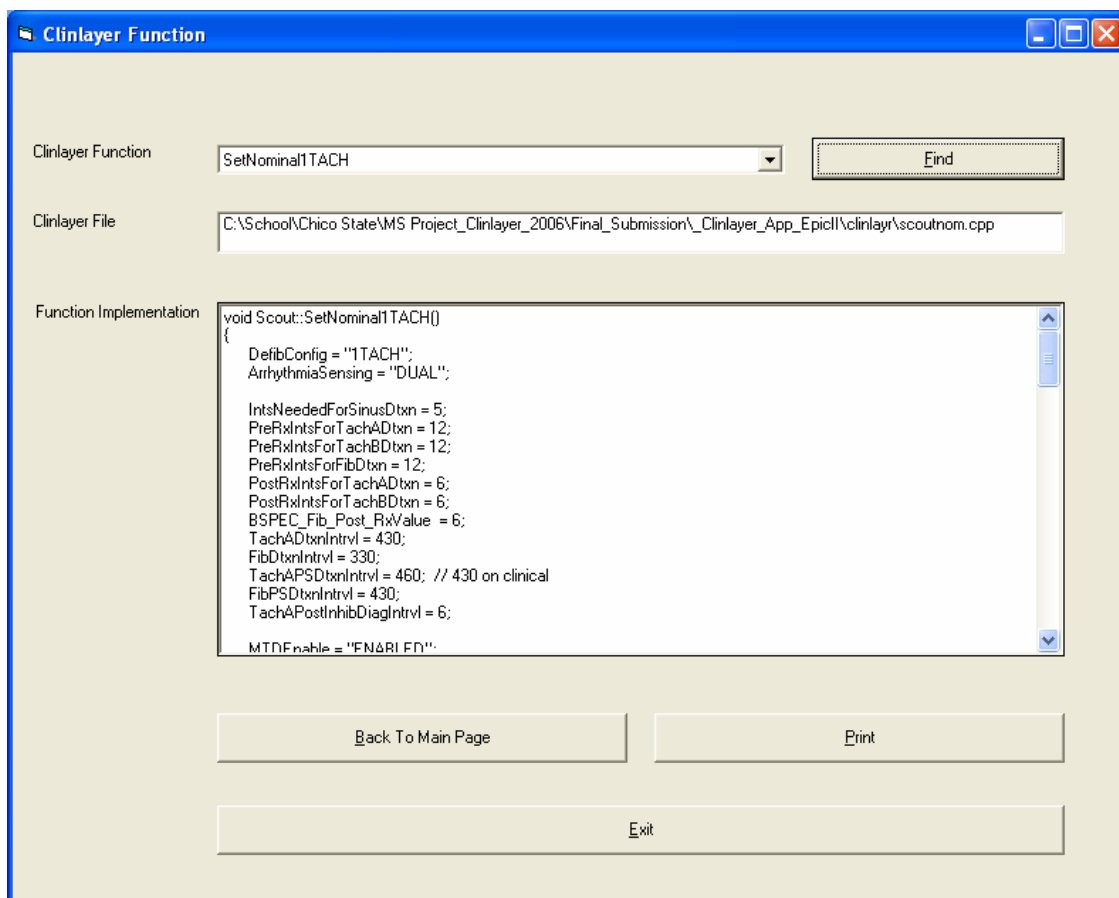


Fig. 13. Clinlayer function user interface.

❑ Select clinlayer function to set the cardiac device configuration. Select the clinlayer function from the available device configuration list. It will parse and get all the parameters and the values associated with the selected configuration in the Function Implementation text box. It also provides the path to the implementation file.

❑ Print clinlayer function report. Clinlayer function report could be printed by clicking on the Print button. It will bring clinlayer function name, path to the implementation file and device configuration parameters set up all on a single report (Appendix B).

- ❑ Go back to the main Clinlayer application UI. Click Back to The Main Page button to go back to the main Clinlayer application UI.
- ❑ Exit from the Clinlayer Function User Interface and main application. Click on the Exit button to exit from the Clinlayer Function and main Clinlayer application.

CHAPTER VI

SUMMARY AND CONCLUSION

Software verification and validation in biomedical companies is a very important process that affirms the quality of the cardiac device software. It increased the quality of the cardiac devices which helped in preventing any device recalls, less failures, reduced harm to cardiac patients, and reduced liability to cardiac device manufacturers. The implementation and availability of the search engine reduced the cost by automating certain operations and by making them error free software operations. It also helped reducing the long term software cost associated with potential recalls and maintainability.

Due to search engine tool availability, it became easier for developers to find CRM device parameters even though EIIS had enormous number of device parameters and implementation functions in test libraries. While writing test scripts, it became easier for developers to find which clinical parameters are available and how to program those using their implemented programming or interrogating objects.

Developers who wanted to look at implementation of any of these Clinlayer parameters, they can now easily search through all these documents and files, without going through manual, error-prone, lengthy and frustrating processes. It helped improve overall software development quality and now, it has made possible to execute and finish multiple projects development work on time.

Benefits from the Project Implementation

Implementation of the search tool has satisfied all requirements [13] mentioned as per project deliverables in Chapter II. Having this project implementation helped a lot to the Device Firmware verification team. The main benefits from the implementation and availability of this project were:

- Implemented a tool that brings data from EIS and Verification libraries on to a single report for reviewing the EIS data and their implementation in verification libraries (Appendices B, C, and D).
- A common interface was defined to search through EIS and Verification libraries with a single click of search capabilities.
- It greatly helped in troubleshooting the programs when they fail due to different errors like Clinlayer errors, unmatched conditions and incorrect use of terms.
- It made the development process easier, faster and error-free.
- While developing test scripts, this tool helped developers to search through available clinical parameters.
- It helped finding the programming objects easily without traversing through many implementation files.
- The developers would not have to depend on the Clinlayer team regarding the Clinlayer functions. For example,

What are those Clinlayer functions?

Where they are located?

What are the programming formulas?

What's on the interrogation side etc.?

- With help of this search engine tool, developers searched for the desired programming parameters, and they got all the Clinlayer files with necessary functions, with all the programming formulas and interrogation formulas.

Measured Impact

The major impacts from the implementation of this project were:

- It provided verification engineers a tool to increase efficiency and effectiveness of their testing process.
- It saved 60-90 minutes of time per verification engineer/day while developing the test cases involving such EHS parameters.
- The tool is used by more than 35 software engineers in cardiac rhythm management device verification organization.
- It helped reducing the implementation complexity.
- It helped reducing the errors.
- It helped continue focusing on the core verification components.

Limitations and Future Scope

This section is intended for the future improvements and opportunities. It is not part of this project implementation.

Dynamic arrays data structure would perform faster data search for limited number of CRM parameters only. If we have more than few thousands and millions of parameters, dynamic arrays would slow down the search performance. To achieve the superior level of performance, CRM parameters should be stored in XML database for

faster indexing. It would also help to output the CRM parameters information in XML report which would help end customers of EIIS.

REFERENCES

REFERENCES

- [1] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management and Performance*. New York, NY: Addison-Wesley, 1999.
- [2] J.A. Larson, *Interactive Software: Tools for Building Interactive User Interfaces*. Englewood Cliffs, NJ: Yourdon Press/Prentice-Hall, 1992.
- [3] C.T. Lin and C.Y. Huang, "Enhancing and Measuring the Predictive Capabilities of Testing-Effort Dependent Software Reliability Models," *Journal of Systems and Software*, accepted for publication.
- [4] Institute of Electrical and Electronics Engineers, *IEEE Standard 1012-1998 Software Verification and Validation*. New York, NY, IEEE, 1998.
- [5] S. Mouchawrab, L.C. Briand, and Y. Labiche, "A Measurement Framework for Object-Oriented Software Testability," *Information and Software Technology*, vol. 47, no. 15, pp. 979-997, 2005.
- [6] US Food and Drug Administration, "General Principles of Software Validation: Final Guidance for Industry and FDA Staff," 2002. Retrieved June 6, 2007 from World Wide Web: <http://www.fda.gov/cdrh/comp/guidance/938.html>.
- [7] M. Roper, "Software Testing—Searching for the Missing Link," *Information and Software Technology*, vol. 41, no. 14, pp. 991-994, 1999.
- [8] P. Weston, *Bioinformatics Software Engineering: Delivering Effective Applications*. New York, NY: John Wiley & Sons, Inc., 2004.
- [9] G. Mandanis and A. Wyatt, *Software Project Management Kit*. Foster City, CA: IDG Books Worldwide, Inc., 2000.
- [10] Institute of Electrical and Electronics Engineers. *IEEE Standard 610.12-1990 Glossary of Software Engineering Terminology*. IEEE, New York, 1990.
- [11] W.E. Perry, *Effective Methods for Software Testing: Includes Complete Guidelines and Checklists, 3rd Edition*. New York, NY: John Wiley & Sons, Inc., 2006.
- [12] R.S. Pressman, *Software Engineering: A Practitioner's Approach, 4th edition*. Berkeley, CA: Osborne/McGraw-Hill, 1997.

- [13] S. Lauesen, *Software Requirements: Styles and Techniques*. New York, NY: Addison-Wesley, 2002.
- [14] R. Stephens, *Advanced Visual Basic Techniques*. New York, NY: John Wiley & Sons, Inc., 1997.
- [15] M. Halvorson, *Microsoft Visual Basic 6.0 Professional: Step by Step*. Redmond, WA: Microsoft Press, 1998.
- [16] M. Fewster and D. Graham, *Software Test Automation: Effective Use of Test Execution Tools*. New York, NY: Addison-Wesley, 1999.
- [17] R.J. Brockmann, *Writing Better Computer User Documentation*. New York, NY: John Wiley & Sons, Inc., 1990.
- [18] International Organization for Standardization/International Electrotechnical Commission, *ISO/IEC 9126. Information Technology: Software Product Evaluation: Quality Characteristics and Guidelines for Their Use*. Geneva, Switzerland: ISO/IEC, 1991.
- [19] J. Hunt, "Testing Control Software Using a Genetic Algorithm," *Engineering Applications of Artificial Intelligence*, vol. 8, no. 6, pp. 671-680, 1995.
- [20] A.M. Salem, K. Rekab, and J.A. Whittaker, "Prediction of Software Failures Through Logistic Regression," *Information and Software Technology*, vol. 46, no. 12, pp. 781-789, 2004.

APPENDIX A

DEFINITIONS

External Instrument / Programmer

Any external device which could communicate to the implanted cardiac device.

Cardiac device (Defibrillator/Pacemaker)

Medical device implanted to the heart via Defibrillator / pacemaker lead which treats tachy arrhythmia / brady arrhythmia.

Interrogation Formula

Interrogation formula is a conversion formula to read the information from the device firmware and to convert it to the user understandable units.

Programming Formula

Programming formula is a conversion formula to read the user requested information and to convert it to the device firmware supported language.

Midlevel

Midlevel is a general term of three function libraries, which provide the testing conditions in an Engineering Mode Testing (EMT) environment.

Clinlayer

The Clinlayer is a function library to simulate programmer to the EMT environment. The Clinical Layer is a software version of the EIIS. There are two halves to the Clinical Layer Code: Clinical Parameters and Ram Variables.

Test runner

Test runner provide the testing mechanism, document testing conditions and results; maintain the links with code, breadboard, BAR, DI, DO, HS and other part of EMT.

ACRONYMS

CRM: Cardiac Rhythm Management

DI: Digital input

DO: Digital output

EIIS: External Instrument Interface Specification

EMT: Engineering mode Tester

FDA: Food and Drug Administration

GUI: Graphical user Interface

HS: Heart Simulator

HW: Hardware

ICD: Implantable Cardioverter Defibrillator

APPENDIX B

CLINLAYER FUNCTION REPORT

CLINLAYER REPORT (Clinlayer Functions)

Clinlayer Function :

SetNominal1TACH

Clinlayer Function File :

C:\School\Chico State\MS Project\clinlayr\scoutnom.cpp

Function Implementation :

```
void Scout::SetNominal1TACH()
{
    DefibConfig = "1TACH";
    ArrhythmiaSensing = "DUAL";

    IntsNeededForSinusDtxn = 5;
    PreRxIntsForTachADtxn = 12;
    PreRxIntsForTachBDtxn = 12;
    PostRxIntsForTachBDtxn = 6;
    FibDtxnIntrvl = 330;
    TachAPSDtxnIntrvl = 460; // 430 on clinical
    FibPSDtxnIntrvl = 430;
    TachAPostInhibDiagIntrvl = 6;

    MTDEnable = "ENABLED";
    MTTEnable = "ENABLED";
    MTDCutoff = 430;
    MTTCutoff = 430;
    MTDDuration = 20;
    MTTDuration = 20;

    PSBasePacingIntrvl = 1203;
    SetResetValues();
}
```

APPENDIX C

INTERROGATION REPORT

CLINLAYER REPORT (Interrogation)

Parameter Name:

BP0_DAO_Is_On

Parameter Type :

Programmed

Description :

Indicates if this algorithm is programmed ON.

Range :

FALSE (DAO is off) or TRUE (DAO is on)

Resolution :

One of the selections

Programmable Object :

DAOMode

Interrogation Class File :

C:\School\Chico State\MS Project\clinlayr\sbrady.cpp

Interrogation Class Formula :

```
void DAO_Mode::Interrogate(Device* Dev)
{
    // Location: DAO_Is_On

    if((*Dev)[Location] == 0x00)
        CurrentValue = "OFF";
    else
        CurrentValue = "ON";
}
```

APPENDIX D

PROGRAMMING REPORT

CLINLAYER REPORT (Programming)

Parameter Name :

BP0_DAO_Is_On

Parameter Type :

Programmed

Description :

Indicates if this algorithm is programmed ON.

Range :

FALSE (DAO is off) or TRUE (DAO is on)

Resolution :

One of the selections

Internal Object :

RV_DAO_Mode

Programming Class File :

C:\School\Chico State\MS Project\clinlayr\rvbrady1.cpp

Programming Class Formula :

```
void DAO_ModeFlag::Program(Device* Dev2)
{
    if(Dev->DAOMode == "ON")
        Temp = 0x01;        // temp[0] = 1
    else
        Temp = 0x00;        // Temp[0] = 0

    (*Dev)[RAMLoc] = Temp;
}
```



Fig. 4. Midlevel clinlayer class diagram.

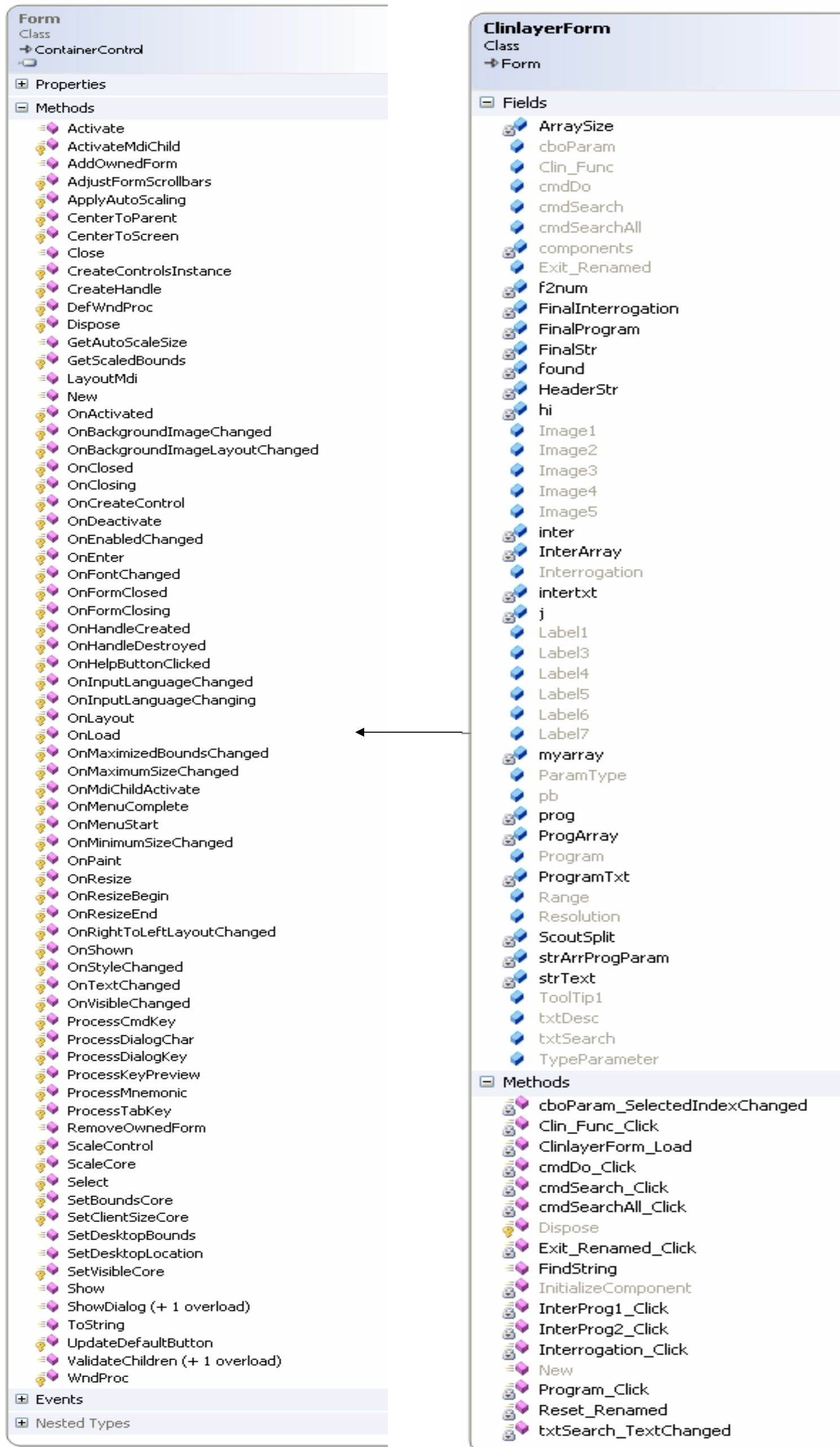


Fig. 5. Clinlayer Class.

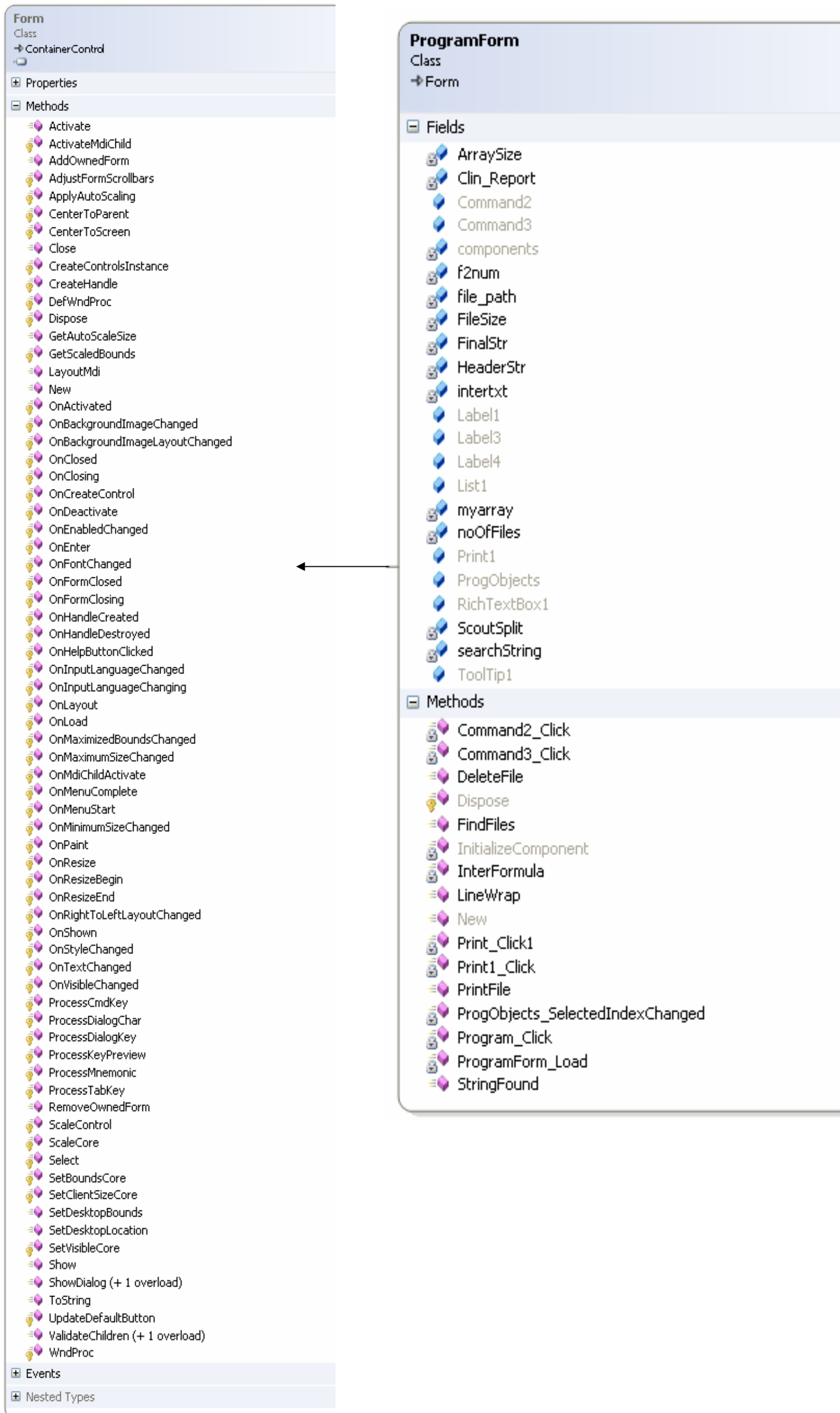


Fig. 6. Program class.

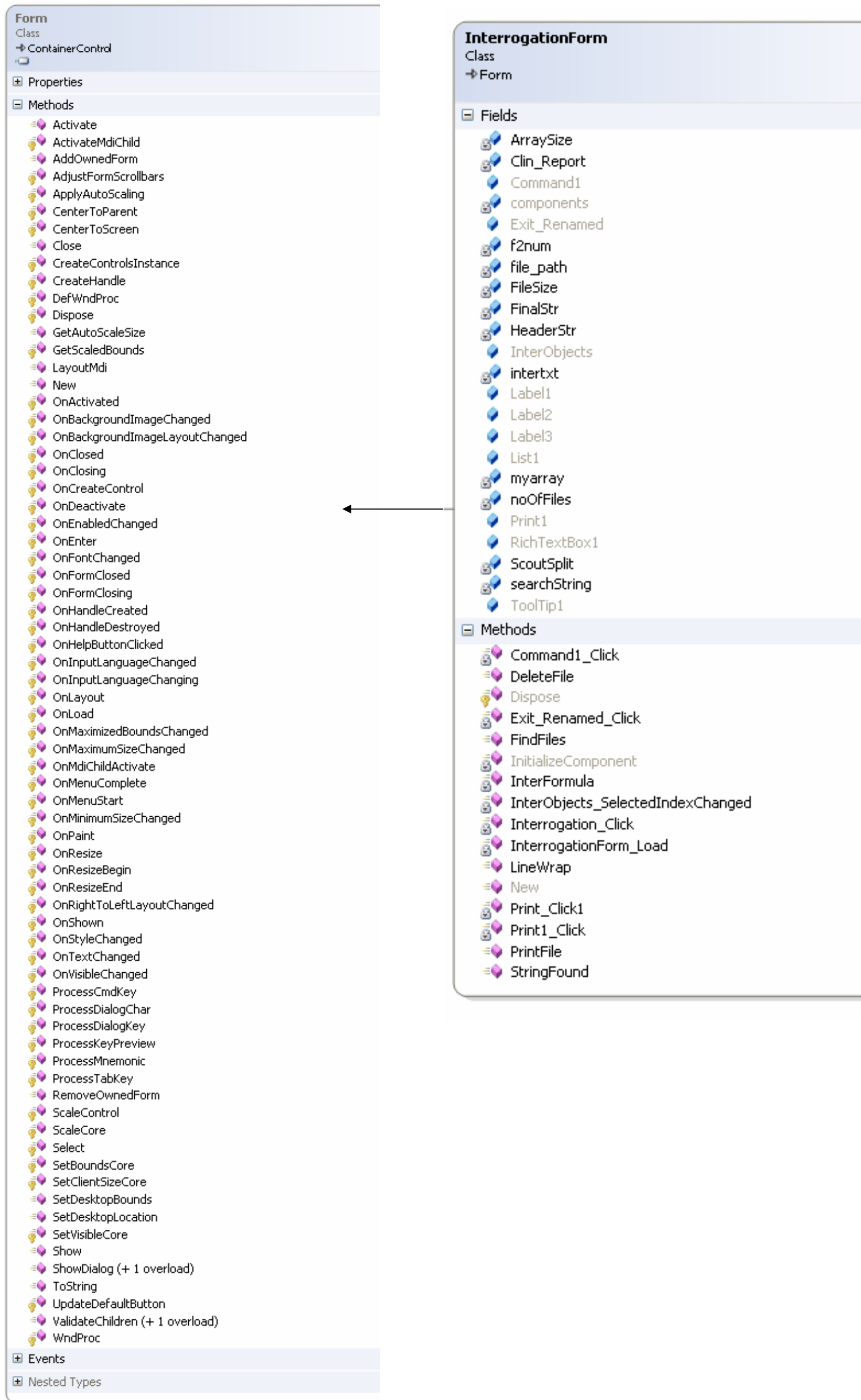


Fig. 7. Interrogation class.

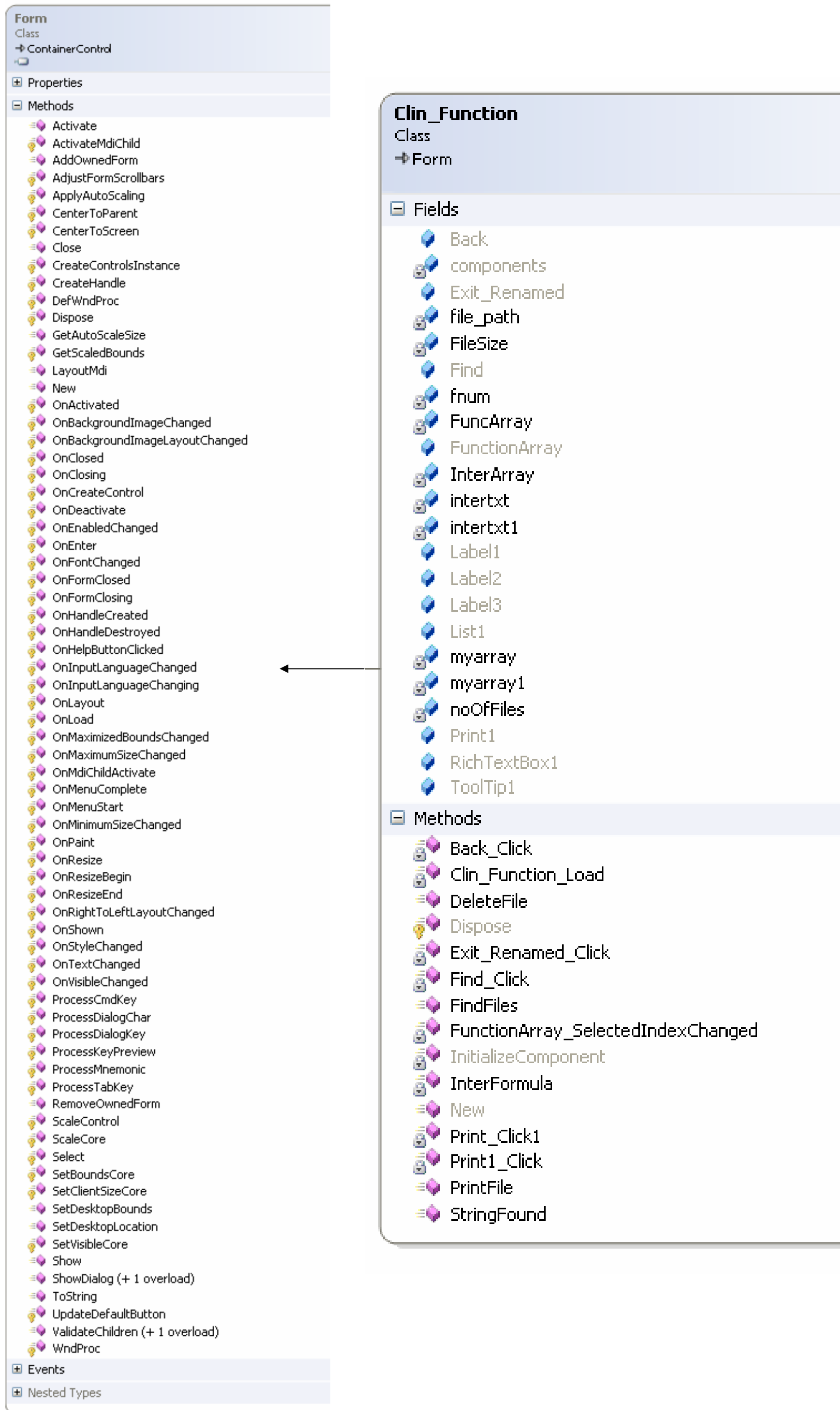


Fig. 8. Clinlayer function class.