A FORMAL MECHANISM FOR ANALYSIS AND RE-IMPLEMENTATION OF LEGACY PROGRAMS

DIMITRIOS V. MAGLARAS

A submission presented in partial fulfilment of the requirements of the University of Glamorgan/Prifysgol Morgannwg for the degree of Master of Philosophy

September 2001

A formal mechanism for analysis and re-implementation of legacy programs

Abstract

The last ten years of this century have been characterized as a period of software crisis. The need for information is growing day by day and the formal development tools have been proven insufficient to serve this need. This led software engineers to the creation of new technologies that would be more efficient in the manipulation of the data and the development of software systems. However, what will happen with all those large software applications that have been developed in the past, under formal development tools such as 3rd GLs? In the real world there are too many software applications, developed using 3rd GLs, still working in business and there are many reasons why these applications need to be modified in order to keep on running effectively. The introduction of EURO as global currency in Europe is a well-known problem concerning these old applications. In most cases the documentation describing the requirements, design and implementation of the legacy software systems does not exist or is too poor to make sense. This thesis will provide a mechanism to regain design and implementation information of a software system examining its source code. The mechanism is based on a software tool that will be able to extract useful information from the source code of an old application. The modularization of the information, concerning the design and implementation analysis of the software system, into smaller pieces of information, describes the scheme that will be used in order to retrieve, manipulate and finally provide this information to the users. This scheme treats the pieces of information, which are gathered from the source code, as separate objects related to each other. These objects together with their relations will be stored into a semantic network (database). The contents of this database will be browsed in such a way that will provide critical and meaningful information about the implementation and design of the software system. A software module, called parser, will be developed, which will be able to extract pieces of information by parsing all the source files of the old application line by line. This information is stored into a semantic network and a separate tool will be configured in order to retrieve information from the semantic network and provide it on the screen using a GUI. In the first chapter an introduction to this research project takes place. In the second chapter the documentation gathered concerning the research area of software analysis and reuse is studied and analyzed. In the third chapter all the requirements and specifications of the proposed mechanism are set. Chapters four and five present the design and implementation of the semantic network that will contain the pieces of information gathered from the source code and the source code parser. In the sixth chapter the developed mechanism is tested against its specifications. Finally, in the seventh chapter the analysis of a large industry, data-processing software application takes place.

Table of contents

A formal mechanism for analysis and re-implementation of legacy programs

Chapter 1

Introduction

1.1 Introduction

The following project has been carried out from Mr. Dimitrios Maglaras. He is a mathematician with a software engineering specialization. His main interests are the structured architecture and design of large software systems and the reengineering of existing software applications. In the past he has been dealing with software components that were using techniques to calculate the solution vector of large linear equation systems with special structures. Nowadays he deals most with 'data-processing applications', the way they are developed, structured and implemented.

The last ten years of this century have been characterized as a period of software crisis. The need for information is growing day by day and the formal development tools have been proven insufficient to serve this need. This led software engineers to the creation of new technologies that would be more efficient in the manipulation of the data and the development of software systems. Large software systems have been developed using these new technologies. These systems have been proven efficient and satisfying. However, what will happen with all those large software applications that have been developed in the past, under formal development tools such as 3rd generation languages (GLs)? In the real world there are too many software applications, developed using 3rd GLs, still working in business. There are many reasons why these applications need to be modified in order to keep on running effectively. The introduction of EURO as global currency in Europe is a well-known problem concerning these old applications. In most cases the documentation describing the requirements, design and implementation of the legacy software systems does not exist or is too poor to make sense. This thesis will provide a mechanism to regain design and implementation information of a software system examining its source code.

There are certain problems that occur in the reengineering process of large data-processing software application systems, which have been developed using legacy software development tools, such as the complexity of the code combined with the absence of computer-aided software engineering (CASE) tools. This project will study the requirements, design and develop a software system that helps, by means of providing critical information, to speed up this process. A real data-processing software system, which has been developed using the RM/COBOL-85 computer language, will be analyzed using this mechanism and the results of this process will be presented.

1.2 Project description

Many large software systems have been developed using COBOL, PASCAL, PL1, BASIC and other software development tools that belong to a very large category named 3rd Generation Languages. In the following when a reference to a software system or application is made, characterizing it as *old* or *legacy*, it means that the referenced software or application has been developed using 3rd GLs. This thesis focuses in the reengineering process of *old* software applications.

It has been more than a decade now that the next generation of software development languages (4th GLs) appeared for the first time. Nowadays, 4th GLs dominate in the software development community. They offer many facilities that make software development easier especially on applications that manipulate large amounts of data. Nowadays, modern software developers use these tools almost exclusively in the development of their software applications.

The mechanism for analysis and re-implementation of legacy programs, which is presented in this thesis, is based on a software tool that will be able to extract useful information from the source code of an *old* application. This information can be used for various purposes, such as: version information, problem determination and problem solution. Furthermore, this information can be utilized for the whole reconstruction of an already implemented and working application. The main idea is to develop a software system in order to extract information concerning the structure, implementation and design of an existing software application instead of having software engineers to read all the source code line by line, in order to extract the desired information.

Software engineers are interested in specialized *pieces of information*, which are spread throughout the source code of the software application, in order to understand the design and implementation analysis of the application. Until now engineers had to read the source code line by line, in order to identify these pieces of information. An automated system should be able to locate these *pieces of information* inside the source code, store them in a database and represent them in a user-friendly manner through a graphical user interface (GUI). The purpose of the proposed thesis is the development of such an automated system.

The meaning of *"pieces of information"* is abstract and subjective. In software engineering, one could find some *"pieces of information"*, about an old application, important and at the same time, someone else could characterize them meaningless. For example, someone would think that access to temporary files is a phenomenon that should be remarked, while someone else couldn't care less about the temporary files that a software application creates and accesses. However, the similarities found across software systems are enormous and undeniable. These similarities indicate common techniques in the design and implementation of software systems. The pieces of information, which will be extracted from the source code, may be used in order to regain information about such techniques, which where applied during the development of the software application. This thesis will concentrate in the extraction of such pieces of information.

The modularization of the information, concerning the design and implementation analysis of the software system, into smaller pieces of information, describes the scheme that will be used in order to retrieve, manipulate and finally provide this information to the users. This scheme treats the pieces of information, which are gathered from the source code, as separate objects related to each other. These objects together with their relations will be stored into a semantic network (database). The contents of this database will be browsed in such a way that will provide critical and meaningful information about the implementation and design of the software system.

A software module, called parser, will be developed, which will be able to extract all this information from the source code of the old application, by parsing all the source files of the old application line by line, gathering all the pieces of information included in the source code. The parser should recognize and understand a number of statements and their arguments specific to the 3rd GL, which was used in the development of the old software application. Not all of the statements of the programming language should be *understood* from the parser. Only the ones regarded to include useful pieces of information will be recognized.

After the parser has finished scanning the source code of the application, it will store the information derived in a text file in the form of transactions. A special tool will be used in order to read the text file, execute the transactions and store the information to the database.

Finally, a separate tool will be configured in order to retrieve information from this database and provide it on the screen. The retrieval of the information will be implemented in the form of query transactions to the database. The results of these transactions will be formed and presented on the screed using a GUI. This tool should be in a position to provide results for a number of predefined queries. For example, if the names of all the datafiles (tables) that are used from a software system were stored in the database a simple query would be '*give me the names of all the files that this software system uses*'. The engine should search in this database, whenever this query is asked and come out with the results.

Figure 1.1 shows the layout of the formal mechanism. In summary, the steps, which will be followed are:

1 The parser will extract the necessary information from the sources of the old application and store it in a text file in a form of transactions.

2 A special tool will read the text file and execute the transactions

3 Another tool will be configured in order to retrieve the information from the database by executing queries to it. The information will be presented on the screen using a GUI.

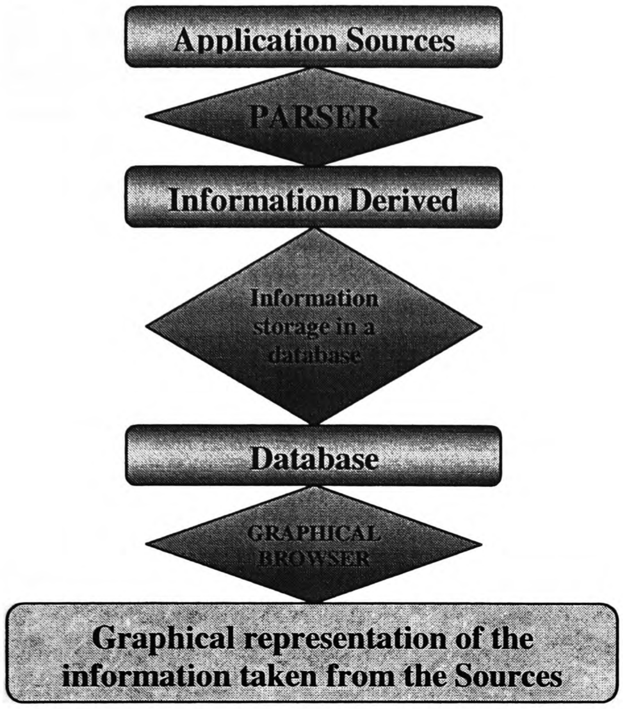

Application Sources
PARSER
Information Derived
Information storage in a database
Database
GRAPHICAL BROWSER
Graphical representation of the information taken from the Sources

Figure 1.1 – Layout of the mechanism for analysis and re-implementation of legacy programs.

1.3 Motivation

The results of this research will help the software engineers, to overcome many difficult problems, which are now facing. Billions of lines of 3rd GL source code have been written for the development of millions of legacy software applications. These legacy applications now are very difficult to maintain or further develop. The development tools of 3rd GLs used for the implementation of software applications (mainly those that manipulate big amounts of data) are primitive compared to the 4th GL development tools. Furthermore, the modern software development tools use an open architecture to store and manipulate data. A separate software engine is responsible for the final data manipulation and storage. The application itself just makes the necessary request to this engine (transaction), in order to retrieve or store data, using a separate software module (driver), to communicate with this engine. This makes access to the same database engine very easy for separate applications that need to retrieve data stored in the same database. Just using a similar *driver* it can achieve communication with the same database engine and access its data. Applications developed using 3rd GL usually communicate with each other using ASCII text files that have special structure. This is a rather painful and time spending procedure because there are no standard rules for the structure of these text files and a special piece of software (usually called *bridge*) should be developed, from both sides, for that purpose. Thus, a lot of time and manpower is wasted in the maintenance or further development of legacy applications.

Due to the robust and effective tools that the 4th GLs provide in the development of software applications and to the simple solutions that have been given to problems that 3rd GL face, there are yet not many software developers left to deal with all those applications that have been developed using 3rd GLs. In addition, 3rd GLs are not taught any more in modern schools and universities. As a result, all new software developers work using modern tools and techniques. Nevertheless, there are many applications that have been developed in the past using old development tools and now only need a minor alterations or improvements. In this case all the necessary information that can be extracted from the old application, is required in a short period of time. The proposed formal software reengineering mechanism will help locating the problem, in order to commit the necessary changes, having as a result an improved, working application.

Other software developers could use this tool in order to keep version information for their applications. This tool will help them to keep truck of the characteristics of every version of their application, by scanning the sources of their application in a regular basis and keeping records of the databases, which are produced each time.

1.4 Theses organization

Chapter 2 presents the documentation that was found to be representative in the research area of these theses. In this chapter the supporting theory, published papers and other relative commercial implementations are discussed. In **Chapter 3** the requirements that this software reengineering mechanism must fulfill are studied and according to those requirements a set of specifications is created. **Chapter 4** presents the design and implementation analysis of the model, which is developed aiming to represent the analysis of each software application. This model is implemented in the form of a specific setup of the SIS software engine. **Chapter 5** contains the design and implementation analysis of the source code parser. The developed mechanism is applied in a real software system and is tested against its specifications in **Chapter 6**. Finally, **Chapter 7** presents the detailed analysis of the software system, which was analyzed while testing the software reengineering mechanism in **Chapter 6**.

A formal mechanism for analysis and re-implementation of legacy programs

Chapter 2

Documentation Study and Analysis

2.1 Introduction

In this chapter the analysis and study of existing documentation in the research field of reengineering and reusing legacy software systems will be investigated. This specific domain of software engineering, *-software reengineering and reuse-* provides an attractive research area for the computer scientists. It covers a very wide range of software applications that are still running in business and they need appropriate modifications to fulfill the new requirements. These applications without proper reengineering tools and techniques would be very hard to modify and the corporations that are using them would face serious problems.

The supporting theory of this research area is presented in the first section of the document. In the second section some papers that are found to be representative for this specific research domain are discussed. Finally, in the third section some existing implementations of tools, which were developed to provide software reengineering and reuse solutions, will be presented.

2.2 The supporting theory

The domain of this project includes general software engineering concepts and especially reengineering concepts. The software engineering community has achieved great progress in software reengineering and reusability. For this reason several computer theories have been developed to provide answers to serious questions in this problem domain. The most important of those theories are presented and discussed in this section.

2.2.1 The theory of software objects

The theory of software objects plays a pivotal role in software reengineering. This theory has brought a revolution in the software engineering community. It does not provide just another tool or another technique to develop software, but the innovation that this theory has brought, lies at a higher level of abstraction. It lies in the software development mentality that every developer will have to follow or stay out of business. Almost every action inside this project is based on the object-oriented philosophy that dominates nowadays. This theory has also inspired the scientists to create several techniques in order to be able to reuse software components and to control big systems in a much more efficient and robust manner.

In the context of this thesis a tool will be developed that will scan the source code of an *old* application. Then it will store the information gathered about the applications design and implementation in a Semantic Network (database), which consists of software objects and relations between them. This way useful information will be provided to the engineers about the software system under analysis in order to modify it.

The Semantic Network used in this project handles data in a form of objects related to each other. Every last piece of information is treated as a separate object and any special characteristic of an object is represented as a relation to another object. Using this principle all the information extracted from the source code of the *legacy application* will be stored into a specially configured TELOS database instance, which forms the implementation of the Semantic Network that will be used in this project. A special tool will be provided in order to present the information, which has been derived from the sources of the *old* application and stored into the database, to the end users.

2.2.2 The software component classification theory

The software component classification is another theory in computer science that is used in the context of this project. The information derived from the source code of the application under analysis, represented in an object form, should be distributed into a class hierarchy schema, which will actually consist an easily manipulated Semantic Network. Inside the database, which forms the implementation of the Semantic Network, objects sharing common properties will be grouped into classes. Classes themselves are treated as generic objects, which their members are instances of. These classes, in turn, can be instances of other more generic objects. In fact every object has to be declared as an instance of at least one class. As a result, an infinite classification hierarchy is created starting with objects that have no instances of their own and they are usually called tokens [12].

The software component classification theory proposes and studies the grouping of elements containing related pieces of information into sets. Using this theory the manipulation of these elements becomes more efficient and the management of the whole system that is created much easier. It has been proved that the management of large software class collections requires addressing problems related to the following categories:

- The representation of classes so as to capture structural and descriptive information and to allow multiple views.
- The representation of relationships and dependencies among classes in a collection.
- The selection and understanding of classes by appropriate querying and browsing facilities.
- The support of class evolution. [9]

2.2.3 The representation of software components

The representation theory is a research domain that has to be studied in this thesis also. The system that will be developed uses a special tool in order to represent the information gathered in a human understandable way. The so-called representation problem in reverse engineering [22] has been taken into account in order to pass information to the users with a human friendly user interface. A browser allows viewing parts of the information that has been derived from the source code and has been stored in the database. The information is presented at various levels of detail and navigation through the presented components in a hypertext-like fashion is available.

2.2.4 The theory of patterns

The theory of patterns is an interesting research domain that software engineers use in order to transfer expertise among them easily. This theory uses standard forms to express problems and solutions that have been successfully applied to them. When the time comes for software engineers to deal with a large application, in order to make some changes or try to extract specific information, this theory helps them mostly in the first steps [24]. Software reengineering faces such problems all the time so the theory of patterns is widely used. It takes a long time for engineers to become experts in a specific domain and this theory helps them in this effort.

Nevertheless, this project will try to regain formal patterns that have been used by the developers of the old application concerning its design and implementation. It will give one more solution to the problem of reengineering large software applications, which have been developed using 3rd generation software development tools. It should be capable to transfer expertise that has been invested in the development of the *legacy* application to the new engineers that have undertaken the responsibility to commit reengineering actions upon it.

2.2.5 The theory of software repositories

Another approach to software maintenance and reengineering is through software repositories [10]. This reengineering effort aims to the creation of a software component repository that will contain useful information about the software system that is being analyzed. When the source code is organized, classified and sorted in a repository it becomes much easier for the developers to control it and thus become more efficient. This way when the time comes for a complete reengineering process (i.e. reconstruction with new and modern development tools) information such as specifications and design analysis is easier to be reused.

In this project a big software repository is created and managed with special tools and techniques so that the information stored in the repository can easily be retrieved. This information can be presented with different ways, each giving a certain point of view of the software system that is under analysis. The relations between the software routines and the data tables they manipulate give a good example to this. If someone focuses on a routine he can easily see all the data tables it affects and if he focuses on a data table he can easily see the routines that affect this table.

2.3. Bibliography overview

The scientific papers found in the bibliography search, in the area of software reengineering, can be divided into three categories. The first category includes those papers that refer to general reengineering issues. The second category consists of papers that provide an approach to software reengineering through the theory of patterns. Finally, the third category contains two papers that provide techniques for testing the results of a software development process.

2.3.1 General software reengineering issues

Arun Lakhotia in his paper entitled "Architecture Recovery Techniques: a unified view and a measure of their goodness"[32] provides a framework for comparing architecture recovering techniques (ART). He developed a scheme to classify ART's. In addition, he described a unifying view of ART's by rephrasing each technique using a common set of symbols and terms. Finally, in this paper a metric to quantify how "close" a recovered architecture is to an expected architecture is developed.

Arun Lakhotia mentions that recovering a system's architecture corresponds to performing cluster analysis. The term *cluster analysis* broadly refers to any method of grouping a set of objects such that objects in the same group are in "some sense" more similar to each other than those in different groups.

While talking about the effectiveness of ART's Lakhotia mentions that even a small variation in a graph, which a human, may easily not detect, could change its "meaning" drastically. Therefore there is a need to automate the comparison of the recovered architecture with an expected architecture.

As a conclusion the author refers that in order to keep up with changes in technology, the re-engineering of large software systems is the only viable alternative. He insists that any re-engineering task involves reverse engineering, i.e. recovering of design and other abstractions from source code.

The BAI Reengineering Team in its paper entitled "Software Reengineering Technique Classification"[27] proposes the creation of a reengineering technique catalogue or a reengineering process fragment library. The team focuses on four software abstraction levels namely axioms, requirements, design and implementation together with the various operations on software represented at these levels.

The BAI reengineering team insists that the reengineering of software seeks to clarify understanding of software, alter characteristics of code, or change the functionality of software. They claim that reengineering is like maintenance, in that it operates on existing software. Both reengineering and maintenance are like software development. Development creates new software requirements, design, and implementation from whole clothe, while reengineering and maintenance mend and alter an existing garment.

Furthermore, a definition of terms about software abstractions and reengineering actions is attempted. The BAI reengineering team claims that there are four *software abstraction levels*. They define these four *software abstraction levels* and two *software primitive definitions*, the forward engineering and reverse engineering. With the use of these definitions a classification of the various software reengineering techniques takes place.

Arun Lakhotia in his paper entitled "A Unified Framework for Expressing Software Subsystem Classification Techniques"[16] presents a unified framework for expressing techniques of classifying subsystems of a software system. The framework consists of a consistent set of terminology, notation and symbols that may be used to describe the input, output and processing performed by these techniques.

Arun Lakhotia claims that unless the architecture of a system is documented, which is very rare, its maintainer has to infer its overall structural organization from its source code. However, the architecture of a software system is not usually apparent from its source code. Having these facts in mind Lakhotia mentions that there is considerable interest in developing automated support for recovering the architecture of a software system from its source code.

He also refers that the crucial problem in recovering the architecture of a software system is classifying its components into subsystems. Next he points out that the emerging field of software architecture aims at formalizing the notion of architectures and developing languages to specify architectures. Finally, in this paper Lakhotia admits that research in subsystem classification is also of significance to research in reengineering procedural programs into object-oriented programs.

"Toward Experimental Evaluation of Subsystem Classification Recovery Techniques"[17] is the title of a paper that has also been published from Arun Lakhotia in cooperation with John M. Gravley. Lakhotia and Gravley introduce through this paper a technique that *measures* the effectiveness of subsystem classification recovery techniques (SCRT). SCRT's are used to classify software system components into subsystems.

The authors insist that though there is a rich body of literature investigating various aspects of classification problems, there is no classification technique that, if directly applied to a new classification problem, will guarantee good results. They justify their claim insisting that classification techniques are essentially heuristic and rely upon knowledge specific to the problem domain. Additionally, Lakhotia and Gravley insist that there is no single, general-purpose classification technique, but research into classification problems has led to classes of techniques such as generic templates that may be customized for individual applications.

Discussing the evaluation of the effectiveness of an SCRT the authors mention that there is no distinction in evaluating the effectiveness of an SCRT and any other heuristic technique. They believe that in order to evaluate a SCRT someone will have to subject it to several different inputs and compare its outputs against to expected outputs.

Lakhotia and Gravley mention that the problem for recovering the modular subsystem classification of legacy systems is an important issue in the reverse engineering of software. As a conclution they claim that several techniques have been proposed for recovering such subsystem classifications and that there is now a considerable collection of techniques. They insist that these techniques should be experimentally evaluated.

Arun Lakhotia is also the author of the paper titled "What is the appropriate abstraction for understanding and reengineering a software system?" [14]. Lakhotia believes that in order to understand a software system for maintaining it, a model (requirement, design or some other view) is created by abstracting certain details away from the source code. He also believes that the model plays a central role in any automated support for software maintenance and that the choice of abstraction (the notation or technique used to represent a model) therefore becomes a crucial decision in the development of any maintenance tool.

Lakhotia insists that the most appropriate abstraction of a software system varies based upon the maintenance task being performed. He mentions (as an example) that in order to make changes to a program such as adding a new feature, an abstraction that permits *concept assignment* and *impact analysis* is more useful. On the other hand, if the need is to reengineer a system (i.e. to move from one programming language to another), abstractions that enable restructuring and redesign of the code would be preferred.

The author makes two critical statements concerning the appropriate abstractions necessary to understand and reengineer a software system. The first is: "For activities (such as reuse and reengineering) requiring an understanding a *what* a software system does, abstractions that where (or should have been) used in the forward engineering of the system should also be most effective to recover by reverse engineering". The second is: "Even if two programs are written in the same language, the same abstractions may not be useful in understanding them for the purpose of reuse and reengineering".

In his conclusions Lakhotia mentions that so far the program understanding community has focused on recovering abstractions such as program cliches, logic or set theoretic expressions, cross-reference databases, control flow graphs and abstract syntax trees. These abstractions differ in the amount of information they hide as well as the effort involved in recovering them. Of these, only logic or set expressions are used, by some forward engineering methods at the requirements and design level.

Burd E. and Munro M. in their paper entitled “A method for the identification of reusable units through the reengineering of legacy code” [4] describe a method for reengineering legacy systems into potential reuse candidates so that they can eventually be replaced by more flexible and maintainable software. Their method consists of 10 steps to obtain the reuse candidates and employs both the analysis of code and the assistance of domain specialists. These steps are:

1. Generate a PERFORM graph from the source code
2. Generate a dominance tree from the PERFORM graph pairs
3. Identify reusable units from the dominance tree
4. Identify data dependencies within the source code
5. Identify data inter-relationships between sub-graphs
6. Identify potential reuse candidates from users/designers of code
7. Identify potential simplification procedures to assist encapsulation
8. Isolate sub-graph(s) to form reusable components using graph slicing
9. Identify data items in reuse units that would reduce data intersections
10. Identify SECTIONs where partitioning could assist separation

The authors mention that it has been estimated that about 70% of the total life cycle cost is spent on software maintenance and this fact makes the redevelopment of code an expensive and time-consuming business. They also mention that one recently proposed approach is to reengineer parts of the code into objects. This involves splitting the functionality of the legacy code into smaller encapsulated objects. The result of this restructuring is believed to aid the maintenance process. They generally believe that reuse has long been thought of as a way of improving productivity and improving quality standards. In addition, they believe that reuse based software development can reduce the costs of maintenance, because the nature of such software is more modular and therefore software updates are more localized.

Burd and Munro point out that similar techniques previous to their work had only been tried on small (averaging several hundred lines) Pascal programs. Their work has investigated its use for up to 30.000 lines of code. The case studies performed have indicated that the only issue of size concerned with the use of the method is the upper limits of the prototype tools used for analysis. However, if industrial strength tools were developed the upper analysis limits will be larger. The issue of size is less important than the issue of complexity. The greater the number of possible paths through the program, the size of reusable components is likely to be smaller.

From performing the case studies they have identified a number of drawbacks with using the method including:

- Complexity with data interactions means that further work must be performed on steps 9 and 10 to reduce the complexities of component interfaces.
- Development of good graphical tools is required to achieve automatic support.

Burd and Munro believe that the benefits of the method contribute to many aspects of software engineering and that users of this approach can expect to see an impact in the areas of software reuse, software maintenance, program understanding, impact analysis and code reengineering/restructuring.

Canfora et al. in their paper entitled "Decomposing legacy systems into objects: an eclectic approach" [5] observed a large consensus within the software maintenance community that identifying objects in existing procedural programs is desirable. They present a number of reasons why object identification within procedural programs is necessary, such as:

- Acquiring a precise knowledge of the data items in a program, the ways these items are created and modified and their relations.
- Understanding system design, testing and debugging.
- Reengineering from a conventional programming language into an object-oriented language.
- Avoiding degradation of the original design during maintenance.
- Facilitating the reuse of existing operations contained in a software system.
- Extending the benefits of recent programming innovations to most systems currently in use.
- Repairing the damages of extensive maintenance.
- Evolving the reverse engineered procedural programs in an object-oriented domain.
- Recovering software architectural representations from source code.

In this paper the authors present their experiences of applying an eclectic approach to identify objects in procedural programs. A premise of their research is that methods and tools proposed to identify objects achieve some level of success, but by no means a single approach can handle the large variety of problems that arise in real-life large-scale projects of identifying objects in legacy systems. They pursue an eclectic approach where a domain expert software engineer is encouraged to select, tailor and combine existing object identification methods to synthesize the method most suitable to the project and the system at hand. Their study demonstrates that using an eclectic approach, where existing object recovery methods are tailored and combined to adapt to the requirements and characteristics of a particular project, tends to produce objects, which contain less spurious connections and require less human effort to comprehend.

Dr. Carma McClure makes some interesting claims in the paper titled "Model-Driven Software Reuse. Practing Reuse Information Engineering Style"[35]. McClure gives out some reasons why reuse should occur in an enterprise. In that paper is mentioned that reuse makes sense because the similarity found across software systems (including code, design, functions and architectures) is enormous and undeniable. The author defines software reuse as the systematic development of reusable components and the systematic reuse of these components as building blocks to create new systems. The author claims that a reusable component may be code, but the bigger benefits of reuse come from a broader and higher-level view of what can be reused (such as software specifications, designs, tests, cases, data, prototypes, plans, documentation, frameworks and templates).

Certain cases that reuse has been applied and succeeded are also mentioned in that paper. McClure insists that reuse is more difficult to implement than other software technologies because it works best when applied above the single system level where there is more opportunity to reuse components and to get the payback from the investment in reuse. It is also claimed that the broader the base in which to practice reuse the better.

The author also states that the requirements of future systems must be identified and reusable components, that can fulfill common requirements, must be created and all this must be done prior to developing these systems. It is also mentioned that reuse must be planned in order to be successful. As a result the earlier in the life cycle process reuse is considered, the greater the benefits actually achieved from reuse will be. Practicing reuse in the early phases often leads to greater reuse in the phases, which follow.

While referring to domain analysis on enterprise models the author mentions that software reuse would be practical and easier to cost-justify if only we could predict what components would be highly reusable prior to new system development. McClure also insists that domain analysis provides a time and place in the software life-cycle to determine what the most valuable reusable components for the domain are likely to be and to create a library containing these components. Furthermore, domain analysis provides an opportunity to link business goals and system planning with reuse planning, enabling the most optimum practice of reuse for the corporation. McClure provides a table showing some reuse metrics together with the scope for practicing reuse. While referring to reuse-driven systems planning she claims that the goal of reuse-driven systems planning is to define the scope for practicing reuse in the corporation (by means of what organizational units, what life cycle phases, what application areas and what types of reusable components will be involved). Finally, she insists that management must establish the basic principles around the reuse of various types of reusable components such as the use of packages, software templates and software reuse libraries, as an alternative to building software from scratch.

Eng Huat NG, M. Sarshar and D. Pountney have published a paper entitled "An object-oriented learning system framework for domain oriented reuse"[8]. The paper is based on a project entitled Intelligent Multimedia Learning System (IMLS). The authors describe software reusability as a vital attribute of a high quality software component. They also insist that software reusability means that ideas and code are developed once, and then used to solve many software problems, thus enchanting productivity, reliability and quality.

While referring to portability the authors mention that this is a special case of software reusability where a whole application is reused from one environment to another without any changes at all, or simply by making small changes and get basically the same results. Then they make a reference to object technology claiming that it offers significant benefits in the construction of individual programs, but the real power of object-oriented information systems is that they are generally easier to modify and maintain so that they can rapidly adjust to changing circumstances.

The authors insist that computer aided learning systems are costly and time-consuming to develop. As a result, there are practical and economic motivations to develop a reusable and portable learning system framework. Referring to the IMLS's framework for domain oriented reuse they mention that it consists of four main domain models, namely: tutor, student, assessor and electronic resource. The IMLS project provides a framework to reuse the components of these domains. The authors insist that this framework can be applied vertically in each of these domains and in some cases horizontally to more than one of them.

Richard J. LcBlank, Stephen B. Ornburn and Spencer Rugaber have published a paper entitled "Recognizing design decisions in programs"[18]. Through this paper the authors describe a framework for documenting and manipulating design information for maintenance and reuse purposes. This framework is based on the analysis of programming language constructs. They claim that the only indication of a decision is its resulting influence on the source code and that in order to effectively maintain an existing system, it is essential to sustain previously made decisions, unless the reasons for the decisions have also changed.

The whole design process is described as a process that is repeatedly taking a description of intended behavior (whether specification, intermediate representation or code) and refining it. Each refinement reflects to an explicit design decision, which limits the solution to a class of implementations within the universe of possibilities. The framework presented introduces six classes that characterize design decisions made in programs. These are: composition and decomposition, encapsulation and interleaving, generalization and specialization, representation, data and procedure and the last is function and relation.

The authors claim that software maintenance and reuse activities require the detection of design decisions in existing code. They insist that this is part of the reverse engineering, which is the process of constructing a higher-level description of a program from a lower level. Typically, this means constructing a representation of the design of a program from its source code. The process is bottom-up and incremental. Low-level constructs are detected and replaced by their high-level counterparts. Then the authors claim that gradually the overall architecture of the program emerges from the programming language details if this process is repeated.

It is pointed out that it is not sufficient to simply recognize design decisions in code and that the decisions recognized must be organized in such a way, that they can be used effectively by maintenance programmers and reuse engineers. The authors insist that the organization chosen serves as a representation for design information. Describing the design decisions in programs the authors admit that they occur where the abstract models and theories of an application domain confront the realities of limited machines and imperfect programming languages. Finally, they mention that if the design decisions could be reconstructed, then there would be greater hope for the software engineers to be able to maintain and reuse the mountains of undocumented software confronting them.

Frank Svoboda has published a paper titled "Application of integrated process definition to reverse engineering"[25] in the context of the Army/STARS/Unisys Demonstration Project. This is a project that has successfully applied integrated process definition to promote understanding of its reverse engineering process and to communicate this understanding among the project team and to external stakeholders. Through this paper the project's reverse engineering process and the impact of an integrated approach to the definition of that process is highlighted.

The author mentions that software processes have traditionally been defined informally through standards and policy manuals. He claims that these process descriptions suffer from their inability to be analyzed, discussed, taught, learned and changed. It is also mentioned that stable environments can hide problems caused by informal process definitions that might surface when subjected to personnel or technological change. The author also insists that integrated process notations help gain control over software efforts and that by enabling quicker access and understanding through coherent graphical organization the impact of change can be minimized.

Referring to reengineering attempts the author claims that as funding for major new projects becomes scarcer, non-developmental efforts, such as reengineering, maintenance and reuse are becoming increasingly popular. It therefore becomes incumbent upon software engineers to understand and perfect the processes by which better use can be made of the legacy systems. He also insists that reverse engineering should capture system understanding from artifacts across the entire life cycle and not just code.

Arun Lakhotia has published a paper analyzing his experiences with modifying large, real-world programs written by other programmers [31]. Lakhotia insists that the functionality of a program is not only understood from its documentation but also by executing it and inferring relations between its inputs and outputs. He also claims that when deleting a function, the code implementing it is not destroyed, only execution paths leading to it are disconnected; leaving behind *dead-code*. The replicated and dead-code segments are major contributors to the difficulty in understanding and modifying programs.

The author insists that software maintenance heavily relies on the programmer's ability to comprehend programs and that this observation has prompted several researchers to investigate the processes involved in understanding programs. He also mentions that if a programmer is not knowledgeable about the program's domain, it will make it harder for him to understand the code and presented briefly some works relevant to the subject of analyzing computer programs.

Lakhotia through this paper gives a narrative of four program modification exercises performed on two real-world programs (GCC and WPIS). Some students performed these exercises, while Lakhotia was structuring and observing them and in the end he came out with some interesting conclusions. He claims that: "A programmer may scan the code in search of information that may be used to create hypotheses about a program's design. Scanning the code is different than *reading* the code to understand each statement. It is also different from tracing control flow paths. Scanning a code means flipping through text in search of *beacons*. Such a scan may also be done by searching for occurrences of certain *words* or character sequences of relevance to the context".

Lakhotia also mentions in his conclusions that: "The ease with which a program may be understood depends on several factors. It depends on the programmer's experience with modifying code and also his knowledge of the domain and the relevant programming concepts. It is influenced by the programmer's ability to create hypotheses about a program's design (or behavior) and the ability to test the hypothesis using observed facts and logical reasoning. A system decomposed into several levels of abstractions is easier to understand, than one with a more coarse grained decomposition. He also insists that organizing the source code in a directory hierarchy that reflects this decomposition can further help in comprehension of a large system".

Spencer Rugaber and Richard Clayton in their paper entitled "The representation problem in reverse engineering"[22] examine the representation problem by presenting taxonomy of solutions. They make a distinction between the mental activities involved in understanding a program and the representations produced. It is also claimed that the reverse engineer is responsible for developing mental models of what a program does and that representations and formal representation techniques are tools for modeling and for expressing the results.

The authors believe that programmers are the people most likely to need a detailed understanding of a software system. They also believe that programmers are trained as model makers and so are likely to have both the skills to construct mental models and an appreciation of the value in doing so. Furthermore, in this paper it is mentioned that for reverse engineering to be successful, a representation must be constructed describing the implementation architecture and how it relates to the problem being solved by the software.

Rugaber and Clayton mention that there is a problem related to the fidelity of the representation. They insist that there are many cases in which the corresponding representation does not retain all of the information available in the source code. They also believe that if the results of the reverse engineering are going to be used to support other activities, such as maintenance and reengineering, then it is desirable to be able to use the same representation for these activities as was used during the reverse engineering. The authors define the application model as a description of the problem the system is trying to solve. They claim that a successful reverse engineering effort must include a description of what a system does and how it does it. They also claim that representations of the initial code models are necessarily low level because they are derived directly from the code. Finally, Rugaber and Clayton make a meaningful observation mentioning that the ability to integrate representations ensures that reverse engineers, using a collection of previously created models, will be able to extract maximum value from them with minimum effort.

Melody Moore, Spencer Rugaber and Phil Seaver through their paper entitled "Knowledge-based User Interface Migration"[20] chronicle a study of user interface migration issues, examining and evaluating current tools and techniques. They also describe a case study undertaken to explore the use of knowledge engineering to aid in migrating interfaces across platforms.

The authors insist that a significant problem in reengineering large systems is adapting the user interface to a new environment and that portability across platforms is a major concern in the software industry today. They claim that legacy systems are being reengineered with portability and multi-platform considerations as priorities.

Referring to the user interface of an application, Melody Moore, Spencer Rugaber and Phil Seaver insist that since many tools are based on high-level abstractions, it is important that the user have the ability to fine-tune the interface generated by a tool. They also believe that graphical user interface builders can drastically speed up the development process, since much of the code can be generated automatically. Finally, they claim that the fundamental problem, while migrating the user interface, is to preserve the functionality of the original interface, while accommodating the differing stylistic conventions.

2.3.2 Theory of patterns

Erich Gamma et al., in their paper entitled "Design Patterns: Abstraction and Reuse of Object-Oriented Design"[6] describe how to express and organize design patterns and introduce a catalog of them. The authors' experience in applying design patterns to the design of object-oriented systems is also presented.

Design patterns are defined as reusable micro-architectures that contribute to an overall system architecture. The authors believe that software architects, who are familiar with a good set of design structures, can apply them immediately to design problems without having to discover them. They also claim that design structures can improve the documentation and maintenance of existing systems, by furnishing an explicit specification of class and object interactions and their underlying intent.

A list of uses of design patterns in the object-oriented development process is presented. Erich Gamma et al., insist that design patterns act as building blocks for constructing more complex designs. They also insist that a good set of design patterns effectively raises the level of the development process and that design patterns help a novice perform more like an expert. Finally, as a conclusion the authors insist that using design patterns early in the lifecycle may avert refactoring at later stages of design.

Perdita Stevens and Rob Pooley in their paper entitled "Software Reengineering patterns"[23] introduce the idea of software reengineering patterns adapting the ideas of design patterns. This work identifies lessons in successful reengineering projects and makes these lessons available to new projects.

The authors believe that the so-called legacy systems are normally, but not necessarily, large systems built in an era before encapsulation and componentization were regarded as fundamental tenets of design. It is also mentioned that software practitioners have adopted patterns enthusiastically, because a pattern is an effectively transferable unit of expertise.

Referring to legacy systems Stevens and Pooley insist that there seems no reason to be confident that today's new systems are not also tomorrow's legacy systems and that the problem of reengineering legacy systems is probably here to stay. The authors also insist that it is almost universally accepted that a system that consists of a loosely coupled collection of highly cohesive components, is easier to adapt than one that is not.

Finally, while referring to the reengineered system, it is mentioned that its architecture and high-level design are not identical with that of the original system and that the best design for a reengineered system may not be achievable in practice, but a sensible compromise may be available.

Perdita Stevens and Rob Pooley have also published a paper entitled "Systems Reengineering Patterns"[24]. In this paper they claim that today's business can only survive if they can adapt rapidly to a changing environment and take advantage of new business opportunities. The authors aim also to understand the way that software practitioners with a lot of experience undertake the reengineering of legacy systems. With this knowledge the authors claim to be able to develop efficient techniques and material for transferring expertise.

Stevens and Pooley believe that the wide range of factors, which must be taken in to account in evaluating candidate solutions, exacerbates the problem of becoming expert. They also believe that software engineers have great difficulty in becoming expert engineers.

The authors define patterns as a successful, recurring solution to a common problem in a given context. They insist that a reengineering pattern embodies expertise about how to guide a reengineering project to a successful conclusion. Using reengineering patterns they intend to address a problem in a way that takes into account the needs of a software engineer who needs to make decisions about reengineering in a reasoned way, taking advantage of the experience of others. Stevens and Pooley regard the reengineering process as the process of applying engineering principles to an existing system in order for it to meet new requirements.

Walter Zimmer in his paper entitled "Experiences Using Design Patterns to Reorganize an Object-Oriented Application"[26] expresses his experiences in the reorganization process of a hypermedia application, in which the design pattern approach was used. Zimmer mentions that the team of the project used design patterns as a major resource in the reorganization process for three main reasons. The first is that the usage of design patterns promised to improve the design quality and comprehensibility by reusing well-designed micro-architectures. The second is that design patterns provide a common vocabulary for discussions and documentation. The third is that design patterns raise the granularity level of design from classes, methods and relationships (inheritances, components) to larger building blocks.

On the contrary, it is also mentioned that design patterns are only one design approach and do not cover all aspects of software design. The author insists that further support is needed for tasks like finding of design flows or the documentation of the application. Concerning the reorganization of an application, the author insists that it requires good knowledge and comprehension of the application. He also insists that in order to be effective, the developer should have a good overview about generic and domain-specific design patterns, their intentions (addressed design problems) and their applicability (typical situation / context in which the design pattern is applied).

Zimmer claims that the existence of a common vocabulary is the main advantage of design patterns. He also claims that as reorganization is a time-intensive task, effort should be focused on really critical issues. Besides experiences from the original developers, scenarios and design metrics can be systematically used to identify application deficiencies and to get suggestions for improvements ideally in the form of design patterns. In addition, Zimmer claims that the complexity of the reorganization task should be reduced as much as possible and that design of important abstractions in the application domain (links, documents) often requires the combination of several, interrelated design patterns.

The author wonders about the problem of organization and manipulation of patterns in the case that their number grows fast. He insists that if larger collections of design patterns (hundreds instead of twenty) are developed in the future and if the application is really large (hundreds of classes), the developer would have to be supported by tools. Those tools would help to manage complexity, to accelerate the exploration of design alternatives, to learn / understand design patterns and to free the developer form error-prone detail work.

2.3.3 Software testing theory

Stephane Barbey et al. in their paper entitled "From Requirements to Tests via Object-Oriented Design"[1] study the testing procedure in an object-oriented (OO) development process. This paper is based on a production cell CASE study. The various problems that have appeared during the CASE study are reported. These problems mainly concern controllability and observability issues. The authors insist that these problems have caused some iteration and backtrack on OO analysis and design.

The authors believe that software testing must be anticipated and prepared during the whole development process, in order to be successful. They insist that testing involves exercising the software by supplying it with input values. They also insist that exhaustive testing is not tractable and that the tester faces the problem of selecting a subset of input domain that is well suited for revealing the possible faults.

Formal testing methods are presented as an approach to reveal faults in a program, by verifying its functionality, without analyzing the details of its code. The authors believe that the goal is to answer the question: *Does the program satisfy its formal specification?* Concerning the development environment of an application, the authors insist that the classical development methods for procedural programs involve a hierarchical decomposition of functions and that on the contrary, OO development methods are characterized by decentralized architectures of objects. The authors also insist that in order for the test to be effective this observation has to be taken into account (i.e. traditional unit and integration levels of testing do not fit well in OO development methods).

Stephane Barbey, Didier Buchs and Cecile Peraire have published a paper entitled "A Theory of Specification-Based Testing for Object-Oriented Software"[2]. Through this paper the authors propose the adaptation to object-oriented software of an existing theory of testing, for stateless abstract data types (ADT), to find errors in a class, by checking that its implementation meets its specification. The authors study the construction of the procedure that analyses the results of the tests, adapted to object-oriented software.

The authors believe that although some people have assumed that object-orientedness leads by itself to quality, experience has proved that object-oriented software cannot escape to a validation and verification process. They also insist that the main problem is to build a test set that has enough significance to find a maximal number of errors, so that a 'yes' answer gives confidence to the programmer that the program meets its specification. Referring to testing systems that have been developed using object-oriented technologies they claim that testing must take into account the specifics of the object-oriented development methods and of the structure of object-oriented software. They also claim that traditional strategies need adaptation to fit object-oriented systems.

This paper is focusing -as mentioned in its title- on specification-based testing methods. The authors call these methods *black box* methods. They define black box methods as an approach to find errors in a program by validating its functionality, without analyzing the details of its code, but by using the specification of the system. It is mentioned that the goal is to answer the question: *Does a program satisfy the requirements of its specification?* or, in accordance to the goal of testing, to find if a program does not satisfy its specification.

Answering the previous question the authors propose a specific strategy. This strategy includes selecting from the specification the services required from the system. For each service, the specification allows the selection of a number of scenarios for the program under test. The set of all these scenarios makes up the test set. Furthermore analyzing the test set, the authors mention that an exhaustive test set should obviously contain all the tests that are required by the specification. Then they admit that an exhaustive test set is generally infinite, and it is necessary to apply a number of reduction hypotheses to the behavior of the program in order to obtain a finite test set of reasonable size.

2.4 Commercial implementations concerning software reengineering

Software reengineering is not a new issue to the software development community. In fact, it appeared along with the first pieces of source code. Initially software reengineering was not well accepted because its results were not regarded as original software but a replicate of others people software. This situation changed as the amount of source code was growing rapidly. The need for software reengineering and reusability became a necessity when software development tools progressed and became mature and sophisticated. As new generations of sophisticated and efficient development tools emerged, systems that were developed with earlier tools were still used in business. This situation led to the so-called software crisis, which characterized the software community during the last decade of this century.

The reactions of the software engineering community against the problem of software crisis had as a result the development of software tools that would be in a position to transform the old software systems to new ones, embedding all the advantages of the modern systems. It is widely admitted that the most time consuming procedure is the definition of the requirements, specifications and design of a software system rather than the implementation of the actual code. The software reengineering tools are able to extract the specification and design information from the source code of the old systems and, therefore, save valuable time in the development process. Three typical systems that have been developed for software reengineering purposes and are available for commercial use will be presented.

2.4.1 The DMS® Software Reengineering Toolkit

Semantic Designs Inc [45] have developed the DMS toolkit. DMS enables the analysis, translation, and/or reverse engineering of large-scale software systems, containing arbitrary mixtures of languages. It can also be used for domain-specific program generation. DMS offers the reengineer the following capabilities:

- Robust parser and lexer generation.
- Automatic construction of abstract (not concrete) syntax trees.
- Semi-automated pretty-printer generation (to reverse the parsing process), according to a specified layout information.
- Multi-pass attribute-evaluator generation from grammar.
- Sophisticated symbol-table construction facilities.
- Control-flow graph construction and data flow analysis framework
- Multiple domains can be represented at the same time.
- Transforms and patterns can be written directly in surface-to-surface domain syntax form.
- A full Associative/Commutative rewrite engine that operates on trees and DAGs, which can be used to apply sets of transforms.
- A metaprogramming language, XCL, provides the ability to control the sequencing of the application of transforms and sets of transforms.
- An algebraic specification subsystem can be used to specify arbitrary algebras (this is just a DMS domain!).
- Industrial Scale: DMS is designed to work on source systems with up to 1 million lines of specification.

While complex legacy grammars can be defined quickly for DMS use, there are grammars for:

- ANSI C and C++ with intelligently managed preprocessor directives
- COBOL85, and IBM VS COBOL II with CICS/SQL, with COPYLIB management, and name and type resolution
- Fortran95/90/77
- Java
- IBM JCL
- ISO Pascal
- PL/1
- Progress (a 4GL)
- SQL (SQL2 aka SQL 1992 and SQL3)
- Verilog
- VHDL
- Visual Basic

Application of DMS:

DMS can enable an organization to carry out a variety of useful analyses and modifications on a large system. Some ideas are listed below:

Program modification

Application evolution, massive regular change, porting, restructuring, optimization and Program Analysis

Program analysis

Metrics, organization style checking, programming information extraction, domain information extraction, semantic faults, test coverage, Domain-specific program generation

Domain-specific program generation

Partial differential equation solvers, factory control synthesis, entity relationship compilers, protocol compilers, automated test generation Legacy code reverse engineering

Legacy code reverse engineering

Design recovery to domain abstractions, incremental design capture, reusable component extraction, component extraction for domains, legacy mergers, business rule extraction

In addition, Semantic Designs Inc have another tool called CloneDR [44]. The CloneDR has a particular configuration designed to find exact and almost-identical blocks of code ("clones") in large systems, and remove them by replacing them with invocations of abstractions (macros, procedures, etc.). The technology is generic enough so it is applied to COBOL, C/C++, Java, Fortran 90, (and recently as an interesting experiment, to VHDL.). Someone can download a demo C/C++ clone detector/remover.

2.4.2 George and James Software

George and James Software [47] provides some software reengineering products. These are:

RE/2000

This is a dedicated analysis tool designed to address year 2000 problems by identifying and classifying date related processing in M source code. RE/2000 analyzes your source code looking for date related processing and produces a set of hyperlinked documents that provide:

- A summary of problem instances, by severity level, for all routines analyzed.
- An index of routines analyzed, with totals by severity level and by application system.
- Routine source code listings in html format marked up with hyperlinks that highlight date related processing.
- Each instance of date related processing has a link to a problem explanation, which identifies possible problems with the highlighted code.
- Each problem page has further links to a problem catalog, which describes known problems in more detail and provides guidance on how to deal with these problems.
- RE/2000 incorporates a library of over 500 problem primitives in 17 different problem classes that indicate date related processing. This library can be user extended to incorporate date formats and algorithms that are unique to your applications.
- The optional RE/parser add-on module allows additional problem classes to be created giving maximum flexibility in extending the capabilities of RE/2000 to meet the most demanding requirements. RE/parser also enables code to be re-engineered allowing frequently occurring problems to be corrected automatically.

RE/m

This is a Reverse Engineering tool that has the following features:

- At the heart of RE/m is a repository containing information about the programs and data that make up an application. This repository contains the documentation needed to maintain and enhance any M application.
- Driving RE/m is a powerful code analyzer that performs the Reverse Engineering process of extracting information from M routines, analyzing it, and loading it into the repository.
- Sophisticated analysis tools then enable the analyst/programmer to access comprehensive and accurate documentation to assist in the support, maintenance and enhancement of an application.
- RE/parser is an add on module that can be user configured to meet customer specific needs including automated syntax conversion and advanced quality control.

RE/data

This tool is a member of the RE/m family of re-engineering tools. It derives data dictionary information from the application code, M databases and other sources, and uses it to create a data dictionary that describes the structure and contents of your database. The dictionary created can be used to automatically populate a range of dictionary driven third party products including any product that can read standard SQL DDL. It has the following features:

- RE/data is the fastest way of building a data dictionary for an existing M application. Using RE/data, in ideal conditions, between 25 and 50 global nodes (tables) can be defined per day.
- RE/data promotes the re-use of attribute definitions. If an attribute such as Patient Number is used throughout a database then RE/data allows all instances of Patient Number to be consolidated into one attribute definition. This achieves a highly consistent dictionary with corresponding improvement and quality.
- RE/data assures quality by enabling the dictionary to be validated against sample databases at any stage of the dictionary creation process. This validation can be used periodically after the dictionary has been created to ensure that your investment is protected.

RE/parser

This is a fully user configurable code parser. It can be used in two ways, firstly as an add-on to RE/m, RE/data or RE/2000 it can help you to get more information out of the code being analyzed by customizing the products parser tables to understand more about the way your applications are written. Secondly, as a stand-alone product RE/parser can be used to re-engineer M code by replacing what it parses with user specified constructs. RE/parser comprises three parts:

- Parser Engine - which reads M code and interprets it in accordance with the rules defined in a Parser Table.
- Parser Table Editor - which enables the user to create and edit Parser Tables.
- Parser Tables - that defines the complete M syntax in accordance with the ANSI/MDC X11.1-1995 standard.

2.4.3 The Metamorphic^{TM} COBOL Converter

The Metamorphic TM COBOL Converter [48] allows users to take existing ANSI 85 COBOL programs and translate them to Java 1.2, Microsoft Visual Basic 5.0 or C++. By allowing the user to select his target language, he can select the appropriate target platform and technology to solve his business problem today. Metamorphic Computing, Corp. (MCC) advertises some sample COBOL source code and how it is converted to Visual Basic and Java sources.

MCC announced a new product that allows programmers to migrate existing legacy applications up to 100 times faster than previously possible. By running COBOL code through the Metamorphic TM Converter the user gets compile-ready code. The conversion is done by the customers or as a service by MCC. The purchase options available are (1) straight conversion by MCC with no tuning - designed for the fastest turnaround (2) conversion by MCC with tuning - designed for customers who need help revolutionizing their existing process (3) direct purchase of the converter.

2.5 Summary

In this chapter the analysis and study of existing documentation in the research domain of reengineering and reuse of legacy software systems has been investigated and presented. This domain includes general software engineering concepts and especially reengineering concepts. Several computer theories have been developed to provide answers to serious questions in this research domain. The most important of those theories were presented and analyzed. These are the theory of software objects, the software component classification theory, the representation theory, the theory of patterns and the theory of software repositories.

The scientific papers found in the bibliography search, in the area of software reengineering, were divided into three categories. The first category includes those papers that refer to general reengineering issues. The second category consists of papers that provide an approach to software reengineering through the theory of patterns. Finally, the third category contains two papers that provide techniques for testing the results of a software development process. Through the presentation of these papers the problems that other engineers have faced in the area of software reengineering and the solutions that they propose were outlined. These problems include theoretic issues related to software reengineering as well as applied solutions provided to specific demands.

Finally, three typical systems that have been developed for software reengineering purposes and are available for commercial use have been presented. These are the DMS reengineering toolkit, the tools from George and James Software Inc and the Metamorphic COBOL Converter. The fact that some software companies have already developed and marked some products concerning this research domain, justifies the purposes for the implementation of this project. Through this thesis it is envisioned that another theoretical point of view and another reengineering tool will be provided adding another small rock in the mosaic of the whole research area of software reengineering and reuse.

Chapter 3

Requirements Analysis and Specifications

3.1 Introduction

The need to reuse pieces of software in the construction of an application appeared in the early years of software development. One of the first steps towards software reuse was the organization of special routines and functions into software libraries, which were distributed to the programmers. The use of those libraries saved time and manpower. The software pieces, which were included in them, were tested and verified, so the possibility of a logical or implementation error during the development of the software application was decreased.

In later years the software engineering community made significant progress in software reuse. Software engineers invented tools and techniques in order to reuse effectively software elements. Not only source code but also designs, requirements, specifications, development processes and decision experiences are now also candidates for reuse [10].

The mechanism, which is presented in this thesis, is another approach to the aims of reengineering large software systems, which have been developed using 3rd generation computer languages. This mechanism focuses in the process of the analysis of software systems by software engineers who did not participate in the development process of those software systems. It brings out information about the design and implementation of a software system by scanning its source code. In order to get information from a software system, it is very helpful to read its documentation. However, it is very rare to find a documented software system. Also when such a document is available it is usually inconsistent since it has not been kept up to date with the modifications done to the software system. Therefore, the only definite document of a program is its source code [32].

The 3rd generation development tools are designed to work using a compiler. The compiler is a piece of software, which reads the source code and produces binary files. These binary files may be directly executed by the operating system or a special program, usually called runtime, is used to execute them. In the real world large software systems are in use consisted of millions of lines of source code, which have no documentation about their design and implementation. The information about their design and implementation is necessary in order to maintain or reengineer those systems. The formal way to regain such information is to have a team of software engineers to read all the source code of the software application and try to understand the architecture and the logic behind each routine. This procedure is time consuming and as a result an expensive process. An alternative way is to develop a software system to extract information concerning the structure, implementation and design of an existing software application. This software system should function in a way similar to the compiler of the computer language. It should be able to read the source code of the application, but instead of producing the applications binary files it should present information about its design and implementation.

In this document the requirements, which this software reengineering mechanism will have to meet, are investigated and studied. Based on the investigation and study of these requirements, a set of specifications, which this mechanism will have to fulfill in order to meet its requirements, is developed.

3.2 Requirements analysis

Software engineers are interested in specialized *pieces of information*, which are spread throughout the source code of the software application, in order to understand the design and implementation analysis of the application. The meaning of *"pieces of information"* is abstract and subjective. In software engineering, one could find some *"pieces of information"*, about an old application, important and at the same time, someone else could characterize them meaningless. For example, if someone needs to gain information about the implementation and design of a software system, which has been developed for the needs of a hotel reception, he would search for information about how this program handles the data of the customers, the rooms and the reservations. This kind of information makes no sense in a software application, which is used to control the productivity of a car factory! Additionally, the computer language and all the software tools used in the development of a software application, play pivotal role in the procedure of its design and implementation analysis.

However, it is possible to provide general information about the design and implementation of a software system by studying the specifications of the computer language used to develop it. As McLure [35] mentions "reuse makes sense because the similarity found across software systems is enormous and undeniable. This includes code, design, functional and architectural similarities". This thesis, focuses on reengineering software systems developed using 3rd generation computer languages. These computer languages have common characteristics. By studying those common characteristics, it is possible to design a software system, which will be used to gain critical information about the design and implementation of a software application, which has been developed using a 3rd generation computer language.

Three cases of common characteristics, of the 3rd generation computer languages, are of major importance because they provide information of the following:

- The database structure of the application
- The control flow of the application
- The organization of its source code

In the first case, the common characteristic is the existence of a separate datafile for each data table of the database. Additionally, the structure of each datafile, accessed by a program of the application, has to be defined in the source code of the program. It is also possible to have two different definitions for the same datafile in two different programs of the software application.

In the second case the commonly used GOTO statement affects the control flow of a program. While this statement is widely used by the programmers, software engineers believe that restructuring in the early days of structured programming implied removing the GOTO statement [34]. There are also a number of statements, which each 3rd generation language uses, in order to pass the control flow of the application to external programs. It is also possible to pass information to the external program, by supplying values to special variables, before the external program starts running.

In the third case the common characteristic of the 3rd generation languages is the organization of the source code in text files. These files could be spread in a directory tree. During the compilation time, the compiler will have to read these source files line by line and produce the binary programs of the application. There are also special statements of the computer language, which instruct the compiler to use additional source files while building the program. This mechanism enables the developer to create his own libraries, which may be used by several programs of the application.

The three cases of common characteristics of the 3rd generation computer languages indicate that the requirements, which this software reengineering mechanism will have to satisfy, should be divided in three main sections. The first is the *database analysis* of the software application, the second is the *control flow analysis* of the application and the third is the *source code organization analysis.*

3.2.1 Database Analysis

Every software application manipulates data. Some applications manipulate small amounts of data while others deal with huge database files. Within the last fifteen years great progress in the process of data manipulation has been achieved. Software developers nowadays use modern tools and techniques to manipulate data. One of the motives of this software reengineering mechanism is to provide information about the database used by the *old* software application and propose tools and techniques to move from the *old* database schema to a new one. This mechanism should be in a position to: a) present the *database structure* of the software application, b) provide information about the *record structure* of each table in the database, and c) provide *information concerning the table relations*, which may occur in the database.

3.2.1.1 Database structure

In order to analyze and reengineer a software application, which manipulates large amounts of data, it is desirable to know the complete set of the datafiles, which constitute its database. For each datafile used it is also important to know the set of the programs or routines, which access this datafile. This kind of information can give useful hints about the structure of the software system. Some datafiles are used to store the basic information that the application manipulates, while others can be used to store some details, which are 'less important'. Creating sets of programs of the software application, which access specific datafiles, can be useful in the analysis of the applications design and implementation. For example, in a hotel management system there could be a set of programs used to manipulate the information about the reservations of the hotel rooms and another set of programs to manipulate information about the tourist offices, which make the reservations.

It is also helpful to find out the source files that define the structure of the datafiles used. The 3rd generation languages need to have the complete definition of the structure of each datafile accessed by a program, defined into the programs source code. If two different programs inside the same software application access the same datafile, then they must include the same definition of this datafile in their source code. Nevertheless, in 3rd generation languages it is possible to have two different programs access the same datafile, with different definitions for the structure of the datafile, in their source code. These programs may execute without run-time errors, but the results may be unpredictable since 3rd generation languages do not guarantee the integrity of the data of the software application. In small software systems, it is easy to control such a demand but in large systems constituted by thousands of programs and hundreds of datafiles this is a very difficult and time spending procedure.

3.2.1.2 Record structure

The identification of the format of each data table used by the software application provides the basic information about the design of the database of the application. Any possible indexes and relations must also be identified in order to analyze the database structure. Almost every modern relational database management system (RDBMS) provides tools that graphically represent the design of the database in forms of tables, records, fields, indexes, relation etc. It is useful to provide the same information about the database of a software system developed under a 3rd GL.

It would be helpful and time saving to identify the source files that contain the definitions of each data table. Many programs inside the same software application may share these source files. It is a common tactic the sharing of the same source file in different programs, in order to define the structure of a specific datafile in these programs. Using this technique the possibility of a mismatch between the two different definitions is minimized. Once the record analysis, in the form of fields and indexes, has occurred, design or implementation errors, which obstruct the normal operation of the software system, can be easily identified. The 3rd GL's do not have any tools to provide information concerning the record structure and the developer is always responsible for the design and implementation of each datafile in every program, which accesses the specific datafile. This may result to design and implementation errors because the development tool does not apply restrictions in the design of the database and people can always make logical errors.

3.2.1.3 Information concerning the table relations

Software engineers can unveil information about the relations that might exist between two datafiles, by analyzing the record descriptions of the data tables and comparing them. Although the 3rd generation languages do not support special handling of relations between two separate datafiles, software developers have the ability to implement such relations by generating special fields in the record that match the related records between the separate datafiles. As a result, there is only one valid procedure in order to investigate the existence of such relations. This procedure is divided in three steps. In the first step the analysis of the structure of the record description of each datafile takes place. In the second step the comparison of the record description of each datafile against the record description of all the other datafiles, which exist in the database, provides clues about the relations that might exist among them. In the third step the existence of those relations is validated by browsing the data of the possible related datafiles and comparing them.

The information concerning the relations among the datafiles can also be used for creating sets of datafiles. Usually the datafiles, which are related to each other, contain similar information. The grouping of datafiles according to the relations found could create sets of datafiles, inside the same software system, which contain similar information. In the previous mentioned example of the hotel management system there could be a set of datafiles used to store information about the reservations of the hotel rooms and another set of datafiles to store information about the tourist offices, which make the reservations.

3.2.2 Control flow analysis

Knowledge of the control flow of a software application gives detailed information about its structure and functionality. The control flow of a software system is usually presented using control flow diagrams. Such diagrams illustrate the control flow of each procedure of the software system in full detail, so studying those diagrams may yield information concerning the applications design and implementation. Nevertheless, it is not always desirable to have a diagram describing the control flow of a software application. The control flow diagrams of large software systems are huge and sometimes can approach infinity.

An alternative approach to present the control flow of a software system is to use static analysis. Static analysis of a large software system presents all of its components and describes their attributes. The information derived from the static analysis of a software application is concise and substantial. It provides briefly critical information about its structure and design so it is very useful for a software engineer to obtain such information about a software system in order to commit reengineering changes on it. It is possible to combine the information derived from the static analysis of a software system and obtain information regarding its control flow.

The information needed from the static analysis of a software application, in order to analyze its control flow, consists of three main parts. First of all, it is necessary to know *the complete set of programs*, which constitute the software application. Next, *the calls among programs* should be identified. Finally, it is necessary to know *the database attributes of each program*.

3.2.2.1 The complete set of programs

It is important to identify all the programs that constitute the software system in order to analyze its structure and design. In order to identify all the programs of the software application, it is necessary to assign a unique name to each program. The name of the program should be the internal name that the computer language uses to reference it. In many 3rd GL this name is defined in the source code (COBOL is one of them) while in others use the name of the source file of the program. The name of the source file of the program is the external name that is used to store the source file in the storage media (filesystem).

In order to compile a program many other components may be required in addition to its source code. For example, it could use libraries, which could have been developed by the programmer or be provided by the computer language used. These libraries may be simple source files, without an internal name, which are patched in to the source code of the main program. This means that each source file does not necessary define an internal name to the program, even if the computer language needs to assign an internal name to each program. As a result it is useful to have a report that will present all the programs of the application and for each program, the name of the source file that the compiler is instructed to read first in order to build it. This is the main source file of the program.

3.2.2.2 The calls among programs

Other programs that are called from each program should also be identified in the static analysis of the software application. For each program there should be a report presenting all the external programs which are called. Using this information it is possible to get a description of the control flow of the application. The information provided by reporting the calls, which take place between the programs of the software application, can illustrate a *higher-level* control flow description of the application. Such information provides a brief view of the control flow of the application and gives basic information about its design and implementation.

Examining the calls, which take place among the programs of the application, it is also possible to create groups of the programs used by the software application. For example, a software system developed to cover the needs of a commercial store could use a set of programs to manipulate information of the customers and suppliers and another set of programs to manipulate information of the product trades. It is always possible to have calls between programs, which are not in the same group. Nevertheless, the most common situation is to have the most calls between programs, which belong in the same group.

3.2.2.3 The database attributes of each program

It is useful to know the datafiles, which are defined in the source code of a program and the datafiles, which are accessed by this program. It is possible to have the definitions of the datafiles in separate source files (user defined libraries), which are referenced during compilation time. This tactic minimizes the possibility to have different definitions referring to the same datafile in two different programs. In adition, by examining the programs, which access common datafiles, it is possible to create groups of programs, which implement similar functions.

Once all these attributes have been identified for each program of the software application, a software engineer may retrieve useful information about the structure and design of the application. First he can get a brief listing of all the programs constituting the software system. Then he can focus on the properties of each program and navigate from one program to another according to the questions he needs to get answers for. For example, he can focus on program P1 and find out that it affects the datafile F1. Then he might be interested in identifying all the programs that affect the datafile F1, for example program P2 is one of them. Then he could focus on program P2 and so on.

3.2.3 Source code organization analysis

The 3rd generation languages use text files to store the source code of the programs. These source code files are usually organized in a directory tree hierarchy depending on their usage. For example, some routines or functions that are common to many programs of the software application might be in a separate directory. While it is interesting to know how the source code is organized in subdirectories, the most important is to know how these source files are combined during compilation. The compiler can be instructed to use additional source files in order to build a program. Such instructions are passed to the compiler using special statements in the source code of the program. This way the 3rd generation languages give the opportunity to the developers to create their own libraries. As a result, for each program of the application, it is necessary to know the complete set of source files used to build it, in order to be able to study and analyze it.

As mentioned in section 2 of this document, the source files of the application usually are located in different subdirectories for organizational reasons. For example, the source files, which are used as user-defined libraries may lie in a separate directory. This software reengineering mechanism should be in a position to locate the source code files in the various subdirectories in order to filter the source code and derive the desired information. In the usual form most software applications use a top-level directory and all the source code files are located in a number of subdirectories below this top-level directory. It is not good practice to have files that are not part of the applications source code, for example the binary files of the application, mixed with the source code files, although this situation is a usual phenomenon. Assuming that the application has its source code files and only them, spread in a separate directory tree, this mechanism should be in a position to scan all the subdirectories, bellow the top-level directory of the source code of the application and locate all the source code files.

3.3 Specifications

After the study of the requirements, which this software reengineering mechanism will have to satisfy, the specifications must be outlined in order to design and implement it. The requirements of this mechanism have been categorized in three main sections namely *database analysis*, *control flow analysis* and *source code organization analysis*. For each section, a number of specifications must be outlined, in order to fulfill these requirements.

Tables 3.11 - 3.3 provide the specifications, which this reengineering mechanism will have to fulfill. **Table 3.1** provides the specifications of the database analysis section, **Table 3.2** provides the specifications of the control flow analysis section and **Table 3.3** provides the specifications for the source code organization analysis section of this reengineering mechanism.

Table 3.1 – Database Analysis Specifications

1. Database analysis	
1.1	**The complete set of datafiles** All the datafiles, which exist inside the database of the software application, must be reported. The software system, which will be developed in order to implement the aims of this software reengineering mechanism, should be capable of giving a report including all the datafiles that the software application uses.
1.2	**Access attributes** For each datafile there should be a separate report showing all programs, which access the specific datafile.
1.3	**Definition attributes** For each datafile there should be a separate report showing all the programs or their source files, which define the specific datafile in their source code. The user of this mechanism should be in a position to find easily these definitions inside the source code of the software application.
1.4	**Record description** For each datafile used by the software application there should be a standard method providing its record description by reporting all the fields, which constitute the record of the datafile.
1.5	**Relation information** This mechanism should provide hints about relations, which may possibly exist between the datafiles of the software application.

Table 3.2 – Control Flow Analysis Specifications

2. Control flow analysis	
2.1	**The complete set of programs** The complete set of programs of the software application will have to be reported. The software system, which will be developed in order to implement the aims of this software reengineering mechanism, should be capable of giving a report including all the programs that the software application uses.
2.2	**Name attributes** Each program has two names. The one is an internal name, which can be defined in its source code. The other is the external name of its source file. For each program both names should be reported.
2.3	**Control flow attributes** Each program may call other programs at run time. For each program, the complete set of programs, which may be called during run time, should be reported.
2.4	**Database access attributes** Each program may affect several datafiles in the form of adding or deleting records in the datafile, or changing the contents of some records. For each program, all the datafiles, which are affected by the program, should be reported.

2.5	**Datafile definition attributes** Each program may include several datafile definitions in its source code. For each program, all the datafiles defined in its source code should be reported.

Table 3.3 – Source Code Organization Analysis Specifications

3. Source code organization analysis	
3.1	**Source file location** This mechanism should be able to locate all the source files of the application, which may exist in a directory tree, below a given top-level directory.
3.2	**Source file report** A report providing all the source code files used by the application should be available.
3.3	**Program-level source file report** For each program all the source files (libraries) that the compiler uses in order to built it should be reported.

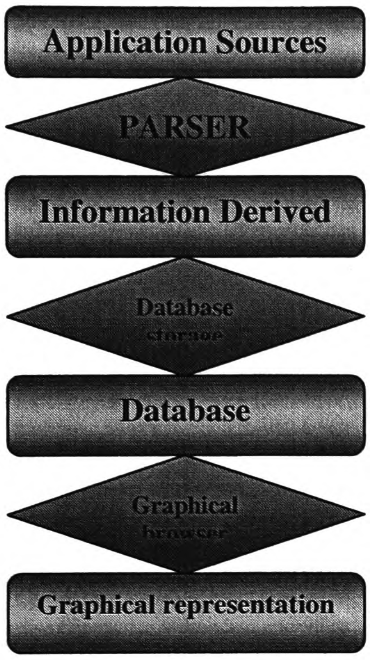
Chapter 4

Design and Implementation Analysis of the SIS Setup

4.1 Introduction

The formal mechanism for analysis and re-implementation of legacy programs, which is presented in this thesis, is divided in three main parts as illustrated in **Figure 4.1**. In the first part, the source code of the legacy software application is scanned line by line in search of useful *pieces of information*. These pieces of information help in the recovery of the implementation and design analysis of the legacy software application. The pieces of information, which have been gathered from the source code of the legacy software application, are stored in a text file in a form of transactions. In the second part of this mechanism the pieces of information, which have been derived from the scanning of the source code of the legacy application, are stored in a database. A special database schema will be developed using the TELOS database [11] in order to store, maintain and provide these pieces of information. Their storage into the TELOS database is implemented by executing the transactions, which are stored in a text file after the end of the first part. The pieces of information are stored in a specially configured TELOS database instance in the form of objects and object attributes. The object attributes define relations between them creating a semantic network. Navigation through this semantic network provides powerful information regarding the implementation and design analysis of the legacy application. In the third part of this mechanism, the information, which has been stored in the TELOS database, is presented using a GUI. The presentation of the information will be implemented using the SIS GAIN browser [12]. This software tool will be configured to commit special queries to the TELOS database and present the results on the screen. The SIS GAIN browser has a GUI with navigation facilities. Once an object is shown on the screen the user can easily select it and see all the other objects, which are related to it. Then he can select one of the related objects and so on.

This formal mechanism can be used to analyze software applications regardless of the computer language used for their development. Nevertheless, for each computer language, integration is necessary in order to setup the tools that will scan the source code and maintain the information derived (parser and database). In this thesis a software reengineering mechanism will be developed, which will be capable to analyze legacy software applications developed using RM/COBOL – 85.



1. The COBOL parser scans the source code of the application and extracts useful pieces of information about the application

2. The information that the parser derived from the sources is stored in a database in order to be effectively browsed and easily accessed.

3. A graphical browser presents the information stored in the database using a user-friendly Graphical User Interface.

Figure 4.1 – A formal mechanism for analysis and re-implementation of legacy programs

4.2 Presentation of tools

A software tool that has been developed by the Foundation of Research and Technology – Hellas will be used in order to manipulate the information concerning the legacy application. This software tool is the Semantic Index System (SIS) and it is constituted by three relative software systems. These software systems are: a database engine called *TELOS*, a query execution mechanism called *Query Interpreter (QI)* and a graphical browser called *Graphical Analysis Interface (GAIN)*.

A software repository is implemented using a special TELOS design. QI and GAIN are used to retrieve and provide the information stored in the repository. TELOS belongs in the family of entity-relationship (E-R) models and it is designed specifically for information system development applications [11]. There are many implementations of the E-R model and the IBM Repository Manager/MVS [19] and PCTE+OMS [3] are two examples of them. There are many reasons why TELOS was chosen among those implementations. First is its treatment to the attributes and to metaclasses, which makes it more expressive and extensible. Second is its simple and elegant formal semantics. Using these semantics TELOS specifies data structures and abstraction mechanisms in terms of a deductive relational database constituted by only a few basic system facts, deduction rules and integrity constraints. This simplicity offers advantages over existing object oriented DBMSs especially when designing multiple related query interfaces such as QI and GAIN. Third is the existence of GAIN and QI. GAIN is a powerful hypertext engine and in combination with QI can execute queries in the database and provide the results on the screen. With this hypertext engine and QI navigation through the objects of the database is possible and thus there can be several approaches in order to design and implement the presentation of the information, which is stored in the software repository.

4.2.1 Description of TELOS

TELOS is a knowledge representation language. Its framework is object oriented. TELOS objects are grouped into *individuals* (entities, concepts, nodes) and *attributes* (relationships, attribute links). It provides three structuring principles, namely the *classification* (inverse instantiation) *specialization* (inverse generalization) and the *aggregation* (inverse decomposition). It does not distinguish between schema and data. Schema changes can be performed without loss of data at any time by simple data-entry statements.

All data in TELOS are grouped into classes. Classes are sets of instances (objects). TELOS has some built in classes that the user cannot change (delete them or change their attributes). These built in classes can only be related by user-defined attributes. Some of them can be directly instantiated while others cannot. **Table 4.1** provides a list of the built in classes of TELOS.

Table 4.1 – Built in TELOS Classes

Classes that cannot be directly instantiated by the user	Object, Individual, Attribute, Class, IndividualClass, AttributeClass
Classes that can be directly instantiated by the user	Token, S_Class, Mn_Class (n=1,2,...), Telos_Integer, Telos_Real, Telos_String, Telos_Time

As every *object* in TELOS, *classes* must have a unique name as identifier. Their instances may be classes again. Instances of a class must be declared explicitly. There is no automatic classification. Classes may have no instances. All the user-defined *classes* are instances of *class*.

All data a user can enter are regarded as instances of the *object* class. Each *object* has a unique logical name as identifier. *Objects* are distinguished into *individuals* and *attributes*. It is not allowed to define *classes*, which mix *individuals* and *attributes*. As a result, *class* is partitioned into *IndividualClass* and *AttributeClass*. The *individual* class includes all the *objects*, which correspond to real things or sets of things or sets of sets of things etc. The *attribute* class includes all the *objects*, which correspond to the relations among *objects* or sets of relations or sets of sets of relations etc. Any *object* must be an *individual* or an *attribute*. One *individual* together with a set of *attributes* and the related *objects* constitute a structured *object*. **Table 4.2** presents the hierarchy of the built-in TELOS *classes*, which cannot be directly instantiated by the user. These *classes* are usually called hereto-defined *classes*.

Table 4.2 – The hierarchy of the hereto-defined TELOS *classes*.

Telos_Class	**isA**	*Telos_Object*
IndividualClass	**isA**	*Telos_Class*
IndividualClass	**isA**	*Individual*
AttributeClass	**isA**	*Telos_Class*
AttributeClass	**isA**	*Attribute*

Token is defined as the *class*, which includes all the "simple" *individuals* or *attributes* (i.e. those that are not *classes*). These "simple" *objects* are said to have the "token instantiation level". Simple *classes* are those *classes*, which have exclusively *tokens* as instances. These *classes* are said to have the "simple instantiation level" and they are all instances of S_*Class*. *Classes*, which have exclusively "simple" *classes* as instances, are metaclasses. They are all instances of *M1_Class* and they are said to have the "metaclass instantiation level". This scheme is deliberately continued and forms the instantiation hierarchy in TELOS. These "level *classes*" constitute a partitioning of the *objects* orthogonal to the *individual-attribute* partitioning. A user can directly instantiate the intersections of one level *class* with either *individual* or *attribute*, together with some pre-defined *classes* and nothing else. Finally, there are four built-in *individual* simple *classes* for primitive values. These are *Telos_Integer, TelosReal, TelosString and TelosTime.* The values of the *individuals*, which belong in those *classes* cannot be created or deleted (an attribute is understood as a relation to an object and not the object itself).

Every object in TELOS has some single and necessary values and two sets. **Table 4.3** shows these properties for each individual object, while **Table 4.4** shows these properties for each attribute object. All the properties that are mentioned in **Tables 4.3** and **4.4** must have a unique value except from the IN_set and ISA_set of the attribute objects.

Table 4.3 – Individual object properties

Value properties of each TELOS individual object	
SYSID	Internal identifier
Sys_name	Logical name
Sys_class	Built-in class it is instance of
Set properties of each TELOS individual object	
IN_set	User defined classes it is an instance of
ISA_set	User defined superclasses

Table 4.4 – Attribute object properties

Value properties of each TELOS attribute object	
SYSID	Internal identifier
Sys_name	Logical name
Sys_class	Built-in class it is an instance of
Sys_from	The relating object
Sys_to	The related object
Set properties of each TELOS attribute object	
IN_set	User defined classes it is an instance of
ISA_set	User defined superclasses

All properties in **Tables 4.3** and **4.4** are implemented by direct bi-directional linkage. In a semantic network, such as the one that TELOS implements, there is no preference of query direction. Furthermore, the IN_set and ISA_set properties are used for the implementation of multiple instantiation, which is useful for classification purposes. Finally, it should be mentioned that TELOS does not support ISA relations at token level.

The logical name of an attribute is also called the label of the attribute and the classes, which an attribute is instance of, are also called the categories of the attribute.

An attribute class relates two classes, Sys_from and Sys_to. All instances of that attribute class must relate objects, which are instances of the Sys_from class, to objects, which are instances of the Sys_to class. Finally, it should be mentioned that an attribute object might relate objects, which are instances of different class level. For example, object p, which is an instance of a S_Class, can be related to object q, which is an instance of a M1_Class.

ISA relations implement the *specialization* structuring principle in TELOS. The user must declare explicitly every ISA relation in the semantic network. ISA relations assume a subset relationship between the corresponding classes and they must not be cyclic. Any two classes related by an ISA relation must belong to the same instantiation level. Inheritance occurs automatically with ISA relations. For example, if a class P is a specialization of class Q and Q is a specialization of class R then P is also a specialization of class R.

TELOS uses two statements to manipulate objects into the database. These are the TELL and the RETELL statements. The TELL statement provides the ability to enter objects or classes of objects and also to define hierarchy relations among them. TELOS provides two alternative ways for associating objects with attributes: a) explicit definition using the Attribute declaration and b) implicit definition within the individual declaration. Explicit definition allows defining all possible fields of an attribute, while implicit definition assigns the individual declared explicitly as Sys_from to all implicitly defined attributes. ISA relations and attributes of attributes cannot be declared implicitly.

The RETELL statement updates the attributes of an object inside the database. **Table 4.5** shows the features of the RETELL statement. In addition, there are two formats of the RETELL statement. The first, which is also simple, updates an object that already exists in the database. The second format of the RETELL statement is closer to that of the TELL statement and is used in order to update an object that does not necessarily exist in the database. For this reason the second format of the RETELL statement redefines all the properties of the object and not only the possible changes. Update operations can be additions, deletions or changes. However, changes are internally implemented in the form of deletions followed by the proper additions.

Table 4.5 – Features of the RETELL statement

1.	Syntax of RETELL is as close as possible to that of the TELL statement.
2.	The RETELL statement might redefine anything that the TELL statement might define except the assignment of an object to the built-in system class (Sys_class).
3.	No redundant information is required. RETELL needs only information to identify the object or the set of objects to be changed added or deleted and the corresponding action to be taken.
4.	Apply several changes to an object within one RETELL statement.
5.	More than one RETELL statement for the same object might exist in the same transaction. In that case the updates are executed in the order they are found inside the transaction.

4.2.2 Description of Query Interpreter

Query Interpreter (QI) is a program that uses the Programmatic Query Interface (PQI) to access a TELOS database. The PQI is a set of functions that can be used to express queries in a TELOS database. Using QI the users can execute queries to the TELOS database. Although QI has its own user interface to enter queries and then execute them to the database, its input can be redirected. Using this mechanism, query macros can be edited in text files and then passed to QI for execution. QI can also be used inside programs as an interpreter for batch execution of queries. Finally, it should be mentioned that the GAIN browser uses QI not only for batch execution of queries, but as a configuration mechanism for its parameters as well.

An existing object in the TELOS database should be set as the *current* object each time QI starts. The *scn* (set current node) PQI command is used for that purpose. Every query command is always applied on the *current* object. The answer of a query command is stored to the *current set* of objects. It is also possible to have a query command applied on every object existing in the *current set* of objects, which has been created by a previous query command. The answer to a query can be viewed by projecting the contents of the *current set* using special PQl projection commands.

Navigation through the objects, which exist in the TELOS database, is possible by executing queries to the *current object* or to the *current set of objects*. The results of the queries are stored in the *current set of objects* so the next queries might be applied on these objects and so on. This is a technique that will be used by the software reengineering mechanism, in order to navigate through the programs of a software application and view their properties.

4.2.3 Description of Graphical Analysis Interface

The combination of the GAIN browser with QI provides a mechanism to retrieve information from the SIS base and present it on the screen in two ways: graphically or textually. The graphical presentation is implemented using the window of the graphical subsystem of the GAIN browser, while the textual presentation is implemented using a simple text screen.

Initially, when the GAIN browser is executed the window shown in **Figure 4.2** appears on the screen. The window is divided into the following areas: a) The *menu bar*, b) The *query info area*, c) The *query results area*, and d) The *output control area*.

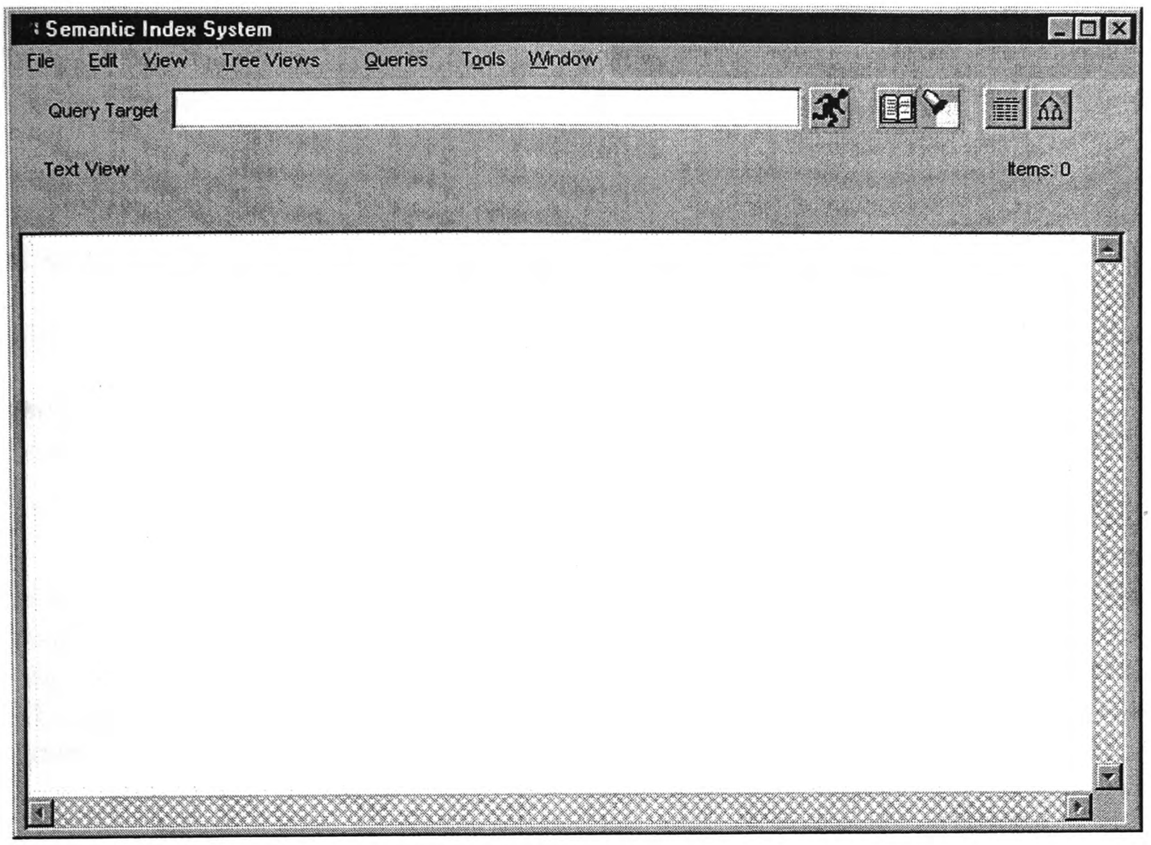


Figure 4.2 – Initial window of the SIS GAIN browser.

The standard *menu bar* of the GAIN browser consists of seven drop-down menus. The user can add menus or items to the standard menus of the menu bar by creating special objects in the TELOS database that the GAIN browser connects to, when it initializes. **Table 4.6** describes the standard drop-down menus of the menu bar of the GAIN browser.

Table 4.6 – The standard drop-down menus of the menu bar of the GAIN browser

File	Includes the options that are used for file and program operations.
Edit	Includes the options offering text operations in the query result area.
View	Includes the options, which control the appearance of the user interface and also settings, which affect the graph and text output.
Tree Views	Includes the options that are used for controlling the display mode and executing predefined recursive queries, which are displayed in graphical mode.
Queries	Includes the options that are used for executing predefined queries, which are displayed in textual mode.
Tools	Includes options that are used for communication with external tools.
Window	Includes the options that give all the available output representations.

The *query info* area consists of six sub-areas: a) The query target text editor, b) The exec button, c) The history button, d) The find button, e) The query type label, and f) The items label. **Table 4.7** describes the six sub-areas of the query info area of the GAIN browser's initial window.

Table 4.7 – The six sub-areas of the query info area

Query text editor	It is a single line text editor. It is used to display the object to which a query is applied on.
Exec button	It is used for executing the last graphical query type selection on the object pointed by the query text editor.
History button	It displays a popup frame, which is used for retrieving previous query commands.
Find button	It displays the Pattern Search Card, which is used to search a pattern in the textual or graphical window.
Query type label	It displays the current query type (textual or graphical).
Items label	It displays the number of objects included in the answer set of the query.

The *query results* area is used to display the objects included in the answer set of the query. It can be scrolled left-right or up-down. The results of a textual query are displayed using a scrollable text window, while the results of a graphical query are displayed using a graphical subsystem, which is responsible for drawing the graphs. The query results area can be toggled between the two display modes (textual and graphical) without loosing its contents.

Finally, the *output control* area consists of just two buttons: Text and Graph. The Text button is used to switch the display mode to text mode, while the Graph button is used to switch the display mode to graph mode.

Initially, the GAIN browser connects automatically to a TELOS database. It reads special environment variables, which instruct the GAIN browser which database it should connect to and how to implement the connection. After it successfully connects to the database, it reads from it special configuration parameters. Specific database objects are used to configure the name and number of the menus in the menu bar. The same mechanism is used to configure the names of the additional graphical or textual queries in the corresponding menus and also describe these queries. It is also possible to give a special name to the window of the GAIN browser using this mechanism.

4.3 Design analysis

The design of the SIS setup, which will serve the needs of the proposed software reengineering mechanism, is divided in two sections namely: a) *GAIN browser setup design* and b) *Database instance design*. Since all the information concerning the design and implementation of a software application is presented using the GAIN browser, the design of the GAIN browser setup will occur first, aiming to cover the specifications of the reengineering mechanism. The database instance design follows, aiming to cover the demands of the GAIN browser setup design.

4.3.1 GAIN browser setup design

The GAIN browser includes two separate menus in its menu bar concerning the database queries. These menus are the *Queries* menu and the *Tree Views* menu. The Queries menu includes four text-view queries, as illustrated in **Figure 4.3** that provide special sets of objects, while the Tree Views menu includes ten graphical-view queries, as illustrated in **Figure 4.4** that can only be applied on the *current* object.

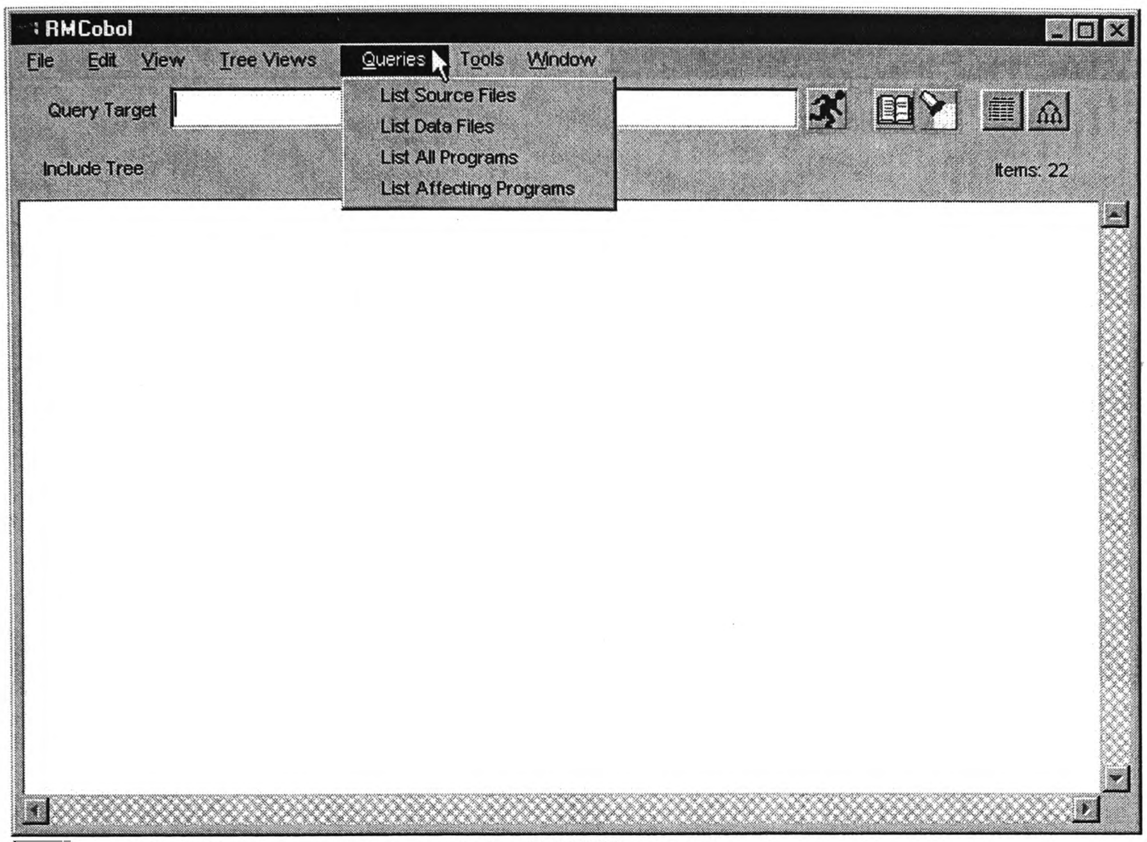


Figure 4.3 – The queries menu of the GAIN browser

The *Queries* menu of the GAIN browser is shown in **Figure 4.3**. This menu includes four text-view queries. The first query of the Queries menu is named *List Source Files* and corresponds to the *Source Code Organization Analysis* section of specifications. It lists all the text files, which include the source code of the software application. The second query is named *List Data Files* and corresponds to the *Database Analysis* section of specifications. It lists all the files that the software application uses to store its data. This query uses the internal name that the application uses in order to reference the datafile in order to represent the datafile in the report. The third query is named *List All Programs* and it lists all the programs of the software application. Beside each program, in a second column, appears the name of the source file that the compiler is instructed to read first, in order to build the program, which is also called the *main* source file of the program. The fourth query is named *List Affecting Programs* and lists all the programs of the software application, which affect at least one datafile and beside each program, in a second column, appears the name of its main source file. The third and fourth queries correspond to the *Control Flow Analysis* section of the specifications (see **Chapter** 3 - Requirements and Specifications Analysis).

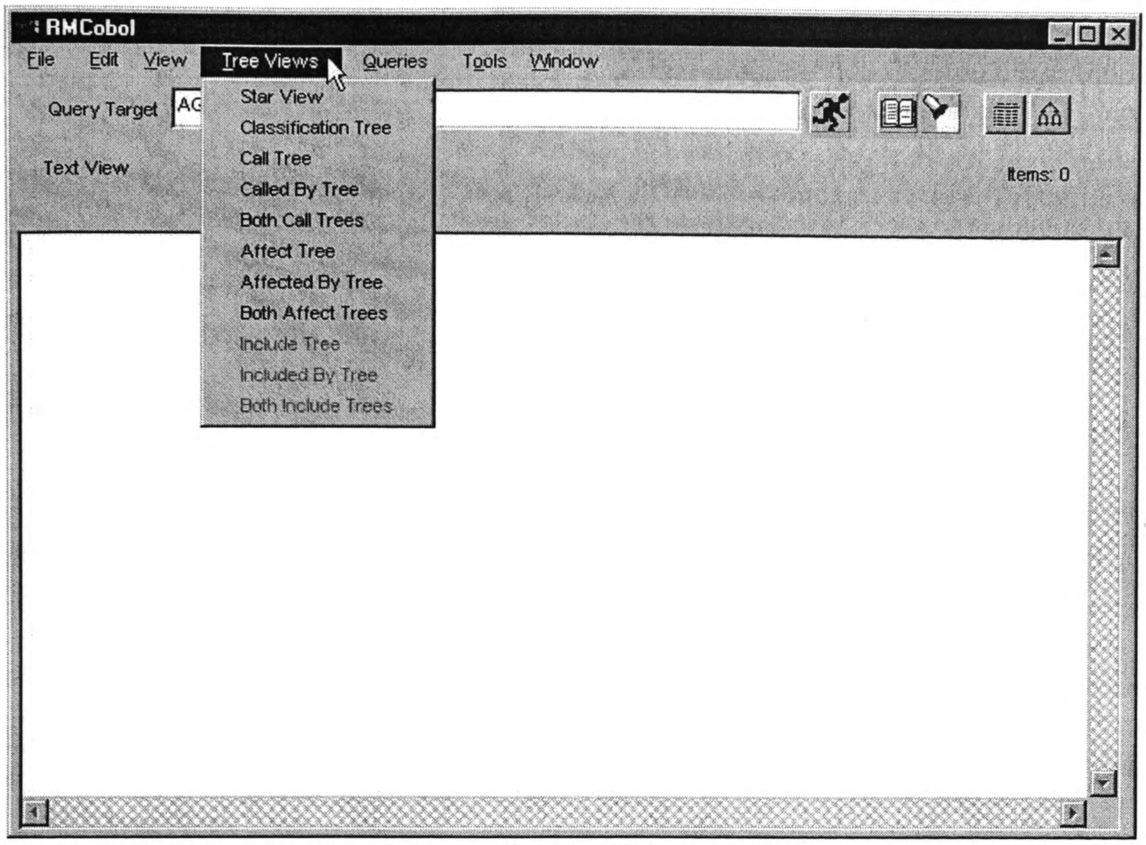


Figure 4.4 – The tree views menu of the GAIN browser

The *Tree Views* menu of the GAIN browser is shown in **Figure 4.4**. This menu includes ten graphical-view queries. Since these queries can only be applied on the current object, the user must edit the name of an existing database object before executing one of these queries. There are two ways to edit the name of an object in the query text editor. The first is to simply type it, after clicking with the mouse in the query text editor area of the GAIN browser. The second is to execute one of the text-view queries and after the objects included in the result set of the query appear on the screen, simply select one of those objects by clicking on it using the mouse.

The first query of the Tree Views menu is named *Star View*. This query presents the current object and all the objects, which are related to it under any of its attributes. The second query is named *Call Tree*. This query presents the current object, which must be a COBOL program and all the COBOL programs, which are called by this program. It is a recursive query and thus it presents all the COBOL programs, which are called by the programs, which are called by this program and so on. The third query is named *Called By Tree*. This query presents the current object, which must be a COBOL program and all the COBOL programs, which call this program. It is a recursive query and thus it presents all the COBOL programs, which call the programs, which call this program and so on. The fourth query is named *Both Call Trees* and it provides the information that both the two previous queries provide for the current object, which must also be a COBOL program. The fifth query is named *Affect Tree*. This query provides the same information with the Call Tree query and additionally for each COBOL program presents all the datafiles, which are accessed by the program. The sixth query is named *Affected By Tree*. This query provides the same information with the Called By Tree query and additionally for each COBOL program presents all the datafiles, which are accessed by the program. The seventh query is named *Both Affect Trees*. This query provides the same information with the Both Call Trees query and additionally for each COBOL program presents all the datafiles, which are accessed by the program. The eighth query is named *Include Tree.* This query presents the current object, which must be a source file and all the source files, which the compiler is instructed to also read (include) while reading this source file. It is a recursive query and thus it presents all the source files, which are included by the source files, which are included by this source file and so on. The ninth query is named *Included By Tree.* This query presents the current object, which must be a source file and all the source files, which include this source file. It is a recursive query and thus it presents all the source files, which include the source files, which include this source file and so on. The tenth and final query of the Tree Views menu is named *Both Include Trees.* This query provides the information that both the two previous queries provide for the current object, which must also be a source file.

Figure 4.5 shows an example of the graph, which is generated by the *Star View* graphical query, applied on an individual object. The *current* object is represented by a color box, which includes the name of the object. The *attribute* objects, which are related to the *current* object, are also represented with a color box and an arrow, starting from the *current* object and pointing to them. An *attribute* object is always used in order to relate two *individual* objects. Thus, the *individual* objects, which are related with the *current* object, are represented with a color box and an arrow starting from the *attribute* object that implements the relation and pointing to them.

In the special case that there are too many *individual* objects related under the same *attribute* object to the *current* object, at the end of the arrow appears only one color box including the word *many* and the name of the *attribute* object. Once this box is clicked with the mouse, a separate window, which presents all the *individual* objects related to the *current* object under this *attribute* object, appears on the screen. This window is called *Many List Card.*

Once the user makes a click with the mouse on an *individual* object appearing on the screen, this object becomes the *current* object and its name appears in the query text editor. If the user makes a double-click on an *individual* object, this object becomes the *current* object, appears in the query text editor and the last executed graphical query is automatically executed again for this object.

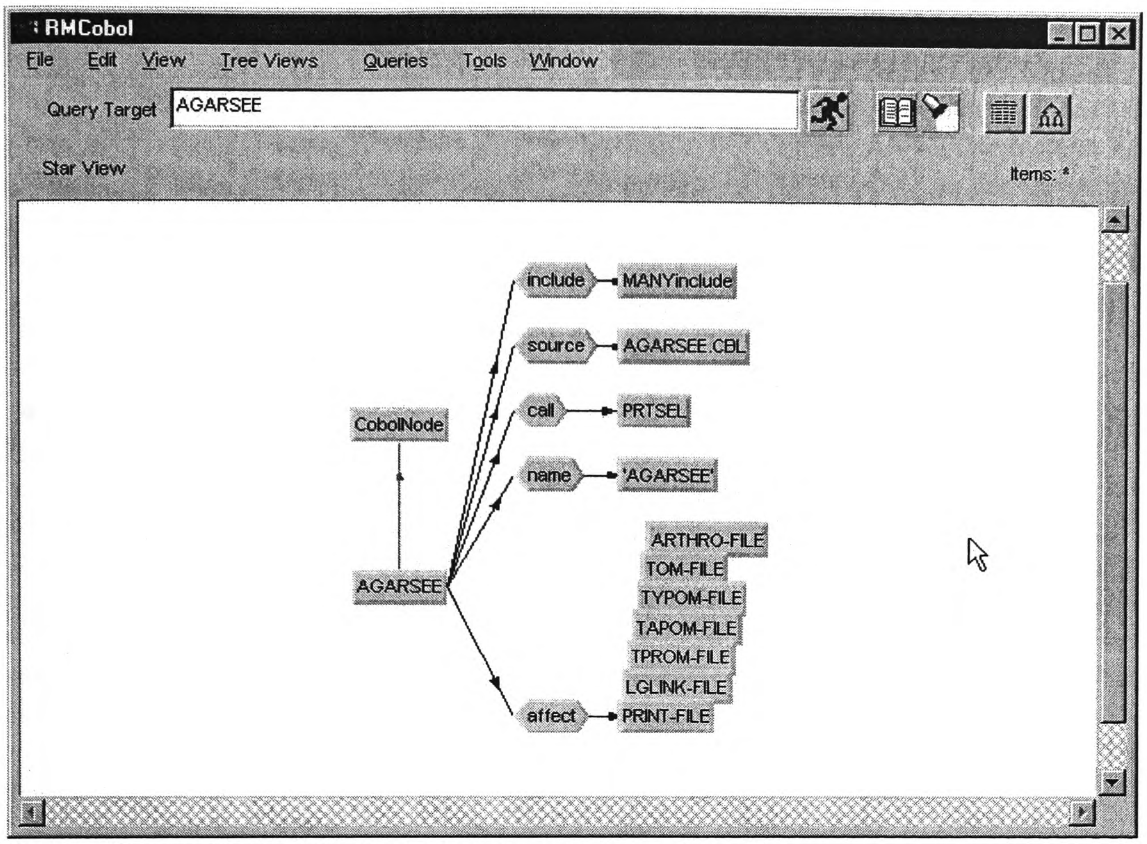


Figure 4.5 – A "Star View" graphical query example.

4.3.2 Database instance design

All the pieces of information extracted from the legacy application source code are stored into the TELOS database in the form of objects. In this section the design of a special model, which has been developed aiming to maintain and represent the analysis of a COBOL software application, is presented. This model has the form of a specially configured TELOS database instance. *Individual* objects represent the components of the software application, such as source files and programs and *attribute* objects represent their properties such as calls between programs and database affects. **Table 4.8** presents the classes of individual objects that have been defined in the database. For each individual object, all the attribute objects, which constitute its properties, are presented. **Figure 4.6** provides a graphical representation of those classes and **Figures 4.7 – 4.11** show a representation of those objects provided by the SIS.

All the software objects of the software application are grouped in to five classes namely *NodeType, SourceType, CobolData, CobolNode* and *SourceFile.* NodeType is a superclass of CobolNode and SourceType is a superclass of SourceFile thus, CobolNode inherits the attributes of NodeType and SourceFile inherits the attributes of SourceType.

NodeType is an instance of the M1_Class built-in TELOS class and also a superclass of the CobolNode class. Since it is possible to use more than one computer language to implement a software application, the various programs of the application should be categorized in more than one class, depending on the computer language used to implement them. NodeType is a superclass of all the classes of programs of the software application. NodeType has an attribute called *Node_ref*, which is a cyclic relation to itself. This attribute is inherited to all the subclasses of NodeType.

Node_ref is used as a category of all the attributes that relate a program of the software application with another program.

SourceType is an instance of the M1_Class built-in TELOS class and also a super class of the SourceFile class. Since many different computer languages may be used in the development of a software application, as mentioned in the previous paragraph, the various source files of the application should be categorized in more than one class, depending on the computer language source code that they include. SourceType contains all the classes of source files of the software application. SourceType has an attribute called *Source_ref*, which is a cyclic relation to itself. This attribute is inherited to all the subclasses of SourceType. Source_ref is used as a category of all the attributes that relate a source file of the software application with another source file.

CobolData is an instance of the S_Class built-in TELOS class and contains all the data files of the software application. The internal names that the software application uses in order to access those data files are used in order to represent those data files in the CobolData class. CobolData has two attributes namely *affect* and *select*. Affect is used to relate the data file with all the programs of the software application that access this data file. Select is used to relate the data file with all those programs that define the structure of the data file in their main source file.

CobolNode is an instance of the S_Class built-in TELOS class and of the NodeType class. It contains all the COBOL programs of the software application. The internal names that the software application uses in order to refer to those programs are used in order to represent them in the CobolNode class. CobolNode has six attributes namely *affect*, *call*, *include*, *name*, *select* and *source*. The affect attribute relates each program with all the data files that it accesses. The call attribute relates each program with all the programs of the software application that it invokes at run time. The include attribute relates each program with all the source files, except the main source file of the program, that are used by the compiler in order to build the program (COBOL COPY statements). The name attribute is the internal name that COBOL demands to be defined for each program (COBOL Program Identification Entry). The select attribute is used to relate each program with all the data files that have their structure defined in the main source file of the program (COBOL SELECT statements). Finally, the source attribute is used in order to relate each program with its main source file.

SourceFile is an instance of the S_Class built-in TELOS class and of the SourceType class. It contains all the source files of the software application, which are regarded to be COBOL source files. The name of the source file that is used in order to store the source file in the directory tree of the source code of the software application is used to represent the source file in the SourceFile class. The SourceFile class has three attributes namely *contain*, *source* and *include*. The contain attribute is used to relate each source file with all the source files that the compiler is instructed to read additionally while reading this source file and building a program. The source attribute is used to relate each main source file of the software application with the COBOL program that is built by it. The include attribute is used to relate each source file with all the programs that use the source file in addition to their main source file.

Table 4.8 – TELOS individual objects and their attributes

Object Name	TELOS SuperClasses	Object Attributes
NodeType	M1_Class	Node_ref (NodeType)
SourceType	M1_Class	Source_ref (SourceType)
CobolData	S_Class	affect (CobolNode), select (CobolNode)
CobolNode	S_Class, NodeType	name (Telos_String), source (SourceFile), affect (CobolData), select (CobolData), call (CobolNode), include (SourceFile)
SourceFile	S_Class, SourceType	contain (SourceFile)

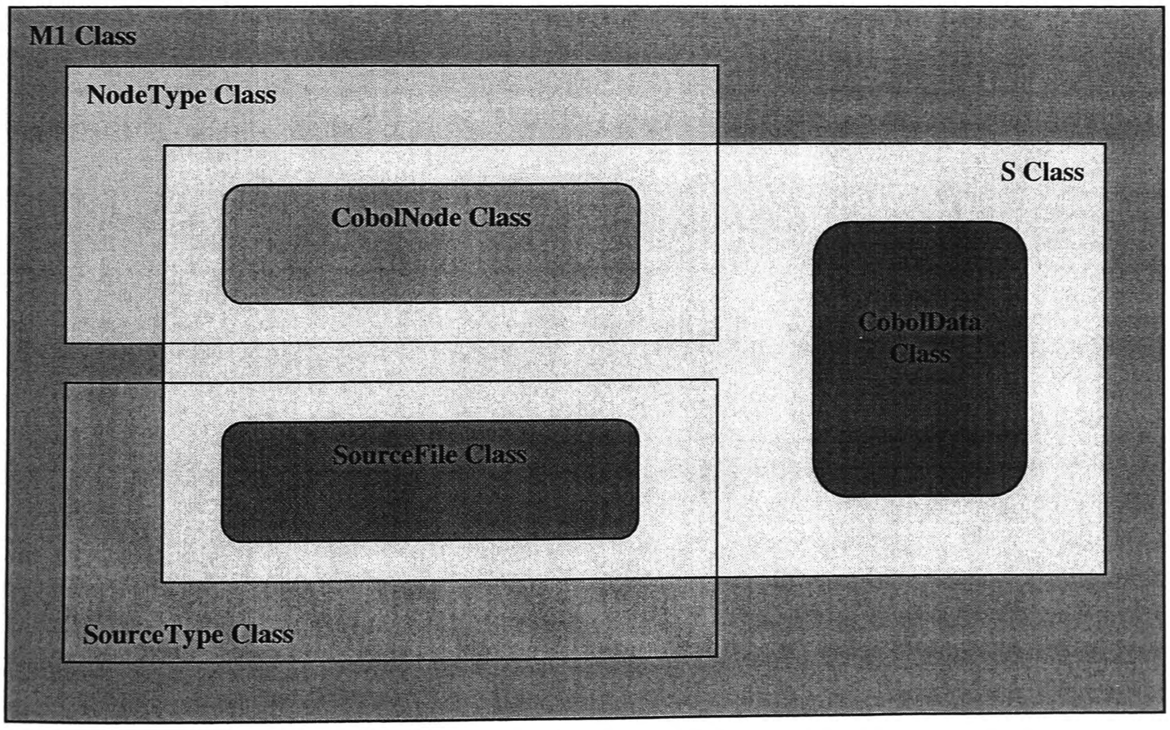


Figure 4.6 – A graphical representation of the TELOS database instance design

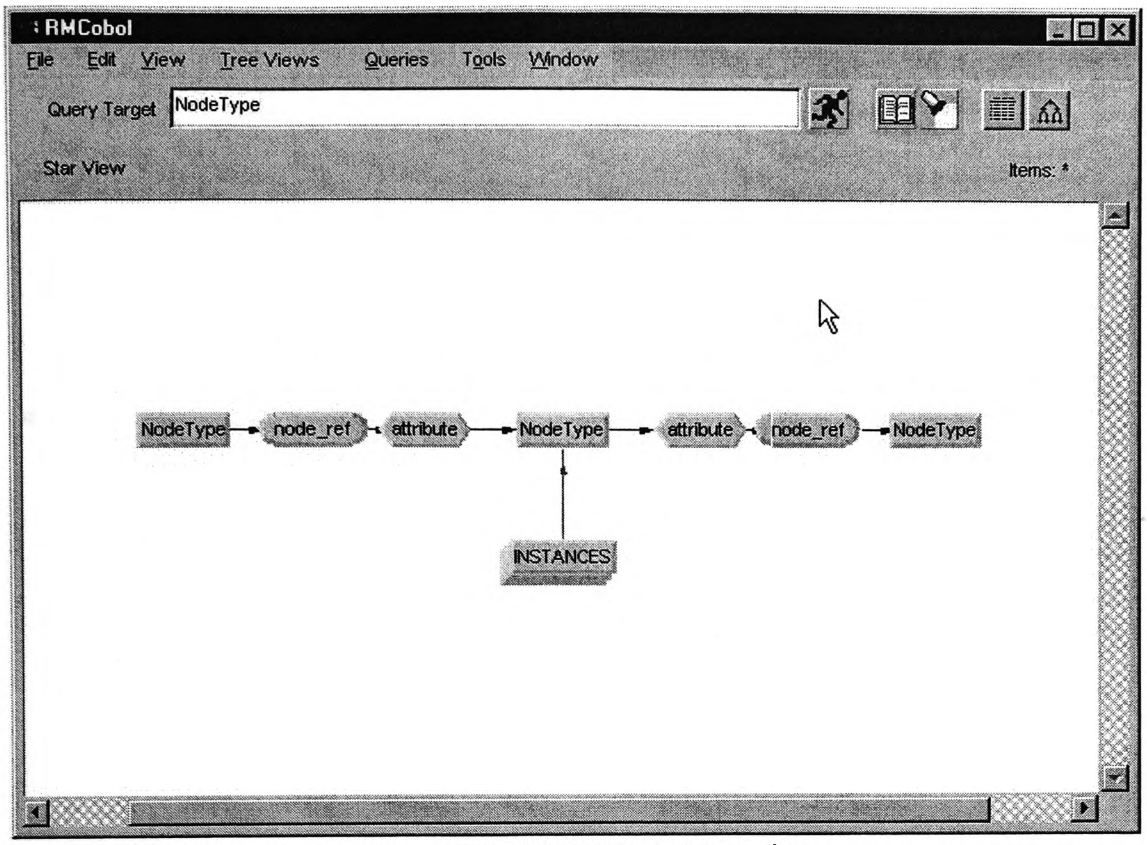


Figure 4.7 – SIS representation of the NodeType object class.

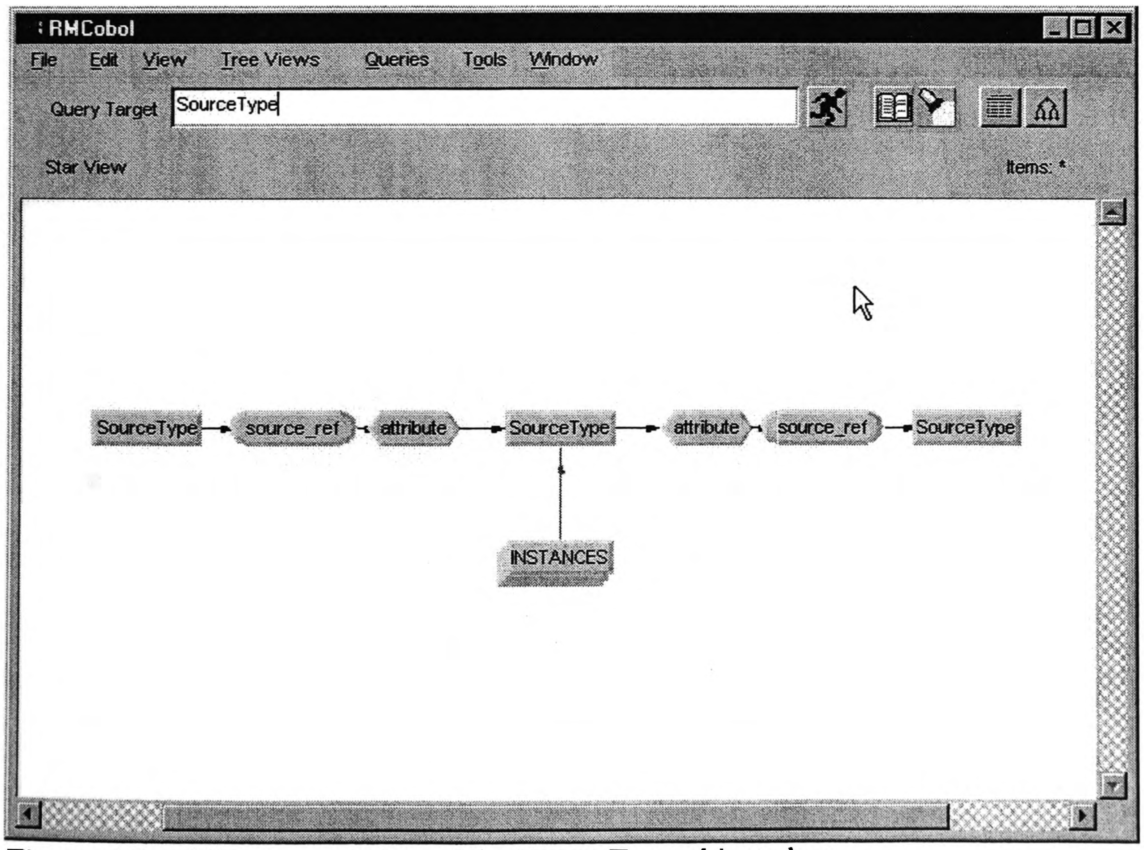


Figure 4.8 – SIS representation of the SourceType object class.

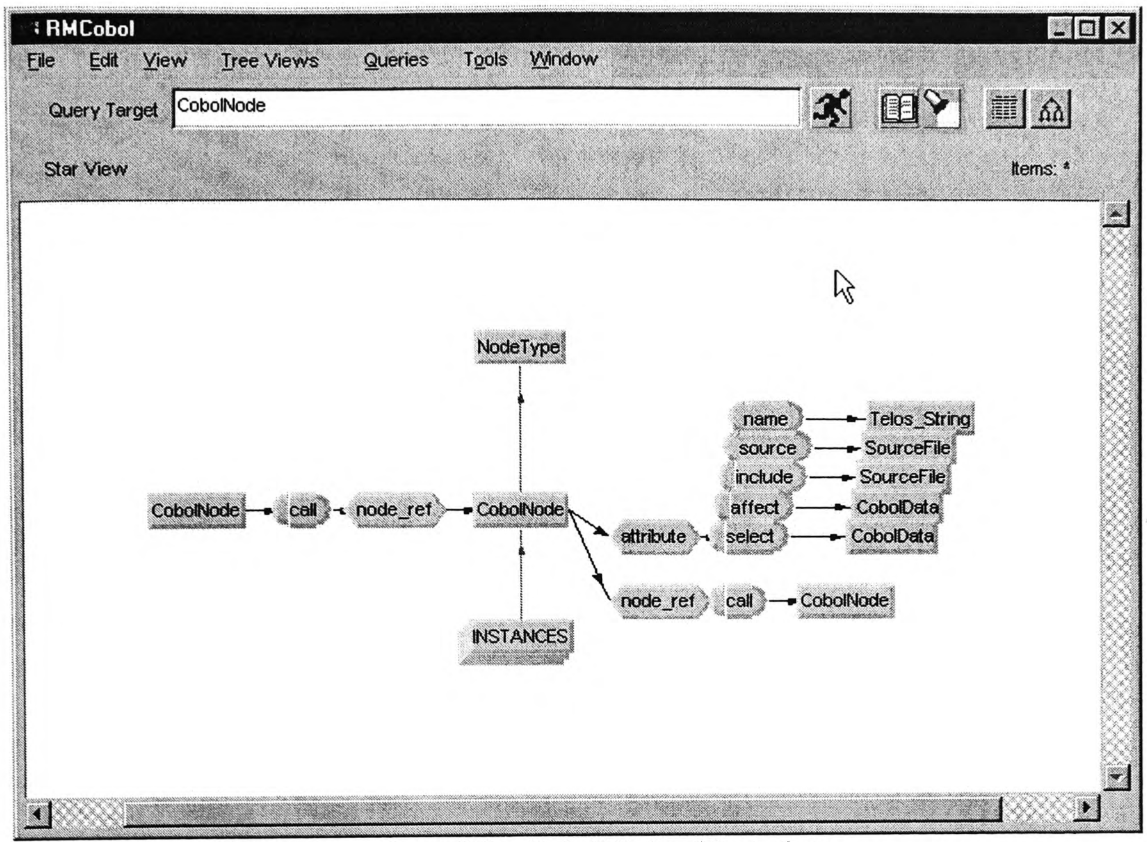


Figure 4.9 – SIS representation of the CobolNode object class.

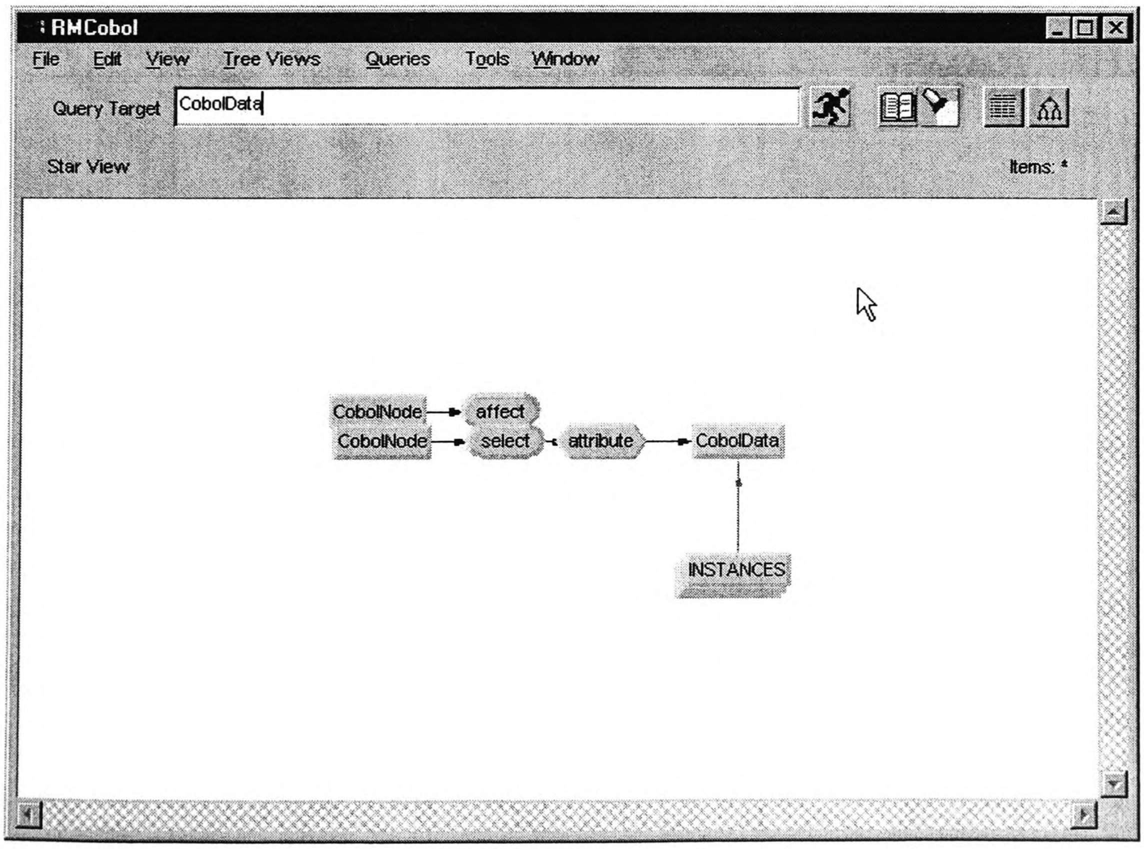


Figure 4.10 – SIS representation of the CobolData object class.

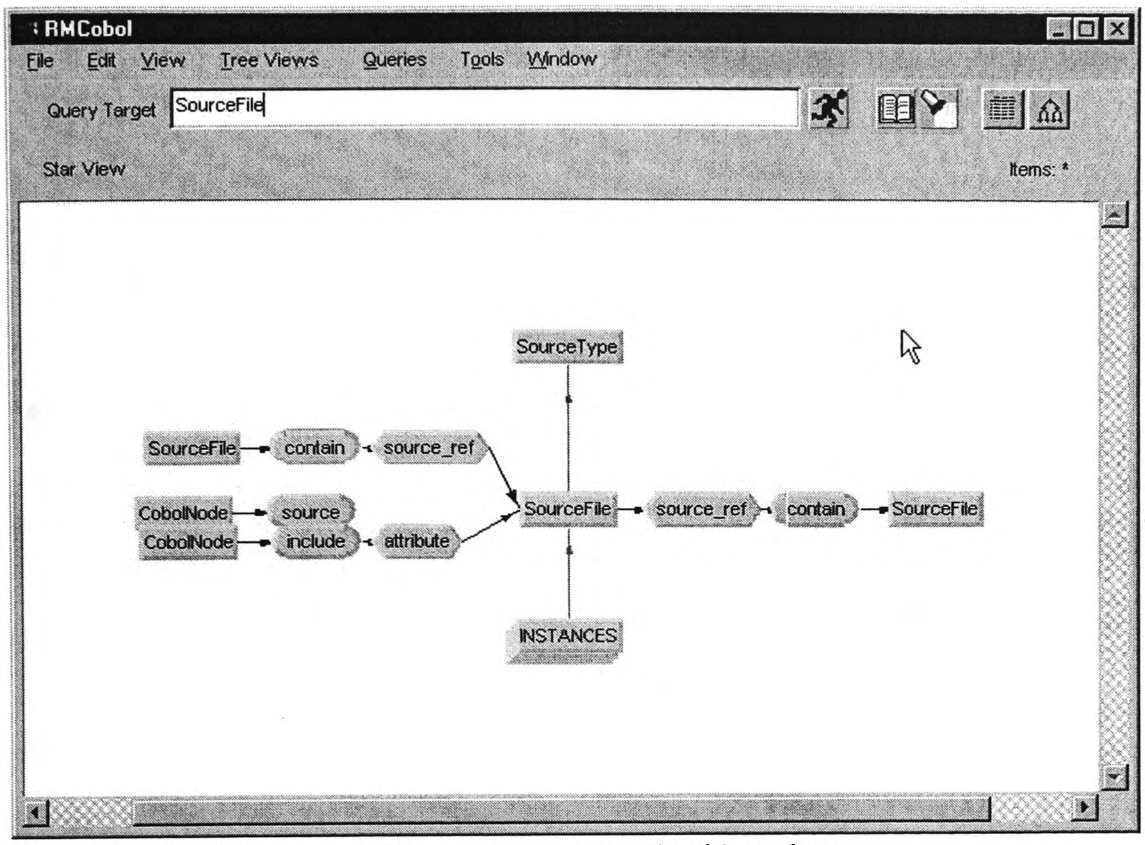


Figure 4.11 – SIS representation of the SourceFile object class.

4.4. Implementation Analysis

The implementation of both GAIN browser setup and database instance structure is achieved by creating special objects and object attributes inside a TELOS database instance. The source code of the transactions that create these objects is stored in text files. TELOS reads those text files, commits the transactions and thus, creates the objects in the database instance.

4.4.1 GAIN browser setup implementation

There are several objects that the GAIN browser must find in the TELOS database when it initializes in order to setup its environment. It is outside the scope of this document to describe all these objects and how they are created in the database except from the Tree Views menu, the Queries menu and their contents. The tree views menu includes ten queries namely *Star View*, *Call Tree*, *Called By Tree*, *Both Call Trees*, *Affect Tree*, *Affected By Tree*, *Both Affect Trees*, *Include Tree*, *Included By Tree* and *Both Include Trees*. **Appendix 1** presents the transactions that define the Tree Views menu, the queries in the Tree Views menu, the Queries menu and the queries in the Queries menu into the database instance. **Appendix 2** presents the source code of the queries included in the Queries menu and **Appendix 3** presents the source code of the queries included in the Tree Views menu.

Whenever a query is selected for execution, the GAIN browser passes its source code and if necessary the current object to QI. QI executes the specified query and returns a set of objects to the GAIN browser. If the executed query is a graphical-view query then the objects included in the resulting set represent software components and attributes, which have the form of relations between the software components of the resulting set. The GAIN browser displays all those objects of the resulting set on the screen using its hypertext engine. If the executed query is a text-view query then the objects included in the resulting set represent only software components and are simply displayed on the screen using a text window.

4.4.2 Database instance implementation

The design of the database instance is implemented by executing a number of TELOS transactions that create the software object classes and their attributes as described in **Table 4.8** inside a new TELOS database instance. The source code of each of those transactions is presented in **Appendix 4**. NodeType and SourceType object classes must be created first inside the TELOS database instance since they are superclasses for the CobolNode and SourceFile object classes. All other object classes may be defined without any special order.

4.4.3 Database instance creation procedure

The procedure followed in order to implement this particular SIS setup is constituted of the seven steps presented in **Table 4.9**.

Table 4.9 – The procedures followed to create the TELOS database.

1.	Set values to environment variables regarding general SIS configuration.
2.	Set values to environment variables regarding special SIS configuration concerning the specific database instance.
3.	Assure that the database directory is empty.
4.	Start the TELOS database.
5.	Execute the transactions that create the objects concerning the GAIN browser setup.
6.	Execute the transactions that create the objects concerning the structure of the database instance.
7.	Execute the transactions that create the objects concerning the queries that the GAIN browser will execute.

In step 1 some environment variables are evaluated to provide general configuration values to SIS such as the name of the server in which SIS is running, the top-level directory of the SIS engine and the directory of the SIS binaries. In step 2 some environment variables are evaluated to provide configuration values to SIS concerning the TCP port that the database server will listen to, the directory of the database files and the name of the application that the database is serving. In step 3 the contents of the directory that includes the database files are erased in order to create new and empty database files when the database engine starts. In step 4 the database server starts executing and initially it creates an empty database. In step 5 a special program, which is called TELOS parser, reads the source files that include the

RETELL statements shown in **Appendices 1-3** and creates the database objects that configure the GAIN browser. In steps 6 and 7 the same procedure if followed in order to create the database objects concerning the structure of the database instance and the queries that the GAIN browser will execute.

A formal mechanism for analysis and re-implementation of legacy programs

Chapter 5

Design and Implementation Analysis of the COBOL Parser

5.1 Introduction

The formal mechanism for analysis and re-implementation of legacy programs, which is presented in this thesis, is divided in three main parts as shown in **Figure 4.1** in **Chapter 4**. The design and implementation analysis of the SIS setup has already been presented in **Chapter 4**. In this chapter the design and implementation analysis of the source code parser is presented. Even though the source code scanning phase belongs in the first part of the software analysis procedure, the design and implementation analysis of the SIS setup is presented first and the design and implementation analysis of the source code parser follows. It is necessary to follow this order because the model, which is used in order to represent the analysis of each software application, has the form of a specially configured TELOS database instance. The source code parser must follow this model and produce the appropriate output that is necessary to store the information derived from the source code into the specially configured TELOS database instance. Thus, the source code parser must be designed and implemented according to the TELOS database instance specifics.

This formal mechanism has been designed and implemented to be capable of analyzing software applications regardless of the computer language used for their development. Nevertheless, for each computer language used, integration is necessary in order to implement the setup of the tools that will scan the source code and maintain the information derived (parser and database instance). In this thesis a software reengineering mechanism will be developed, which will be capable to analyze legacy software applications developed using RM/COBOL–85 and thus the source code parser will also be referred to as COBOL parser.

5.2 Presentation of tools

The COBOL parser has been developed in a Unix operating system environment. The main computer language used in its development is the AWK pattern scanning and processing language (see **Appendix 5**). AWK has been chosen since it is a very powerful filtering language. Its ability in filtering text files, recognizing multiple patterns and formatting the output text, which is the result of the filtering process, are the main reasons for the selection of AWK among other computer languages. The simple and powerful mechanism of the AWK language, regarding the scanning of text files, is used in order to scan the source files of the old software application. The powerful mechanism of the AWK language, regarding the pattern recognition inside text files, is used in order to identify the COBOL statements that include the necessary pieces of information, which will be stored into the TELOS database instance in the form of software objects and attributes. In addition, the powerful mechanism of the AWK language regarding the formatting of text is utilized for the creation of the TELOS statements that constitute the TELOS transaction, which will store the pieces of information of the old software application into the properly designed TELOS database instance. Many other computer languages, such as C++, could be used for the development of the COBOL parser, but the mentioned features of AWK should be implemented as separate routines in that case.

The programs that have not been developed using AWK are script programs that have been developed using the bash (Bourne Again Shell) Unix shell environment and some Unix native tools such as sort and grep.

A shareware distribution of RedHat Linux-6.1 has been used for the development of the COBOL parser. This distribution includes the GNU Project versions of AWK, bash, sort and grep*.

5.3 Design analysis of the COBOL Parser

The COBOL parser is a program that works as a *mini* compiler of the source code, which was used to build the old software application. The COBOL parser is able to read the source code line by line, is able of *understanding* some of the RM/COBOL–85 statements and for each statement can recognize some of its operands. According to the specifications that have been outlined for this software reengineering mechanism a small number of RM/COBOL-85 statements have been identified that include the necessary information concerning the implementation and design of the software application (see **Table 5.2**).

5.3.1 The COBOL Parser segments

All the files that constitute the COBOL parser lie in a separate directory tree. The COBOL parser is constituted of five segments namely: *awk*, *bin*, *log*, *tls* and *tmp*. The awk segment of the COBOL parser contains the AWK pattern scanning and processing language programs. All the programs that are used to scan line by line the source code of the old software application have been implemented using AWK. These programs are not directly executable by the operating system but the AWK run-time system is needed in order to read these source files and execute the programs. The bin segment contains all the executable programs of the COBOL parser. The log segment contains all the log files that the COBOL parser generates at execution time. The tls segment contains all the TELOS source files that are generated while scanning the source code. Finally, the tmp segment contains all the temporary information that the COBOL parser needs while scanning the source code.

There is a separate directory inside the top-level directory of the COBOL parser for each segment. The awk directory contains the text files that include the source code of the AWK programs. The bin directory contains the text files that include the source code of the bash (Bourne Again Shell) script programs of the COBOL parser. The log directory contains the log files that are generated during the scanning procedure of the applications source code. The tls directory contains the

* The GNU Project offers free, open source software. More information can be found in the location http://www.gnu.org

resulting TELOS source files that constitute the TELOS transaction, which includes all the pieces of information gathered by the parser from the source code of the software application. Finally, the tmp directory contains temporary files that the COBOL parser generates while executed. The top-level directory tree can be anywhere in the filesystem. The names of the files included in each directory are presented in **Table 5.1**. **Appendix 6** provides a detailed description of each of the files referred in **Table 5.1**.

Table 5.1 – The contents of the top-level directory of the COBOL parser

Directory name	Files contained
awk	Attributes.awk, CobolData.awk, CobolNode.awk, SourceFile.awk, SourceFileName.awk, checktls.awk, make-dependencies.awk
bin	CreateSisTransaction, find-Attributes, find-CobolData, find-CobolNode, find-SourceFile, loginit, parser, tlsinit, tmpinit
log	CheckDuplicates.log, attributes.log, coboldata.log, cobolnode.log
tls	1-SourceFile.tls, 2-CobolData.tls, 3-CobolNode.tls, 4-Attributes.tls, application-structure.tls
tmp	CobolData.tmp, CobolNode.tmp, SourceFile.tmp, coboldata.tmp, cobolnode.tmp, sourcefile.tmp, srcfilenames.tmp

5.3.2 The COBOL Parser phases

The presentation and analysis of the control flow of a software application always provides critical information about its structure and design. Nevertheless, one of the aims of this software reengineering effort is to provide information regarding the control flow of a software application. **Figure 5.1** provides the control flow of the COBOL parser. It presents the phases of the COBOL parser and the programs that constitute each phase. It also presents the order that these programs are invoked while the COBOL parser is executed. This is a *higher-level* control flow presentation since it does not present the control flow of each of the programs that constitute each phase. All these programs are analyzed in this chapter and a control flow chart is provided, whenever it is necessary. In addition, **Appendix 6** provides a description of each of these programs and can be used as a quick reference.

Figure 5.1 shows that the COBOL parser consists of three phases. These are namely *Initialization*, *Application Scanning*, and *Final Merging* phase. The application scanning phase is the main phase of the COBOL parser and is divided in two sub-phases. These are namely *Filesystem Scanning* and *Source Code Scanning* phase. Although **Figure 5.1** presents all these phases as separate phases, information is exchanged among them using parameter values and temporary files.

During the initialization phase the COBOL parser sets up the environment and the workspace which will be used by the processes that take place in the phases that will follow. This includes information such as: the filesystem location where the COBOL parser has been installed, the filesystem location where the source code of the application, which is about to be scanned, lies and the cleanup of the temporary space.

The application scanning phase is the main phase of the COBOL parser. It is divided in two sub-phases, as shown in **Figure 5.1**, which are: the filesystem scanning phase and the source code scanning phase. In the filesystem scanning phase the COBOL parser scans the directory tree of the source code of the old application to locate all the source files. After the end of this phase the TELOS statements that define the corresponding SourceFile objects in the TELOS database are created and stored in a separate text file. Some temporary files are also created in order to pass information, which was found while scanning the directory tree, to the next phase of the COBOL parser, which is the source code scanning phase. In the source code scanning phase the parser scans each file that contains source code line by line three times. The first time the parser scans each sourcefile searching for the data files that the software application uses to store its data, locates the CobolData objects and creates the TELOS statements that define the corresponding CobolData objects in the TELOS database instance. The second time the parser scans each sourcefile searching for the programs of the software application, locates the CobolNode objects and creates the TELOS statements that define the corresponding CobolNode objects in the TELOS database instance. The third time the parser scans each source file searching for the attributes of each CobolNode and SourceFile object and creates the TELOS statements that redefine the corresponding CobolNode and SourceFile objects together with all their attributes in the TELOS database instance.

The final merging phase merges the output that the previous phases have stored in text files while scanning the software application in one file. This file is then filled with some statements of the TELOS Data Entry Language in order to take the final form of a transaction that will be used to store the software objects in the specially designed TELOS database instance.

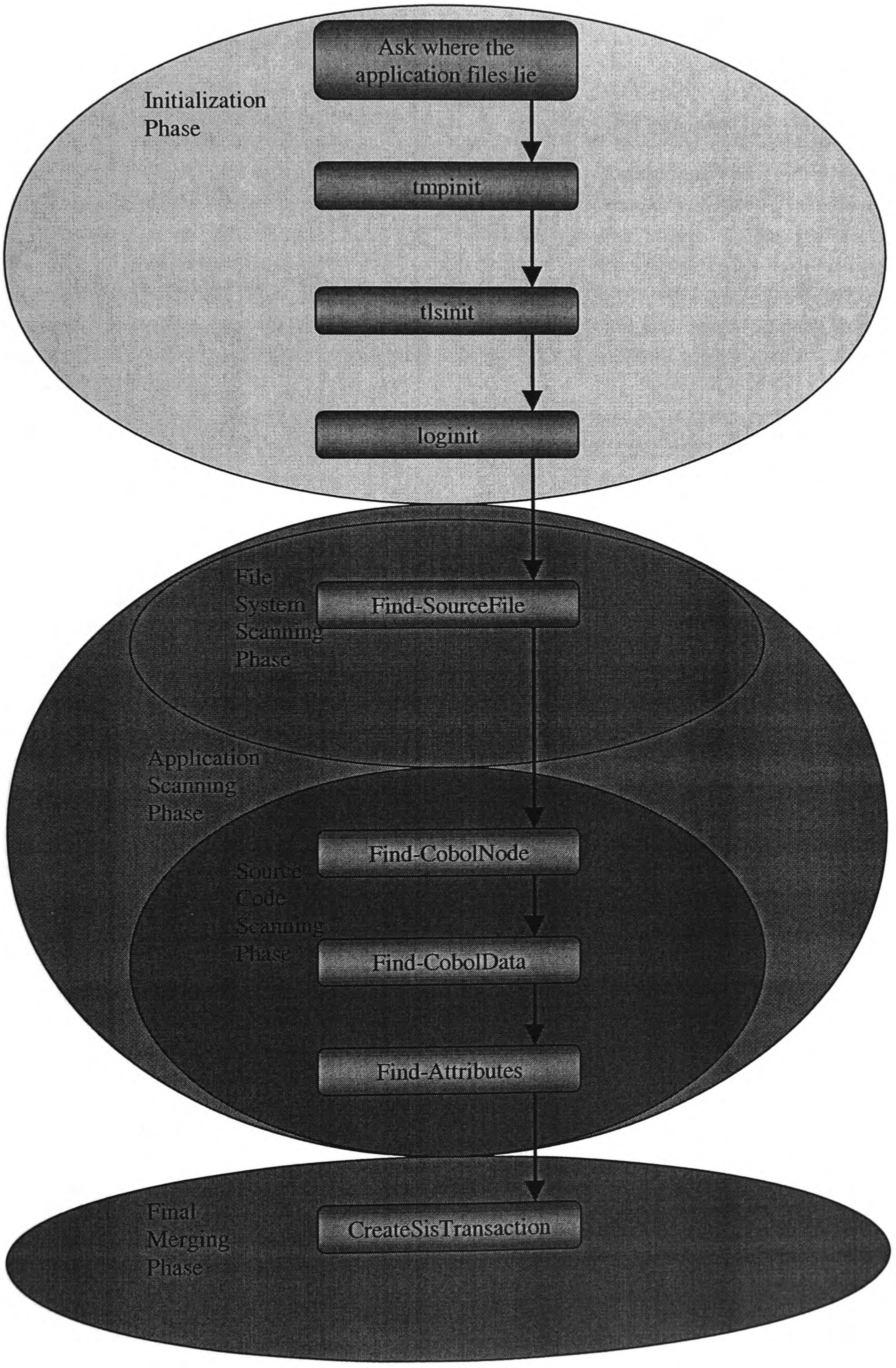


Figure 5.1 – Control flow of the COBOL parser

5.4. Implementation analysis of the COBOL Parser

5.4.1 The initialization phase

The initialization phase of the COBOL parser begins as soon as the parser starts to execute. Initially the parser gives standard values to its variables PATH and PARSERHOME. The PATH variable is used for the location of the executable files. This variable is inherited from the operating system environment and the "." directory is appended to it. The PARSERHOME variable has the value of the top-level directory of the parser directory tree. The parser prompts the user to enter the value of the cobolroot variable. This variable declares the top-level directory of the directory tree that the source code of the software application resides. All other subdirectories that might exist below this directory are also supposed to contain source files of the software application.

The first program that the parser invokes during the initialization phase is the tmpinit. This program initializes the workspace where the temporary files reside. The temporary files are created from the parser and used during the source code parsing procedure. The procedure of initializing the temporary space of the COBOL parser includes the removal of any old temporary files that might exist in the temporary workspace area. It also includes the initialization of all the temporary files (see **Appendix 6**) that will be used while the parser is executed.

The second program that the parser invokes during the initialization phase is the tlsinit. This program initializes the workspace where the TELOS files reside. This workspace area is also called the *output area* of the COBOL parser since it contains the files, which are assumed to be the output of the COBOL parser. These files are created while scanning the source code of the old software application. This initialization procedure includes the removal of any old TELOS files that might exist in the output workspace area of the parser. It also includes the initialization of all the TELOS files (see **Appendix 6**) that will be generated while the parser is executed.

Finally, the third program that the parser invokes during this phase is the loginit. This program initializes the workspace where the log files reside. The log files are generated during the software application parsing procedure. This workspace area is also called the *logging area* of the COBOL parser. This initialization procedure includes the removal of any old log files that might exist in the logging workspace area of the parser. It also includes the initialization of all the log files (see **Appendix 6**) that will be generated while the parser is executed.

5.4.2 The filesystem scanning phase

The filesystem scanning phase together with the source code scanning phase constitute the application scanning phase, which is the main part of the COBOL parser. The software tools that are used in this phase are the AWK pattern scanning and processing language together with a few other (mainly Unix) utilities such as grep and sort.

The only program that the COBOL parser invokes entering this phase is the find-SourceFile program. The parser invokes this program passing the top-level directory of the source code directory tree of the application as a parameter. At this point the procedure of locating all the source code files that constitute the software application (that the parser aims to analyze) starts. The filenames of the source code files are stored in the *srcfilenames.tmp* temporary file in a full path form.

After the location of all the source files, which constitute the software application, the COBOL parser creates the *SourceFile.tls* file. The SourceFile.tls file is a text file that contains all the TELOS statements that define the SourceFile individual objects into the TELOS database instance. The COBOL parser creates this file by reading the srcfilenames.tmp temporary file and using the AWK pattern scanning and processing language. The name that is used to store the source file in the filesystem is also used to represent the source file in the TELOS database instance.

The temporary file srcfilenames.tmp contains the names of the source files in a full-path form. The SourceFile.awk AWK program reads each line of the srcfilenames.tmp temporary file, discards the preceding directory names, which are separated by the "/" character and keeps the final name which is the name of the source file. The TELOS statement that is generated in order to define the SourceFile individual objects in the TELOS database instance has the form:

```
RETELL Individual (<filename>) in Token, SourceFile end
```

For each SourceFile individual object found by the parser a separate statement of the previous form is appended in the SourceFile.tls file.

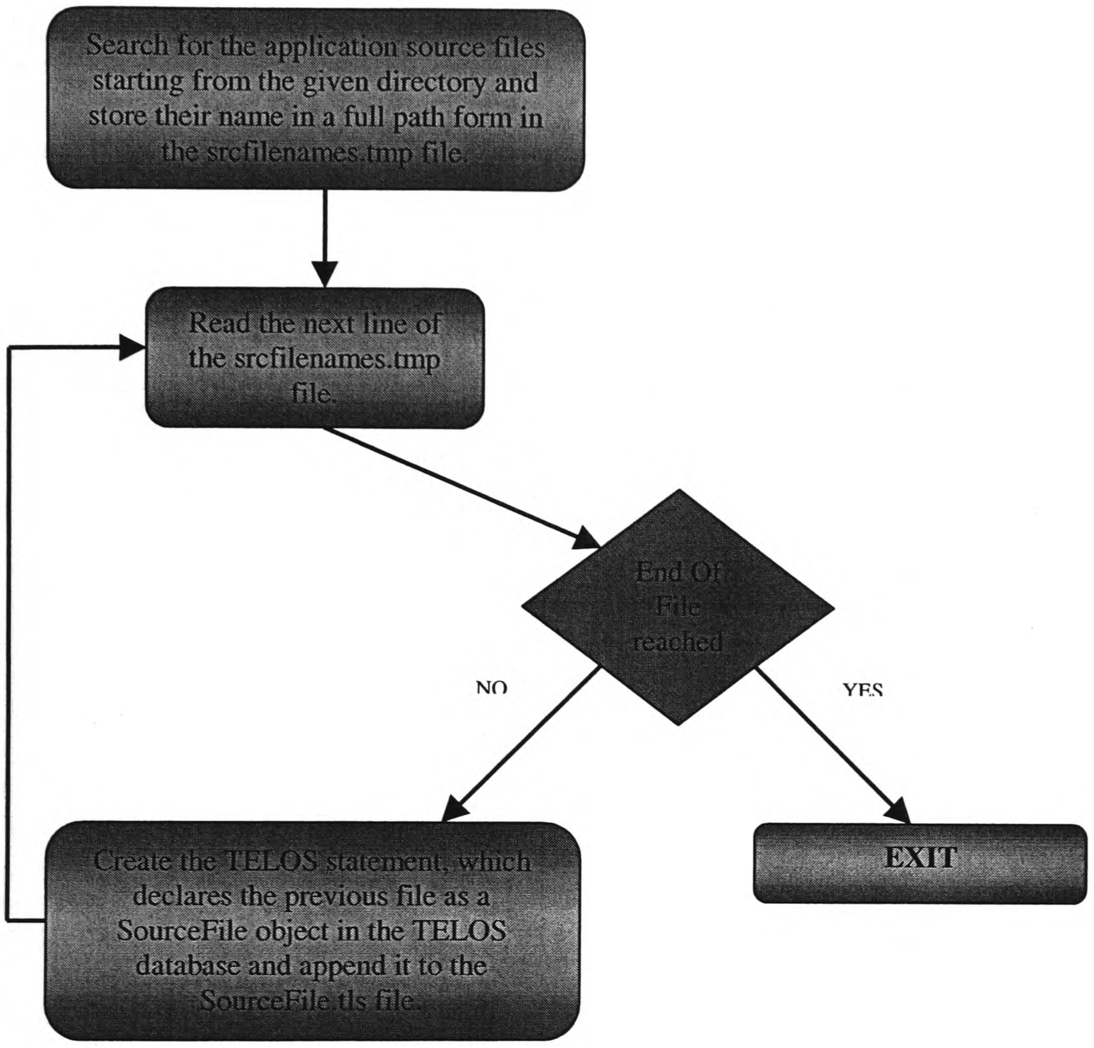

Search for the application source files starting from the given directory and store their name in a full path form in the srcfilenames.tmp file.
Read the next line of the srcfilenames.tmp file.
End Of File reached
NO
YES
Create the TELOS statement, which declares the previous file as a SourceFile object in the TELOS database and append it to the SourceFile.tls file.
EXIT

Figure 5.2 – Control flow of the find-SourceFile program

5.4.3 The Source Code Scanning Phase

The second part of the Application Scanning Phase is the Source Code Scanning Phase (see **Figure 5.1**). During this phase the parser scans the source code of the application line by line three times. The first time that the parser scans the source code it searches for CobolNode individual objects. The second time that the parser scans the source code of the application it searches for the CobolData individual objects. Finally, the third time that the parser scans the source code of the application it redefines the CobolNode and SourceFile individual objects and all of their attributes. The reason why the CobolNode and SourceFile individual objects are initially defined into the TELOS database instance without any attributes is that the object attributes are implemented inside TELOS in the form of relations to other objects. If a relation between two database objects is defined and one of the related objects does not exist into the database instance then the transaction gets discarded. As a result all the database objects are defined initially without any attributes and then a redefinition of them together with all of their attributes occurs.

The find-CobolNode program is the first program that the COBOL parser invokes during the source code scanning phase. The find-CobolNode program scans line by line the srcfilenames.tmp file and for each entry invokes the *CobolNode.awk* AWK program to scan line by line corresponding source file to determine whether this source file is the main source file of a program of the old software application. Each main source file of a program contains a PROGRAM-ID paragraph. The internal name of the COBOL program is defined in the PROGRAM-ID paragraph and if that paragraph is present in the source file then it is a main source file of a program. If the source file is identified as a main source file of a program then the find-CobolNode program creates a TELOS statement that has the following form:

```
RETELL Individual (<CobolNodeName>) in Token, CobolNode end
```

The previous TELOS statement defines the corresponding CobolNode object into the TELOS database instance. The find-CobolNode program appends the statement in the CobolNode.tmp temporary file. The name that is used in order to define the CobolNode object into the TELOS database instance is the same as the name of the corresponding SourceFile individual object but without any ".cbl" or ".CBL" extensions. After all the source files have been scanned then the CobolNode.tmp temporary file is checked against the possibility to contain duplicate entries. If duplicate entries are found then only one of them is kept and the fact is logged into the CheckDuplicates.log log file. Then the CobolNode.tls text file is created with the same contents with the CobolNode.tmp temporary file but without any duplicate entries. All other error messages that may be produced while the find-CobolNode program is executed are logged into the cobolnode.log log file.

While analyzing the procedure of computing the TELOS statement shown above it is easy to see why it is possible to have duplicate lines in the CobolNode.tmp temporary file. The name of each source file of the application is stored in the srcfilenames.tmp temporary file in a full path form. While computing the TELOS statement that defines the name of the source file as a CobolNode in the TELOS database the preceding directory names are discarded. It is possible though to have two different files, located in different directories, sharing the same filename. Although it is not a good practice to have two programs sharing the same name while located in different directories, this fact has to be faced by the COBOL parser.

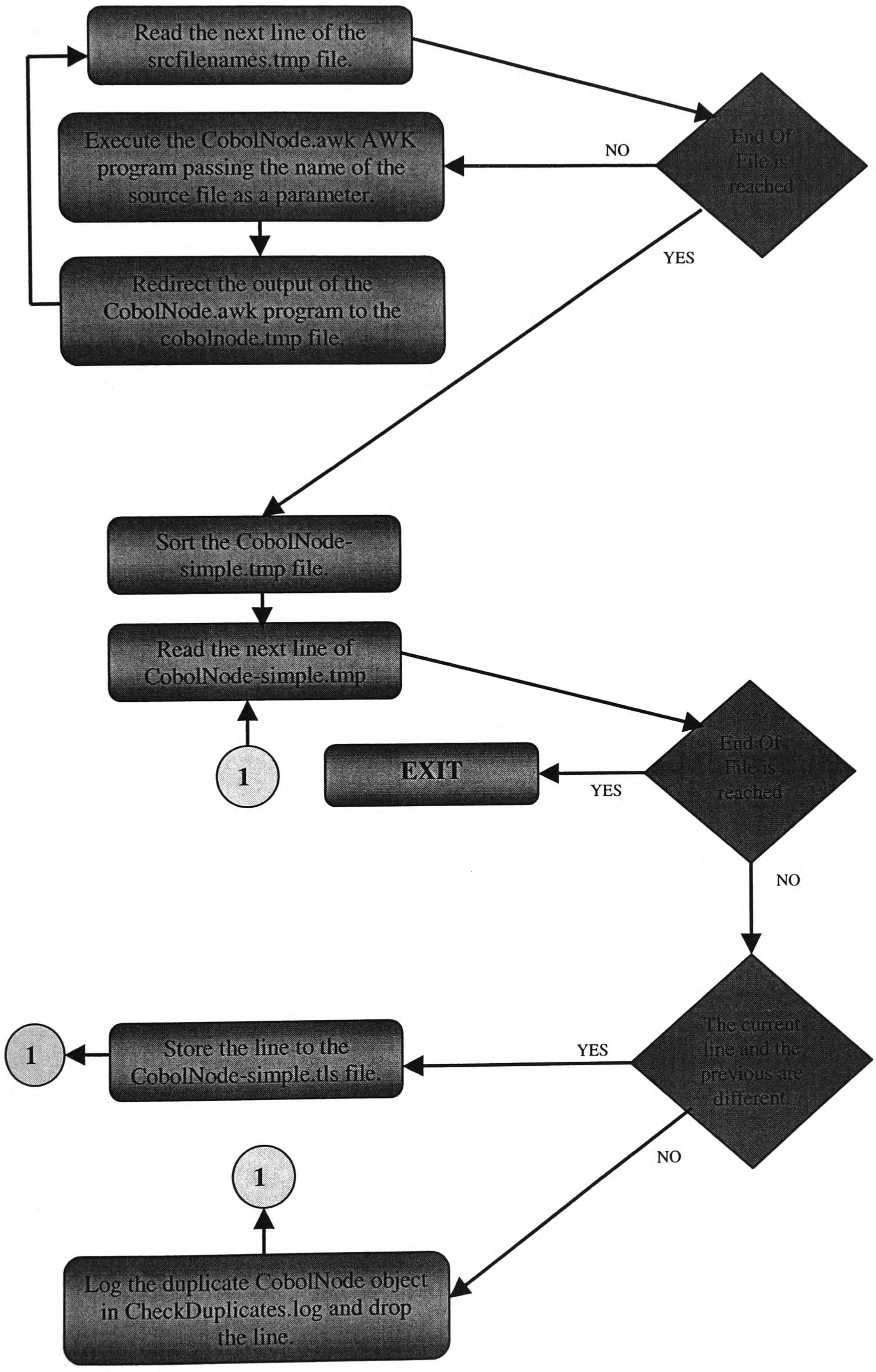

Read the next line of the srcfilenames.tmp file.
End Of File is reached
NO
Execute the CobolNode.awk AWK program passing the name of the source file as a parameter.
Redirect the output of the CobolNode.awk program to the cobolnode.tmp file.
YES
Sort the CobolNode-simple.tmp file.
Read the next line of CobolNode-simple.tmp
1
End Of File is reached
YES
EXIT
NO
The current line and the previous are different
YES
Store the line to the CobolNode-simple.tls file.
1
NO
1
Log the duplicate CobolNode object in CheckDuplicates.log and drop the line.

Figure 5.3 – Control flow of the find-CobolNode program.

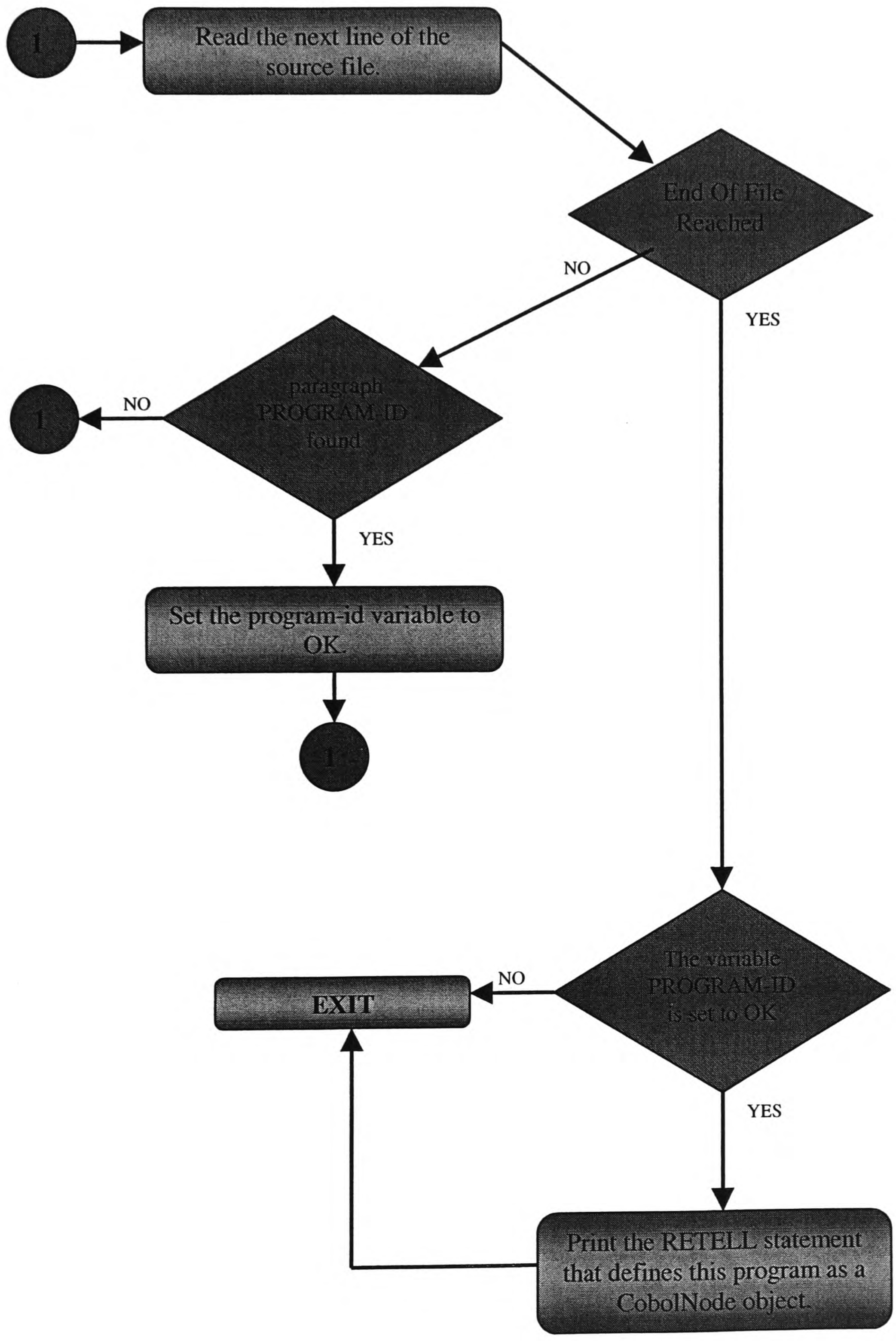

1
Read the next line of the source file.
End Of File Reached
NO
YES
paragraph PROGRAM-ID found
NO
1
YES
Set the program-id variable to OK.
1
The variable PROGRAM-ID is set to OK
NO
EXIT
YES
Print the RETELL statement that defines this program as a CobolNode object.

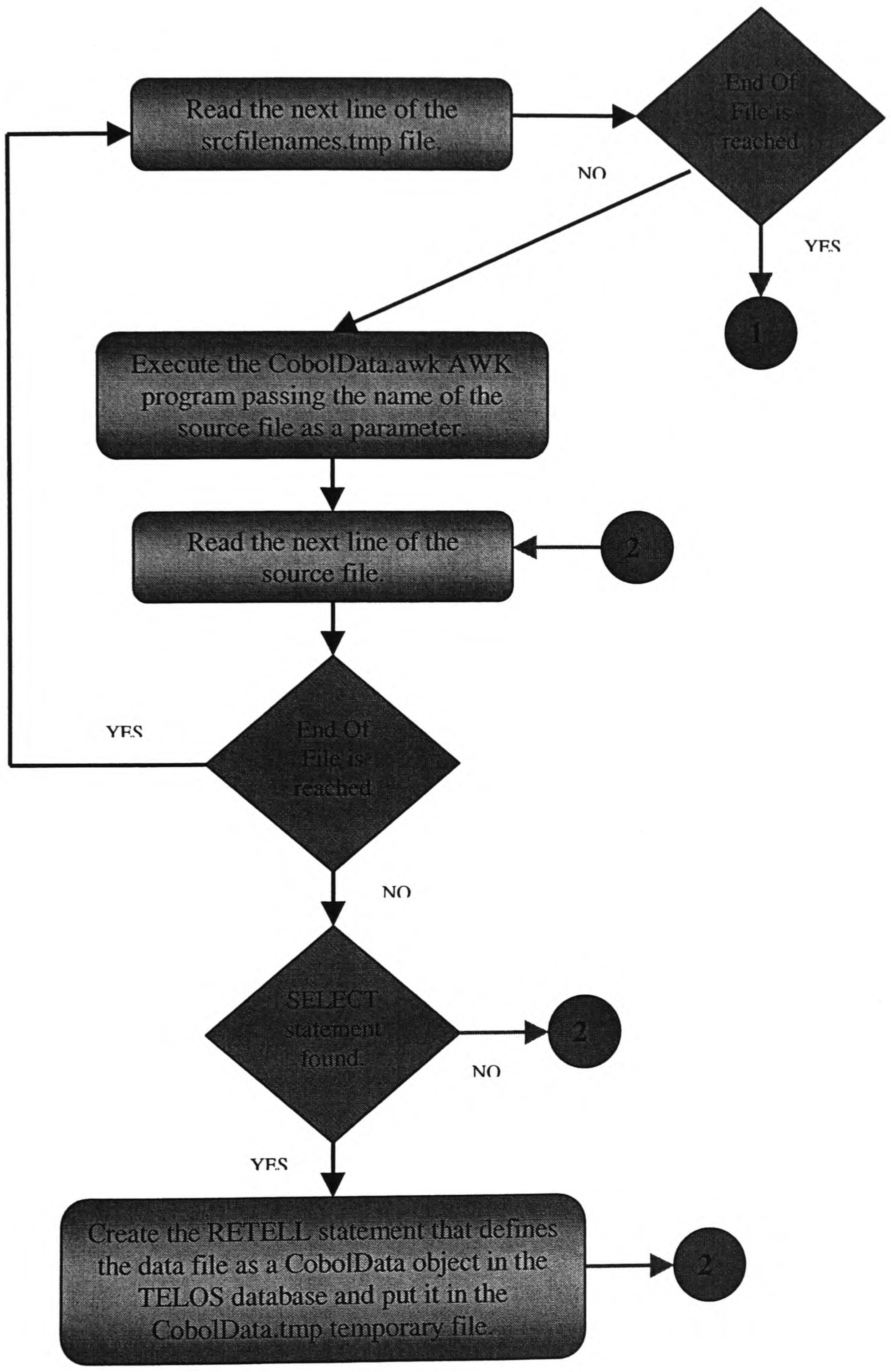
Figure 5.4 – Control flow of the CobolNode.awk AWK program.

The second program the COBOL parser invokes in the source code scanning phase is the find-CobolData program. This program reads line by line the srcfilenames.tmp temporary file and for each entry in this temporary file it executes the CobolData.awk AWK program passing the name of the source file to be scanned as a parameter to it.

The CobolData.awk AWK program reads line by line the file indicated by the find-CobolData executable in search for SELECT statements. The SELECT statement defines the structure of a COBOL data file. This statement links the filename of the data file in the filesystem with an internal name that is used whenever this data file is referenced from within the COBOL application. The name, which COBOL uses internally to reference the data file, is also used to identify this data file inside the TELOS database. The CobolData.awk AWK program understands the syntax of the COBOL SELECT statements and identifies the data files defined with these statements. For each SELECT statement found in the source file a RETELL statement is produced that has the following form:

```
RETELL Individual (<data file>) in Token, CobolData end
```

The previous RETELL statement defines the <data file> as a CobolData object in the TELOS database instance. **Figure 5.5** presents the control flow of the find-CobolData executable including the CobolData.awk AWK program.


Read the next line of the srcfilenames.tmp file.
End Of File is reached
NO
YES
1
Execute the CobolData.awk AWK program passing the name of the source file as a parameter.
Read the next line of the source file.
2
YES
End Of File is reached
NO
SELECT statement found.
2
NO
YES
Create the RETELL statement that defines the data file as a CobolData object in the TELOS database and put it in the CobolData.tmp temporary file.
2

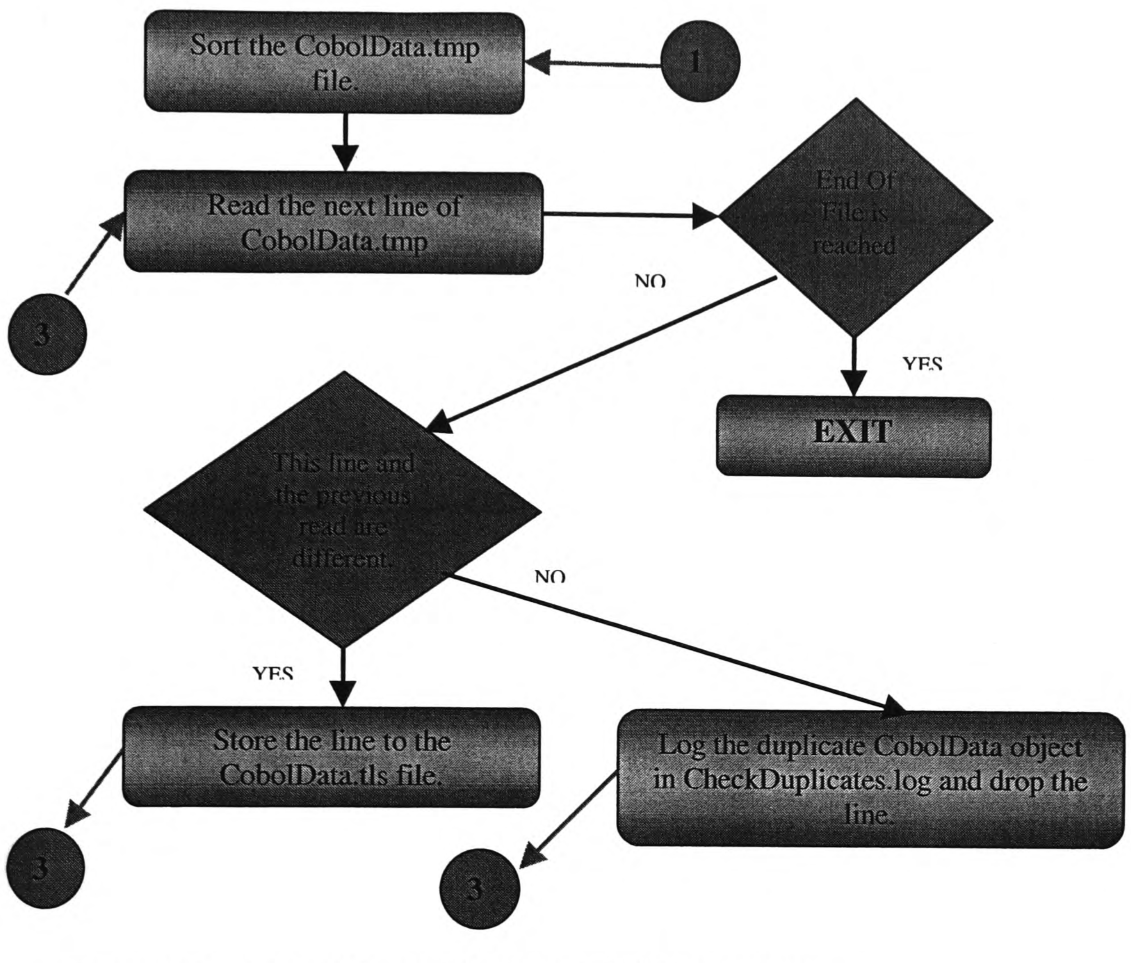

Sort the CobolData.tmp file.
1
Read the next line of CobolData.tmp
End Of File is reached
NO
YES
EXIT
3
This line and the previous read are different.
NO
YES
Store the line to the CobolData.tls file.
Log the duplicate CobolData object in CheckDuplicates.log and drop the line.
3
3

Figure 5.5 – Control flow of the *find-CobolData* program.

The last program that the COBOL parser invokes during the application scanning phase is the find-Attributes program. This program scans every source file of the application line by line and if the scanned source file is the main source file of a program of the software application then it redefines the corresponding CobolNode individual object together with all of its attributes. If the source file includes attributes regarding the corresponding SourceFile individual object then the find-Attributes program redefines the corresponding SourceFile individual object with all of its attributes. All the attributes that a CobolNode individual object might have are *name*, *include*, *call*, *affect* and *select*. The only attribute that a SourceFile individual object might have is *contain*. **Table 5.2** provides the COBOL statements that assign the attributes to the CobolNode individual objects*.

* For a complete description of the COBOL statements referred in the above Table see RM/COBOL-85 Reference Manual.

Table 5.2 – The COBOL statements that assign attributes to the CobolNode objects

Attribute	**Description**
Name	This attribute is defined by the Program-Id COBOL paragraph, which defines the internal name of the COBOL program.
Include	The COBOL COPY statement defines this attribute. COBOL uses the COPY statement to include in the program source code that is located in another source file indicated by the operands that follow the COPY statement.
Call	The COBOL CALL statement defines this attribute. COBOL uses the CALL statement to invoke other programs during run-time.
Affect	The COBOL OPEN statement defines this attribute. COBOL uses the OPEN statement to affect (read, write or append) a data file.
Select	The COBOL SELECT clause defines this attribute. COBOL uses the SELECT clause to define the attributes (such as the internal name, location in the filesystem etc) of a data file.

By the time the find-Attributes program is invoked, it creates two temporary files. The first temporary file created is named cobolnode.tmp. This file is created by reading the CobolNode.tls file line by line and dropping just the names of the CobolNode objects, which are defined by the RETELL statements included in the CobolNode.tls file, in it. The temporary file created next is named coboldata.tmp. This file is created by reading the CobolData.tls file line by line and dropping just the names of the CobolData objects, which are defined by the RETELL statements included in the CobolData.tls file, in it. As a result these two temporary files contain only the names of the CobolNode and CobolData individual objects identified so far by the Cobol parser.

These temporary files are created for data integrity reasons. Whenever an attribute is defined, the TELOS database checks the types of the related objects. If the types don't match the transaction, which defines the relation, gets discarded. For example whenever an affect relation is defined, TELOS expects it to be between a CobolNode and a CobolData object. So if an OPEN statement is found while scanning the source code the names of the files referred by the OPEN statement have to be validated within the names of the CobolData objects defined so far. If no match is found then the relation is not defined and a log entry is entered in the coboldata.log log file. This is a necessary validation check mainly because a lot more than one attribute or software object is defined per transaction. In fact, the definition into the TELOS database of all the software objects and attributes, which were identified while scanning the software application, occurs in just one transaction. As a result, the definition of even one attribute, referring to non-existing object, will destroy the whole application parsing process!

For each program of the old software application, which has previously been defined as a CobolNode object, the find-Attributes program creates a RETELL statement of the following form:

```
RETELL Individual (<CobolNode>) in Token. CobolNode
        with source
                : (<SourceFile>)
        with name
                : "<Internal Name>"
        with include
                : (<SourceFile 1>);
                : (<SourceFile 2>);


        with call
                : (<CobolNode 1>);
                : (<CobolNode 2>);


        with affect
                : (<CobolData 1>);
                : (<CobolData 2>);


        with select
                : (<CobolData 1>);
                : (<CobolData 2>);


        end
```

Table 5.3 – Inside the RETELL statement every keyword in the right of a *with* keyword represents the kind of the relations that follow (see **Table 5.2**).

The find-Attributes program uses the Attributes.awk AWK program to scan each source file. The Attributes.awk AWK program is invoked with the name of the source file to be scanned as a parameter. Before the Attributes.awk AWK program starts scanning the source file reading it line by line, it gets into an initialization phase. This phase divided in two parts, which are namely the *variable initialization* part and the *function definition* part.

All the variables that will be used during the scanning process are initialized with desirable values. Although the AWK programming language does not require the initialization or declaration of all variables in a certain part of the program, such as COBOL does, this initialization occurs in this program to make it easier to understand whenever someone reads its source code and for structural reasons. **Appendix 7** describes the usage of each variable inside the Attributes.awk AWK program, while **Appendix 8** describes its functions.

After the variable initialization and function declaration the Attributes.awk AWK program starts reading the source file line by line in search of attributes for the corresponding CobolNode or SourceFile individual objects. The first check in the loop procedure for the *current* line identifies whether the line continues an OPEN statement, found earlier in a previously scanned line. Normally, every COBOL statement ends with a trailing "." character. It is possible though for the "." character not to be used in the end of a COBOL statement, for example whenever this statement is inside an IF statement. In this case some operands of the OPEN statement, which indicate CobolData objects, may be continued in the next line. The read_opens Boolean variable is used to indicate such a case. Whenever a line starting with an OPEN statement is found it is checked against the possibility to end with the "." character. If this happens then this variable is set to the FALSE value, indicating that the next line is not a part of the *current* OPEN statement. If no "." character is found in the end of the line this variable is set to the TRUE value indicating that the next line is possibly continuing the OPEN statement. When the next line is read and the read_opens variable is TRUE then the line is checked against the possibility to start with a valid COBOL statement. If yes, this is the case that the previous OPEN statement finished in the previous line without a trailing "." character. In this case the value of the read_opens variable is turned to FALSE and the scanning process continues. If not then the Attributes.awk program regards that the words that belong in this line are CobolData objects defined by the OPEN statement of a previous line and it continues with the declaration of the corresponding affect relations inside the RETELL statement.

Before a word of the line is finally identified as a CobolData object, a number of checks take place to validate the identification. First, the word is checked against the possibility to be a phrase of the OPEN statement such as *input*, *output* and *io*. If so the word is discarded. Next, the word is checked against the possibility not to be one of the CobolData objects identified in previous phases, which are included in the coboldata.tmp temporary file. If the word is not included in the coboldata.tmp temporary file, it also gets discarded. Finally, the word is checked to find out whether the same CobolData object has also been defined earlier, while scanning this source file. This can happen by a previously found OPEN statement. It is not desirable to define the same affect relation more than one time although it is absolutely normal for a COBOL program to *open* many times, using more than one time the OPEN statement, the same data file. If no one of the three above cases happens then the word is regarded to be a valid CobolData object and the affect attribute of the corresponding CobolNode object is updated to include the specified value.

Next, the line is filtered to determine if the PROGRAM-ID paragraph is defined in it. If it is, then the name attribute of the corresponding CobolNode object is updated to include the specified value.

Next, the line is checked against the possibility to begin with a COPY statement. If the line starts with a COPY statement then the word following the COPY statement is compared with the names of the valid SourceFile objects, which have been stored into the sourcefile.tmp temporary file. If it is a valid SourceFile object then the include attribute of the corresponding CobolNode object and the contain attribute of the corresponding SourceFile object are updated to include the specified value. If the word is not a valid SourceFile object, it gets discarded.

Next, the line is filtered in search for a CALL statement in the beginning of it. In this case the word following the CALL statement is compared with the valid

CobolNode objects, which have been stored into the cobolnode.tmp temporary file. If the word is identified between the valid CobolNode objects, then the call attribute of the corresponding CobolNode object is updated to include the specified value.

Next, the line is filtered in search for an OPEN statement in the beginning of it. In this case the same procedure that has been described earlier for the definition of an affect relation is also used here to define the affect relations indicated by the OPEN statement. After their definition, the "." character is sought in the end of the current line. If the "." character is found then the read_opens variable gets the FALSE value indicating that the next line of the source file does not append the OPEN statement found in the current line. Otherwise, the read_opens variable gets the TRUE value indicating the possible continuation of the current OPEN statement in the next line.

Finally, the line is checked in search for a SELECT statement in the beginning of the line. In this case the word following the SELECT statement is checked against the valid CobolData objects the names of which have already been stored into the coboldata.tmp temporary file. If the word represents a valid CobolData object, then the select attribute of the corresponding CobolNode object is updated to include the corresponding value. This is the last examination of a line. After it has finished a new loop starts examining the next line of the source file.

After the last line of the source file has been examined the Attributes.awk program examines if the source file is the main source file of a program of the software application. If it is, then the RETELL statement that redefines the corresponding CobolNode object together with all of its attributes into the TELOS database instance is printed in the standard output. In addition the Attributes.awk program examines whether any attributes of the corresponding SourceFile object have been identified during the parsing procedure and. In this case the RETELL statement that redefines the corresponding SourceFile object together with all of its attributes into the TELOS database instance is printed in the standard output. The find-Attributes program redirects the standard output of the Attributes.awk program into the Attributes.tls file.

It has to be mentioned that during the source code parsing procedure the CobolData objects are not directly assigned any attributes. Only the CobolNode objects are assigned attributes referring to CobolData objects. The CobolData objects just inherit these attributes from the CobolNode objects.

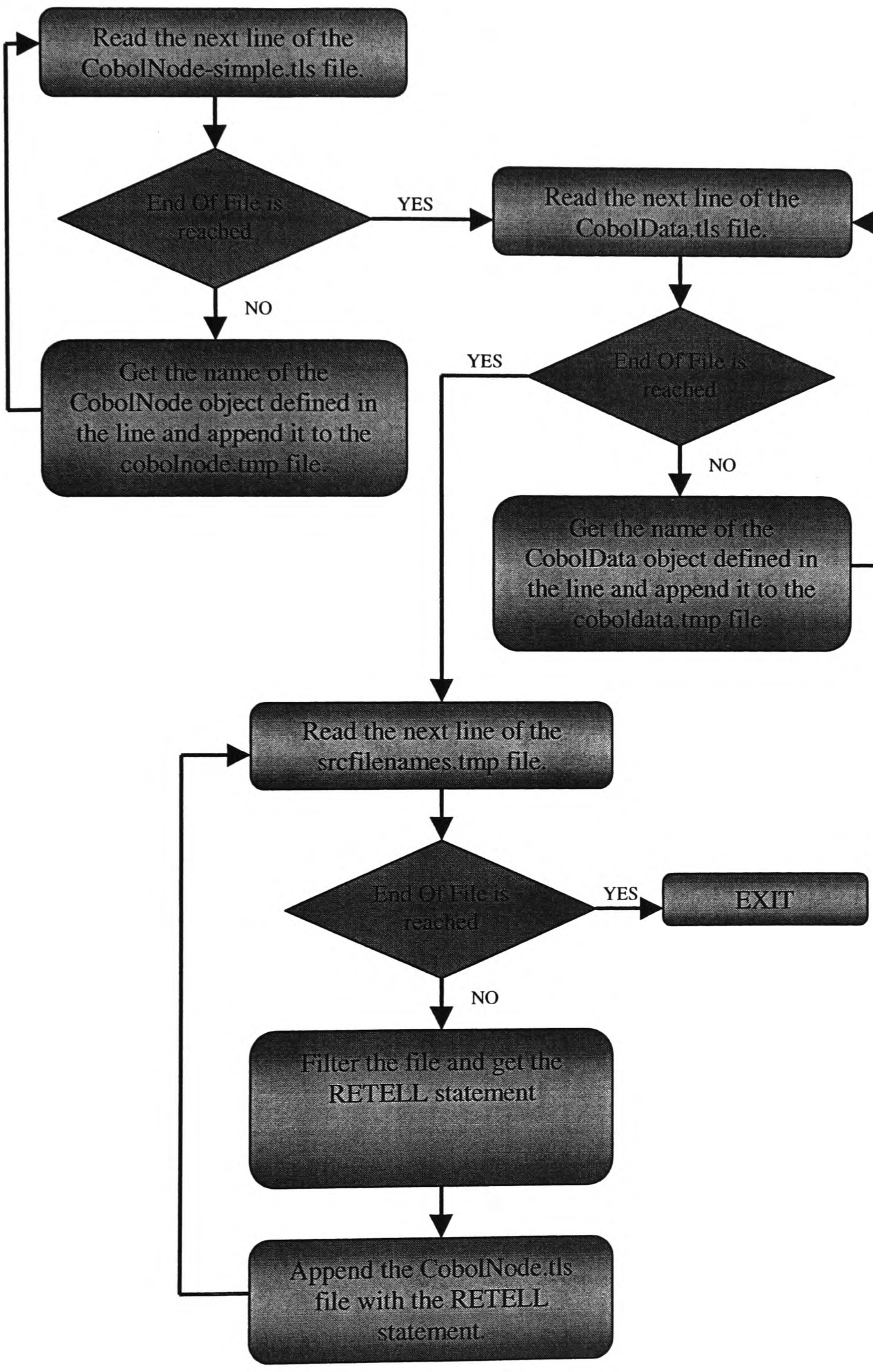
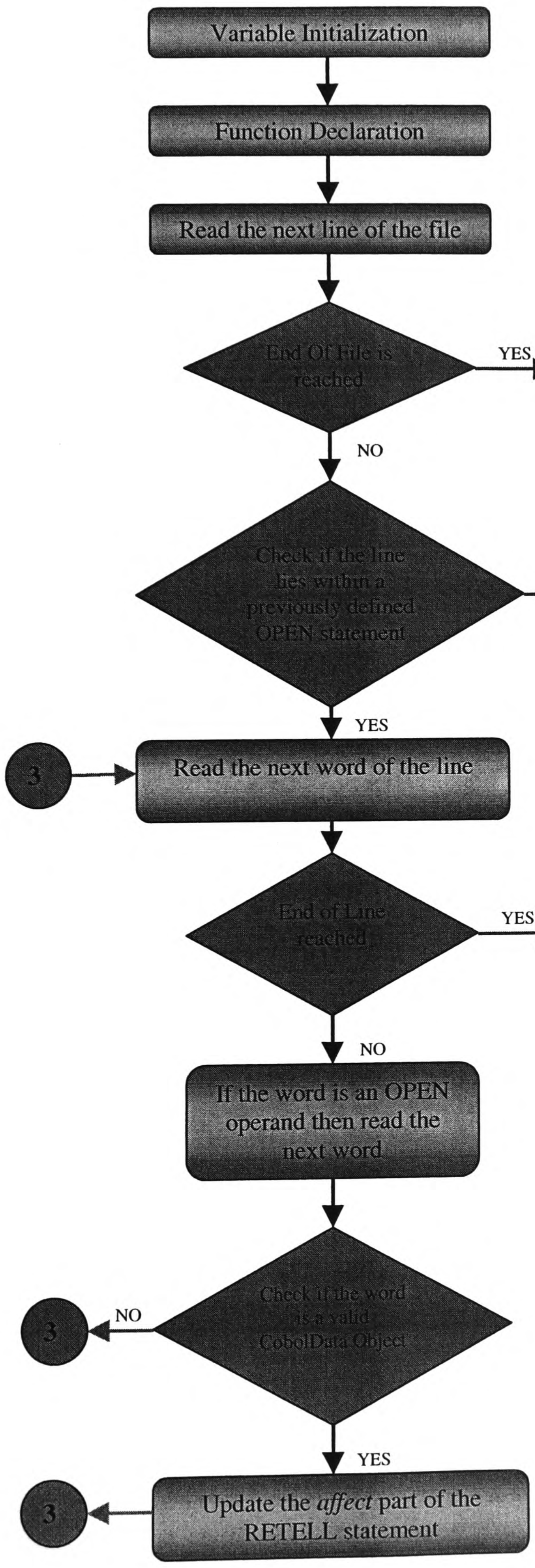
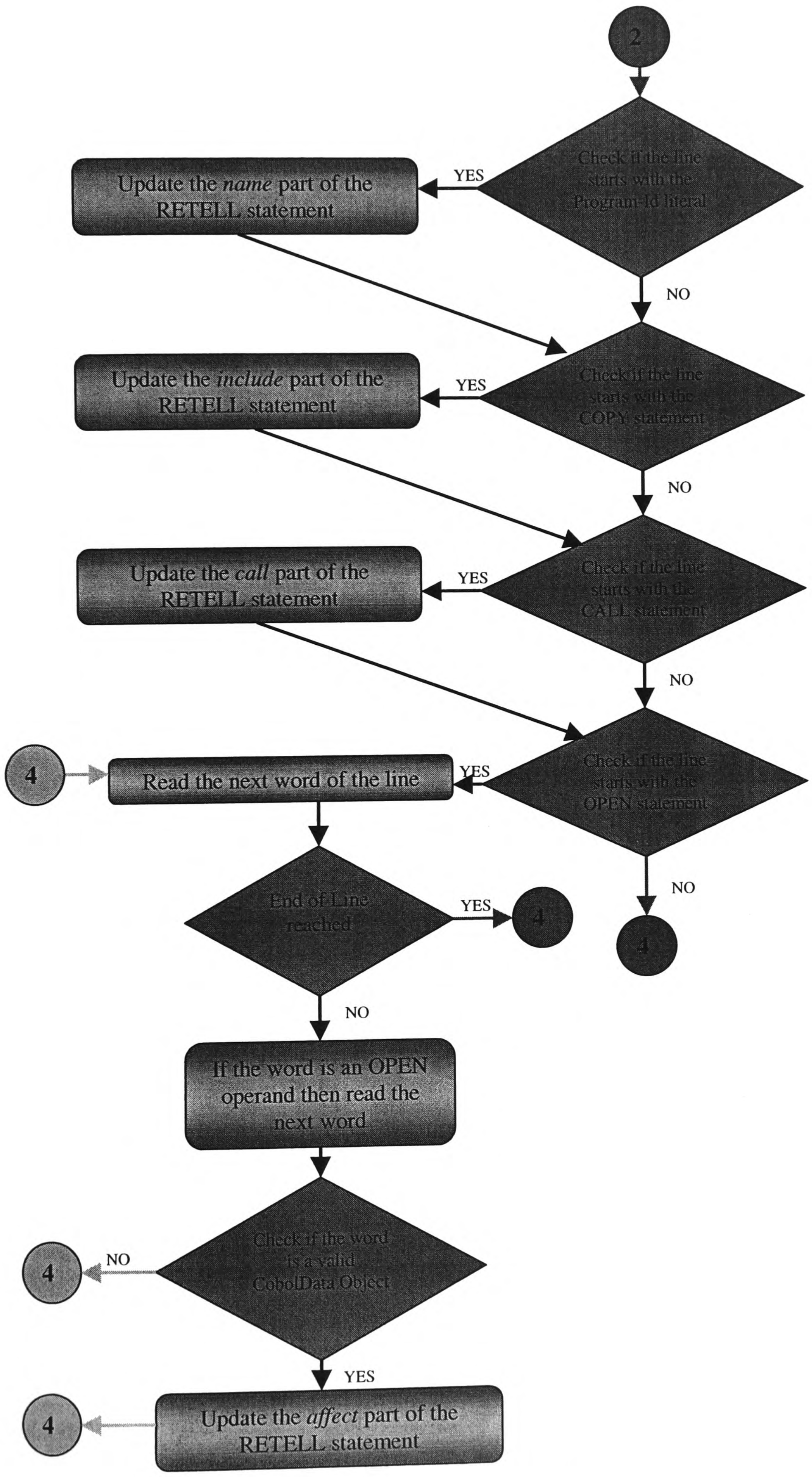

Read the next line of the CobolNode-simple.tls file.
End Of File is reached
YES
NO
Get the name of the CobolNode object defined in the line and append it to the cobolnode.tmp file.
Read the next line of the CobolData.tls file.
End Of File is reached
YES
NO
Get the name of the CobolData object defined in the line and append it to the coboldata.tmp file.
Read the next line of the srcfilenames.tmp file.
End Of File is reached
YES
EXIT
NO
Filter the file and get the RETELL statement
Append the CobolNode.tls file with the RETELL statement.

Figure 5.6 – Control flow of the find-Attributes program.


Variable Initialization
Function Declaration
Read the next line of the file
End Of File is reached
YES
1
NO
Check if the line lies within a previously defined OPEN statement
NO
2
YES
3
Read the next word of the line
End of Line reached
YES
2
NO
If the word is an OPEN operand then read the next word
Check if the word is a valid CobolData Object
NO
3
YES
Update the *affect* part of the RETELL statement
3


2
Check if the line starts with the Program-Id literal
YES
Update the *name* part of the RETELL statement
NO
Check if the line starts with the COPY statement
YES
Update the *include* part of the RETELL statement
NO
Check if the line starts with the CALL statement
YES
Update the *call* part of the RETELL statement
NO
Check if the line starts with the OPEN statement
YES
4
Read the next word of the line
NO
4
End of Line reached
YES
4
NO
If the word is an OPEN operand then read the next word
Check if the word is a valid CobolData Object
NO
4
YES
4
Update the *affect* part of the RETELL statement

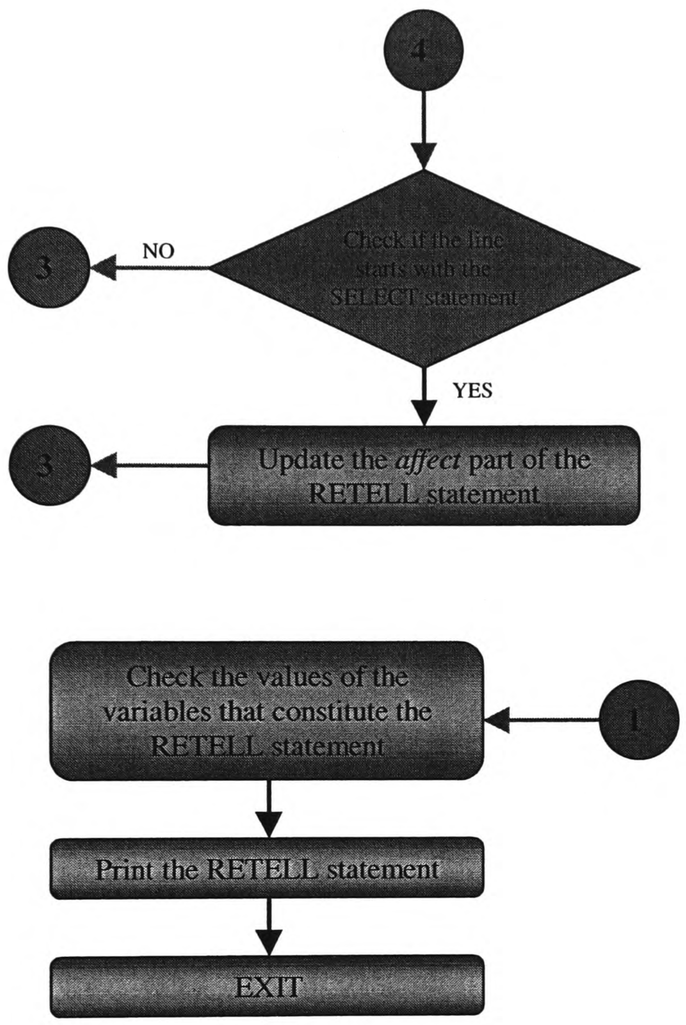

4
Check if the line starts with the SELECT statement
NO
3
YES
Update the *affect* part of the RETELL statement
3
Check the values of the variables that constitute the RETELL statement
1
Print the RETELL statement
EXIT

Figure 5.7 – Control flow of the *Attributes.awk* AWK program.

5.4.4 The final merging phase

The final merging phase is the last phase of the COBOL parser. It includes only one program named CreateSisTransaction. This program creates the application-structure.tls file, which includes the TELOS transaction that will define all the software objects and attributes, which have been found while scanning the source code of the old software application, into the properly designed TELOS database instance.

The CreateSisTransaction program implements six tasks. In the first task it appends a line including the BEGINTRANSACTION instruction of TELOS into the application-structure.tls file. In the second task it appends the contents of the SourceFile.tls file into the application-structure.tls file. In the third task it appends the contents of the CobolNode.tls file into the application-structure.tls file. In the fourth task it appends the contents of the CobolData.tls file into the application-structure.tls file. In the fifth task it appends the contents of the Attributes.tls file into the application-structure.tls file. Finally, in the sixth task the CreateSisTransaction program appends a line including the ENDTRANSACTION instruction of TELOS into the application-structure.tls file and exits.

After the completion of the CreateSisTransaction program the COBOL parser displays a message on the screen informing the user for the location of the application-structure.tls file and exits.

Chapter 6

Application and Testing of the Software Reengineering Mechanism

6.1 Introduction

The formal mechanism for analysis and re-implementation of legacy programs is divided in three main parts as illustrated in **Figure 4.1** in **Chapter 4**. The subject of this chapter is to apply the software reengineering mechanism in a real software system and test its results. A large software application, which is consisted of approximately 650000 lines of source code, will be used as a pilot in order to test the efficiency of this formal software reengineering mechanism. The information that this mechanism will present concerning the design and implementation of this specific legacy software application will be compared against the specifications that have been set for this software reengineering mechanism. The information, which is presented in this chapter regarding the XPERT Hotel software system, is brief and only for the purposes of testing the software reengineering mechanism. A more detailed analysis of XPERT Hotel takes place in **Chapter 7**.

6.2 Application

The application of the software reengineering mechanism on the legacy software application is implemented in three stages as shown in **Figure 4.1**. Initially, the COBOL parser scans the source code of the legacy software application and extracts useful pieces of information concerning its design and implementation. Next, this information is stored in a specially designed TELOS database instance. Finally, the SIS GAIN browser is used in order to present the information concerning the implementation and design of the legacy software application, which has been stored into the TELOS database.

6.2.1 The source code parsing procedure

The name of the legacy software application that will be analyzed is XPERT HOTEL. This software application manages the repositories and the financial issues regarding the supplies of a hotel. The source code of this software application is spread in three directories. The top directory of the directory tree of the source code is named *src* and contains 2349 source files. A separate directory named *dvp* lies below the top-level directory and contains only one subdirectory named *ken400*, which contains 202 source files. All the source files contain approximately 650000 lines of source code.

Initially, the COBOL parser prompts the user to enter the top-level directory of the directory tree of the software application. Then the parser scans all the source files of the software application, identifies the software objects and their attributes and creates the TELOS statements that will define those objects together with their attributes in the specially designed TELOS database instance. The parser exits informing the user for the location of the application-structure.tls file, which is a text file that contains all the TELOS statements created by the parser. The application-structure.tls file that the parser created for the XPERT HOTEL application contains 6309 TELOS statements.

After the completion of the source code parsing procedure of the XPERT HOTEL application, the log files that the COBOL parser has created are examined. The coboldata.log and cobolnode.log files only mention that the COBOL parser could not open three files for scanning, which are namely hotelsrc, dvp and ken400. This information is expected since these files are the three directories, which include the source files of the XPERT application and not regular text files. The same information is included in the attributes.log file. Additionally, the attributes.log file mentions repeatedly (seven entries) that the PRG(WHAT-WAY string could not be located in the contents of the cobolnode.tmp temporary file and that the command that tried to locate the string exited with error. The error is justified by the fact that the COBOL parser uses the *grep* Unix command to identify the cobolnode objects within the cobolnode.tmp file and that this command handles parenthesis as a special character inside the string that is to be matched and thus exits with an error code. Finally, the CheckDuplicates.log file indicates that seventeen duplicate source files, seven of which are main source files, have been identified while locating the source files of the legacy software application. Obviously these source files reside both in the hotelsrc and ken400 directories. The CheckDuplicates.log file also indicates that there are 632 duplicate definitions of the data files of the software application!

All this information that has been logged while scanning the source code provides helpful hints while analyzing the implementation and design of the software application. For the XPERT HOTEL application the coboldata.log and cobolnode.log files do not contain information that must be analyzed furthermore. The attributes.log file includes some hints that should be analyzed, while the CheckDuplicates.log file offers information that should be carefully studied while analyzing the software application. The information included in the log files will be examined in detail in the next chapter while analyzing the XPERT HOTEL software application.

6.2.2 Information storage into the TELOS database instance

In order to store the software objects that the COBOL parser has identified regarding the legacy software application, it is necessary to create a new and empty TELOS database instance. After its creation this database instance is specially configured for the needs of the software reengineering mechanism. Finally, the software objects that the COBOL parser has identified together with their attributes are stored in this database instance.

In order to create a new and empty TELOS database instance all the contents of the data directory of the TELOS database engine are deleted when the database engine is inactive. When the database engine initializes, it checks the contents of its data directory and creates a new and empty database instance if this directory is found empty.

After the new TELOS database instance is created, it is configured specially for the needs of this software reengineering mechanism by creating special objects that define the structure of the database instance. For the definition of the structure of the database instance, initially, the TELOS parser executes the TELOS statements that create the objects that configure the menus of the GAIN browser. Next, the TELOS parser executes the TELOS statements that create the classes of the software objects and their attributes. Finally, the TELOS parser executes the TELOS statements that create the objects that define the queries that the GAIN browser is able to execute. This procedure is described in detail in **Chapter 5**.

After the new TELOS database instance is created and configured, the TELOS parser reads the application-structure.tls file and executes the 6309 TELOS statements that the COBOL parser has created, while scanning the source code of the software application. After the execution of all these TELOS statements, all the software objects that the COBOL parser has identified, regarding the old software application, together with their attributes have been defined into the specially designed TELOS database instance. **Appendix 9** describes the programs, the batch files and the order they are executed, in order to accomplish the described procedure.

6.2.3 Presentation of the information using the SIS GAIN browser

The third and final part of this software reengineering mechanism includes the presentation of the information, regarding the design and implementation of the legacy software application. The presentation of the information is accomplished using the GAIN browser in cooperation with the TELOS database instance. First, the TELOS database instance starts executing and waits for query transactions. Then the GAIN browser starts executing, connects to the database instance and waits for the user to trigger the available queries. **Appendix 10** describes the programs, the batch files and the order they are executed, in order to start the TELOS database instance and the GAIN browser.

The four text view queries of the GAIN browser provide general information regarding the software components (source files, programs and data files), which apart the XPERT HOTEL application (see **Chapter 4** named "Design and Implementation Analysis of the SIS Setup"). After the identification of all the software components it is easy to concentrate on each of those components and get information regarding their properties using the graphical view queries of the GAIN browser.

The first text view query in the Queries menu of the GAIN browser is named List Source Files. This query reports the names of 2537 source files that constitute the XPERT HOTEL application. In section 2.1 of this document is mentioned that the XPERT HOTEL application is constituted by 2349 source files in the hotelsrc directory and 202 source files in the ken400 directory. In section 2.2 of this document is mentioned that there are 17 duplicate source files that reside in both hotelsrc and ken400 directories of the XPERT HOTEL application. As a result the GAIN browser should report 2349 + 202 – 17 = 2534 source files. Examining the names of the source files, which are reported by the GAIN browser, it is observed that the names of the hotelsrc, dvp and ken400 directories are included in the report. This fact justifies the resulting summary of 2537 source files. Nevertheless, it is not necessary and maybe not right to include the names of the directories of the source code of a software application in the report of its source files. This can be one of the future improvements of the COBOL parser.

The second text view query in the Queries menu of the GAIN browser is named List Data Files. This query reports the names of 288 data files that constitute the database of the XPERT HOTEL application. In section 2.1 of this document is mentioned that there are 632 duplicate definitions of the datafiles of the XPERT HOTEL application. Comparing these two numbers it is easy to see that many data files have duplicate definitions throughout the source code of the XPERT HOTEL application. This fact leads in the examination of those definitions in search for possible errors as long as enhancements of the source code of this software application.

The third text view query is named List All Programs. This query reports that 1155 programs constitute the XPERT HOTEL application. The comparison of the number of programs with the number of source files, which is 2534, provides a clue concerning the organization of the source code of the application. Since the number of source files is more than twice the number of programs, it is assumed that some pieces of source code that are identical among different programs have been isolated in separate source files. Nevertheless, it has already been observed that many data files have duplicate definitions throughout the source code. These duplicate definitions should also be isolated in separate text files in order to minimize structural database problems and errors. As a result, it is expected that the source code of the XPERT HOTEL application is not very well organized. Some of the identical source code pieces may be isolated in separate source files, while other pieces of source code that should be identical are located in more than one source files, increasing the possibility of a logical or implementation error.

The fourth text view query is named List Affecting Programs. This query reports that 967 of the 1155 programs that constitute the XPERT HOTEL application affect the contents of at least one datafile. The fact that approximately 84% of the complete set of programs affect at least one datafile identifies that the XPERT HOTEL application is mainly a data processing application. As a result, the process of the analysis regarding the implementation and design of the XPERT HOTEL application should concentrate in the analysis of the database of the application and the data flow within each program rather than the control flow of each program individually.

6.3 Testing

The *Documentation Study and Analysis* section of this thesis presents two research papers concerning special techniques for testing software [1] [2]. There are a great number of research papers within the research area of software testing. The two selected research papers propose a method for testing object oriented software systems to certify if the software systems meet their specifications or not. The software testing method that these two papers propose has been chosen, since the software reengineering mechanism is an object oriented software system, which has been developed in order to satisfy a predefined set of specifications (see **Chapter 3** entitled "Requirements Analysis and Specifications").

6.3.1 The testing method

One of the most powerful benefits of the proposed software reengineering mechanism is that with the use of the COBOL parser it is able to identify software objects and software object attributes within the source code of a procedural computer language such as COBOL. These software objects together with their attributes are stored in a specially designed TELOS database instance. The GAIN browser connects to the database instance and is capable of executing special queries, the results of which provide critical information in the process of the implementation and design analysis of the old software application. These queries can be faced as events that require specific input values and produce the desired output information when triggered. Concerning the development environment of an application, the authors of [1] and [2] insist that the classical development methods for procedural programs involve a hierarchical decomposition of functions and that on the contrary, OO development methods are characterized by decentralized architectures of objects. The authors claim that in order for the test to be effective this observation has to be taken into account.

The requirements that this software reengineering mechanism should meet are studied in **Chapter 3**. In the same chapter a set of specifications that the proposed software reengineering mechanism should fulfil has been developed according to these requirements. Stephane Barbey, Didier Buchs and Cecile Peraire [2] focus on specification-based testing methods. The authors call these methods *black box* methods. They define black box methods as an approach to find errors in a program by validating its functionality, without analyzing the details of its code, but by using the specification of the system. They insist that the goal is to answer the question: *Does a program satisfy the requirements of its specification?* or, in accordance to the goal of testing, *to find if a program does not satisfy its specification.*

In order to identify whether a program meets its specification or not, Stephane Barbey, Didier Buchs and Cecile Peraire propose a specific method. This method includes selecting from the specification the services required from the system. For each service, the specification allows the selection of a number of scenarios for the program under test. The set of all these scenarios makes up the test set. Furthermore analyzing the test set, the authors mention that an exhaustive test set should obviously contain all the tests that are required by the specification. Then, they admit that an exhaustive test set is generally infinite and it is necessary to apply a number of reduction hypotheses to the behavior of the program in order to obtain a finite test set of reasonable size.

The proposed test procedure is as follows:

Given:

- SPEC Class of all specifications
- PROG Class of all programs
- TEST Class of all tests
- $\models$ Satisfaction relationship on PROG X SPEC, expressing the validity of a program with respect to the specification
- $\models_O$ Satisfaction relationship on PROG X TEST, deciding if the tests are successful or not

Steps:

Step1	Selection of a test set from a specification of the system and from a set of hypothesis on the program under test
Step2	Execution of the program under test using the test set
Step3	Analysis of the results obtained during the execution of the program

The test procedure is successfully finished as long as:

$$(\forall P \in \mathbf{PROG} \text{ and } \forall T \in \mathbf{TEST}) \Rightarrow P \models_O T$$

6.3.2 Applying the testing method to the software reengineering mechanism

The specifications of the software reengineering mechanism are divided in three sections. These sections are namely: I) database analysis, II) control flow analysis and III) source code organization analysis (see section named *Requirements Analysis and Specifications*). For each section of specifications a separate test set is built and verified. The XPERT HOTEL application that has already been parsed will be used in order to implement and verify the test sets. The fact that the mechanism will be tested only for its application to the XPERT HOTEL software system does not guarantee that it is error-free to every software system that has been developed with RM/COBOL-85. Nevertheless, as mentioned in [2] it is always necessary to apply some reduction hypothesis to the behavior of the program, to obtain a test set of *reasonable* size. The reduction hypothesis of applying the software reengineering mechanism to the XPERT HOTEL software system and then testing it against its specifications, is regarded fair enough, since the XPERT HOTEL software system does not have any specifics that have been additionally studied while designing and implementing the software reengineering mechanism. The mechanism is supposed to provide equivalent results if tested when applied to any other software system that has been developed with RM/COBOL-85. Another reduction hypothesis is that each test will not be applied to all the software objects (programs, datafiles and source files) that it is referred to, since the XPERT HOTEL application consists of several thousands of software objects, which constitute a huge test set. The software objects that are candidates for testing will be sorted based on the number of attribute relations that they have and the top three of them will constitute the test set. **Appendix 11** presents the software objects of the XPERT application that are candidates for testing. The fact that only three of the complete set of software objects are tested increases the possibility of an undetected error condition inside the whole software system. On the contrary the fact that these three software objects have the most attribute relations increases the possibility of an error condition to exist in one of these software objects than any other software object of the software system. Nevertheless, the process of applying the tests to the complete set of software objects of the XPERT HOTEL application would require a whole team of software programmers in order to apply these tests in a reasonable period of time. Thus, this reduction hypothesis is judged to be fair enough for the purposes of this project.

6.3.2.1 Specifications regarding the database analysis

The specifications of the software reengineering mechanism, regarding the database analysis of the legacy software application, are shown in **Table 3.1** in **Chapter 3**. A detailed analysis of the specifications of the software reengineering mechanism occurs in **Chapter 3** named *Requirements Analysis and Specifications*.

As indicated by specification 1.1 the mechanism should provide a report of all the datafiles used by the legacy software application. This report is provided by the *List Data Files* text-view query of the GAIN browser. The report represents the datafiles using the names that the old application uses to reference them in its source code. According to **Table A11.2** in **Appendix 11** programs EMP-APO, MAKOIKO and YPOK-APO are the three top programs of the XPERT HOTEL application that define the most datafiles in their main source file. The source code of these programs has been examined in search for SELECT statements. The datafiles that are defined in the source code of the programs were located inside the report produced by the *List Data Files* text-view query of the GAIN browser. The result of this test indicates that the information provided by this report is valid.

Specification 1.2 requires a separate report for each datafile presenting all the programs that access the datafile. The GAIN browser provides this report by focusing on the specific datafile and executing the *Star View* graphical query. This query will produce a graph having the datafile in the center of the graph and all the objects related to it categorized by the type of relation. All the programs that are related with the datafile with the *affect* relation are those programs that access the datafile. According to **Table A11.6** in **Appendix 11** datafiles APO-FILE, APOMA-FILE and APANAL-FILE are the top affected datafiles of the XPERT HOTEL application. The information provided by the *Star View* graphical query, concerning the programs that access a specific datafile, is incomplete. After the location of all the OPEN statements inside the source code of the XPERT HOTEL application that refer to these three datafiles it was proved that all the programs mentioned in the report were actually affecting these datafiles. In addition there were a number of additional programs affecting the contents of these datafiles, which were not reported by the GAIN browser. The problem is focused in the fact that the COBOL parser creates an *affect* relation to each program, after examining the main source file of the program. The source code that is included to the program but exists in additional source files is not examined for *affect* relations. This result indicates that the software reengineering mechanism did not pass successfully this specific test and needs to be improved in this point.

Specification 1.3 requires a separate report for each datafile that presents all the programs that define the specific datafile in their source code. The GAIN browser provides this report by executing the *Star View* graphical query after focusing on each datafile. All the programs related with the datafile with the *select* relation define the datafile in their source code. According to **Table A11.7** in **Appendix 11** datafiles

SEQ-FILE, PRINT-FILE and APO-FILE are the top three files with the most *select* relations. The same problem with the previous report has also been identified in the case of this report. The COBOL parser creates a *select* relation to each program, after examining the main source file of the program. The source code that is included to the program but exists in additional source files is not examined for *select* relations. This result indicates that the software reengineering mechanism did not pass successfully this test.

Specification 1.4 requires a standard method to provide for each datafile its record description. The method that the software reengineering mechanism provides for this purpose is to find all the programs that define the specific datafile in their source code and find the record description of the datafile by reading their source code. This method is valid as long as the software reengineering mechanism reports correctly the programs that define each datafile.

Finally, specification 1.5 requires a method to provide hints about relations, which may possibly exist among the datafiles of the legacy application. The method that the software reengineering mechanism provides for this purpose is to isolate the record descriptions of the datafiles and compare them. The fields of different datafiles that have the same size and similar names are candidates for the implementation of a relation. This method is valid as long as the software reengineering mechanism reports correctly the programs that define each datafile.

6.3.2.2 Specifications regarding the control flow analysis

The specifications of the software reengineering mechanism, regarding the control flow analysis of the legacy software application, are shown in **Table 3.2** in **Chapter 3**.

As indicated by specification 2.1 the mechanism should provide a report of all the programs that constitute the legacy software application. This report is provided by the *List All Programs* text-view query of the GAIN browser. This report represents each program using the name that the XPERT HOTEL application uses to reference the program in its source code (internal name). The main source files of all the programs listed in the report were examined one by one and the results of the report were validated. The rest source files of the XPERT HOTEL application were also examined one by one and none of them was found to constitute a valid RM/COBOL-85 program. This result indicates that the information provided by this report is valid.

Specification 2.2 requires both the internal name and the name of the main source file of the program to be reported. The *List All Programs* text-view query provides the name of the main source file of each program in a second column in the right of the internal name of each program of the legacy software application. While applying the previous test, regarding the complete set of programs of the XPERT HOTEL application, this information was also validated.

Specification 2.3 requires for each program of the legacy software application all the programs that are called by it at run time to be reported. The GAIN browser provides this report in two ways. The first is to execute the *Star View* graphical query after focusing on a specific program. As mentioned earlier this query will produce a graph having the selected object in the center of the graph and all the objects related to it categorized by the type of relation. All the objects related with the selected object with the *call* relation are the programs that are called by the selected program. This query also presents the programs that call the selected program separately. The second is to execute the *Call Tree* graphical query of the GAIN browser. After focusing to a specific program of the legacy application and executing the *Call Tree* graphical query a graph is generated showing all the programs that are called by this program. This query is recursive so the programs that are called by the programs that are called by the focused program are also reported and so on. According to **Table A11.4** in **Appendix 11** programs APOPEN, APIN and TIMOPEN are the top three programs with the most *call* relations. For these three programs the information provided by the software reengineering mechanism was validated, by examining the CALL COBOL statements inside their source code. Nevertheless, this result was regarded to be incidental since it is already known from the previous tests that the COBOL parser examines only the main source file of a program when it identifies its relations. Thus, the programs APOMASTN, TIMEIS and TIMAG, which according to **Table A11.1** in **Appendix 11** are the three top programs with the most *include* relations, were also examined. For these programs the results were not validated since there was found to be several calls to external programs that were coded in the additional source files of the programs that were not included in the report. This result indicates that the software reengineering mechanism did not pass successfully this test.

Specification 2.4 requires a separate report for each program of the old software application showing all the datafiles that are affected by the program. The GAIN browser provides this report in two ways. The first is to execute the *Star View* graphical query after focusing on a specific program. All the objects related with the selected program with the *affect* relation are the datafiles that are affected by the program. The second is to execute the *Affect Tree* graphical query of the GAIN browser. After focusing to a specific program of the legacy application and executing the *Affect Tree* graphical query a graph is generated showing all the datafiles that the program affects, all the programs that are called by this program and for each called program all the datafiles that it affects. This query is recursive so the programs that are called by the programs that are called by the focused program and the datafiles that they affect, are also reported and so on. According to **Table A11.2** in **Appendix 11** programs V3-APO, TIMEIS and TIMEKD are the top three programs with the most *affect* relations. For these three programs the information provided by the report was found to be correct for the V3-APO program and incomplete for TIMEIS and TIMEKD. The datafiles that were reported for TIMEIS and TIMEKD were actually affected by these programs, but additional datafiles were found to be affected by these two programs that were not included in the report. This problem occurs since the COBOL parser does not examine the additional source files that constitute a program when it identifies its attributes. The OPEN statements that were not identified by the software reengineering mechanism were located in the additional source files that were used to build these two programs. This result indicates that the software reengineering mechanism did not pass successfully this test.

Finally, specification 2.5 requires a separate report for each program of the old software application showing all the datafiles that are defined in its source code. This report is provided by the *Star View* graphical query of the GAIN browser. After focusing on a specific program and executing the *Star View* graphical query all the objects that are related with the focused program are shown on the screen. The datafiles that are related with the *select* relation with the focused program are those that are defined in the source code of the program. According to **Table A11.3** in **Appendix 11** programs EMP-APO, MAKOIKO and YPOK-APO are the top three programs with the most *select* relations. For these three programs the information provided by the report was found to be correct for the EMP-APO and MAKOIKO programs and incomplete for YPOK-APO. The datafiles that were reported for

YPOK-APO were actually defined in the source code of the program, but additional datafiles were found to be defined in the source code that were not included in the report. This problem occurs since the COBOL parser does not examine the additional source files that constitute a program when it identifies its attributes. The SELECT statements that were not identified by the software reengineering mechanism were located in the additional source files that were used to build these two programs. This result indicates that the software reengineering mechanism did not pass successfully this test.

6.3.2.3 Specifications regarding the source code organization analysis

The specifications of the software reengineering mechanism, regarding the source code organization analysis of the legacy software application, are shown in **Table 3.3** in **Chapter 3**.

Specification 3.1 requires that the mechanism should be able to locate all the source files of the application, given the top-level directory of the directory tree of the source code of the old application. When the COBOL parser starts executing, it prompts the user to enter the top-level directory of the directory tree, where the source code of the software application resides. Then it uses the UNIX command *find* to locate all the source files that exist in the directory tree and stores their names in a full path form in a text file (srcfilenames.tmp). After locating one by one all the source files of the XPERT HOTEL application, by browsing the lists of the contents of the directory tree of the application, the information that the COBOL parser identified, concerning the complete set of source files of the application, was validated.

Specification 3.2 requires a report of all the source files that are used by the old software application. This report is provided by the *List Source Files* text-view query of the GAIN browser. This query reports the names of 2537 source files that constitute the XPERT HOTEL application. In section 2.1 of this document is mentioned that the XPERT HOTEL application is constituted by 2349 source files in the hotelsrc directory and 202 source files in the ken400 directory. In section 2.2 of this document is mentioned that there are 17 duplicate source files that reside in both hotelsrc and ken400 directories of the XPERT HOTEL application. As a result the GAIN browser should report 2349 + 202 – 17 = 2534 source files. Examining the names of the source files, which are reported by the GAIN browser, it is observed that the names of the hotelsrc, dvp and ken400 directories are included in the report. This fact justifies the resulting summary of 2537 source files. The information provided by this report was validated, by browsing the contents of the directories, where the source files of the XPERT HOTEL application reside.

Finally, specification 3.3 requires a separate report for each program showing all the source files that the compiler is instructed to read in order to build the program. This report is provided by the *Include Tree* graphical query of the GAIN browser. After focusing on the main source file of a specific program and executing the *Include Tree* graphical query, all the source files that the compiler is instructed to read (COBOL COPY statement) in order to build the program are reported recursively. According to **Table A11.1** in **Appendix 11**, APOMASTN, TIMEIS and TIMEKD are the three top programs with the most include relations. The information provided by this report was validated, by reading all the source code of these programs and locating the COPY statements. This result indicates that the software reengineering mechanism passed the test successfully.

6.4 Conclusions

After studying the results of the tests performed to the software reengineering mechanism, it has been proved that the mechanism covers all the specifications, for which it has been designed and implemented, but in many cases it provides incomplete information. The incompleteness of the information is located in the fact that the COBOL parser assigns the attributes to the programs of the software application by parsing only their main source file. In order for the information to be complete it is necessary for each program to also parse all the additional source files that are used by the compiler while building the program. No other errors were detected while testing the software reengineering mechanism, using this specific software testing method.

The revision of the software reengineering mechanism, which was used to apply the tests, is revision 1.2. After the identification of the incomplete information that was provided by special reports, the software reengineering mechanism was corrected and improved. The final revision that passes all the previous tests successfully is revision 1.8. The improvements and corrections that were implemented in the intermediate revisions until revision 1.8 are described in **Appendix 12**. The revision 1.8 of the software reengineering mechanism is regarded to be stable and will be used in the analysis of the XPERT HOTEL application that takes place in the next chapter.

Chapter 7

Analysis of the XPERT Hotel Application

7.1 Introduction

The analysis of the XPERT Hotel software system with the help of the formal reengineering mechanism is the subject of this chapter. The XPERT Hotel software application is mainly a data processing application, which has been developed for the management of hotel repositories. The revision 1.8 of the software reengineering mechanism will be used in order to commit the analysis of the XPERT Hotel software system. As mentioned in **Chapter 6** named "Application and Testing of the Software Reengineering Mechanism", revision 1.8 is the final stable revision of the software reengineering mechanism, which covers all the specifications that were set in **Chapter 3** named "Requirements Analysis and Specifications". The software testing procedure, which was performed to revision 1.2 of the COBOL parser in **Chapter 6**, was also performed to revision 1.8 and all the tests were passed successfully.

The same procedure, which has been presented in the previous chapter for the revision 1.2 of the formal reengineering mechanism, is used in order to apply the revision 1.8 to the XPERT Hotel software system. After the scanning of the source code by the COBOL Parser, a TELOS database instance is created and four log files. The detailed examination of the contents of the TELOS database instance and the information included in the log files will lead the process of the XPERT Hotel software system analysis.

7.2 Description of the XPERT Hotel software system

The XPERT Hotel software system manipulates the data concerning all the products bought from the suppliers of a hotel company, the internal manipulation of these products by the hotel staff and finally, the selling of the new products to the clients of the hotel. For example, one such procedure is the food and beverage system. The hotel company buys the ingredients of the food, the chef cooks the food and finally, the food is served to the hotel clients. The developers of the XPERT Hotel software application mention that the application has been designed and implemented in such a way that it is capable of covering the needs of hotel companies that own more than one hotel. Such companies need to have complex information regarding the various segments of all their hotels combined in a single report.

Too many programmers have been involved in the development of the XPERT Hotel software system from time to time. According to information that the current developers provided the quality of the source code is not good even after the willing efforts to improve it while fixing the Y2K problem. For example, the developers believe that a lot of source files have been unlinked from the main application, without being completely removed from the set of source files, leaving behind dead code [31]. The main reason that justifies the existence of such a low quality source code is the fact that too many programmers have contributed in the development of this software system without being supervised by a fairly qualified software engineer.

7.3 Source code level analysis

7.3.1 Source code location

The source code of the XPERT Hotel application is organized in three directories. The top-level directory of the directory tree of the XPERT Hotel application is named *hotelsrc*. This directory contains 2350 source files and 1 directory named *dvp*. Directory *dvp* contains only one directory named *ken400*. Finally, directory *ken400* contains 202 source files. **Table 7.1** shows the three directories of the XPERT Hotel application and their contents.

Table 7.1 - The directories of the XPERT Hotel application source code

Directory Name	Number of source files
hotelsrc	2349 source files, 1 directory
dvp	1 directory (ken400)
ken400	202 source files

The *checkduplicates.log* log file points out that there are 17 duplicate source files. These source files exist both in the *src* and *ken400* directories of the XPERT Hotel application. It is not easy to find out which of the duplicate source files is *really* used in the working system, since there is no standard procedure for building the XPERT Hotel application. The RM/COBOL-85 manual mentions that the value of a special environment variable named RMPATH determines the order by which the source code directories are searched in order to locate the appropriate source code files during compilation. While inspecting the environment that is used for the development of the XPERT Hotel application, it was pointed out that a special Unix account is used by all the programmers, which contribute to the development of the application. Each time a programmer logs into the system using this account, the initialization scripts assign a specific value to this variable. According to this value the *src* directory has a priority against the *src/dvp/ken400* directory and as a result the *real* source code files should be the ones which are found in the *src* directory. Nevertheless, since someone could manually change the value of this environment variable right before compiling a program, it is not absolutely safe to make decisions about which of the duplicate source code files is the correct depending on the value that the user initialization scripts give to this variable. Thus, this information should be used as a hint and not as a standard. The final decision should be made by reading the duplicate source code files, compiling them, running the programs which use these source code files and finally, comparing the results against those of the real application. As mentioned by the developers of the application, directory ken400 contains the source code of programs or routines that are common to all the software applications of the company. For example, one such routine is LOXWOOD.PRC and is used to count the number of days between two certain dates. This information is useful to all software applications of the company. One question that comes out, having in mind this information, is what does the source code of a program or routine, which is commonly used to all software applications of the company, have to do among the main source code of it, which is lying in the hotelsrc directory. As mentioned earlier there are 17 source code files existing in both directories and this fact points out signs of slovenly implementation.

7.3.2 Organization of the source code

As shown in **Appendix 13** RM/COBOL source files are distinguished in three main categories. These are the main source files, the source code library files and the structured text files containing global parameters of the software application. The main source files are those that are directly compiled by the RM/COBOL compiler and specifically those that contain the PROGRAM-ID statement. The source code library files contain RM/COBOL source code that cannot be directly compiled by the RM/COBOL compiler, but they are patched into main source files at compilation time (COBOL COPY statements) instead. These source files generally contain special routines or datafile definitions that are commonly used throughout the entire software application.

XPERT Hotel software application is totally constituted by 2534 source files with a total of 643716 lines of source code. There are 1153 (45,5%) main source files with a total of 494732 lines of source code, 1289 (50,8%) library files with a total of 127267 lines of source code and 92 (3,7%) parameter files with a total of 21717 lines of text. These numbers indicate that XPERT Hotel is a large software application. A software application of this size must be very well designed and implemented in order to keep on running successfully and effectively. If a software application of this size contains design or implementation errors then the process of maintenance or further development becomes very difficult. In cases like this the need for a software tool that implements the process of automated software analysis becomes crucial.

While examining the complete list of source files (see **Appendix 13**) it was observed that many of those source code files have special suffixes. Each suffix is used to declare a special kind of source file depending on its usage. For example all the source code files that have the '.prc' suffix contain global procedures that are used by many programs of the software application. **Table 7.2** presents the five categories of source files identified throughout the XPERT Hotel software application according to the suffix of their name.

It is now possible to check the usage of these libraries throughout the entire software application and make decisions regarding the normal development of the software application. For example it is expected to find out that all the definitions of the datafiles of the software application are stored in special .SEL and .REC source code files and all the programs that access a datafile get its definition from the same library file. This example is studied in more detail in section five of this document where the database analysis of the XPERT Hotel software application occurs. Here it is just mentioned that many datafiles, according to **Appendix 17**, have several definitions throughout the source code of the XPERT Hotel application. This fact indicates data mismatches, runtime errors and generally unexpected conditions, which usually lead to program crashes and data corruption.

Table 7.2 – The source code file categories of the XPERT Hotel application

Prefix	**Description of category**
.CBL	The main source file of a complete program
.PRC	The source file contains procedures used by many programs of the software application
.WOR or .WS	The source file contains variable declarations (Working storage section of RM/COBOL-85 language), which are that same to many different programs of the software application
.SEL	The source file contains datafile structure definitions (Input output section of RM/COBOL-85 language)
.REC	The source file contains datafile record descriptions (File section of the Data division of RM/COBOL-85 language)

The fact that a systematic organization of the source code libraries inside the software application exists, declares that this software application has been well designed in the beginning. The fact that these source code libraries are not systematically used throughout the entire software application declares that this software application was bad implemented.

7.3.3 Dead code

The compilation process of each main source code file of the XPERT Hotel software application referred that 82 of 1153 (7,11%) source code files did not compile successfully. These main source files consist of 85059 lines of source code out of 494732 (17,2%) total lines of source code in main source files. These main source code files have to be programs that were never successfully completed, because of change of plans in the development of the software application, or stopped being used for some reason and so they were never maintained. Some of them are also backups of programs that came out during the development or maintenance process of the actual programs and the programmers did not erase them in the end. Such main source files usually have the .STD suffix, while the actual source code file has the .CBL suffix.

While checking each source code file of the XPERT Hotel software application it was observed that many of them where not main source files of a complete program inside the application nor were used by any program in the form of a library. This fact means that these source files were used in the past as library files and now they have been unlinked from the main source code files leaving behind dead code. This form of dead code occupies disk space and makes the development of the software application more complex since the programmers don't know for sure if these source code files are actually used in the application or not. Nevertheless, at compilation time this dead code is not included in any active program of the software application and thus it does not consume computer power at run time. As shown in **Appendix 13** 382 useless library source code files where identified among the source files of the XPERT Hotel software application. These are 29,64% of the total library source code files of the application or 15,07% of the total application source files. After counting the source code lines included in these source files it was observed that 43463 source code lines are included in these useless files out of totally 127267 (34,15%) source code lines included in source code library files.

The main source code files that do not compile successfully are useless files and their existence increases the level of difficulty in the process of distinguishing the dead code apart from the healthy source code. When determining whether a library source file is still used by the application it is necessary to check if any other main source file or library source file includes the code of this source file at compilation time (COBOL COPY statements). At this time it is not checked if the main program compiles successfully or not. As a result it is necessary to remove all the main source files that do not compile successfully before the COBOL parser starts scanning the source code of the software application. For research reasons the COBOL parser scanned the source code of the XPERT Hotel software application two times. The first time including the broken main source code files and the second without them. The second time where identified 64 more unlinked library files (dead code) out of 318 (20,12%) unlinked library files that were identified the first time.

In many programs of the XPERT Hotel software application too many datafiles are defined in their source code, which are not actually accessed by the program. This fact can easily be observed by executing the 'open tree' graphical query of the GAIN browser for each main source file of the application. The reason for this fact is that the datafiles of the application have been categorized and there are certain source files that define all the datafiles of each category. For example the source file PRO.REC contains all the record descriptions of the datafiles concerning the suppliers of the hotel. When a new program is built in order to manipulate data concerning the hotel suppliers all the definitions of the files concerning the hotel suppliers are copied into it. However, only a few of the defined files are actually affected by the program. This tactic generates dead code and the only way to remove it is to identify which datafiles are actually affected by the program and keep only those datafile definitions in its source code. Otherwise the program occupies much more memory at run time and executes much slower.

7.4 Program level analysis

7.4.1 The complete set of programs

As shown in **Appendix 14** the XPERT Hotel software application consists of 1054 programs. 15 of these programs have two main source files while one of them has three. As a result there is a total of 1071 main source files and this information is in accordance to **Appendix 13** where it is mentioned that the XPERT Hotel application has 1071 main source files (not including those that do not compile successfully).

RM/COBOL has two mechanisms when calling an external program. The first is used to call a program, which is part of a library consisting of several programs. The second is used to call a separately compiled program. In the first case the PROGRAM-ID paragraph of the main source file defines the name of the program that is used in order to identify it. In the second case, the name of the main source file of the program is used without the .CBL or .COB suffix. In both cases the name of the program is case insensitive. The XPERT Hotel software application does not have any program libraries including more than one programs. It is parted of separate programs and the second mechanism is always used when calling an external program. This information provides a clue that the 17 duplicate main source files, which exist inside the XPERT Hotel application are probably useless (dead code), but which of those are the good ones and which are useless? They all compile successfully so only the XPERT Hotel programmers may know the answer. Probably the one that has the latest modification date is the good one but this is not for sure. Since there is no documentation describing the organization and architecture of the XPERT Hotel software application, the only way to be sure in this case is to read the source code, then compile each main source file and check the program against the one of the working application.

7.4.2 Program names

Appendix 14 lists all the programs of the XPERT Hotel application referring to them using the name of the main source file without the .CBL or .COB suffix. As mentioned above this is the name that is used in order to call a program from another program inside the software application. The name that is given to a program in the PROGRAM-ID paragraph of its main source file is also available but not in **Appendix 14**. in order to find out the name, which is assigned to a program in the PROGRAM-ID paragraph it is necessary to start the TELOS database and the GAIN browser, edit the program name in the Query Target text line editor and execute the star view graphical query. This query provides all the attributes of a program. The 'name' attribute shows the name that was found in the PROGRAM-ID paragraph of the main source file of the program when it was scanned line by line by the COBOL parser (see **Figure 7.1**).

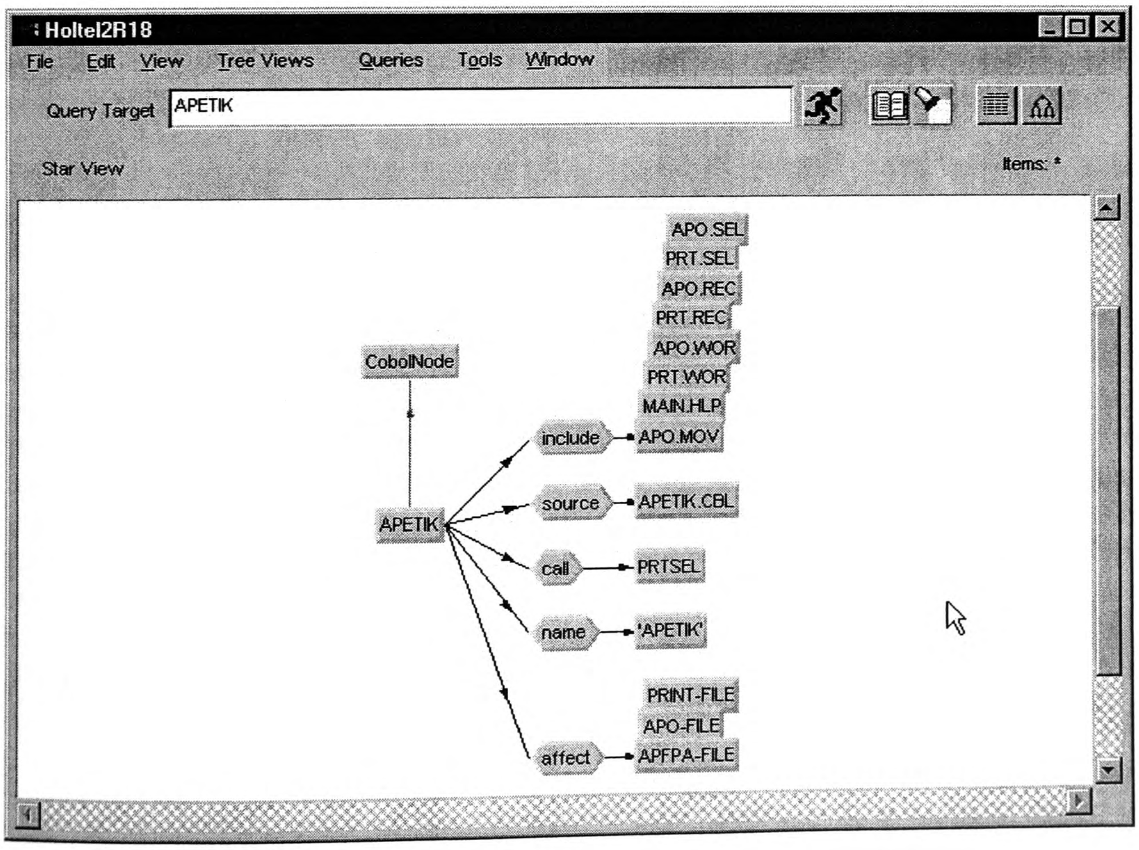


Figure 7.1 – Program attributes as provided by the GAIN Browser.

Many programs of the XPERT Hotel application have different name in the PROGRAM-ID paragraph than the name of their main source file. This fact is not abnormal for RM/COBOL development but the general policy in the development of this application is to have the same name for both the main source file and the PROGRAM-ID paragraph. Nevertheless, since the name defined in the PROGRAM-ID paragraph is not used, as mentioned earlier, in order to call the program from another one inside the application, many of the main source files are just copies of other main source files of similar programs that where simply modified in order to create the new program. Thus, the name defined in the PROGRAM-ID paragraph remained the same as the one in the old program.

7.4.3 Control flow description

The XPERT Hotel software system is constituted by 1054 standalone (separately compiled) programs. These programs are loaded using a special menu driven system, which is implemented by the MENRTS program. This program reads a configuration file (Xbas.mtd) describing the programs that are available in the menus and can be selected by the user. As a result in order to study the control flow of the software application it is necessary to study the control flow of each standalone program.

The 'call' attribute of a CobolNode (program) object inside the semantic network points out that a COBOL CALL statement was identified within the main source file of the program and joins the two programs, the calling and the called program. It is also possible for a program to include one or more additional source files and inside the source code of these additional source files to have CALL statements. The 'invoke' attribute of a SourceFile object inside the semantic network points out that a COBOL CALL statement was identified within the source file and joins the source file with the called CobolNode object.

Appendix 18 presents the information given by the GAIN browser after executing the 'call tree' graphical query for the PRSTAT program. The graph contains 53 objects, which are programs that are called at run time or additional source files used at compilation time. This is a small example that was chosen just because the output graph fits in a single page. Many programs of the XPERT Hotel software application produce a much bigger graph when executing the 'call tree' graphical query to them. For example, the graph produced for the TIMEIS program contains 533 objects! It is much easier for a new programmer to modify a program like this after studying the produced graph than to read all of the source code. Even the old developers of the application may have forgotten many programs that are called by such a big program or many additional source files (libraries) that are used in order to compile it.

7.4.4 Database access of each program

It is possible to have datafile access in a program and the source code statements that implement the datafile access to exist in additional library files that are used at compilation time. It is also possible to have run time calls to external programs, which in turn access additional datafiles. As a result it is very difficult in large programs, which use many additional library files at compilation time or make many external program calls at run time, to find out the complete set of datafiles, which are affected at run time by the program. In such a case the value of the information, which is presented by the 'affect tree' query of the GAIN browser is inestimable. One small example of this case is presented in **Appendix 18**. The graph, which is produced by the GAIN browser after executing the 'affect tree' graphical query to the PRSTAT program contains 61 software objects. These software objects are additional libraries that are used at compilation time, external programs called at run time or datafiles affected at run time. At this point it should be mentioned that if a software object is referenced more than once, the GAIN browser counts the object as many times as it is referenced by the other objects of the semantic network in the total number of objects retrieved by the query. Then it presents the object on the screen once and the total number of counts can be justified by the total number of 'TO' relations of the object. For example EKT-FILE is affected both by the PRSEL and the PRTEIS external programs, which are both called at run time by the PRSTAT program. The GAIN browser has counted twice this datafile in the total number of software objects included in the PRSTAT program but it has presented the software object on the screen once showing two affect relations, one with the PRTSEL and another with the PRTEIS external programs.

The PRSTAT program is a small program, which was chosen as an example because it produces a representative and at the same time small graph that fits in a single page. Nevertheless, the usual situation in the XPERT Hotel software application is to have a lot more than 61 software objects related with one program. The TIMEIS program is one of the biggest programs of the XPERT Hotel application and the graph that is produced by the GAIN browser, after executing the 'affect tree' graphical query on this program, is constituted by 651 software objects related with each other. It would really need a big plotter to print such a big graph on the paper, but the information that comes out of such a graph is invaluable. It would really take too much time and manpower to a new programmer to get all this information by just reading the source code of this program, the source code of all the external libraries that are used by it at compilation time, the source code of all the external programs that are called by it at run time and finally the source code of the external libraries that are used at compilation time by the programs that are called at run time. Even the old programmers of the application may not have such a higher-level view of a big program, like TIMEIS is, in their mind. In cases like this it is impossible for a human to have in his mind all this complex information and it is very easy to make an error when modifying such a program and spend much more time than necessary to find out where the error comes from.

7.4.5 Datafile definitions

The definition of the structure or the record description of each datafile within the source code of a program plays pivotal role in software development when using 3rd GL's such as COBOL. Specifically, in the XPERT Hotel software application the definitions of the most datafiles exist in separate library source code files, which in general are used by the programs of the application at compilation time. Nevertheless, many programs have their own definitions for some datafiles. **Appendix 15** provides the complete list of source code files, which contain datafiles record descriptions or datafile structure definitions. **Appendix 17** provides the complete list of datafiles and for each datafile all the source code files that contain record descriptions or structure definitions for the datafile. The fact that many programs have their own definitions for some datafiles, may cause data corruption but it is also possible to be on purpose, in order to implement some special interventions to the datafile. For this reason it is necessary to know whether a program has a special definition for a specific datafile, which is affected by it and in this case to inspect whether this special definition should exist in the source code of the program or not. The information provided in **Appendix 15** comes out by focusing on each source file and checking its *select* and *define* relations. The information provided in **Appendix 17** comes out by focusing on each datafile and checking its *select* and *define* relations.

According to **Appendix 15**, 163 main source files contain datafile structure definitions or datafile record descriptions. According to **Appendix 13** the total number of main source files of the XPERT Hotel software application is 1071, which means that the 15,21% of the application main source files have their own definitions for some datafiles. Additionally there are 87 source files, which have been unlinked from the main application or are main source files that do not compile successfully (dead code in both cases) that also contain datafile structure definitions or datafile record descriptions. After excluding the dead code there are 123 source code files containing datafile record descriptions, 98 source code files containing datafile structure definitions and 131 source files containing both datafile record descriptions and datafile structure definitions. After excluding the main source code files there are 121 library source code files that contain datafile record descriptions and 68 that contain datafile structure definitions.

7.5. Database level analysis

7.5.1 The complete set of datafiles

The XPERT Hotel software application has 286 datafiles, which are listed in **Appendix 16**. This information comes out after executing the 'List All Datafiles' text-view query of the GAIN browser. An observation that comes out after reading the list of the datafile names in **Appendix 16** is that the most datafiles have the –FILE suffix in their name. This tactic was followed in order to easily recognize the name of a datafile from the name of a routine or external program call within the source code of the software application. Another observation is that many of these datafiles have similar names. One example is AGD-FILE, AGDXFILE and WAGD-FILE. In this example the filename that is used to store the data into the filesystem (physical storage media) is the same (agdelt.dat) for each datafile. These datafiles have also similar structure definitions and the same record description. This observation points out that these three datafiles are not completely separate. It is one main datafile (AGD-FILE) and the other two are just used to commit special interventions in some abnormal circumstances and are not used within the normal control flow of the application. There are totally 37 datafiles that have the same name with other datafiles plus the letter W in the front of their name and other 36 that have the XFILE suffix instead of the –FILE suffix in their name. This observation decreases the actual total number of datafiles from 286 to 213 (25,52%).

7.5.2 Access attributes

Specification 1.2 (see **Chapter 3** named "Requirements Analysis and Specifications") of the software analysis mechanism refers that all the datafiles of the analyzed software application should be reported and additionally, for each datafile all the programs that access the datafile should also be reported. This information can be retrieved by the GAIN browser after focusing in a specific datafile and executing the 'Tree View' graphical-view query. The 'affect' relations are those that identify the programs that include an OPEN statement for the datafile in their main source file. The 'open' relations are those that identify the source files that include an OPEN statement for the datafile. In order to avoid executing 286 times the 'Tree View' graphical-view query (one for each datafile) a new text-view query has been created named 'List All Datafile Affects'. This query produces a three column report. In the first column appears the name of each datafile. In the second column, beside the name of each datafile, appear the names of all the programs that include an OPEN statement for the datafile in their main source file. In the third column appear the names of the source files that include an OPEN statement for the datafile.

By checking the contents of the report produced by the "List All Datafile Affects" text-vew query of the GAIN Browser for the XPERT Hotel software application, it can easily be identified which datafiles are mostly accessed by the programs of the application. **Table 7.3** shows the top ten datafiles, which are accessed by the most programs. While looking at the contents of **Table 7.3**, it must be mentioned that PRINT-FILE, which is affected by 171 programs, has no standard definition as it is just used to produce printing output. ANAL-FILE, which is affected by 113 programs, uses two different external filenames to store the data in the physical storage media. According to the information provided in **Appendix 17** these filenames are "pelanal.dat" and "promanal.dat". The contents of "pelanal.dat" include information regarding the customers of the hotel company, while the contents of "promanal.dat" include information regarding the suppliers of the hotel company. In this point it must be mentioned that it is not a good tactic to share the same name between two separate datafiles because this way it is impossible to access both datafiles from the same program. Nevertheless, the developers of the XPERT Hotel software application insist that it will never be necessary to access the contents of "pelanal.dat" and "promanal.dat" at the same time from the same program since no program handles customer and supplier data at the same time. On the other hand, too many programs and library files affect those datafiles and thus it is a time spending procedure to change the internal name assigned to them. This change is also risky since it is possible to have program run time errors and crashes by a single mistake while implementing this change in a source file. The expence of time and the risk for errors are the two main reasons why these two datafiles still share the same name in the XPERT Hotel software application.

Table 7.3 – Top ten accessed datafiles

DataFile	**Affects**
APO-FILE	382
APOMA-FILE	261
APANAL-FILE	206
APTM-FILE	196
APTEAM-FILE	187
APO1-FILE	182
PELAT-FILE	176
PRINT-FILE	171
PROM-FILE	154
ANAL-FILE	113

The report produced by the "List All Datafile Affects" text-view query shows that there are 43 datafiles out of 286 total (15,03%), which are defined in the source code of the XPERT Hotel software application, but have no "affect" or "open" relations. This fact indicates that their contents are not accessed by any program of the software application and as a result they are useless datafiles. A normal explaination for the existence of those datafile definitions inside the source codeis that in the past there might be some programs accessing their contents, but these programs were unlinked some other time from the main application and their source code was destroyed. This observation leads to the result that the source code, which is used for those datafile definitions, is also dead code that must be removed.

7.5.3 Datafile structure definition and datafile record description analysis

Specifications 1.3 & 1.4 of the software analysis mechanism refere that for each datafile there should be a separate report presenting all the programs or library source code files, which contain structure definitions or record descriptions for the datafile. This information can be retrieved for each datafile fom the GAIN browser by focusing on a specific datafile and then executing the "Tree View" graphical-view query. The "select" relations of the datafile point out which source code files contain structure definitions for the datafile and the "define" relations of the datafile point out which source code files contain record descriptions for the datafile. Since there are too many datafiles in the XPERT Hotel software application, a new text-view query has been created, which is named "List All Datafile Definitions". The report produced by this query has three columns. In the first column appears the name of each datafile. For each datafile in the second column appear all the source files that contain structure structure definitions for the datafile. Finally, for each datafile in the third column appear all the source files that contain record descriptions for the datafile.

Appendix 17 presents the report produced by the "List All Datafile Definitions" text-vew query for the XPERT Hotel software application. The red source files are those that are characterized in **Appendix 13** as dead code. The green source files are those that are main source files of a program. For each datafile that has more than one structure definitions or record descriptions, a comparison among them has taken place and in the fourth column of the report is mentioned if any mismatches were found or not. Finally, in the fifth column of the report is presented the external name of the datafile, which is used to store the actual data in the physical storage media. The information that appears in columns 4 & 5 was retrieved by just reading the appropriate source code files. In this case the software analysis mechanism only helped in the process of identification of the appropriate source code files for each datafile in order to retrieve the desired information.

The report, which is presented in **Appendix 17** points out that there are 33 datafiles out of 286 total (11,53%) that are defined only in source code files that are characterized in **Appendix 13** as dead code. This fact points out that these datafiles are useless. This information can be combined information provided in section 5.2, which mentions that there are 43 datafiles that are not affected by any program of the XPERT Hotel software application. The total number of datafiles that are not affected by any program or their definitions lye in dead code is 60 out of 286 total (20,97%), which means that there are 16 datafiles that are not affected by any program and their definitions lye in dead code.

Table 7.4 reports the top ten datafiles, for which the most datafile structure definitions were identified throughout the XPERT Hotel software application source code. It has to be mentioned that SEQ-FILE, PRINT-FILE and TMP-FILE are just used as temporary files and have no standard definitions or external names. For example the PRINT-FILE file is used to store printing output. The external name of the datafile is usually passed as a parameter from the program that created the printing output to another external program that is responsible for passing the printable data, which are the contents of PRINT-FILE datafile, to the printing device. This is the standard printing procedure of the XPERT Hotel software application.

Table 7.4 – Top ten datafiles with the most structure definitions

Datafile Name	**Structure Definitions**
SEQ-FILE	47
PRINT-FILE	33
APO-FILE	22
PELAT-FILE	20
WAPO-FILE	18
TMP-FILE	17
PROM-FILE	17
WANAL-FILE	16
ANAL-FILE	16
ARTHRO-FILE	16

Another observation that can be made by examining the contents of **Table 7.4** is that two couples of datafiles are identified among them, which are: APO-FILE with WAPO-FILE and ANAL-FILE with WANAL-FILE. As mentioned in section 5.1, datafiles that have the same name with an additional W prefix are redefinitions of the same datafile, which are used in order to commit special datafile interventions. The fact that a datafile appears in the contents of **Table 7.4** indicates that too many special interventions take place to the contents of that datafile or the datafile is accessed by too many bad implemented programs. The fact that these two couples appear in the contents of **Table 7.4** indicates that much too many special interventions take place to the contents of APO-FILE and ANAL-FILE.

Table 7.5 reports the top ten datafiles, for which the most datafile record descriptions were identified throughout the XPERT Hotel software application source code. The same observation for datafiles SEQ-FILE, PRINT-FILE, and TMP-FILE that was made earlier while studying the contents of **Table 7.4**, regarding the datafile structure definitions, can also be made for these datafiles regarding their record descriptions, since they are also included in **Table 7.5**.

The two datafile couples that were identified in **Table 7.4**, APO-FILE with WAPO-FILE and ANAL-FILE with WANAL-FILE, are also identified within the contents of **Table 7.5**. This fact reensures the observation that much too many special interventions take place to the contents of APO-FILE and ANAL-FILE datafiles.

Table 7.5 – Top ten datafiles with the most record descriptions

Datafile Name	**Record Descriptions**
SEQ-FILE	38
PRINT-FILE	33
WAPO-FILE	18
APO-FILE	16
PELAT-FILE	16
WANAL-FILE	16
TMP-FILE	15
WAPANAL-FILE	14
ANAL-FILE	13
APTEAM-FILE	12

As mentioned in section 3.2, it is a good tactic to have two separate library source code files regarding the definition of each datafile. One to contain the structure definition of the datafile and another to contain the record description of the datafile. The normal procedure, when a standalone program needs to access the contents of a datafile, is to include these two source files, which define the structure and record description of the datafile in its main source code (COBOL COPY statements). Nevertheless, many datafiles of the XPERT Hotel software application do not have separate librarly source code files to include their definitions. **Table 7.6** presents the datafiles that actualy have two separate library source code files, one including just their structure definition and one including just their record description. As shown in **Table 7.6**, 27 datafiles out of 286 total (9,44%) have separate library source code files for their definition.

As mentioned in sectrion 5.2 of this document, ANAL-FILE uses two different external filenames (pelanal.dat and promanal.dat according to **Appendix 17**) to store the data in the physical storage media. This fact justifies the existence of two different source files for the structure definition and for the record description of this datafile in the contents of **Table 7.6**.

Table 7.6 – Datafiles which have their definitions in separate library source code files

Datafile	**Structure Definition**	**Record Description**
ANAL-FILE	PEL.SEL, PR2.SEL	PELANAL.REC, PRANALREC.CBL
APANAL-FILE	APANAL.SEL	APANAL.REC
APO-FILE	APOMAST.SEL	APOMAST.REC
APO1-FILE	APOM1.SEL	APOM1.REC
APOGR-FILE	APOGR.SEL	APOGR.REC
APOMA-FILE	APOMA.SEL	APOMA.REC
APTIMCH-FILE	APTIMCH.SEL	APTIMCH.REC
APTM-FILE	APTM.SEL	APTM.REC
APUPD-FILE	APUPD.SEL	APUPD.REC
APTR-FILE	APTR.SEL	APTR.REC
ASUPD-FILE	ASUPD.SEL	ASUPD.REC
EPIT-FILE	EPI.SEL	EPI.REC
GRL-FILE	GRL.SEL	GRL.REC
MENSPT-FILE	MENSPT.SEL	MENSPT.REC
MENU-FILE	MEN.SEL	MEN.REC
ORPH-FILE	ORPH.SEL	ORPH.REC
ORPI-FILE	ORPI.SEL	ORPI.REC
PARM-FILE	PARM.SEL	PARM.REC
PELAT-FILE	PELAT.SEL	PELMAST.REC
SYNT-FILE	APSYN.SEL	APSYN.REC
SYSENV-FILE	SYSENV.SEL	SYSENV.REC
TAMDEL-FILE	TAMDEL.SEL	TAMDEL.REC
TRA-FILE	TRA.SEL	TRAREC.CBL
TRWH-FILE	TRWH.SEL	TRWH.REC
VAR-FILE	VAR.SEL	VAR.REC
WAPO1-FILE	WAPO1.SEL	WAPO1.REC
WINSEL-FILE	WINSEL.SEL	WINSEL.REC

An observation, which can be made by studying the contents of **Appendices 15 & 17**, is that there are many library source files that contain structure definitions or record descriptions for more than one datafile. This points out that the developers of the XPERT Hotel software application have created groups of datafiles. Each time they need to access a specific group of datafiles, they just include the appropriate library source code file into the main source file in order to define all the datafiles of the group. Nevertheless, it is also observed that in many programs too many datafiles are defined but only a few of them actually affected by the program. This observation points out that when the programmers wanted to access the contents of a subset of the group of datafiles, they included all the code, which defines the whole group of datafiles, into the main source file of the program. This is a bad programming technique, which leads to dead code that makes a programm bigger in size, memory consuming and slower at execution time. **Table 7.7** presents the groups of datafiles that were identified in the XPERT Hotel software application and the library source code files that led to the identification of each group.

Table 7.7 – Groups of Datafiles identified in the XPERT Hotel software application

Group	Description	Sourcefiles
A	Datafiles containing information concerning the hotel repositories	APO.SEL, APO.REC
B	Datafiles containing information concerning the hotel clients	PEL.SEL, PEL.REC
C	Datafiles containing information concerning the hotel suppliers	PRO.SEL, PRO.REC
D	Datafiles containing information concerning the hotel buys	AGO.SEL, AGO.REC
E	Datafiles containing information concerning the hotel sales	TIM.SEL, TIM.REC

7.5.4 Relation information

According to specification 1.5 (see **Chapter 3** – Requirements Analysis and Specifications), this mechanism should provide hints about relations, which may possibly exist between the datafiles of the software application. In **Chapter 6** it is mentioned that the method that the software reengineering mechanism provides for this purpose is to isolate the record descriptions of the datafiles and compare them. The fields of different datafiles that have the same size and similar names are candidates for the implementation of a relation. After all the candidate relations are identified, the validation of existence of these relations is investigated by browsing the data contained in the possibly related datafiles and checking the contents of the relating fields.

Using this method, 6 relations where easily identified in the XPERT Hotel database. The datafile, which was mostly related with other datafiles, is APANAL-FILE. The external name for this datafile is apanal.dat and is used to keep detailed information for each product sale taking place in the hotel. It is possible to exist more datafile relations in the database of the XPERT Hotel software application than those already identified, since the investigation of the hints that lead to the identification of these relations was not extensive. Nevertheless, the most complicated and important set of datafile relations is the one including APANAL-FILE and this information was validated by the developers of the application.

Table 7.8 presents the relations that were identified in the XPERT Hotel database. Columns 1 and 4 of the table contain the names of the related datafiles. Columns 2 and 3 contain the names of the relating fields.

Table 7.8 – Datafile relations identified in the XPERT Hotel software application

APANAL-FILE	PELAT-KOD	ANAL-KOD	ANAL-FILE
	PELAT-KOD	PELAT-KEY	PELAT-FILE
	APANAL-KOD	APO-KEY	APO-FILE
	APANAL-APTM	APTM-KEY	APTM-FILE
ANAL-FILE	ANAL-KOD	PELAT-KEY	PELAT-FILE
ANAL-FILE	ANAL-KOD	PROM-KEY	PROM-FILE

As mentioned in sectrion 5.2 of this document, ANAL-FILE uses two different external filenames (pelanal.dat and promanal.dat according to **Appendix 17**) to store the data in the physical storage media. This fact justifies the appearance of this datafile twice in the contents of the 1st column of **Table 7.8**. The external name of ANAL-FILE related with PELAT-FILE is pelanal.dat and the external name of ANAL-FILE related with PROM-FILE is promanal.dat. ANAL-FILE also appears related with APANAL-FILE according to **Table 7.8**. The external name of ANAL-FILE related with APANAL-FILE is pelanal.dat.

7.6 Conclusions

The XPERT Hotel software system manipulates the data concerning all the products bought from the suppliers of a hotel company, the internal manipulation of these products by the hotel staff and finally, the selling of the new products to the clients of the hotel. The application has been designed and implemented in such a way that it is capable of covering the needs of hotel companies that own more than one hotel and need to have complex information regarding the various segments of all their hotels combined in a single report.

As mentioned in section 3.2 of this chapter, the XPERT Hotel software application consists of 643.716 lines of RM/COBOL-85 source code. 45,5% of the source code is located in main source files, 50,8% in source code library files and 3,7% in parameter files. The application has 1054 standalone, separately compiled programs, which are loaded directly by a special menu-driven mechanism or indirectly, in the form of external program calls. Finally it must be mentioned that the XPERT Hotel database consists of 286 datafiles. All these facts make XPERT Hotel a large data-processing software application of industrial level.

The source files of the application are grouped into 5 categories according to **Table 7.2**. Nevertheless, a big part of the application does not follow this grouping. In addition, according to **Appendix 13**, approximately 22% of the application source code is dead code.

As shown in **Table 7.7**, a grouping of the datafiles takes place in the source code of the application. This is a good organizational tactic but the fact that many programs contain in their source code the definitions of all the datafiles of a group, while affecting only a few of them, shows signs of bad implementation. In addition, according to section 5.3, approximately 21% of the defined datafiles are useless, meaning that they are not affected by any program or their definition lies in dead code.

The conclusion that comes out, after studying the evidence reported in this chapter, is that XPERT Hotel is a large data-processing software application of industrial level that was well designed in the beginning but it was not perfectly implemented and maintained. This fact is typical to software applications of this level and is one of the main motives for the development of this research project.

Chapter 8

Conclusions and Future improvements

8.1 Conclusions

8.1.1 Philosophical matters regarding software reengineering

The last ten years of this century have been characterized as a period of software crisis. The need for information is growing day by day and the formal development tools have been proven insufficient to serve this need. This led software engineers to the creation of new technologies that would be more efficient in the manipulation of the data and the development of software systems. Large software systems have been developed using these new technologies. These systems have been proven efficient and satisfying. However, what will happen with all those large software applications that have been developed in the past, under formal development tools such as 3rd Generation Languages (GLs)? In the real world there are too many software applications, developed using 3rd GLs, still working in business. There are many reasons why these applications need to be modified in order to keep on running effectively. The introduction of EURO as global currency in Europe is a well-known problem concerning these old applications. In most cases the documentation describing the requirements, design and implementation of the legacy software systems does not exist or is too poor to make sense. This thesis provided a mechanism to regain design and implementation information of a software system examining its source code. There are certain problems that occur in the reengineering process of large data-processing software application systems, which have been developed using legacy software development tools, such as the complexity of the code combined with the absence of computer-aided software engineering (CASE) tools. This project studied the requirements, design and developed a software system that helps, by means of providing critical information, to speed up this process. A real data-processing software system, which has been developed using the RM/COBOL–85 computer language, has been analyzed using this mechanism.

8.1.2 Who would need this mechanism and why

Billions of lines of 3rd GL source code have been written for the development of millions of legacy software applications. These legacy applications now are very difficult to maintain or further develop. The development tools of 3rd GLs used for the implementation of software applications (mainly those that manipulate big amounts of data) are primitive compared to the 4th GL development tools. Thus, a lot of time and manpower is wasted in the maintenance or further development of legacy applications. Thus it is necessary to gain the design and implementation analysis of a legacy software application in small period of time in order to reengineer it or create a new one based on the recovered analysis.

Due to the robust and effective tools that the 4th GLs provide in the development of software applications and to the simple solutions that have been given to problems that 3rd GLs face, there are yet not many software developers left to deal with all those applications that have been developed using 3rd GLs. In addition, 3rd GLs are not taught any more in modern schools and universities. As a result, all new software developers work using modern tools and techniques. Nevertheless, there are many applications that have been developed in the past using old development tools and now only need a minor alterations or improvements. In this case all the necessary information that can be extracted from the old application, is required in a short period of time. The proposed formal software reengineering mechanism helps locating the problem, in order to commit the necessary changes, having as a result an improved, working application.

Other software developers could use this tool in order to keep version information for their applications. This tool can help them to keep truck of the characteristics of every version of their application, by scanning the sources of their application in a regular basis and keeping records of the databases, which are produced each time.

8.1.3 Documentation

In this chapter the analysis and study of existing documentation in the research domain of reengineering and reuse of legacy software systems has been investigated and presented. The research domain of reengineering and reusing legacy software systems includes general software engineering concepts and especially reengineering concepts. Several computer theories have been developed to provide answers to serious questions in this research domain. The most important of those theories are the theory of software objects, the software component classification theory, the representation theory, the theory of patterns and the theory of software repositories.

The scientific papers found in the bibliography search, in the area of software reengineering, were divided into three categories. The first category includes those papers that refer to general reengineering issues. The second category consists of papers that provide an approach to software reengineering through the theory of patterns. Finally, the third category contains two papers that provide techniques for testing the results of a software development process. Through the presentation of these papers the problems that other engineers have faced in the area of software reengineering and the solutions that they propose were outlined. These problems include theoretic issues related to software reengineering as well as applied solutions provided to specific demands.

Finally, three typical systems that have been developed for software reengineering purposes and are available for commercial use have been presented. These are the DMS reengineering toolkit, the tools from George and James Software Inc and the Metamorphic COBOL Converter. The fact that some software companies have already developed and marked some products concerning this research domain, justifies the purposes for the implementation of this project.

8.1.4 Testing the software reengineering mechanism

After studying the results of the tests performed to the revision 1.2 of the software reengineering mechanism, it has been proved that the mechanism covers all the specifications, for which it has been designed and implemented, but in many cases it provides incomplete information. The incompleteness of the information is located in the fact that the COBOL parser assigns the attributes to the programs of the software application by parsing only their main source file. In order for the information to be complete it is necessary for each program to also parse all the additional source files that are used by the compiler while building the program. No other errors were detected while testing the software reengineering mechanism, using this specific software testing method.

After the identification of the incomplete information that was provided by special reports, the software reengineering mechanism was corrected and improved. The final revision that passes all the previous tests successfully is revision 1.8. The improvements and corrections that were implemented in the intermediate revisions until revision 1.8 are described in **Appendix 12**. The revision 1.8 of the software reengineering mechanism is regarded to be stable and was used in the analysis of the XPERT HOTEL application that takes place in the next chapter.

8.1.5 The XPERT Hotel software application

The XPERT Hotel software system manipulates the data concerning all the products bought from the suppliers of a hotel company, the internal manipulation of these products by the hotel staff and finally, the selling of the new products to the clients of the hotel. The application has been designed and implemented in such a way that it is capable of covering the needs of hotel companies that own more than one hotel and need to have complex information regarding the various segments of all their hotels combined in a single report.

The XPERT Hotel software application consists of 643.716 lines of RM/COBOL-85 source code. 45,5% of the source code is located in main source files, 50,8% in source code library files and 3,7% in parameter files. The application has 1054 standalone, separately compiled programs, which are loaded directly by a special menu-driven mechanism or indirectly, in the form of external program calls. Finally it must be mentioned that the XPERT Hotel database consists of 286 datafiles. All these facts make XPERT Hotel a large data-processing software application of industrial level.

The source files of the application are grouped into 5 categories according to **Table 7.2**. Nevertheless, a big part of the application does not follow this grouping. In addition, according to **Appendix 13**, approximately 22% of the application source code is dead code.

As shown in **Table 7.7**, a grouping of the datafiles takes place in the source code of the application. This is a good organizational tactic but the fact that many programs contain in their source code the definitions of all the datafiles of a group, while affecting only a few of them, shows signs of bad implementation. In addition, according to section 5.3, approximately 21% of the defined datafiles are useless, meaning that they are not affected by any program or their definition lies in dead code.

All the mentioned facts indicate that XPERT Hotel is a large data-processing software application of industrial level that was well designed in the beginning but it was not perfectly implemented and maintained. This fact is typical to software applications of this level and was one of the main motives for the development of this research project.

8.2 Future improvements

A software application may be developed using more than one computer language. Nevertheless, the software reengineering mechanism was designed and implemented only to recognize and present the analysis of RM/COBOL-85 programs. The fact that it would be necessary to update the mechanism in order to support more than one computer language has already been faced. The model that represents the analysis of a software application includes superclasses, which contain the classes of COBOL programs (CobolNode) and COBOL source code files (SourceFile). These superclasses are NodeType for CobolNode and SourceType for SourceFile. Into these superclasses could be added two more classes such as PascalNode as a subclass of NodeType class for the Turbo Pascal programs and PascalSource as a subclass of SourceType class for the Turbo Pascal source code files.

Not only the model, but the source code parser should be updated in order to recognize the source code of the additional 3rd GL. The current implementation of the source code parser does not support this improvement so it would require to add an additional segment to recognize Pascal keywords. Nevertheless, it would be very interesting to improve it by making it support simultaneously another 3rd GL such as Turbo Pascal.

References

[1] Barbey, S.; Buchs, D.; Gaudel, M.; Matte, B.; Peraire, C.; Fosse, P.; Waeselynck, H. (1998): *From Requirements to Tests via Object-Oriented Design*, DeVa Third Year Report, December 1998, pp. 331-384

[2] Barbey, S.; Buchs, D.; Peraire, C. (1996): *A Theory of Specification-Based Testing for Object-Oriented Software*, Proc. 2nd European Dependable Computing Conference (EDCC2), Taormina, Italy, October 1996

[3] Boudier, G. (1988): *An overview of PCTE and PCTE+*, Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, ACM Software Engineering Notes, Vol. 13 No. 5, November 1988, pp. 248-257

[4] Burd, E.; Munro, M. (1998): *A Method for the Identification of Reusable Units through the Reengineering of Legacy Code*, The Journal of Systems and Software, Vol. 44 No. 2, December 1998, pp. 121-134

[5] Canfora, G.; Cimitile, A.; De Lucia, A.; Di Lucca, G. (2001): *Decomposing Legacy Systems into Objects: an Electic Approach*, Information and Software Technology, Vol. 43 No. 6, May 2001, pp. 401-412

[6] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1993): *Design Patterns: Abstraction and Reuse of Object-Oriented Design*, ECOOP 1993, pp. 406-431

[7] Gravley, J.; Lakhotia, A. (1996): *Identifying Enumeration Types Modeled with Symbolic Constants*, Proc. 3rd Working Conference on Reverse Engineering, © IEEE CS Press, November 1996

[8] Huat, E.; Sarshar, M.; Poutney, D. (1995): *An Object-Oriented Learning System Framework for Domain Oriented Reuse*, 3rd Annual Conference on the Teaching of Computing, CTI'95, Dublin, Ireland, September 1995

[9] Konstantopoulos, P.; Doerr, M. (1995): *Component Classification in the Software Information Base*, Object-Oriented Software Composition, Prentice Hall, 1995

[10] Konstantopoulos, P.; Doerr, M.; Vassiliou, Y. (1993): *Repositories for Software Reuse: The Software Information Base*, Proc. IFIP WG 8.1 Conference on Information System Development Process, Como, Italy, September 1993

[11] Konstantopoulos, P.; Jarke, M.; Mylopoulos, J.; Vassiliou, Y. (1995): *The Software Information Base: A Server for Reuse*, VLDB Journal, Vol. 4, No. 1, January 1995, pp. 1-43

[12] Konstantopoulos, P.; Pataki, E. (1992): *A Browser for Software Reuse*, Proc. 4th International Conference on Advanced Information Systems Engineering (CAiSE '92), Manchester, UK, May 1992

[13] Lakhotia, A. (1993): *Construction of Call Multigraphs Using Dependence Graphs*, Proc. 20th POPL, ACM Press, November 1993, pp. 273-284

[14] Lakhotia, A. (1994): *What is the Appropriate Abstraction for Understanding and Reengineering a Software System?*, Reverse Engineering Newsletter, IEEE Computer Society, November 7, September 1994, pp. 1-2

[15] Lakhotia, A. (1995): *Wolf: A Tool to Recover Dataflow Oriented Designs of Software Systems*, 5th Systems Reengineering Technology Workshop, CA, February 1995

[16] Lakhotia, A. (1997): *A Unified Framework for Expressing Software Subsystem Classification Techniques*, Journal of Systems and Software, © Elsevier Science Inc., Vol. 36, March 1997, pp. 211-231

[17] Lakhotia, A.; Gravley, J. (1995): *Toward Experimental Evaluation of Subsystem Classification Recovery Techniques*, Proc. 2nd Working Conference on Reverse Engineering, © IEEE CS Press, July 1995

[18] LeBlank, R.;Ornburn, S.; Rugaber, S. (1990): *Recognizing design decisions in programs*, IEEE Software Vol. 7 No. 1, 1990, pp. 46-54

[19] Low, C. (1988): *A Shared, Persistent Object Store*, Proc. ECOOP'88, 1988

[20] Moore, M.; Rugaber, S.; Seaver, P. (1994): *Knowledge-based User Interface Migration*, Proc. International Conference on Software Maintenance, Victoria, British Columbia, September 1994

[21] Peraire, C.; Barbey,S.; Buchs, D. (1998): *Test Selection for Object-Oriented Software Based on Formal Specifications*, IFIP Working Conference on Programming Concepts and Methods (PROCOMET'98), Shelter Island, New York, USA, June 1998, Chapman & Hall, 1998, pp. 385-403

[22] Rugaber, S.; Clayton, R. (1993): *The Representation Problem in Reverse Engineering*, Proc. 1st Working Conference on Reverse Engineering, Baltimore, Maryland, May 21-23, 1993

[23] Stevens, P.; Pooley, R. (1998): *Software Reengineering patterns*, SEBPC Workshop, 1998

[24] Stevens, P.; Pooley, R. (1998): *Systems Reengineering Patterns*, Proc. ACM-SIGSOFT, 6th International Symposium on the Foundations of Software Engineering, pp.17-23, ISBN 1-58113-108-9

[25] Svoboda, F. (1994): *Application of Integrated process Definition to Reverse Engineering*, SEI Software Engineering Workshop, Pittsburgh, PA, May 1994

[26] Zimmer, W. (1995): *Experiences Using Design Patterns to Reorganize an Object-Oriented Application*, FZI Publication, Forschungszentrum Informatik Karlsruhe, January 1995

Bibliography

[27] BAI Reengineering Team (1996): *Software Reengineering Technique Classification*, URL: http://users.erols.com/bai/srtclass.html#771613

[28] Barbey, S.; Buchs,D.; Peraire, C. (1996): *Issues and Theory for Unit Testing of Object-Oriented software*, Tagungsband, Basel, October 1996, pp. 73-112

[29] Breuer, P.; Lano, K. (1991): *Creating Specifications from Code*, Journal of Software Maintenance: Research and Practice, Vol. 3, 1991, pp. 145-162

[30] Demeyer, S.; Rieger, M.; Tichelaar, S. (1998): *Three Reverse Engineering Patterns*, Writing Workshop at EuroPLOP'98, April 1998

[31] Lakhotia, A. (1993): *Analysis of Experiences with Modifying Computer Programs*, URL: http://www.cacs.usl.edu/~arun/ftp-index.html#publications, May 1993

[32] Lakhotia, A. (1994): *Architecture Recovery Techniques: a unified view and a measure of their goodness*, URL: http://www.cacs.usl.edu/~arun/papers/TR-94-5-9.pdf, May 1994

[33] Lakhotia, A. (1998): *DIME: A Direct Manipulation Environment for Evolutionary Development of Software*, Proc. 6th International Workshop on Program Comprehension (IWPC'98), Ischia, Italy, IEEE Computer Society Press, June 1998, pp. 72-79

[34] Lakhotia, A; Deprez, J. (1999): *Restructuring Functions with Low Cohesion*, Proc. 6th Working Conferenceon Reverse Engineering (WCRE'99), Atlanta, GA, IEEE Computer Society Press, October 1999

[35] McLure, C. (1995): *Model-Driven Software Reuse. Practing Reuse Information Engineering Style*, Extended Intelligence Inc., URL: http://www.reusability.com/papers2.html, 1995

[36] Nierstrasz, O.; Papathomas, M. (1990): *Viewing Objects as Patterns of Communicating Agents*, OOPSLA/ECOOP 1990, pp. 38-43

[37] Spanoudakis, G.; Konstantopoulos, P. (1993): *Similarity for Analogical Software Reuse: A Conceptual Modeling Approach*, Proc. 5th International Conference on Advanced Information Systems Engineering (CAiSE '93), Paris, France, June 1993

[38] Spanoudakis, G.; Konstantopoulos, P. (1994): *Measuring Similarity between Software Artifacts*, Proc. 6th International Conference on Software Engineering and Knowledge Engineering (SEKE'94), Jurmala, Latvia, June 1994

[39] Spanoudakis, G.; Konstantopoulos, P. (1994): *On Evidential Feature Salience*, Proc. 5th International Conference on Database and Expert Systems Applications (DEXA '94), Athens, Greece, September 1994

[40] Spanoudakis, G.; Konstantopoulos, P. (1994): *Similarity for Analogical Software Reuse: A Computational Model*, Proc. 11th European Conference on Artificial Intelligence (ECAI '94), Amsterdam, The Netherlands, August 1994

[41] Stevens, P. (1998): *Report of Working Group on Reengineering Patterns*, VMCAT'98, August 1998

[42] Svoboda, F. (1994): *Reuse Based Reengineering*, Software Technology Conference, Salt Lake City, UT, April 1994

[43] Svoboda, F. (1994): *Reuse-Based Reengineering. Notes from the Underground*, 4th Systems Reengineering Technology Workshop, Monterey, CA, February 1994

[44] Svoboda, F. (1995): *The 3 "R's" of Mature System Development: Reuse, Reengineering and Architecture*, 5th Systems Reengineering Technology Workshop, February 1995

Commercial Implementations

[45] Semantic Designs Inc - **The DMS Software Reengineering Toolkit**
Location: http://www.semdesigns.com/Products/DMS/DMSToolkit.html

[46] Semantic Designs Inc – **CloneDR**
Location: http://www.semdesigns.com/Products/Clone/index.html

[47] George and James Software
Location: (http://www.georgejames.com/mar1/gjs220.htm)

[48] Metamorphic Computing Corporation
Location: http://www.metamorphic.com/html/cobol.html

Appendix 1

The following RETELL statements define the Queries and Tree Views objects and their contents in the TELOS database. When the GAIN browser initializes will recognize these objects and configure its Queries and Tree Views menus.

Definition of both **Queries** and **Tree Views** menus

```
RETELL Individual COBOLMenus in Token, MenuDescription
      with queryMenu
             (Queries)                : COBOLTextMenus
      with viewMenu
             (Tree_Views)             : COBOLTreeQuery
end
```

Definition of the **Queries** menu contents

```
RETELL Individual COBOLTextMenus in Token, SubMenu
      with commands
             (List_Source_Files)      : COBOLFileList;
             (List_Data_Files)        : COBOLDataList;
             (List_All_Programs)      : COBOLProgList;
             (List_Affecting_Programs)      : COBOLIOProgList
end
```

Definition of the **Tree Views** menu contents

```
RETELL Individual COBOLTreeQuery in Token, SubMenu
      with commands
             (Call_Tree)              : COBOLCallTree;
             (Called_By_Tree)         : COBOLCallTreeInv;
             (Both_Call_Trees)        : COBOLCallTreeBoth;
             (Affect_Tree)            : COBOLAffTree;
             (Affected_By_Tree)       : COBOLAffTreeInv;
             (Both_Affect_Trees)      : COBOLAffTreeBoth;
             (Include_Tree)           : COBOLInclTree;
             (Included_By_Tree)       : COBOLInclTreeInv;
             (Both_Include_Trees)     : COBOLInclTreeBoth
end
```

Appendix 2

The following RETELL statements store the source code of the queries included in the Queries menu in the TELOS database. Whenever one of these queries is triggered the corresponding TELOS statements are passed to QI with the order that the numbers before the statements define. For example whenever the List Source Files query is triggered by the user, QI is instructed by the GAIN browser to execute first the "scn SourceFile" statement and then the "gai" statement and return all the corresponding objects to the GAIN browser.

The **List Source Files** query

```
RETELL Individual COBOLFileList in Token, QueryMacro
      with code
      (1)    : "scn SourceFile";
      (2)    : "gai"
end
```

The **List Data Files** query

```
RETELL Individual COBOLDataList in Token, QueryMacro
      with code
      (1)    : "scn CobolData";
      (2)    : "gai"
end
```

The **List All Programs** query

```
RETELL Individual COBOLProgList in Token, QueryMacro
      with code
      (1)    : "scn CobolNode";
      (2)    : "gai";
      (3)    : "sc CobolNode source 1 end end"
      with iterator
             : "rf"
      with outputHeader
             : "Program Sourcefile"
end
```

The **List Affecting Programs** query

```
RETELL Individual COBOLIOProgList in Token, QueryMacro
      with code
      (1)    : "scn CobolData";
      (2)    : "gai";
      (3)    : "gltc CobolNode affect";
      (4)    : "gfv";
      (5)    : "sc CobolNode source 1 end end"
      with iterator
             : "rf"
      with outputHeader
             : "Program Sourcefile"
end
```

Appendix 3

The following RETELL statements store the source code of the queries included in the Tree Views menu in the TELOS database. Whenever one of these queries is triggered the corresponding TELOS statements are passed to QI with the order that the numbers before the statements define. For example, whenever the Call Tree query is triggered by the user, QI is instructed by the GAIN browser to execute first the "sc CobolNode call 1 end end" statement then the "sdep –1" statement and finally the "tc 0" statement. After the execution of those statements QI returns all the corresponding objects to the GAIN browser. The fact that the inputType attribute is set to CobolNode defines that this query is available by the GAIN browser whenever the current object belongs in the CobolNode class.

The **Call Tree** query

```
RETELL Individual COBOLCallTree in Token, QueryMacro
      with inputType
            : "CobolNode"
      with code
      (1)   : "sc CobolNode call 1 end end";
      (2)   : "sdep -1";
      (3)   : "tc 0"
      with iterator
            : "rn"
end
```

The **Called By Tree** query

```
RETELL Individual COBOLCallTreeInv in Token, QueryMacro
      with inputType
            : "CobolNode"
      with code
      (1)   : "sc CobolNode call 2 end end";
      (2)   : "sdep -1";
      (3)   : "tc 0"
      with iterator
            : "rn"
end
```

The **Both Call Trees** query

```
RETELL Individual COBOLCallTreeBoth in Token, QueryMacro
      with inputType
            : "CobolNode"
      with code
      (1)   : "sc CobolNode call 3 end end";
      (2)   : "sdep -1";
      (3)   : "tc 0"
      with iterator
            : "rn"
end
```

The **Affect Tree** query

```
RETELL Individual COBOLAffTree in Token, QueryMacro
      with inputType
            : "CobolNode"
      with code
      (1)   : "sc CobolNode call 1 CobolNode affect 1 end end";
      (2)   : "sdep -1";
      (3)   : "tc 0"
      with iterator
            : "rn"
end
```

The **Affected By Tree** query

```
RETELL Individual COBOLAffTreeInv in Token, QueryMacro
      with inputType
            : "CobolNode"
      with code
      (1)   : "sc CobolNode call 2 CobolNode affect 1 end end";
      (2)   : "sdep -1";
      (3)   : "tc 0"
      with iterator
            : "rn"
end
```

The **Both Affect Trees** query

```
RETELL Individual COBOLAffTreeBoth in Token, QueryMacro
      with inputType
            : "CobolNode"
      with code
      (1)   : "sc CobolNode call 3 CobolNode affect 1 end end";
      (2)   : "sdep -1";
      (3)   : "tc 0"
      with iterator
            : "rn"
end
```

The **Include Tree** query

```
RETELL Individual COBOLInclTree in Token, QueryMacro
      with inputType
            : "SourceFile"
      with code
      (1)   : "sc SourceFile contain 1 end end";
      (2)   : "sdep -1";
      (3)   : "tc 0"
      with iterator
            : "rn"
end
```

The **Included By Tree** query

```
RETELL Individual COBOLInclTreeInv in Token, QueryMacro
      with inputType
            : "SourceFile"
      with code
      (1)    : "sc SourceFile contain 2 end end";
      (2)    : "sdep -1";
      (3)    : "tc 0"
      with iterator
            : "rn"
end
```

The **Both Include Trees** Query

```
RETELL Individual COBOLInclTreeBoth in Token, QueryMacro
      with inputType
            : "SourceFile"
      with code
      (1)    : "sc SourceFile contain 3 end end";
      (2)    : "sdep -1";
      (3)    : "tc 0"
      with iterator
            : "rn"
end
```

Appendix 4

The following RETELL statements create the software object classes and their attributes as shown in **Table 9** of **Chapter 4** into the TELOS database instance.

The **NodeType** object class

```
RETELL Individual NodeType in M1_Class
      with attribute
            node_ref     : NodeType
end
```

The **SourceType** object class

```
RETELL Individual SourceType in M1_Class
      With attribute
            source_ref   : SourceType
end
```

The **CobolData** object class

```
RETELL Individual CobolData in S_Class end
```

The **SourceFile** object class

```
RETELL Individual SourceFile in S_Class, SourceType
      with source_ref
            contain      : SourceFile
end
```

The **CobolNode** object class

```
RETELL Individual CobolNode in S_Class, NodeType
      with attribute
            name         : Telos_String;   {Program Internal Name}
            source       : SourceFile;     {Program Source File Name}
            include      : SourceFile;     {COBOL COPY statements}
            affect       : CobolData;      {COBOL OPEN statements}
            select       : CobolData       {COBOL SELECT statements}
      with node_ref
            call         : CobolNode       {COBOL CALL statements}
end
```

Appendix 5

A brief description of GAWK

GAWK is the GNU Project's implementation of the AWK programming language. It conforms to the definition of the language in the POSIX 1003.2 Command Language And Utilities Standard. This version is based on the description of The AWK Programming Language, by Aho, Kernighan and Weinberger, with the additional features found in the System V Release 4 version of UNIX AWK. GAWK also provides more recent Bell Labs awk extensions, and some GNU-specific extensions.

An AWK program consists of a sequence of pattern-action statements and optional function definitions.

```
pattern {action statements}
function name(parameter list) { statements }
```

AWK first reads the program source from the program file(s) if specified, from arguments to --source, or from the first non-option argument on the command line. The –f and --source options may be used multiple times on the command line. AWK will read the program text as if all the program-files and command line source texts had been concatenated together. This is useful for building libraries of AWK functions, without having to include them in each new AWK program that uses them. It also provides the ability to mix library functions with command line programs.

The environment variable AWKPATH specifies a search path to use when finding source files named with the -f option. If this variable does not exist, the default path is ".:/usr/local/share/awk". (The actual directory may vary, depending upon how AWK was built and installed.) If a file name given to the -f option contains a ``/" character, no path search is performed.

Gawk executes AWK programs in the following order. First, all variable assignments are performed. Next, AWK compiles the program into an internal form. Then, AWK executes the code in the BEGIN block(s) (if any), and then proceeds to read each file specified in the command line. If there are no files specified in the command line, AWK reads the standard input.

If a filename on the command line has the form var=val it is treated as a variable assignment. The variable var will be assigned the value val. This happens after any BEGIN block(s) have been run. Command line variable assignment is most useful for dynamically assigning values to the variables AWK uses to control how input is broken into fields and records. It is also useful for controlling state if multiple passes are needed over a single data file.

For each record in the input, gawk tests to see if it matches any pattern in the AWK program. For each pattern that the record matches, the associated action is executed. The patterns are tested in the order they occur in the program.

Finally, after all the input is exhausted, gawk executes the code in the END block(s) (if any).

Appendix 6

The following five Tables describe the usage and operation of each of the files that are located in the five subdirectories of the top-level directory of the directory tree of the COBOL parser.

Table A6.1 – The contents of subdirectory awk

File Name	Description of operation
CobolData.awk	Defines the CobolData objects. It is used by the find-CobolData executable.
CobolNode-simple.awk	Defines all the CobolNode objects in a simple way (without any of their attributes). It is used by the find-CobolNode-simple executable.
CobolNode.awk	Redefines all the CobolNode objects together with their attributes. It is used by the find-CobolNode executable.
SourceFile.awk	Defines all the SourceFile objects. It is used by the find-SourceFile executable.
SourceFileName.awk	It is used as a filter. It takes as its input the name of a file in full path representation and returns its name discarding the full path representation. It is used by the find-CobolNode executable.
Checktls.awk	Scans a file in search for contiguous duplicate lines. Its output is the same file without the duplicate lines. Every duplicate line found in the input file is logged in a separate log file that is generated for that purpose. It is used by the find-CobolData and find-CobolNode-simple executables.
make-dependencies.awk	It is used by the find-CobolNode executable in order to create two temporary files. The first file contains the names of the CobolNode objects found by find-CobolNode-simple executable and the second file contains the names of the CobolData objects found by the find-CobolData executable.

Table A6.2 – The contents of subdirectory bin

File Name	Description of operation
CreateSisTransaction	Merges the four files, that describe the TELOS objects found by the COBOL parser and creates a file that includes the final SIS transaction, which describes the structure of the scanned application.
find-CobolData	Scans the source files of the application in search of CobolData objects. It uses srcfilenames.tmp, CobolData.awk, CobolData.tmp, Checktls.awk. The CobolData objects found are stored in CobolData.tls file.
find-CobolNode	Scans the source files of the application and defines the CobolNode objects and their relations. It uses make-dependencies.awk, CobolNode-simple.tls, CobolData.tls, cobolnode.tmp, coboldata.tmp, srcfilenames.tmp, SourceFileName.awk, CobolNode.awk and CobolNode.tls.
find-CobolNode-simple	Defines all the CobolNode objects in a simple way (without any of their attributes). It uses the find-CobolNode-simple.awk, srcfilenames.tmp, CobolNode-simple.tmp, Checktls.awk and CobolNode-simple.tls.
find-SourceFile	Locates the source file names of the application starting from a predefined (by the user) directory. Then it defines the SourceFile objects. It uses srcfilenames.tmp, SourceFile.awk and SourceFile.tls.
loginit	Initializes the workspace where the log files, which are created from the parser during the source code parsing procedure, reside.
parser	The main executable file that is used to start and control the parsing procedure. This executable is used to invoke separate executable files that reside in directory bin in a certain order to handle the several parts of the COBOL source code parsing procedure.
tlsinit	Initializes the workspace where the various TELOS files, which are created from the parser during the source code parsing procedure, reside.
tmpinit	Initializes the workspace where the temporary files, which are created from the parser and used during the source code parsing procedure, reside.

Table A6.3 – The contents of subdirectory log

File Name	Content description
CheckDuplicates.log	Instances that are found to be duplicate instances during the parsing procedure are logged in this file to inform the user about this special case.
coboldata.log	Warnings and errors, which are produced during the execution of the source code parsing procedure, which examines the database of the application, are logged in this file.
cobolnode.log	Warnings and errors, that are produced while the parser executes the part of the source code parsing procedure that examines the internal structure of the application, are logged in this file

Table A6.4 – The contents of subdirectory tls

File Name	Content description
SourceFile.tls	The parser stores in this file the SourceFile individuals (objects) that are recognized during the application parsing procedure.
CobolNode-simple.tls	The parser stores in this file the CobolNode individuals (objects) that are recognized during the application parsing procedure. In this file these objects are identified without any attributes (relations with other objects etc) for internal processing reasons.
CobolNode.tls	The parser stores in this file the CobolNode individuals (objects) that are recognized during the applications source code parsing procedure. In this file these objects are identified together with their attributes (relations with other objects etc).
CobolData.tls	The parser stores in this file the CobolData individuals (objects) that are recognized during the applications source code parsing procedure.
application-structure.tls	The CreateSisTransaction executable concatenates the four files above in this file with a certain order. This executable also fills the file with some TELOS Data Entry Language statements in order to give to the file the final form of a TELOS transaction. This file contains all the information that the COBOL parser has derived from the application while scanning it (except from the logging information, which is just used for notification).

Table A6.5 – The contents of subdirectory tmp

File Name	Content description
CobolData.tmp	The parser stores in this file the CobolData individuals (objects) that are recognized during the applications source code parsing procedure before creating the CobolData.tls file. The CobolData.tls file is produced from this file after sorting it and discarding any possible duplicate entries.
CobolNode-simple.tmp	The parser stores in this file the CobolNode individuals (objects) that are recognized during the applications source code parsing procedure before creating the CobolNode-simple.tls file. The CobolNode-simple.tls file is produced from this file after sorting it and discarding any possible duplicate entries.
coboldata.tmp	The COBOL parser stores in this temporary file only the names of the CobolData objects that have been found during the source code scanning procedure just for internal processing reasons.
cobolnode.tmp	The COBOL parser stores in this temporary file only the names of the CobolNode objects that have been found during the source code scanning procedure just for internal processing reasons.
Srcfilenames.tmp	The COBOL parser stores in this temporary file the name of each source file of the application in a *full path* form to identify the exact position of each source file in the filesystem.

Appendix 7

Description of variables used by the Attributes.awk program.

Variable Name	Description
telos_state	String – Has the constant value of "RETELL Individual (".
telos_class	String – Has the constant value of ") in Token, CobolNode".
var_source	String – Has the value of "with source (<filename>)"
var_programid	String – Is set to the "ProgramId" attribute (if any) of the CobolNode being scanned.
var_individualname	String – Is set to the name of the source code file being scanned.
var_call	String – Is set to the "call" attribute (if any) of the CobolNode being scanned.
var_open	String – Is set to the "affect" attribute (if any) of the CobolNode being scanned.
var_select	String – Is set to the "select" attribute (if any) of the CobolNode being scanned.
var_if	Numeric – The calling parameter of the "if_fix" function (see *Table 9*).
read_opens	Boolean – 1 if the *current* line is part of an OPEN statement, which started in a previous line and is not over yet. 0 otherwise.
var_end	String – Has the constant value of "end".
kw	String (Table of 46 variables) – Each variable is a COBOL statement. All COBOL (RM/COBOL – 85) statements are stored in this Table.
op_kw	String (Table of 11 variables) – Each variable is a *phrase* used by the OPEN (RM/COBOL – 85) statement. All the *phrases* used by the OPEN statement are stored in this Table.
coboldata	String (Table of 1 line and variable length of columns) – It contains the names of the data files (CobolData objects) *affected* by this CobolNode. Each CobolData object found is checked against the contents of this Table. If at least one match is found it gets discarded. If no match is found an "affect" relation is created and then its name is appended to this Table. It is used just not to make twice the same attribute declaration (a specific dada file can be OPENed more than one time in a COBOL program).
mtr_individualname	String (Table of 2 variables) – It is used just to split the name of the source file in two pieces (divided by the "." character). If the second variable (of the Table) has the value "CBL" or "cbl" the value of the first variable is given as a "name" of the CobolNode. Otherwise the CobolNode is named after the name of the source file.
cdin	Counter.
c	String – It is used by the "strfix" function.
i	Counter

Appendix 8

Description of functions defined and used by the Attributes.awk program.

Function name	**Description**
strfix (string)	String – The string parameter is checked against the possibility to have heading or trailing characters that are not (capital or lower case) letters or numbers. If so these characters are discarded. The new string (without any of the *bad* heading or trailing characters) is returned.
offspaces (string)	String – The string parameter is checked against the possibility to contain trailing *space* characters. If so the trailing *space* characters are discarded. The new string (without any trailing *space* characters) is returned.
arrfind (string,string array)	Boolean – The string parameter is checked against each of the elements of the second parameter (string array). The function returns the "TRUE" value (1) if at least one match is found. "FALSE" otherwise.

Appendix 9

The procedure followed to create a new and empty TELOS database instance, configure it and then store all the information gathered by the COBOL parser into the database instance is described in the next steps. More information concerning the detailed description of all these steps and why they must take place can be found in the SIS manual page.

Step1 Execute the mainsetup.bat batch file, which is located in the \sis\env directory. This batch file sets all the environment variables that the TELOS database and the GAIN browser examine when they initialize, concerning values that describe the characteristics of the computer (e.g. the computer name).

Step2 Execute the HotelR12_Setup.bat batch file, which is located in the \sis\env directory. This batch file sets all the environment variables that the TELOS database and the GAIN browser examine when they initialize, concerning values that describe the characteristics of the specific TELOS database instance (e.g. the TCP port that the TELOS database instance will listen to).

Step3 Delete all the contents of the data directory of the TELOS database instance.

Step4 Start the TELOS database instance by executing the command "start sisserver %sis_port%" from the \sis\bin directory.

Step5 Execute the gr_dos_script, which resides in the \sis\applications\HotelR12\Telos_Sources directory. This batch script invokes the TELOS parser and executes the queries, which are located in the query-menus.tls, cobol-nodes.tls and query-transactions.tls files (*see "Design and Implementation of the TELOS Database Instance"*).

Appendix 10

The procedure followed to start TELOS database engine and the GAIN browser is described in the next steps. More information concerning the detailed description of all these steps and why they must take place can be found in the SIS manual page.

Step1	Execute the mainsetup.bat batch file, which is located in the \sis\env directory. This batch file sets all the environment variables that the TELOS database and the GAIN browser examine when they initialize, concerning values that describe the characteristics of the computer (e.g. the computer name).
Step2	Execute the HotelR12_Setup.bat batch file, which is located in the \sis\env directory. This batch file sets all the environment variables that the TELOS database and the GAIN browser examine when they initialize, concerning values that describe the characteristics of the specific TELOS database instance (e.g. the TCP port that the TELOS database instance will listen to).
Step3	Execute the runserver.bat batch file that initializes the TELOS database engine.
Step4	Execute the rungain.bat batch script that initializes the GAIN browser.

Appendix 11

Table A11.1 – Programs with the most *Include* relations

	Number of relations				
Program	**Include**	**Affect**	**Select**	**Call**	**Summary**
APOMASTN	66	16	0	6	88
TIMEIS	62	19	0	6	87
TIMAG	56	8	0	2	66
APDIAK	55	4	0	17	76
ORDEIS	55	10	0	4	69
APEVRKIN	54	13	0	3	70
TDPREIS	53	18	0	2	73
PAREIS1	51	10	0	3	64

Table A11.2 – Programs with the most *Affect* relations

	Number of relations				
Program	**Include**	**Affect**	**Select**	**Call**	**Summary**
V3-APO	19	28	14	0	61
TIMEIS	62	19	0	6	87
TIMEKD	46	19	0	0	65
YPOK-APO	3	19	23	0	45
TDPREIS	53	18	0	2	73
TGEKD	45	18	0	1	64

Table A11.3 – Programs with the most *Select* relations

	Number of relations				
Program	**Include**	**Affect**	**Select**	**Call**	**Summary**
EMP-APO	2	13	24	0	39
MAKOIKO	1	4	24	0	29
YPOK-APO	3	19	23	0	45
EMP-TIM	2	12	15	0	29

Table A11.4 – Programs with the most *Call* relations

Program	Number of relations				
	Include	**Affect**	**Select**	**Call**	**Summary**
APOPEN	1	0	0	25	26
APIN	0	0	0	16	16
TIMOPEN	3	0	0	16	19
TIMIN	0	0	0	16	16
AGOPEN	3	0	0	15	18
AGIN	0	0	0	15	15

Table A11.5 – Source files with the most *Contain* relations

Source File	**Number of *Contain* relations**
APOMASTN.CBL	66
TIMEIS.CBL	62
TIMAG.CBL	56
APDIAK.CBL	55
ORDEIS.CBL	55
APEVRKIN.CBL	54
TDPREIS.CBL	53
PAREIS1.CBL	51

Table A11.6 – Data files with the most *Affect* relations

Data File	Number of relations		
	Affect	**Select**	**Summary**
APO-FILE	345	20	365
APOMA-FILE	239	2	241
APANAL-FILE	193	9	202
APTM-FILE	177	1	178
APTEAM-FILE	175	10	185
APO1-FILE	165	2	167

Table A11.7 – Data files with the most *Select* relations

	Number of relations		
Data File	**Affect**	**Select**	**Summary**
SEQ-FILE	45	47	92
PRINT-FILE	156	30	186
APO-FILE	345	20	365
PELAT-FILE	149	18	167
TMP-FILE	15	17	32
SORT-FILE	0	16	16
WAPO-FILE	18	16	34

Appendix 12

The following table presents all the improvements and corrections (bug fixes) made to the software reengineering mechanism from its first stable revision (Rev 1.2) until its final stable revision (Rev 1.8)

Table A12.1 – Improvements and corrections made to each revision

Revision	Improvements and corrections
Rev 13	Added the "open" relation to the SourceFile object.
Rev 14	Moved the "select" relation from the CobolNode object to the SourceFile object. (Bug Fix) Added the "define" relation to the SourceFile object.
Rev 15	Log all the discarded relations.
Rev 16	Support for .cbl and .CBL extentions in COPY statements. (Bug Fix)
Rev 17	Make CobolNode and CobolData objects to be case insensitive (Bug Fix)
Rev 18	Added the invoke attribute to the SourceFile objects.

Appendix 13

In this appendix the presentation and analysis of source code files of the XPERT Hotel software application takes place.

Sign	Description
√	**Libraries.** RM/COBOL source code or RM Panel binary libraries, which are included in programs that compile successfully.
√	**Dead Code.** The source file has been unlinked from the main application or the only program using it compiles with errors.
✖	**Main Source Code File.** The file is the main source file of a program that compiles successfully.
✖	**Dead Code.** Program compiles with errors.
✚	**Structured Text File.** The file contains global application parameters or is an executable shell script.
✚	**Garbage.** Completely useless text or binary.

Sign	Total Files	Subtotal %	Total %	Total Text Lines	Subtotal %	Total %
√	907	70,36	35,79	83804	65,85	13,01
√	382	29,64	15,07	43463	34,15	6,75
Subtotal	1289			127267		
✖	1071	92,89	42,26	409633	82,80	63,64
✖	82	7,11	3,24	85099	17,20	13,22
Subtotal	1153			494732		
✚	33	35,87	1,3	8753	40,30	1,36
✚	59	64,13	2,34	12964	59,70	2,02
Subtotal	92			21717		
Total Green Code	2011		79,36	502190		78,01
Total Red Code	523		20,64	141526		21,99
Totals	2534			643716		

✖ ACHK.CBL	√ ADP.PRC	✖ AFM.CBL	√ AFM.PRC
√ AFM.WOR	✖ AFMFIX.CBL	✖ ag-ekt.cbl	✖ AG2HMER.CBL
✖ AGARSEE.CBL	✖ AGARSEEP.CBL	✖ AGARTOP.CBL	✖ AGARTP.CBL
√ AGDAPEN.CBL	✖ AGDATOP.CBL	✖ AGDCHK.CBL	✖ AGDEFPAR.CBL
✖ AGDEIS.CBL	✖ AGDEL.CBL	✖ AGDEVR.CBL	✖ AGDEVR4.CBL
✖ AGDEVR23.CBL	✖ AGDEVRDT.CBL	✖ AGDOP.CBL	✖ AGDPAR.CBL
√ AGDREC.CBL	✖ AGDTIM.CBL	√ AGEPIB.PRC	√ AGEPIK.CBL
✖ AGEVR.CBL	✖ AGFLCHK.CBL	✖ AGFORTEID.CBL	✖ AGFORTSYN.CBL
√ AGHMER.WOR	✖ AGHMEROL.CBL	✖ AGIN.CBL	✖ AGINVOP.CBL
√ agkk-ap.prc	√ agkk-ap.ws	√ agkk-pel.prc	√ agkk-pel.ws
√ AGKOST.PRC	✖ AGLINKOP.CBL	√ AGO.MOV	√ AGO.REC
√ AGO.SEL	√ AGO.WOR	✖ AGOPCNT.CBL	✖ AGOPEN.CBL
✖ AGOPOL.CBL	√ AGOVAR.WOR	✖ AGPAR.CBL	√ agparmtr.prc
√ agparmtr.ws	✖ AGPCNTOP.CBL	√ AGPCNTREC.CBL	✖ AGPDEL.CBL
✖ AGPEIS.CBL	✖ AGPEVR4.CBL	✖ AGPEVR123.CBL	✖ AGPOP.CBL
√ AGPREC.CBL	✖ AGPTIM.CBL	✖ AGTCHK.CBL	✖ AGTCHK1.CBL

✖ AGTDEL.CBL	√ agtdisc.win	✖ AGTEIS.CBL	√ AGTEIS.LNK
✖ AGTEIS.LST	✖ AGTEIS.STD	✖ AGTEIS1.CBL	✖ AGTEIS2.CBL
✖ AGTEIS3.CBL	✖ AGTOMEIS.CBL	✖ AGTOMOP.CBL	✖ AGTOP.CBL
✖ AGTPAR.CBL	√ AGTREC.CBL	✖ AGTSYGDT.CBL	√ agtvalue.win
✖ AGTYPEIS.CBL	✖ AGTYPOP.CBL	✖ AGVIEW.CBL	✖ AGVIEW1.CBL
✖ AIT-TIM.CBL	✖ AITEXAG.CBL	✖ AITEXALL.CBL	√ ALLAPF.WOR
√ ALLFEAT.WOR	√ ALLPELF.WOR	✖ ALPHATEST.CBL	✖ ALPHATEST2.CBL
√ AMUPADTE.CBL	√ AMUPD.DCL	√ AMUPD.MOV	√ AMUPD.PRC
√ AMUPD.REC	√ AMUPD.SEL	√ AMUPD.WOR	✖ AMUPDATE.CBL
✚ amupdmv	✖ AMUPREAD.CBL	✖ ANALMAKE.CBL	√ answers.wor
√ ANTO.PRC	√ any-win.prc	√ any-win.wor	✖ AP--DEL.CBL
✖ AP-AP.CBL	✖ AP-DEL.CBL	✖ AP-GET.CBL	✖ AP-MOVE.CBL
✖ AP-MOVE1.CBL	✖ AP-UPD.CBL	✖ AP.CBL	✖ AP0.CBL
✖ AP1CLEAR.CBL	✖ AP2HMER.CBL	✖ apaddtrs.cbl	✖ APAGLAST.CBL
✖ APAIT2EX.CBL	√ APAL.REC	√ APAL.WIN	✖ APALBRI.CBL
√ APALBRI.REC	√ APALBRI.SEL	√ APALBRI.WOR	✖ APALBRI9.CBL
✖ APALBRI9.STD	✖ APALEIS.CBL	√ APALEIS.PRC	√ APALEIS.WIN
√ APALEIS.WOR	✖ APALGCH.CBL	√ APALGCH.PRC	√ APALGCH.WOR
✖ APALOP.CBL	✖ APAN-MTK.CBL	✖ APAN.CBL	✖ APANAHTL.CBL
✖ APANAL.CBL	√ APANAL.MOV	√ APANAL.REC	√ APANAL.SEL
√ APANAL.WOR	✖ APANALAA.CBL	√ apanalho.rec	✖ APANALN.CBL
✖ APANALX.CBL	✖ APANALY.CBL	✖ APANAL_N.CBL	✖ APANCHK.CBL
✖ APANFL1.CBL	✖ APANFL2.CBL	✖ APANLGOP.CBL	✖ APANLOG.CBL
√ APANLOG.PRC	√ APANLOG.REC	√ APANLOG.WOR	✖ APANREAD.CBL
✖ APANREAD1.CBL	√ APANREC.CBL	✖ APANRPRV.CBL	✖ APANTEST.CBL
✖ APANUSER.CBL	✖ APAPOGR1.CBL	✖ APAPOGR2.CBL	√ apaptm.prc
√ apaptm.ws	√ apar.prc	√ apar.ws	✖ APARTPOP.CBL
✖ APASFAL.CBL	✖ APCHANCE.CBL	✖ APCHANGE.CBL	✖ APCHK.CBL
✖ apchk.cbl	✖ APCLEAR.CBL	√ apcode2.win	√ apcode3.win
✖ APCOSTAN.CBL	✖ APCOSTS.CBL	√ APCOSTS.PRC	√ APCOSTS.WOR
√ APD-JOB.PRC	√ APD.REC	✖ APDEL.CBL	✖ APDEL1.CBL
✖ APDGEN.CBL	✖ APDGEN1.CBL	✖ APDIAFOR.CBL	✖ APDIAK.CBL
✖ APDIAK.LST	√ apdiak.prc	√ apdiak.ws	✖ APDIEK.CBL
√ APDMAT.PRC	√ APDMAT.WOR	✖ APDOP.CBL	√ APDPR.PRC
√ APDPR.REC	√ apdpr.win	✖ APDPREVR.CBL	✖ APDPRICE.CBL
✖ APDPROP.CBL	√ APDPRREC.CBL	√ APDREC.CBL	✖ APEIDFP3.CBL
✖ APEIDFPA.CBL	√ apeidfpa.prc	√ apeidfpa.ws	✖ APEIDMA3.CBL
✖ APEIDMAN.CBL	√ apeidman.prc	√ apeidman.ws	✖ APEIDMTK.CBL
√ apeidmtk.prc	√ apeidmtk.ws	✖ APEIDSTK.CBL	√ apeidstk.prc
√ apeidstk.ws	✖ APEIDSYN.CBL	√ apeidsyn.prc	√ apeidsyn.ws
✖ APEIDTIM.CBL	√ apeidtim.prc	√ apeidtim.ws	√ APENHM.PRC
√ APENHM.TIM	√ APENHM.WOR	√ APENMTK.PRC	✖ APEPANEK.CBL
✖ APETIK.CBL	✖ APETIKCH.CBL	✖ APEVMON.CBL	✖ APEVR00.CBL
√ APEVR00.PRC	√ APEVR00.REL	√ APEVR00.WOR	✖ APEVR01.CBL
✖ APEVR01.LST	√ APEVREI.ACC	✖ APEVREI1.CBL	✖ APEVREI1.STD
✖ APEVREI2.CBL	✖ APEVREI6.CBL	✖ APEVRET.CBL	✖ APEVRET1.CBL
✖ APEVRGEN.CBL	✖ APEVRICL.CBL	✖ APEVRKI2.CBL	✖ APEVRKIN.CBL
✖ APEVRKIN.STD	✖ APEVRMAN.CBL	✖ APEVRSTK.CBL	√ APEVRSTK.PRC
√ APEVRSTK.WOR	✖ APEVRTA1.CBL	✖ APEVRTA2.CBL	✖ APEVRTAM.CBL
✖ APEXAG.CBL	✖ APEXAHTL.CBL	✖ APEXEKT.CBL	√ APEXPREC.CBL
√ APF.PRC	√ APF.REC	√ APF.WIN	√ APF.WOR
√ APF1.REC	√ apf1.win	√ APF1.WIN	✖ APF1EIS.CBL
√ APF1L.REC	✖ APF1LINK.CBL	√ apf1lnk.win	√ APF1LREC.CBL
✖ APF1OP.CBL	√ APF1REC.CBL	✖ APFEACH.CBL	√ APFEACH.PRC
√ APFEACH.WOR	√ apfeat1.prc	√ apfeat1.ws	✖ APFEIS.CBL
√ APFEREC.CBL	✖ APFILTER.CBL	√ APFILTER.PRC	√ APFILTER.WOR
✖ APFIXCN.CBL	√ APFL.PRC	√ APFL.REC	√ APFL.WOR
√ APFL1.PRC	✖ APFLCHK.CBL	✖ APFLEIS.CBL	√ apfleis.win
✖ APFLOP.CBL	√ apflprd.win	√ APFLREC.CBL	√ APFLTR.PRC
√ APFLTR1.PRC	√ APFLTR2.PRC	√ APFLTR3.PRC	✖ APFOP.CBL
✖ APFPA.CBL	√ APFPA.PRC	√ APFPA.REC	√ APFPA.WIN
√ APFPA.WOR	√ APFPAREC.CBL	✖ APFPR.CBL	√ APFPR.MOV
√ apfpr.prc	√ APFPR.REC	√ APFPR.SEL	√ APFPR.WOR
√ apfpr.ws	✖ APFPREIS.CBL	✖ APFPREN1.CBL	✖ APFPRENH.CBL
✖ APFPREVR.CBL	√ APFPRT.PRC	√ APFPRT.REL	√ APFREC.CBL
✖ APG1CNLT.CBL	✖ APG1DIAF.CBL	✖ APG1EIS1.CBL	✖ APG1EIS2.CBL
✖ APG1LST1.CBL	✖ APG1LST2.CBL	✖ APG1LST3.CBL	✖ APG1TAKT.CBL
✖ APG1TO3.CBL	✖ APG2CNLT.CBL	✖ APG2DIAF.CBL	✖ APG2EIS1.CBL
✖ APG2EIS2.CBL	✖ APG2LST1.CBL	✖ APG2LST2.CBL	✖ APG2LST3.CBL
✖ APG2TAKT.CBL	✖ APG3CNLT.CBL	✖ APG3DIAF.CBL	✖ APG3EIS1.CBL
✖ APG3EIS2.CBL	✖ APG3LST1.CBL	✖ APG3LST2.CBL	✖ APG3LST3.CBL
✖ APG3LST4.CBL	✖ APG3TAKT.CBL	✖ APGARXH.CBL	✖ APGARXH1.CBL

✖ APGARXH2.CBL	✖ APGCNLT3.CBL	✖ APGCNLTA.CBL	✖ APGCONV1.CBL
✖ APGDEL1.CBL	✖ APGDEL2.CBL	✖ APGDEL3.CBL	✖ APGDELET.CBL
✖ APGDIAF1.CBL	✖ APGEIS1.CBL	✖ APGEIS2.CBL	√ APGEIS2.CPY
√ apgeis2.prc	√ APGEIS2.WOR	√ apgeis2.ws	✖ APGENDEL.CBL
✖ APGENYP.CBL	√ APGENYP.PRC	√ APGENYP.REL	√ APGENYP.WOR
√ APGK.MOV	√ APGK.REC	√ APGK.SEL	√ APGK.WIN
√ APGK.WOR	✖ APGKEIS.CBL	√ apgkeis.prc	√ apgkeis.ws
√ apgkey.win	✖ APGKOPEN.CBL	✖ APGLIST1.CBL	✖ APGLIST2.CBL
✖ APGLIST2.RST	✖ APGLIST2.STD	✖ APGLIST3.CBL	✖ APGLIST4.CBL
√ APGLP.PRC	√ APGLP.REL	√ APGLP.WOR	✖ APGOTHER.CBL
√ apgper.win	✖ APGSTAR1.CBL	✖ APGSTAR2.CBL	✖ APGSTART.CBL
✖ APGSTART.STD	✖ APGTACNL.CBL	✖ APGTAKT.CBL	✖ APGTAKT3.CBL
✖ APHMEROL.CBL	√ aphosp.prc	√ aphosp.ws	✖ APHOSPHT.CBL
✖ APHOSPOP.CBL	✖ APHOSPP2.CBL	✖ APHOSPPL.CBL	✖ APHOSPTR.CBL
✖ APHOTEL.CBL	√ aphotel.prc	√ aphotel.ws	✖ APHOTELO.CBL
√ aphpl.prc	√ aphpl.ws	✖ APHPLEIS.CBL	✖ APHPLEVR.CBL
✖ APHSPHTL.CBL	✖ APHTLCOS.CBL	✖ APIN.CBL	✖ APISOZ.CBL
✖ APISOZ1.CBL	✖ APK1EIDA.CBL	✖ APK1ISOZ.CBL	√ APK1REC.CBL
✖ APK1SYG.CBL	✖ APK1YPOL.CBL	✖ APKARTEL.CBL	✖ APKARTHT.CBL
✖ APKATEL.CBL	✖ APKATEL1.CBL	✖ APKDCHK.CBL	√ APKK.PRC
√ apkk.prc	√ APKK.WOR	√ apkk.ws	√ APKKFIL.CBL
√ APKO2.REC	√ APKO2REC.CBL	✖ APKODCHK.CBL	√ APKODE.REC
√ APKODEREC.CBL	✖ APKODTAM.CBL	√ APKOS.REC	√ APKOSREC.CBL
✖ APKOSTEIS.CBL	✖ APKOSTKER.CBL	√ APKOSTME.CBL	✖ APKSHOW.CBL
✖ APLABEL.CBL	√ APLABEL.PRC	√ APLABEL.WOR	✖ APLGCH.CBL
√ APLGCH.PRC	√ APLGCH.WOR	✖ APLGEIDOS.CBL	✖ APLGEVR.CBL
√ APLTIMAG.PRC	√ APLTIMAG.WOR	✖ APM1JOIN.CBL	✖ APM2JOIN.CBL
✖ APMANTAM.CBL	√ apmaster.prc	√ apmaster.ws	✖ APMCLEAR.CBL
✖ APMEZEV1.CBL	✖ APMEZEV2.CBL	✖ APMEZEVR.CBL	✖ APMEZHTL.CBL
✖ APMEZHTT.CBL	✖ APMEZMTM.CBL	✖ APMHN.CBL	✖ APMHNST.CBL
✖ APMHNSTA.CBL	✖ APMHNSTF.CBL	✖ APMHNYP.CBL	√ APMHNYP.PRC
√ APMHNYP.WOR	√ APMOD.REC	✖ APMODEL.CBL	√ apmodel.win
✖ APMODOP.CBL	✖ APMODPRT.CBL	√ APMODREC.CBL	√ APMON.PRC
√ APMON.REC	√ APMON.WOR	✖ APMONEIS.CBL	✖ APMONHTL.CBL
✖ APMONOP.CBL	√ APMONREC.CBL	✖ APMRCLS.CBL	✖ APMRCRE.CBL
✖ APMRINS.CBL	✖ APMRMET.CBL	✖ APMRMOV.CBL	✖ APMRMOV0.CBL
✖ APMRREAD.CBL	√ APMTIM.PRC	√ APMTIM.WOR	√ APMTIMPK.PRC
√ APMTK.LNK	√ APMTK.PRC	✖ APMTK000.CBL	✖ APMTKAPL.CBL
✖ APMTKCMP.CBL	✖ APMTKCMP.RUN	✖ APMTKEID.CBL	✖ APMTKSYN.CBL
✖ APNCLEAR.CBL	✖ APNEW.CBL	✖ APNOKIN.CBL	✖ APNREAD.CBL
✖ APO-DESC.CBL	√ APO-P.WIN	√ APO-TAM.WIN	✖ APO-TR.CBL
√ APO.DCL	√ apo.metaxa.doc	√ APO.MOV	√ APO.PRC
√ APO.REC	√ APO.SEL	√ APO.WIN	√ APO.WOR
√ APO.WRK	√ APO1.PRC	√ APO1.SEL	√ APO1.WOR
✖ APO1CHK.CBL	√ APO2.WIN	✖ APOCOMP.CBL	✖ APODEL.CBL
✖ APOGCHK.CBL	✖ APOGEIS2.CBL	✖ APOGLST.CBL	✖ APOGOP.CBL
✖ APOGOP0.CBL	✖ APOGOP1.CBL	✖ APOGPR.CBL	✖ APOGR-OP.CBL
√ APOGR.LNK	√ APOGR.MOV	√ apogr.prc	√ APOGR.RE0
√ APOGR.REC	√ APOGR.SEL	√ APOGR.WOR	√ apogr.ws
✖ APOGRDEL.CBL	✖ APOGRMOV.CBL	√ APOGRNEW.REC	√ APOHO.SEL
√ APOINS.PRC	✖ APOKARTA.CBL	√ APOKK.WOR	√ APOKODE.PRC
√ APOM1.REC	√ APOM1.SEL	√ APOM1REC.CBL	✖ APOMA-TR.CBL
√ APOMA.PRC	√ APOMA.REC	√ APOMA.SEL	√ APOMA.WIN
√ APOMA.WOR	✖ APOMA2TR.CBL	√ APOMA3.WIN	✖ APOMACHK.CBL
√ apomacp.prc	√ apomacp.ws	✖ APOMAFIX.CBL	✖ APOMANAF1.CBL
✖ APOMANEW.CBL	✖ APOMANO0.CBL	✖ APOMAREAD.CBL	√ APOMAST.LNK
√ APOMAST.MOV	√ APOMAST.REC	√ APOMAST.SEL	√ APOMAST.WIN
√ APOMAST.WOR	✖ APOMAST1.CBL	√ APOMAST1.SEL	✖ APOMAST2.CBL
✖ APOMAST3.CBL	✖ APOMASTN.BKP	✖ APOMASTN.CBL	✖ APOMASTN.LST
✖ APOMASTN.NEW	√ apomastn.prc	✖ APOMASTN.STD	√ apomastn.ws
√ APOMAX.DCL	√ APOMAX.REC	√ APOMAX.SEL	√ APOMAX.WOR
✖ APOMEIS.CBL	✖ APOMEVR.CBL	√ APOMT.REC	✖ APOP1MR.CBL
✖ APOPANAL.CBL	✖ APOPEN.CBL	✖ APOPFPA.CBL	✖ APOPFPR.CBL
✖ APOPMR.CBL	✖ APOPTIMCH.CBL	√ APOREAD.PRC	√ APOREC.CBL
√ APOTAM1.WIN	√ APOTEST.PRC	✖ APOUPD1.CBL	√ APOX.REC
√ APOX.SEL	√ APOX.WOR	√ APP.REC	✖ APPAR.CBL
✖ APPEIS.CBL	✖ APPEL.CBL	√ APPEL.PRC	√ APPEL.WOR
✖ APPEL0.CBL	√ APPEL0.PRC	√ APPEL0.WOR	✖ APPEL2.CBL
√ APPEL2.PRC	√ APPEL2.WOR	✖ APPEL3.CBL	✖ APPEL22.CBL
✖ APPFPA.CBL	✖ APPFPA1.CBL	√ APPIC.CBL	√ APPL.PRC
√ APPL.WOR	✖ APPL5A.CBL	✖ APPL5B.CBL	√ APPL5B.PRC
√ APPL5B1.PRC	√ APPL5C.PRC	√ APPL5C1.PRC	✖ APPOL3.CBL

✖ APPOP.CBL	√ appos.prc	√ appos.ws	√ APPREC.CBL
✖ APPRICE.CBL	✖ APPROFIT.CBL	✖ APPROM.CBL	√ APPROM.REL
√ APPROM.WOR	✖ APPROM2.CBL	✖ APPROM3.CBL	✖ APPROM4.CBL
✖ APPROM5.CBL	√ APPROM5.PRC	√ APPROM5.WOR	√ APRCSYN.PRC
√ APRD.REC	✖ APRDOP.CBL	√ APRDREC.CBL	✖ APRECOS1.CBL
✖ APRECOS1.LST	√ APRECOS1.PRC	√ APRECOS1.WOR	✖ APRECOST.CBL
√ APRECOST.PRC	√ APRECOST.WOR	✖ APRECSY1.CBL	✖ APRECSYN.CBL
√ APRECSYN.PRC	√ APRECSYN.STD	√ APRECSYN.WOR	√ APREL.MOV
√ APREL.REC	√ APREL.SEL	√ APREL.WOR	✖ APRELATE.CBL
✖ APRELEIS.CBL	√ APRELEIS.PRC	√ APRELEIS.WIN	√ APRELEIS.WOR
✖ APRELOP.CBL	✖ APREPHTL.CBL	✖ APREPORT.CBL	✖ APREPSTK.CBL
✖ APROL3.CBL	✖ APSALES.CBL	√ apsales.prc	√ apsales.ws
✖ APSCLEAR.CBL	✖ APSCREEN.CBL	√ APSELTM.PRC	√ APSELTM.WOR
✖ APSORT.CBL	✖ APSTAT.CBL	✖ APSTAT1.CBL	✖ APSTATUS.CBL
✖ APSTKEID.CBL	✖ APSTKMET.CBL	✖ APSTKPER.CBL	✖ APSTOCK9.CBL
✖ APSTOCK9.LST	✖ APSYG.CBL	✖ APSYGYP.CBL	√ APSYN.REC
√ APSYN.SEL	√ APSYNREC.CBL	√ APSYNT.PRC	✖ APSYNTEV.CBL
√ APT.REC	✖ APTAMDEL.CBL	√ aptameis.prc	√ aptameis.ws
✖ APTAMMAN.CBL	√ aptammon.prc	√ aptammon.ws	✖ APTAMOV.CBL
✖ APTAMPOS.CBL	√ aptampos.prc	√ aptampos.ws	√ APTAMPOS.WS
✖ APTAMPR.CBL	✖ APTAMPR.STD	✖ APTAMTIM.CBL	√ aptamtim.prc
√ aptamtim.ws	✖ APTCHADD.CBL	√ APTE1.REC	✖ APTE1OP.CBL
√ APTE1REC.CBL	√ APTEAM.PRC	√ APTEAM.REC	√ APTEAM.WRK
✖ APTEAMOP.CBL	√ APTEAMREC.CBL	✖ APTEAMS.CBL	✖ APTIMCH.CBL
√ APTIMCH.PRC	√ APTIMCH.REC	√ APTIMCH.SEL	√ APTIMCH1.PRC
✖ APTIMCHF.CBL	√ APTIMCHREC.CBL	✖ APTIMEID.CBL	✖ APTIMMEZ.CBL
✖ APTIMO.CBL	✖ APTIMP0.CBL	✖ APTIMTAM.CBL	√ aptm-ext.prc
√ aptm-ext.ws	√ aptm-win-wor	√ APTM.CPY	√ aptm.hlp
√ APTM.MOV	√ APTM.PRC	√ APTM.REC	√ APTM.SEL
√ APTM.WIN	√ APTM.WOR	✖ APTMEIS.CBL	✖ APTMENU.CBL
✖ APTMEVR.CBL	✖ APTMFILL.CBL	✖ APTMJOIN.CBL	✖ APTMOP.CBL
√ aptmsel.win	√ aptmwor.win	√ APTMX.PRC	√ APTMX.REC
✖ APTMXEIS.CBL	√ APTMXREC.CBL	✖ APTOTAL.CBL	✖ APTOTAL.LST
✖ APTPR2MZ.CBL	√ APTR.REC	✖ APTRANS.CBL	√ APTREC.CBL
✖ APTREIS.CBL	✖ APTROP.CBL	√ APTRREC.CBL	✖ APTRWHOP.CBL
✖ APTURN.CBL	√ APUNITS.PRC	√ APUNITS.WOR	√ APUPD.DCL
√ APUPD.MOV	√ APUPD.PRC	√ APUPD.REC	√ APUPD.SEL
√ APUPD.WOR	✖ APUPDAT3.CBL	✚ apupdate	✖ APUPDATE.CBL
✚ apupdmv	✖ APUPDMV.CBL	✖ APUPDSAY.CBL	✖ APWEEK.CBL
✖ APWEEKST.CBL	√ APWEVOL.PRC	✖ APYPOL.CBL	✖ APYPOL0.CBL
√ APYPOL0.PRC	√ APYPOL0.WOR	✖ APYPOL1.CBL	✖ APYPOLB.CBL
√ APYPREC.CBL	✖ APZERO.CBL	✖ AP_CHK.CBL	✖ ap_chk.cbl
√ ARTP.MOV	√ ARTP.REC	√ ARTP.SEL	√ ARTP.WOR
✖ ASC-AP.CBL	✖ ASC-AP0.CBL	✖ ASC-AP1.CBL	✖ ASC-AP2.CBL
✖ ASC-APAN.CBL	✖ ASC-CODE.CBL	✖ ASC-LGAN.CBL	✖ ASC-PEL.CBL
✖ ASC-PEL1.CBL	✖ ASC-PEL3.CBL	✖ ASC-PEL4.CBL	✖ ASC-PROM.CBL
✖ ASC-PROM1.CBL	✖ ASC-PROM3.CBL	✖ ASC-TEAM.CBL	✖ ASC-TEAM2.CBL
✖ ASC.CBL	√ ASUPD.DCL	√ ASUPD.MOV	√ ASUPD.PRC
√ ASUPD.REC	√ ASUPD.SEL	√ ASUPD.WOR	✖ ASUPDATE.CBL
✖ BACKUP.CBL	✚ BACKUP.WRN	✖ BAR.CBL	✚ BB
✖ boxask.cbl	√ boxask.lib	√ boxask.wor	✖ calc.cbl
✖ calc.std	√ calc.wor	✖ calctest.cbl	✖ CALL-EIS.CBL
✖ CALL.CBL	✖ CALL1.CBL	✖ CALLAPREL.CBL	✚ CANCPR.WRN
✖ CANCPRT.CBL	✚ CANCPRT.WRN	✚ candia	✚ candiain
✚ candiarm	✚ candiatr	✚ cbl	✚ cbllist
√ cent-y.prc	√ cent-y.wor	√ CKEY.WOR	√ ckeys.lib
√ CLOCK.PRC	√ CLOCK.WOR	✖ COMPANY.ALX	✖ COMPANY.CBL
✖ COMPANY.LST	✚ COMPANY.WRN	✚ compile	✚ compilelist
✖ COMPMEN.CBL	√ confirm.lib	√ confirm.prc	√ confirm.ws
✚ cpl	✖ CR-L.CBL	✖ CR-L1.CBL	✖ CS-L.CBL
√ cu-apo.prc	√ cu-apo.ws	√ CUSTOM.REC	√ DATE.ACC
√ DATE.PRC	√ DATE.WOR	√ DATE1.PRC	√ DATE1.WOR
√ DATECN.WOR	√ DATECTRL.PRC	√ DATECTRL.WOR	√ DATECTRX.PRC
✚ datelist	✖ DAY.CBL	√ DAY.PRC	√ DAY.WOR
√ DAYNAME.PRC	√ DAYNAME.WOR	✖ DB.CBL	√ DB.MOV
√ DB.REC	√ DB.SEL	√ DB.WOR	√ dbase.win
✖ DBEXEC.CBL	√ DBEXEC.PRC	√ dbf.win	√ dbfdesc.win
√ dbfld.win	√ dblink.win	✖ DBMARK.CBL	✖ DBPRNT.CBL
√ dbscrn.win	√ DECL.PRC	√ DED.MOV	√ DED.REC
√ DED.SEL	√ DED.WOR	✖ DEDEIS.CBL	√ deductio.prc
√ deductio.win	√ deductio.ws	√ delt.prc	√ delt.ws
√ deltio.prc	√ deltio.ws	✚ demo	√ digit3.prc

√ digit3.ws	√ dimen.win	√ DOC.MOV	✚ doc.prt
√ DOC.REC	√ DOC.SEL	√ DOC.TXT	√ DOC.WOR
√ dpp_parm.win	dvp	✖ EKTEIS.CBL	✖ EKTEPIL.CBL
√ EKTREC.CBL	√ emf.cbl	✖ EMP-AGO.CBL	✖ EMP-APO.CBL
✖ EMP-PEL.CBL	✖ EMP-PRO.CBL	✖ EMP-TIM.CBL	✖ entypo.cbl
✖ entypo2.cbl	✖ EP2PROM.CBL	√ EPAGREC.CBL	√ EPAGREC1.CBL
✖ EPDATE.CBL	✖ EPDATE1.CBL	✖ EPDEL.CBL	✖ EPDEL1.CBL
✖ EPDEL3.CBL	✖ EPDEL4.CBL	✖ EPESYN.CBL	√ EPI.MOV
√ EPI.REC	√ EPI.SEL	√ EPI.WOR	√ EPI1.MOV
√ EPI1.REC	√ EPI1.SEL	√ EPI1.WOR	√ EPIMASTER.CBL
✖ EPIT1OPMR.CBL	√ EPIT1REC.CBL	✖ EPITDEL2.CBL	✖ EPITEIS.CBL
✖ EPITEIS1.CBL	✖ EPITKAT.CBL	✖ EPITKAT1.CBL	✖ EPITKEY.CBL
✖ EPITMEN.CBL	✖ EPITMEN1.CBL	✖ EPITOPMR.CBL	✖ EPITOPTRAN.CBL
✖ EPITPEL.CBL	✖ EPITPROM.CBL	√ EPITREC.CBL	✖ EPITS.CBL
✖ EPITS1.CBL	✖ EPITSYN.CBL	√ EPITTRAN.CBL	√ EPITTRANSR.CBL
✖ EPMET.CBL	✖ EPMET1.CBL	✖ EPSYN.CBL	✖ EPSYN1.CBL
✖ EPTBL3.CBL	✖ EPTBL4.CBL	✖ EPTRAP.CBL	✖ EPTRAP1.CBL
✖ EPWEIS.CBL	✖ EPWEIS1.CBL	✚ ERR	√ EXEC
✖ EXEC.CBL	√ exeis.prc	√ exeis.ws	✖ EXFEIS.CBL
✖ EXMONEIS.CBL	√ expeis.prc	√ expeis.ws	√ expeis2.prc
√ expeis2.ws	√ expeis3.prc	√ expeis3.ws	√ expense.prc
√ expense.ws	√ expsale.prc	√ expsale.ws	√ expsale2.prc
√ expsale2.ws	√ f-keys.wor	✖ FEANEW.CBL	√ FEATURE.PRC
✖ fhandle.cbl	√ fhandle.lib	✚ file.grep	✚ file1
√ filename.cbl	√ filepara.cbl	√ FILTER.PRC	√ filter.win
✖ FINIT.CBL	✖ FINIT1.CBL	✖ FIXAPOLG.CBL	✖ FIXMTK1.CBL
✖ FLAG0.CBL	✖ FLAG1.CBL	✖ FLAG2.CBL	√ FLT.MOV
√ FLT.REC	√ FLT.SEL	√ FLT.WOR	√ fnext.prc
√ fnext.wor	✚ form.men	√ form.wor	√ formbin.rec
√ formbin.sel	√ formbin.wor	√ formbin1.wor	√ formvar.rec
√ formvar.sel	√ formvar.wor	√ formvar1.wor	√ frm-dmy.prc
√ frm-head.prc	√ frm-tail.prc	✖ FRM.CBL	√ frm.dcl
√ FRM.PRC	√ frm.prc	√ FRM.SPT	√ FRM.WIN
√ frm.wor	√ FRM.WOR	√ GEN.PRC	√ GEN.WRK
√ GEN1.PRC	√ GEN1.WRK	✚ ggg.93	✚ ggg.94
✚ ggg.95	✚ ggg.96	✚ ggg.97	✖ GNFLCHK.CBL
✖ GP.CBL	√ GR-US.PRC	✖ GR1CNT.CBL	√ GRA.MOV
√ GRA.REC	√ GRA.SEL	√ GRA.WOR	√ GRA1.MOV
√ GRA1.REC	√ GRA1.SEL	√ GRA1.WOR	✖ GRADATE.CBL
✖ GRADATE1.CBL	✖ GRADEL.CBL	✖ GRADEL1.CBL	✖ GRAKAT1.CBL
✖ GRAM1OPEN.CBL	✖ GRAM1OPMR.CBL	✖ GRAM1OPTR.CBL	√ GRAM1REC.CBL
√ GRAM1TREC.CBL	✖ GRAMDATE.CBL	✖ GRAMDEL.CBL	✖ GRAMEIS.CBL
✖ GRAMEIS1.CBL	✖ GRAMEN.CBL	✖ GRAMEN1.CBL	✖ GRAMET.CBL
✖ GRAMET1.CBL	✖ GRAMKAT.CBL	✖ GRAMOPEN.CBL	✖ GRAMOPMR.CBL
✖ GRAMOPTRAN.CBL	✖ GRAMPEL.CBL	√ GRAMPLHR.CBL	✖ GRAMPROM.CBL
√ GRAMREC.CBL	✖ GRAMTRAP.CBL	√ GRAMTRREC.CBL	✖ GRATRAP.CBL
✖ GRATRAP1.CBL	✖ GRCMENU.CBL	✖ GRCNT.CBL	✖ GRCNT1.CBL
√ GRCNT1REC.CBL	√ GRCNTREC.CBL	✖ GRDEL1.CBL	✖ GRDEL2.CBL
✖ GREPIN.CBL	✖ GREPOPEN.CBL	✖ GRIN.CBL	√ GRL.MOV
√ GRL.REC	√ GRL.SEL	√ GRL.WOR	√ GRPROMREC.CBL
✖ GRTBEX2.CBL	√ grtbex2.win	✖ GRTBL1.CBL	✖ GRTBL2.CBL
√ GSEL.WOR	√ GWIN.REC	√ GWIN.SEL	√ GWIN.WOR
√ H232.WOR	√ HEAD.CBL	√ HEAD.PRC	√ HEAD.REL
√ HEAD.WOR	√ HEAD0.PRC	√ HEAD0.REL	√ HEAD0.WOR
√ HEAD80.WOR	√ HEAD163.WOR	√ HEADAP.PRC	√ HEADAP.REL
√ HEADAP.WOR	√ header.win	√ header1.win	√ headplf.win
√ headprf.win	✚ help.men	√ hlp	√ hlp.rec
√ hlp.sel	√ hlp.wor	√ hlp1	√ hlpwin.wor
√ HOSP.MOV	√ HOSP.REC	√ HOSP.SEL	√ HOSP.WIN
√ HOSP.WOR	√ HOTEL.LIB	√ HOTEL.LNK	✚ hotel.men
√ HOTEL.MOV	√ HOTEL.REC	√ HOTEL.SEL	√ HOTEL.WIN
√ HOTEL.WOR	hotelsrc	√ HPL.MOV	√ HPL.REC
√ HPL.SEL	√ HPL.WIN	√ HPL.WOR	√ HPL2.WIN
✖ HTLEXAG.CBL	✚ htlexag.prt	✚ h_apo.men	✖ ICL-ALL.CBL
✖ ICL-CHK.CBL	✖ ICL-CHK.STD	✖ ICL-CHR.CBL	✚ icl-in
✖ ICL-IN.CBL	✖ ICL-OUT.CBL	√ icl-seq.rec	✖ ICLREAD.CBL
✖ ICLSALES.CBL	√ imd.lib	√ INSUPDEL.PRC	√ IOSP.MOV
√ IOSP.REC	√ IOSP.SEL	√ IOSP.WOR	✖ K1NEW.CBL
✖ K2NEW.CBL	✚ ken.doc	✚ ken.men	✚ ken.mtd
ken400	√ KEY.VAL	√ KEYS.WOR	√ kk-ap.prc
√ kk-ap.ws	√ kk-pel.prc	√ kk-pel.ws	√ kk-prom.prc
√ kk-prom.ws	✖ KODACC.CBL	✖ KODE.CBL	√ KODE.GET

√ KODE.PRC	√ KODE.READ	√ KODE.REC	√ KODE.SEL
√ KODE.WRK	✖ KODEOP.CBL	√ kost.prc	√ kost.ws
√ LEM.MOV	√ LEM.REC	√ LEM.SEL	√ LEM.WOR
✖ lg-ekt.cbl	✖ LG-PL.CBL	✖ lg.cbl	✖ lgag.cbl
✖ LGDEL.CBL	✖ lgempupd.cbl	√ lgink.cbl	√ lglink.cbl
✖ lgpelupd.cbl	✖ lgproupd.cbl	√ link.cbl	√ LINK.HLP
✖ LINKMENU.CBL	√ LINKREC.CBL	+ list	√ log.rec
√ log.sel	√ log1.nam	√ log1.par	√ log1.rec
√ log1.sel	√ LOX6.PRC	√ LOX6.WOR	√ LOXWOOD.PRC
√ LOXWOOD.WOR	√ LOXWOOD.WRK	✖ MAIN.ALX	✖ MAIN.CBL
√ MAIN.HLP	√ MAIN.HLP.STD	√ MAIN.LIB	√ main.prc
√ MAIN.REC	√ MAIN.SEL	√ main.ws	✖ MAINDOS.CBL
✖ MAK+APAN.CBL	✖ MAK+POLHS.CBL	✖ mak-ap.cbl	✖ MAK-AP.CBL
✖ MAK-APN.CBL	✖ mak-apy.cbl	✖ MAK-PEL.CBL	✖ MAK-PR.CBL
✖ MAK999KE.CBL	✖ MAK999MA.CBL	✖ MAKAG.CBL	✖ makag.cbl
✖ MAKAGD.CBL	✖ MAKAGD1.CBL	✖ MAKAGDF.CBL	✖ MAKAGP.CBL
✖ MAKagp.CBL	✖ MAKagt.CBL	✖ MAKAGT0-1.CBL	✖ MAKANAL2.CBL
✖ MAKAP-CS.CBL	✖ makap.cbl	✖ MAKAP.CBL	✖ MAKAP0-1.CBL
✖ MAKAP0-11.CBL	✖ MAKAP1-2.CBL	✖ makap1.cbl	✖ MAKAP1.CBL
✖ MAKAP2.CBL	✖ MAKAPAN.CBL	✖ MAKAPAN0.CBL	✖ MAKAPAN1.CBL
✖ MAKAPAN2.CBL	✖ MAKAPAN3.CBL	✖ MAKAPAN4.CBL	✖ MAKAPANAL.CBL
✖ makapch.cbl	✖ MAKAPCH.CBL	✖ MAKAPD1.CBL	✖ MAKAPD2.CBL
✖ MAKAPDF.CBL	✖ MAKAPDF.STD	✖ MAKAPF.CBL	✖ MAKAPFE.CBL
✖ MAKAPFL.CBL	✖ MAKAPGK.CBL	✖ MAKAPGO.CBL	✖ MAKAPLG.CBL
✖ MAKAPLST.CBL	✖ MAKAPMON.CBL	✖ MAKAPMR.CBL	✖ makapmr.cbl
✖ MAKAPMR1.CBL	✖ makapo.cbl	✖ MAKAPO1.CBL	✖ makapo1.cbl
✖ MAKAPOGR.CBL	✖ MAKAPOM1.CBL	✖ MAKAPOMA.CBL	✖ MAKAPPO.CBL
✖ MAKAPPOL.CBL	✖ MAKAPPOL1.CBL	✖ MAKAPPOL2.CBL	✖ MAKAPPR.CBL
√ MAKAPPR.PRC	√ MAKAPPR.WOR	✖ MAKAPRD.CBL	✖ MAKAPREL.CBL
✖ makapseq.cbl	✖ MAKAPSY.CBL	✖ MAKAPSYN.CBL	✖ MAKAPT0-1.CBL
✖ MAKAPT0-11.CBL	✖ MAKAPTCH.CBL	✖ MAKAPTE.CBL	✖ MAKAPTE1.CBL
✖ MAKAPTEAM.CBL	✖ MAKAPTIMCH.CBL	✖ MAKAPTY2.CBL	✖ MAKAPTYP.CBL
✖ MAKAPYP.CBL	✖ MAKAR.CBL	✖ MAKARADD.CBL	✖ MAKARDEL.CBL
✖ MAKARG.CBL	✖ MAKart.CBL	✖ MAKART.CBL	✖ MAKARTHRO.CBL
✖ MAKCNT.CBL	✖ MAKCOUNT.CBL	✖ MAKDISC.CBL	✖ MAKDPCNT.CBL
✖ MAKDPD.CBL	+ Makefile	✖ MAKEMP.CBL	✖ MAKEPIT.CBL
✖ MAKEPIT1.CBL	✖ MAKGRAM.CBL	✖ MAKGRAM1.CBL	✖ MAKGRX.CBL
✖ MAKINV.CBL	✖ MAKLINK.CBL	✖ MAKMAIN.CBL	✖ MAKMENU.CBL
✖ MAKOIKO.CBL	✖ MAKOYBA.CBL	✖ MAKPAR0-1.CBL	✖ MAKpel.CBL
✖ MAKPEL.CBL	✖ MAKPL-CS.CBL	✖ MAKPL.CBL	✖ MAKPL1.CBL
✖ MAKPL2.CBL	✖ MAKPLAN.CBL	✖ MAKPLAN1.CBL	✖ MAKPLAN3.CBL
✖ MAKPLAN4.CBL	✖ MAKPLAN10.CBL	✖ MAKPLC-D.CBL	✖ MAKPLCD.CBL
✖ MAKPLEVR.CBL	✖ MAKPLHELP.CBL	✖ MAKPLMR.CBL	✖ MAKPLNPO.CBL
✖ MAKPLPR.CBL	✖ MAKPOL.CBL	✖ MAKpol.CBL	✖ makpol.cbl
✖ MAKPOL1.CBL	✖ MAKPOLD.CBL	✖ MAKPOLD1.CBL	✖ MAKPOLD2.CBL
✖ MAKPOLDF.CBL	✖ MAKPOLHS.CBL	✖ MAKPR-CS.CBL	✖ MAKPR.CBL
✖ makpr.cbl	✖ MAKPR8.CBL	✖ MAKPRALT.CBL	✖ MAKPRAN.CBL
✖ MAKPRAN3.CBL	✖ MAKPRMR.CBL	✖ MAKpro.CBL	✖ MAKPROM.CBL
✖ MAKSYN1.CBL	✖ MAKSYN2.CBL	✖ MAKSYN3.CBL	✖ MAKSYN5.CBL
✖ MAKSYNT.CBL	✖ MAKtim.CBL	✖ MAKTIM1.CBL	✖ MAKTRAN.CBL
✖ MAKTYPE.CBL	✖ MAKUNIT2.CBL	✖ mak_agd.cbl	✖ MAK_AGD.CBL
✖ MAK_AGDAT.CBL	✖ mak_agt.cbl	✖ mak_agt1.cbl	✖ MAK_APAN.CBL
✖ mak_apan.cbl	✖ MAK_PLAN.CBL	✖ mak_plan.cbl	✖ mak_pln1.cbl
✖ mak_plpol.cbl	✖ mak_pol.cbl	✖ mak_pol1.cbl	✖ mak_pol2.cbl
✖ MAK_PRAN.CBL	✖ mak_pran.cbl	✖ mak_prn1.cbl	√ mama
+ maris.prt	✖ MARK.CBL	√ MEN.REC	√ MEN.SEL
+ MENCPL	✖ MENCPL.CBL	√ MENCPL.REC	+ mencpl.scr
√ MENCPL.SEL	+ MENCPL.WRN	+ MENDOC	✖ MENDOC.CBL
√ MENDOC.REC	√ MENDOC.SEL	✖ MENEXT.CBL	✖ MENFYI.CBL
✖ MENREFR.CBL	✖ MENRTS.CBL	✖ MENRTS.LNX	✖ MENRTS.OLD
+ MENRTS.WRN	√ MENSEL.PRC	√ MENSEL2.PRC	√ MENSPT.PRC
√ MENSPT.REC	√ MENSPT.SEL	√ MENSPT.WOR	+ menu
✖ MERSRT.CBL	√ metag.win	✖ MINIE-APO.CBL	✖ MINIE-PEL.CBL
√ ML.WOR	√ mlcust.win	√ MLDNLD.DCL	√ MLDNLD.REC
√ MLDNLD.SEL	√ MLDNLD.WOR	√ mlfilter.win	√ mlfilter.wor
√ mlprod.win	√ mlprom.win	√ MLTERM.DCL	√ MLTERM.LNK
√ MLTERM.MOV	√ MLTERM.REC	√ MLTERM.SEL	√ mlterm.win
√ MLTERM.WOR	√ MNFYI.REC	√ MNFYI.SEL	√ MNFYI.WOR
√ MNFYIMAS.REC	√ MNFYIMAS.SEL	√ MNFYIMAS.WOR	+ mnrefr.dat
√ MNREFR.REC	√ MNREFR.SEL	√ MNREFR.WOR	√ MNUSR.MOV
√ MNUSR.PRC	√ MNUSR.REC	√ MNUSR.SEL	√ mnusr.win
√ mnusr.wor	√ MNUSR.WOR	✖ MNUSREIS.CBL	√ MNUSREIS.DCL
√ MNUSREIS.PRC	√ MNUSREIS.PRG	√ MNUSREIS.WOR	√ MNUSRPER.REC

√ MNUSRPER.SEL	√ MNUSRPER.WOR	√ MON.PRC	√ MON.WIN
√ MON.WRK	✖ msg.cbl	√ msg.lib	✖ MSMAKAN.CBL
✖ MSSEQ.CBL	✖ MYF.CBL	✖ MYMAIN.CBL	√ NAIOXI.WOR
√ NAME.CBL	√ ndcust.win	√ ndfilter.win	√ ndfilter.wor
√ ndprod.win	√ ndremark.prc	√ ndremark.win	√ ndremark.ws
√ ndterm.win	✖ NEAXR.CBL	✖ NEAXR.STD	✖ NEAXRDOS.CBL
✖ NENREFR.CBL	√ NEOS-APO.FIL	√ NEOS-APO.SEL	√ NEOS-PEL.FIL
√ NEOS-PEL.SEL	√ NEOS-PRO.FIL	√ NEOS-PRO.SEL	✖ NEWANAL.CBL
✚ newcpl	✖ NEWPRG.CBL	✖ nor-create.cbl	✖ nor-get.cbl
√ nor-lnk.wor	✖ nor-menu.cbl	√ nor-menu.rec	√ nor-menu.sel
✚ nor-menu.spt	√ nor-pelat.rec	√ nor-pelat.sel	✚ norand.men
✖ normark.cbl	√ num2str.wor	✖ numdis.cbl	✖ numtext.cbl
√ numtext.lib	√ OFF.MOV	√ OFF.REC	√ OFF.SEL
√ OFF.WOR	√ offers.prc	√ offers.ws	√ OFFH.MOV
√ OFFH.REC	√ OFFH.SEL	√ OFFH.WIN	√ OFFH.WOR
√ OFFI.MOV	√ OFFI.REC	√ OFFI.SEL	√ OFFI.WOR
√ oikod.prc	√ oikod.ws	√ OKCANC.PRC	✚ olddates
✖ oldnum.cbl	✖ OM-OM.CBL	√ ORD-TLN.WOR	√ ORD-VAR1.PRC
✖ ORDAPTM1.CBL	✖ ORDDATE2.CBL	✖ ORDEID.CBL	✖ ORDEIDDT.CBL
✖ ORDEIS.CBL	√ ORDER.REC	√ order.win	✖ ORDOP.CBL
✖ ORDSCAN.CBL	√ ORP.MOV	√ ORP.REC	√ ORP.SEL
√ ORP.WOR	√ ORPH.MOV	√ ORPH.REC	√ ORPH.SEL
√ ORPH.WIN	√ ORPH.WOR	√ ORPI.MOV	√ ORPI.REC
√ ORPI.SEL	√ ORPI.WOR	✚ out	√ over.cbl
√ OVFLOW.PRC	✚ par-anf.prt	✚ par-mas.prt	✚ par-std.prt
√ PAR-VAR1.PRC	√ PAR-VAR2.PRC	✖ PARACHK.CBL	✖ PARAGOP.CBL
√ PARAGREC.CBL	√ parametr.prc	√ parametr.ws	√ PARCNTREC.CBL
✖ PARDATE2.CBL	✖ PARDATE2.MARK	✖ PAREIS1.CBL	√ parm-ago.prc
√ parm-ago.ws	√ parm-apo.prc	√ parm-apo.ws	√ parm-gen.prc
√ parm-gen.ws	√ parm-pel.prc	√ parm-pel.ws	√ parm-pro.prc
√ parm-pro.ws	√ parm-sys.prc	√ parm-sys.ws	√ parm-tim.prc
√ parm-tim.ws	✖ PARM.CBL	√ PARM.PRC	√ PARM.REC
√ PARM.SEL	√ PARM.WOR	✖ PAROP.CBL	✖ PAROPCNT.CBL
✖ PARPEL3.CBL	✖ PARPEL3.MARK	√ PASS.REC	√ PASSWD.PRC
√ PASSWD.WOR	√ PEL.DCL	√ PEL.MOV	√ PEL.PRC
√ PEL.REC	√ PEL.SEL	√ pel.sel	√ PEL.WOR
√ PEL.WRK	√ PEL1.SEL	✖ PEL2HPL.CBL	√ PELANAL.REC
√ PELAT.MOV	√ PELAT.REC	√ PELAT.SEL	√ PELAT.WIN
√ PELAT.WOR	√ PELAT2.WIN	✖ PELDEL.CBL	✖ PELDISC.CBL
√ PELDISC.PRC	√ PELDISC.WOR	√ PELDREC.CBL	√ PELF.PRC
√ PELF.WOR	✖ PELFEIS.CBL	✖ PELFOP.CBL	√ PELFPRT.PRC
√ PELFREC.CBL	✖ PELIDX.CBL	✖ PELINK.CBL	√ PELINS.PRC
√ PELMAST.REC	✖ PELNEW.CBL	√ pelnme.prc	√ PELNMEREC.CBL
✖ PELOPEN.CBL	✖ PELPROD.CBL	√ pelprod.win	√ PELREAD.PRC
✖ PELSEQ.CBL	✖ PELTEAM.CBL	✖ PELYPEVR.CBL	✖ PELYPOM.CBL
✖ PELYPOP.CBL	√ PERIOD.WIN	✖ PFORTEID.CBL	✖ PFORTEID2.CBL
✖ PFORTSYN.CBL	✖ pkmenu.cbl	√ pkmenu.wor	√ pkwin.prc
√ pkwin.wor	✖ PL-AFM.CBL	✖ PL-MOVE.CBL	✖ PL-PL.CBL
√ PLAFM.PRC	√ PLAGE.PRC	√ PLAGE.WOR	✖ PLAIM1.CBL
✖ PLAIM2.CBL	✖ PLANAL.CBL	√ PLANAL.PRC	√ PLANAL.WOR
✖ PLANAL1.CBL	√ PLANAL1.PRC	√ PLANAL1.WOR	✖ PLANAL2.CBL
√ PLANAL2.PRC	√ PLANAL2.WOR	✖ PLANAL3.CBL	√ PLANAL3.PRC
√ PLANAL3.WOR	✖ PLANCHK.CBL	✖ PLANREAD.CBL	✖ PLANSRT.CBL
✖ PLAPEMP.CBL	✖ PLAPOGR.CBL	✖ PLCHK.CBL	✖ plchk.cbl
✖ PLCHKAFM.CBL	√ PLCHKAFM.PRC	√ PLCHKAFM.WOR	√ PLCOM.PRC
√ PLCOM.WOR	✖ PLCOMMEN.CBL	✖ PLCRDB.CBL	√ PLDECL.PRC
✖ PLEKPT.CBL	√ PLEKPT.PRC	√ PLEKPT.WOR	√ PLEP.PRC
√ PLEP.WOR	✖ PLEPYPOL.CBL	✖ PLETIK.CBL	✖ PLEVRET.CBL
✖ PLEVRET1.CBL	✖ PLEVRET2.CBL	✖ PLFAIM.CBL	✖ PLFAIMEV.CBL
√ PLFAIMEV.PRC	√ PLFAIMEV.WOR	✖ PLFAIML.CBL	√ PLFAIMREC.CBL
✖ PLFILTER.CBL	√ PLFILTER.PRC	√ PLFILTER.WOR	✖ PLFLCHK.CBL
√ PLGFP.PRC	√ PLGLP.PRC	√ PLGLP.WOR	√ PLGPP.PRC
√ PLGPP.WOR	√ PLGPP1.PRC	√ PLGPP1.PRDC	√ PLGPP1.WOR
√ PLHELPREC.CBL	✖ PLHSMHN.CBL	✖ PLHSMHNP.CBL	✖ PLIN.CBL
✖ PLISOZ.CBL	√ PLISOZ.PRC	√ PLISOZ.WOR	✖ PLKATEL.CBL
√ PLLIST.PRC	√ PLLIST.WOR	✖ PLLOGI.CBL	✖ PLMHN.CBL
√ PLMHN.PRC	√ PLMHN.WOR	✖ PLMHNP.CBL	✖ PLMHNPE.CBL
✖ PLMHNPEP.CBL	✖ PLMHNSP.CBL	✖ PLMHNSP.MARK	√ PLMHNSP.PRC
√ PLMHNSP.WOR	✖ PLMHNSPP.CBL	✖ PLMHNSYN.CBL	✖ PLMREIS.CBL
✖ PLMYF.CBL	√ PLNM.PRC	✖ PLNMDT.CBL	✖ PLNME.CBL
✖ PLNMOP.CBL	✖ PLNS.CBL	✖ PLOPANAL.CBL	✖ PLOPMR.CBL

✖ PLPELISOZ.CBL	✖ PLPEREVR.CBL	✖ PLPERISOZ.CBL	√ PLPERISOZ.PRC
√ PLPERISOZ.WOR	✖ PLPERSYG.CBL	✖ PLPERYPOL.CBL	√ PLPERYPOL.PRC
√ PLPERYPOL.WOR	✖ PLPOLREP.CBL	✖ PLPRANAL.CBL	√ PLPRANAL.PRC
✖ PLPREAD.CBL	✖ PLPRYP.CBL	√ PLPRYP.PRC	√ PLPRYP.WOR
✖ PLRANGE.CBL	✖ PLREAD.CBL	✖ PLSALE1.CBL	✖ PLSCREEN.CBL
√ PLSRT.WOR	✖ PLSTAT.CBL	√ PLSTAT.PRC	√ PLSTAT.WOR
√ PLSTD.CBL	✖ PLSYG.CBL	✖ PLSYGALFA.CBL	√ pltmove.win
✖ PLTREIS.CBL	✖ PLYPAGE.CBL	✖ PLYPFAST.CBL	√ PLYPL.PRC
√ PLYPL.WOR	✖ PLYPOL.CBL	√ PLYPOL.PRC	√ PLYPOL.WOR
√ PLYPREC.CBL	✖ PLYPSEND.CBL	√ PLYPSEND.PRC	√ PLYPSEND.WOR
✖ PLZERO.CBL	✖ pl_chk.cbl	√ POLD-JOB.PRC	√ polhs.win
✖ POLREAD.CBL	√ POLREC.CBL	√ POS.MOV	√ POS.REC
√ POS.SEL	√ POS.WIN	√ POS.WOR	✖ PR-AFM.CBL
✖ PR-MOVE.CBL	✖ PR-PR.CBL	√ PR1.MOV	√ PR1.REC
√ PR1.SEL	√ PR1.WOR	√ PR2.MOV	√ PR2.SEL
√ PR2.WOR	√ PR3.MOV	√ PR3.SEL	√ PR3.WOR
√ PR4.MOV	√ PR4.SEL	√ PR4.WOR	√ PR5.MOV
√ PR5.SEL	√ PR5.WOR	√ PR6.MOV	√ PR6.SEL
√ PR6.WOR	√ PR11.SEL	√ PRAFM.PRC	✖ PRAFMAK.CBL
√ PRAGE.PRC	√ PRAGE.WOR	✖ PRANAL.CBL	√ PRANAL.PRC
√ PRANAL.WOR	√ PRANALREC.CBL	✖ PRANCHK.CBL	✖ PRANREAD.CBL
✖ PRAPEMP.CBL	✖ PRAPOGR.CBL	✚ PRC	√ prc-date.lib
✖ prchk.cbl	✖ PRCHK.CBL	✖ PRCHKAFM.CBL	√ PRCHKAFM.PRC
√ PRCHKAFM.WOR	√ PRCOM.PRC	√ PRCOM.WOR	✖ PRDEL.CBL
√ prdsel.prc	√ prdsel.ws	✖ PREVRET.CBL	✖ PREVRET1.CBL
✖ PREVRET6.CBL	√ PREVRET6.PRC	√ PREVRET6.WOR	√ PRF.PRC
√ PRF.WOR	✖ PRFEIS.CBL	✖ PRFLCHK.CBL	✖ PRFOP.CBL
√ PRFPRT.PRC	√ PRFREC.CBL	✖ PRG-TIM.CBL	√ PRGEIS.PRC
√ PRGEIS.WOR	√ PRGFP.PRC	√ PRGLP.PRC	√ PRGLP.REL
√ PRGLP.WOR	√ PRGLP132.PRC	√ PRGLP132.WOR	√ pricetyp.win
✖ PRIN.CBL	✚ printer.lin	√ printer.var	√ printer.wor
✖ PRISOZ.CBL	√ PRISOZ.PRC	√ PRISOZ.WOR	✖ PRKATEL.CBL
√ PRLIST.PRC	√ PRLIST.WOR	✖ PRLOGI.CBL	√ PRMASTREC.CBL
✖ PRMERGE.CBL	✖ PRMHN.CBL	√ PRMHN.PRC	√ PRMHN.WOR
✖ PRMHNSYN.CBL	✖ PRMREIS.CBL	✖ PRMYF.CBL	√ PRNM.PRC
√ PRO.MOV	√ PRO.REC	√ pro.rec	√ pro.sel
√ PRO.SEL	√ PRO.WOR	√ PRO1.SEL	✚ PROBLEMS
✖ PROD-ASC.CBL	✖ PROD-ASC.STD	✖ PROD-ASC1.CBL	✖ PROFF-OP.CBL
✖ PROFFCHK.CBL	✖ PROFFERS.CBL	✖ PROFFEV1.CBL	✖ PROFFEV2.CBL
✖ PROFFEVP.CBL	√ PROM.WIN	√ PROM2.WIN	✖ promhq.cbl
√ PROMX.DCL	√ PROMX.MOV	√ PROMX.REC	√ PROMX.SEL
√ PROMX.WOR	✖ PROMX2PR.CBL	✖ PROPANAL.CBL	✖ PROPEN.CBL
✖ PROPMR.CBL	✖ PRORD-OP.CBL	√ prord.acc	√ PRORD.MOV
√ PRORD.REC	√ PRORD.SEL	√ PRORD.WIN	√ PRORD.WOR
✖ PRORDEKD.CBL	✖ PRORDERS.CBL	√ prorders.prc	✚ prorders.prt
√ prorders.ws	✖ PRPEREVR.CBL	✖ PRPERISOZ.CBL	√ PRPERISOZ.PRC
√ PRPERISOZ.WOR	✖ PRPERSYG.CBL	✖ PRPERYPOL.CBL	√ PRPERYPOL.PRC
√ PRPERYPOL.WOR	✖ PRPLYP.CBL	√ PRPLYP.PRC	√ PRPLYP.WOR
✖ PRRANGE.CBL	✖ PRSCREEN.CBL	✖ PRSTAT.CBL	✖ PRSTAT.GEN
√ PRSTAT.PRC	√ PRSTAT.WOR	√ PRSTD.CBL	✖ PRSYG.CBL
✖ PRSYGALFA.CBL	√ prt&bin.win	√ PRT.REC	√ PRT.SEL
√ PRT.WOR	√ prt132.rec	√ prt132.sel	√ prt132.wor
√ prt232.rec	√ prt232.sel	√ prt232.wor	✖ PRTEIS.CBL
✖ PRTEIS.LST	✖ PRTEIS.OLD	✚ PRTEIS.WRN	√ prtlist.rec
√ prtlist.sel	√ prtlist.wor	✖ PRTPLYP.CBL	✖ PRTREIS.CBL
✖ PRTSEL.CBL	✖ PRTSEL.OLD	√ PRTSTYAP.PRC	√ PRTSTYLE.PRC
√ PRUPD.DCL	√ PRUPD.MOV	√ PRUPD.PRC	√ PRUPD.REC
√ PRUPD.SEL	√ PRUPD.WOR	✖ PRUPD1.CBL	✖ PRUPDATE.CBL
✖ PRUPREAD.CBL	✖ PRYPAGE.CBL	✖ PRYPAGE.MARK	✖ PRYPFAST.CBL
✖ PRYPOL.CBL	√ PRYPOL.PRC	√ PRYPOL.WOR	✖ PRZERO.CBL
✖ pr_chk.cbl	√ PSCR2.CBL	√ PSCR3.CBL	✖ puzzle.cbl
√ pwin_dow.wor	√ REC.REC	√ REC.SEL	√ REC.WOR
√ REC1.REC	√ REC1.SEL	√ REC1.WOR	✚ reclist
√ recprn.wor	✚ recs	√ rel300.mov	√ rel300.prc
√ rel300.rec	√ rel300.sel	√ rel300.wor	✖ RELAPO.CBL
✖ RELPEL.CBL	✖ RELPRO.CBL	✖ reply.cbl	√ reply.lib
✖ RESTLIST.CBL	✚ RESTLIST.WRN	✖ RESTORE.CBL	✚ RESTORE.WRN
✖ rhandle.cbl	√ rhandle.lib	✖ rhandle.lst	✖ rhandle.old
√ rmpanel.ws	√ rmpanels.ws	✚ rmpath	✖ SALES-ASC.CBL
✖ SALES-ASC.STD	✖ SALES-ASC1.CBL	√ SAORSU.PRC	√ SCRPRT.PRC
√ SCRPRT.WOR	√ seira.win	√ SELECTKK.PRC	√ SELECTKK.WOR

✖ SEQ-OTH.CBL	√ SEQKST.REC	√ SEQMST.REC	√ SEQPRO.REC
✖ SEQREAD.CBL	√ SETYPE.PRC	✖ SKT.CBL	√ SNF.MOV
√ SNF.REC	√ SNF.SEL	√ SNF.WIN	√ SNF.WOR
√ SNF1.MOV	√ SNF1.REC	√ SNF1.SEL	√ SNF1.WIN
√ SNF1.WOR	√ SORT.PRC	√ SORT.REC	√ SORT.SEL
√ SPECIAL.NAM	✚ srcupd	✖ STATAPS.CBL	✖ STATBAR.CBL
✖ STATBAR1.CBL	✖ STATBAR2.CBL	✖ STATBAR3.CBL	✖ STATPRT.CBL
✖ STATPRT1.CBL	✖ STATPRT2.CBL	✖ STATPRT3.CBL	✖ STATPSAL.CBL
✖ STATPSAP.CBL	✖ STATPSAS.CBL	✖ STATPSSP.CBL	✖ STATSALE.CBL
✖ STELIOS.CBL	✖ STELIOS1.CBL	✖ str-fcts.cbl	√ str-fcts.prc
√ str-fcts.wor	√ STYAP.PRC	√ STYLE.CBL	√ STYLE.PRC
✖ SYNCHK.CBL	✖ SYNT.CBL	√ synt.prc	√ synt.ws
✖ SYNT00.CBL	√ synthof.prc	√ synthof.ws	✖ SYNTOP.CBL
✖ SYNTPR.CBL	√ SYS.WOR	✖ SYSCALL.CBL	✖ SYSDESC.CBL
✚ SYSDESC.WRN	√ SYSDOS.WOR	✖ SYSENV.CBL	√ SYSENV.REC
√ SYSENV.SEL	√ SYSENV.WOR	√ TAG.MOV	√ TAG.REC
√ TAG.SEL	√ TAG.WOR	√ TAG1.REC	√ TAG2.REC
√ TAG3.REC	✖ TAMAPTM.CBL	√ tamcdsel.prc	√ tamcdsel.ws
√ TAMDEL.MOV	√ TAMDEL.PRC	√ TAMDEL.REC	√ TAMDEL.SEL
√ TAMDEL.WOR	√ TAPOM.WIN	✖ TAPOMEIS.CBL	✖ TAPOMOP.CBL
√ TAPOMREC.CBL	✖ TAPQNTY.CBL	√ TAPQNTY.MOV	√ TAPQNTY.REC
√ TAPQNTY.SEL	√ TAPQNTY.WOR	✖ TAPQNTYR.CBL	√ TARTHREC.CBL
√ TARTHROREC.CBL	✖ TARTHROSEE.CBL	✖ TARUNL.CBL	√ TBL.PRC
√ TBL.WOR	✖ TCNTGEN.CBL	✖ TCOMEIS.CBL	√ TCOMMENT.CBL
✖ TCOMOP.CBL	√ TCOMREC.CBL	√ TCOUNTREC.CBL	✖ TDEFPAR.CBL
√ TDLN.WOR	✖ TDMET.CBL	√ TDP.MOV	√ TDPCREC.CBL
✖ TDPEKT.CBL	✖ TDPEKT1.CBL	✖ TDPEKT2.CBL	✖ TDPINF.CBL
√ TDPREC.CBL	✖ TDPREIS.CBL	√ team-nam.prc	√ team-nam.ws
√ temp-kk.wor	√ TEMP.MOV	√ TEMP.REC	√ TEMP.SEL
√ TEMP.WOR	√ TEMPA.REC	√ TEMPB.REC	✚ terminfo.cfg
✖ TEST.CBL	✖ TESTPRG.CBL	✖ TESTPRG.CBL,v	✚ TESTPRG.WRN
√ TFEAT.PRC	✖ TFORTEID.CBL	✖ TFORTEID2.CBL	✖ TGAKYR.CBL
✖ TGEIS.CBL	✖ TGEKD.CBL	✖ TGEVR.CBL	✖ TGEVR1.CBL
✖ TGGEN.CBL	✖ TGGEN1.CBL	✖ TGGENRP1.CBL	✖ TGGENRP2.CBL
√ TGGENRP2.PRC	√ TGGENRP2.WOR	✖ TGMET.CBL	✖ TGSEE.CBL
✖ THELPOP.CBL	√ THELPREC.CBL	√ THLP.CBL	√ THLPREC.CBL
√ thmpa.prc	√ thmpa.ws	√ TIM.MOV	√ tim.prc
√ TIM.REC	√ TIM.SEL	√ TIM.WOR	√ tim.ws
✖ TIMAG.CBL	✖ TIMAGHLP.CBL	✖ TIMAKYR.CBL	✖ TIMCHK.CBL
✖ TIMCNT.CBL	√ TIMCNTREC.CBL	✖ TIMDEL.CBL	✖ TIMEBAR.CBL
✖ TIMEIS.CBL	✖ TIMEKD.CBL	✖ TIMEVR.CBL	✖ TIMEVR1.CBL
✖ TIMGEN.CBL	✖ TIMGEN1.CBL	✖ TIMIN.CBL	✖ TIMINF.CBL
✖ TIMOPEN.CBL	√ timpelf.win	√ TIMREC.CBL	√ timunit.win
✖ TINVLIST.CBL	✖ TINVOP.CBL	√ TINVREC.CBL	✖ TIPAR.CBL
✖ title.cbl	✖ titleden.cbl	✖ titledet.cbl	✖ titlekar.cbl
✖ titlent.cbl	✖ titlepka.cbl	✖ titleter.cbl	✖ TITLOP.CBL
✖ TLINKCP.CBL	✖ TLINKOP.CBL	√ TLN-PAR.WOR	√ TLN.WOR
✖ TMETEIS.CBL	✖ TMETOP.CBL	√ TMETREC.CBL	✖ TMFLCHK.CBL
√ TMOV1.CBL	√ TMOV2.CBL	✚ tmp	✚ tmp.cp
√ TMP.REC	√ tmp.sel	√ TMP.SEL	√ TMP.WOR
√ TMP1.WIN	✚ tmplist	✖ TOMEIS.CBL	✖ TOMOP.CBL
√ TOMREC.CBL	√ tom_win.prd	√ tom_win.wor	✖ TOPARTHR.CBL
✖ TOPCNT.CBL	✖ TOPPOLHS.CBL	✖ TPLOMEIS.CBL	✖ TPLOMOP.CBL
√ TPLOMREC.CBL	✖ TPOL2HMER.CBL	✖ TPOL3HMER.CBL	✖ TPOLDOP.CBL
√ TPOLDREC.CBL	✖ TPOLHMEROL.CBL	√ TPOLHSREC.CBL	✖ TPOLSEE.CBL
✖ tpolsee.cbl	✖ TPROMEIS.CBL	✖ TPROMOP.CBL	√ TPROMREC.CBL
√ TRA.MOV	√ TRA.REC	√ TRA.SEL	√ TRA.WOR
√ TRAN.REC	√ TRAN.SEL	√ TRAN.WOR	√ TRAPANREC.CBL
✖ TRAPCMENU.CBL	✖ TRAPEIS.CBL	✖ TRAPEVR.CBL	✖ TRAPOP.CBL
√ TRAPREC.CBL	√ TRAREC.CBL	√ TRGETREC.CBL	√ TRWH.MOV
√ TRWH.REC	√ TRWH.SEL	√ TRWH.WIN	√ TRWH.WOR
✖ TSTEIS.CBL	✖ TSTEIS1.CBL	√ TSYMP.CBL	✖ TTMETEIS.CBL
✖ TTMETOP.CBL	✖ TVIEW.CBL	✖ TVIEW1.CBL	√ typln.wor
✖ TYPOMEIS.CBL	✖ TYPOMOP.CBL	√ TYPOMREC.CBL	√ typom_win.prd
√ typom_win.wor	√ updprice.win	✖ UPDTRS.CBL	√ UPPER-LOWER
✚ usr	✖ V3-AGO.CBL	✖ V3-APO.CBL	✖ V3-EPGR.CBL
✖ V3-EPGR1.CBL	✖ V3-PEL.CBL	✖ V3-PRO.CBL	✖ V3-TIM.CBL
✖ V3-XPERT.CBL	✖ V5MAKUPD.CBL	√ valid-ch.prc	√ valid-ch.ws
√ VAR.MOV	√ VAR.REC	√ VAR.SEL	√ VAR.WOR
√ VAR1.MOV	√ VAR1.PRC	√ VAR2.PRC	√ VARLAP.PRC
√ VARML1.PRC	√ VARML2.PRC	√ vars.win	√ VARTD1.PRC
√ VARTD2.PRC	√ W-APANAL.REC	√ WAPO.MOV	√ WAPO.REC

√ WAPO.SEL	√ WAPO.WOR	√ WAPO1.PRC	√ WAPO1.SEL
√ WAPO1.WOR	√ WAPOM1.REC	√ wcb.prc	√ wcb.wor
√ win.wor	√ wind-sta.prc	√ wind-sta.ws	√ window.alla
√ window.allf	√ window.allp	√ window.alpr	√ window.apf
√ window.apt	√ window.cd1	√ window.cd2	√ window.cd3
√ window.comm	√ window.dp	√ window.exaf	√ window.excr
√ window.exed	√ window.exm	√ window.exmn	√ window.exp
√ window.expf	√ window.expr	√ window.exsd	√ window.faim
√ window.fea	√ window.fpa	√ window.ftr	√ window.hlp
√ window.kpar	√ window.mon	√ window.oma	√ window.pdp
√ window.pkey	√ window.pkp	√ window.plf	√ window.ply
√ window.pper	√ window.prc	√ window.prd	√ window.prf
√ window.psel	√ window.timp	√ window.type	√ window.unit
√ window.wor	√ WINDOWS.REC	√ window_f.all	√ window_p.all
✖ WINSEL.CBL	√ WINSEL.PRC	√ WINSEL.REC	√ WINSEL.REC@
√ WINSEL.SEL	√ WINSEL.SEL@	√ WINSEL.WOR	√ WINSEL.WOR@
✖ WINSEL1.CBL	✖ WINSEL2.CBL	√ wintap.prc	√ win_buf.wor
√ win_dow.apf	√ win_dow.apg	√ win_dow.apk	√ win_dow.app
√ win_dow.apt	√ win_dow.com	√ win_dow.dpp	√ win_dow.ep1
√ win_dow.ep2	√ win_dow.epa	√ win_dow.ept	√ win_dow.hlp
√ win_dow.inv	√ win_dow.kk	√ win_dow.kod	√ win_dow.met
√ win_dow.par	√ win_dow.pco	√ win_dow.plc	√ win_dow.prc
√ win_dow.prck	√ win_dow.prd	√ win_dow.prdk	√ win_dow.prm
√ win_dow.prmk	√ win_dow.tig	√ win_dow.tim	√ win_dow.tom
√ win_dow.tra	√ win_dow.wor	√ win_dow.wor.ol	√ win_dow1.inv
√ win_dow1.par	√ win_dow1.prd	√ win_dow1.tim	√ win_dow2.inv
√ win_dow2.prd	√ win_dow3.prd	√ win_dow4.prd	√ win_lap.prc
√ win_ord.prc	√ win_par.prc	√ win_plf.prc	√ win_plh.prc
√ win_plm.prc	√ win_plm1.prc	√ win_prf.prc	√ win_tap.prc
√ win_tim.prc	✚ worlist	√ wpel.cbl	√ WPEL.MOV
√ wpel.prc	√ WPEL.REC	√ WPEL.SEL	√ WPEL.WOR
√ wpel.wor	√ wpel2.prc	√ wpel3.prc	√ WPRO.MOV
√ WPRO.REC	√ WPRO.SEL	√ wpro.wor	√ WPRO.WOR
√ wprom.prc	√ wprom2.prc	√ wprom3.prc	√ ws-date.lib
✚ Xago.CANDIA	✚ Xago.men	✚ Xapo.CANDIA	✚ Xapo.men
✚ Xbas.doc	✚ Xbas.men	✚ Xbas.mtd	✚ Xcp
✚ Xemp.men	✚ Xgra.men	✚ Xlog.men	✖ XP0-1.CBL
✖ XP1-1.CBL	✚ Xpel.men	√ XPERT-APO.PRC	√ XPERT-PEL.PRC
√ XPERT-PRO.PRC	√ XPERT-TIM.PRC	✚ xpert06	✖ XPERTDB.CBL
✚ Xpro.CANDIA	✚ Xpro.men	✚ Xsta.men	✚ Xtim.men
✖ YEARMEN.CBL	✚ YEARMEN.WRN	✖ YP-YP.CBL	✖ YPOK-APO.CBL
✖ YPOK-PEL.CBL	✖ YPOK-PRO.CBL	✖ YPOK-TIM.CBL	✖ YPOKGET.CBL
✖ YPOMKNEW.CBL	✖ YPOMNEW.CBL	✖ _IL01.CBL	✖ _IL01CHK.CBL
✖ _IL01TEST.CBL			

Appendix 14

This appendix presents the complete set of programs of the XPERT Hotel software application and their main source code file(s).

Program	Source File(s)	Program	Sourcefile(s)
ACHK	ACHK.CBL	AFM	AFM.CBL
AFMFIX	AFMFIX.CBL	AG-EKT	ag-ekt.cbl
AG2HMER	AG2HMER.CBL	AGARSEE	AGARSEE.CBL
AGARSEEP	AGARSEEP.CBL	AGARTOP	AGARTOP.CBL
AGARTP	AGARTP.CBL	AGDATOP	AGDATOP.CBL
AGDCHK	AGDCHK.CBL	AGDEFPAR	AGDEFPAR.CBL
AGDEIS	AGDEIS.CBL	AGDEL	AGDEL.CBL
AGDEVR	AGDEVR.CBL	AGDEVR4	AGDEVR4.CBL
AGDEVR23	AGDEVR23.CBL	AGDEVRDT	AGDEVRDT.CBL
AGDOP	AGDOP.CBL	AGDPAR	AGDPAR.CBL
AGDTIM	AGDTIM.CBL	AGEVR	AGEVR.CBL
AGFLCHK	AGFLCHK.CBL	AGFORTEID	AGFORTEID.CBL
AGFORTSYN	AGFORTSYN.CBL	AGHMEROL	AGHMEROL.CBL
AGIN	AGIN.CBL	AGINVOP	AGINVOP.CBL
AGLINKOP	AGLINKOP.CBL	AGOPCNT	AGOPCNT.CBL
AGOPEN	AGOPEN.CBL	AGOPOL	AGOPOL.CBL
AGPAR	AGPAR.CBL	AGPCNTOP	AGPCNTOP.CBL
AGPDEL	AGPDEL.CBL	AGPEIS	AGPEIS.CBL
AGPEVR4	AGPEVR4.CBL	AGPEVR123	AGPEVR123.CBL
AGPOP	AGPOP.CBL	AGPTIM	AGPTIM.CBL
AGTCHK	AGTCHK.CBL	AGTCHK1	AGTCHK1.CBL
AGTDEL	AGTDEL.CBL	AGTEIS	AGTEIS.CBL
AGTEIS.STD	AGTEIS.STD	AGTEIS1	AGTEIS1.CBL
AGTEIS2	AGTEIS2.CBL	AGTEIS3	AGTEIS3.CBL
AGTOMEIS	AGTOMEIS.CBL	AGTOMOP	AGTOMOP.CBL
AGTOP	AGTOP.CBL	AGTPAR	AGTPAR.CBL
AGTSYGDT	AGTSYGDT.CBL	AGTYPEIS	AGTYPEIS.CBL
AGTYPOP	AGTYPOP.CBL	AGVIEW	AGVIEW.CBL
AGVIEW1	AGVIEW1.CBL	AIT-TIM	AIT-TIM.CBL
AITEXAG	AITEXAG.CBL	ALPHATEST	ALPHATEST.CBL
ALPHATEST2	ALPHATEST2.CBL	AMUPDATE	AMUPDATE.CBL
AMUPREAD	AMUPREAD.CBL	ANALMAKE	ANALMAKE.CBL
AP	AP.CBL	AP--DEL	AP--DEL.CBL
AP-AP	AP-AP.CBL	AP-DEL	AP-DEL.CBL
AP-GET	AP-GET.CBL	AP-MOVE	AP-MOVE.CBL
AP-MOVE1	AP-MOVE1.CBL	AP-UPD	AP-UPD.CBL
AP0	AP0.CBL	AP1CLEAR	AP1CLEAR.CBL
AP2HMER	AP2HMER.CBL	APADDTRS	apaddtrs.cbl
APAGLAST	APAGLAST.CBL	APAIT2EX	APAIT2EX.CBL
APALBRI	APALBRI.CBL	APALBRI9	APALBRI9.CBL
APALBRI9.STD	APALBRI9.STD	APALEIS	APALEIS.CBL
APALGCH	APALGCH.CBL	APALOP	APALOP.CBL
APAN	APAN.CBL	APAN-MTK	APAN-MTK.CBL
APANAHTL	APANAHTL.CBL	APANALAA	APANALAA.CBL
APANALN	APANALN.CBL	APANALX	APANALX.CBL

APANALY	APANALY.CBL	APANAL_N	APANAL_N.CBL
APANCHK	APANCHK.CBL	APANFL1	APANFL1.CBL
APANFL2	APANFL2.CBL	APANLGOP	APANLGOP.CBL
APANLOG	APANLOG.CBL	APANREAD	APANREAD.CBL
APANREAD1	APANREAD1.CBL	APANRPRV	APANRPRV.CBL
APANTEST	APANTEST.CBL	APANUSER	APANUSER.CBL
APAPOGR1	APAPOGR1.CBL	APAPOGR2	APAPOGR2.CBL
APARTPOP	APARTPOP.CBL	APASFAL	APASFAL.CBL
APCHANCE	APCHANCE.CBL	APCHANGE	APCHANGE.CBL
APCHK	APCHK.CBL apchk.cbl	APCLEAR	APCLEAR.CBL
APCOSTAN	APCOSTAN.CBL	APCOSTS	APCOSTS.CBL
APDEL	APDEL.CBL	APDEL1	APDEL1.CBL
APDGEN	APDGEN.CBL	APDGEN1	APDGEN1.CBL
APDIAFOR	APDIAFOR.CBL	APDIAK	APDIAK.CBL
APDIEK	APDIEK.CBL	APDOP	APDOP.CBL
APDPREVR	APDPREVR.CBL	APDPRICE	APDPRICE.CBL
APDPROP	APDPROP.CBL	APEIDFP3	APEIDFP3.CBL
APEIDFPA	APEIDFPA.CBL	APEIDMA3	APEIDMA3.CBL
APEIDMAN	APEIDMAN.CBL	APEIDMTK	APEIDMTK.CBL
APEIDSTK	APEIDSTK.CBL	APEIDSYN	APEIDSYN.CBL
APEIDTIM	APEIDTIM.CBL	APEPANEK	APEPANEK.CBL
APETIK	APETIK.CBL	APETIKCH	APETIKCH.CBL
APEVMON	APEVMON.CBL	APEVR00	APEVR00.CBL
APEVR01	APEVR01.CBL	APEVREI1	APEVREI1.CBL
APEVREI1.STD	APEVREI1.STD	APEVREI2	APEVREI2.CBL
APEVREI6	APEVREI6.CBL	APEVRGEN	APEVRGEN.CBL
APEVRICL	APEVRICL.CBL	APEVRKI2	APEVRKI2.CBL
APEVRKIN	APEVRKIN.CBL	APEVRKIN.STD	APEVRKIN.STD
APEVRMAN	APEVRMAN.CBL	APEVRSTK	APEVRSTK.CBL
APEVRTA1	APEVRTA1.CBL	APEVRTA2	APEVRTA2.CBL
APEVRTAM	APEVRTAM.CBL	APEXAHTL	APEXAHTL.CBL
APEXEKT	APEXEKT.CBL	APF1EIS	APF1EIS.CBL
APF1LINK	APF1LINK.CBL	APF1OP	APF1OP.CBL
APFEACH	APFEACH.CBL	APFEIS	APFEIS.CBL
APFILTER	APFILTER.CBL	APFIXCN	APFIXCN.CBL
APFLCHK	APFLCHK.CBL	APFLEIS	APFLEIS.CBL
APFLOP	APFLOP.CBL	APFOP	APFOP.CBL
APFPA	APFPA.CBL	APFPR	APFPR.CBL
APFPREIS	APFPREIS.CBL	APFPREN1	APFPREN1.CBL
APFPRENH	APFPRENH.CBL	APFPREVR	APFPREVR.CBL
APG1CNLT	APG1CNLT.CBL	APG1DIAF	APG1DIAF.CBL
APG1EIS1	APG1EIS1.CBL	APG1EIS2	APG1EIS2.CBL
APG1LST1	APG1LST1.CBL	APG1LST2	APG1LST2.CBL
APG1LST3	APG1LST3.CBL	APG1TAKT	APG1TAKT.CBL
APG1TO3	APG1TO3.CBL	APG2CNLT	APG2CNLT.CBL
APG2DIAF	APG2DIAF.CBL	APG2EIS1	APG2EIS1.CBL
APG2EIS2	APG2EIS2.CBL	APG2LST1	APG2LST1.CBL
APG2LST2	APG2LST2.CBL	APG2LST3	APG2LST3.CBL
APG2TAKT	APG2TAKT.CBL	APG3CNLT	APG3CNLT.CBL
APG3DIAF	APG3DIAF.CBL	APG3EIS1	APG3EIS1.CBL
APG3EIS2	APG3EIS2.CBL	APG3LST1	APG3LST1.CBL
APG3LST2	APG3LST2.CBL	APG3LST3	APG3LST3.CBL
APG3LST4	APG3LST4.CBL	APG3TAKT	APG3TAKT.CBL
APGARXH	APGARXH.CBL	APGARXH1	APGARXH1.CBL

APGARXH2	APGARXH2.CBL	APGCNLT3	APGCNLT3.CBL
APGCNLTA	APGCNLTA.CBL	APGCONV1	APGCONV1.CBL
APGDEL1	APGDEL1.CBL	APGDEL2	APGDEL2.CBL
APGDEL3	APGDEL3.CBL	APGDELET	APGDELET.CBL
APGDIAF1	APGDIAF1.CBL	APGEIS1	APGEIS1.CBL
APGEIS2	APGEIS2.CBL	APGENDEL	APGENDEL.CBL
APGENYP	APGENYP.CBL	APGKEIS	APGKEIS.CBL
APGKOPEN	APGKOPEN.CBL	APGLIST1	APGLIST1.CBL
APGLIST2	APGLIST2.CBL	APGLIST2.RST	APGLIST2.RST
APGLIST2.STD	APGLIST2.STD	APGLIST3	APGLIST3.CBL
APGLIST4	APGLIST4.CBL	APGOTHER	APGOTHER.CBL
APGSTAR1	APGSTAR1.CBL	APGSTAR2	APGSTAR2.CBL
APGSTART	APGSTART.CBL	APGSTART.STD	APGSTART.STD
APGTACNL	APGTACNL.CBL	APGTAKT	APGTAKT.CBL
APGTAKT3	APGTAKT3.CBL	APHMEROL	APHMEROL.CBL
APHOSPHT	APHOSPHT.CBL	APHOSPOP	APHOSPOP.CBL
APHOSPP2	APHOSPP2.CBL	APHOSPPL	APHOSPPL.CBL
APHOSPTR	APHOSPTR.CBL	APHOTEL	APHOTEL.CBL
APHOTELO	APHOTELO.CBL	APHPLEIS	APHPLEIS.CBL
APHPLEVR	APHPLEVR.CBL	APHSPHTL	APHSPHTL.CBL
APHTLCOS	APHTLCOS.CBL	APIN	APIN.CBL
APISOZ	APISOZ.CBL	APISOZ1	APISOZ1.CBL
APK1EIDA	APK1EIDA.CBL	APK1ISOZ	APK1ISOZ.CBL
APKARTEL	APKARTEL.CBL	APKATEL	APKATEL.CBL
APKATEL1	APKATEL1.CBL	APKDCHK	APKDCHK.CBL
APKODCHK	APKODCHK.CBL	APKODTAM	APKODTAM.CBL
APKOSTEIS	APKOSTEIS.CBL	APKOSTKER	APKOSTKER.CBL
APKSHOW	APKSHOW.CBL	APLABEL	APLABEL.CBL
APLGCH	APLGCH.CBL	APLGEIDOS	APLGEIDOS.CBL
APLGEVR	APLGEVR.CBL	APM1JOIN	APM1JOIN.CBL
APM2JOIN	APM2JOIN.CBL	APMANTAM	APMANTAM.CBL
APMCLEAR	APMCLEAR.CBL	APMEZEV1	APMEZEV1.CBL
APMEZEV2	APMEZEV2.CBL	APMEZEVR	APMEZEVR.CBL
APMEZHTL	APMEZHTL.CBL	APMEZHTT	APMEZHTT.CBL
APMEZMTM	APMEZMTM.CBL	APMHN	APMHN.CBL
APMHNST	APMHNST.CBL	APMHNSTA	APMHNSTA.CBL
APMHNSTF	APMHNSTF.CBL	APMHNYP	APMHNYP.CBL
APMODEL	APMODEL.CBL	APMODOP	APMODOP.CBL
APMODPRT	APMODPRT.CBL	APMONEIS	APMONEIS.CBL
APMONHTL	APMONHTL.CBL	APMONOP	APMONOP.CBL
APMRCLS	APMRCLS.CBL	APMRCRE	APMRCRE.CBL
APMRINS	APMRINS.CBL	APMRMET	APMRMET.CBL
APMRMOV	APMRMOV.CBL	APMRMOV0	APMRMOV0.CBL
APMRREAD	APMRREAD.CBL	APMTK000	APMTK000.CBL
APMTKAPL	APMTKAPL.CBL	APMTKCMP	APMTKCMP.CBL
APMTKEID	APMTKEID.CBL	APMTKSYN	APMTKSYN.CBL
APNCLEAR	APNCLEAR.CBL	APNOKIN	APNOKIN.CBL
APNREAD	APNREAD.CBL	APO-DESC	APO-DESC.CBL
APO-TR	APO-TR.CBL	APO1CHK	APO1CHK.CBL
APOCOMP	APOCOMP.CBL	APODEL	APODEL.CBL
APOGCHK	APOGCHK.CBL	APOGEIS2	APOGEIS2.CBL
APOGLST	APOGLST.CBL	APOGOP	APOGOP.CBL
APOGOP1	APOGOP1.CBL	APOGPR	APOGPR.CBL
APOGR-OP	APOGR-OP.CBL	APOGRDEL	APOGRDEL.CBL

APOGRMOV	APOGRMOV.CBL	APOKARTA	APOKARTA.CBL
APOMA-TR	APOMA-TR.CBL	APOMA2TR	APOMA2TR.CBL
APOMACHK	APOMACHK.CBL	APOMAFIX	APOMAFIX.CBL
APOMANAF1	APOMANAF1.CBL	APOMANEW	APOMANEW.CBL
APOMANO0	APOMANO0.CBL	APOMAREAD	APOMAREAD.CBL
APOMAST1	APOMAST1.CBL	APOMAST2	APOMAST2.CBL
APOMAST3	APOMAST3.CBL	APOMASTN	APOMASTN.CBL
APOMASTN.BKP	APOMASTN.BKP	APOMASTN.NEW	APOMASTN.NEW
APOMASTN.STD	APOMASTN.STD	APOMEIS	APOMEIS.CBL
APOP1MR	APOP1MR.CBL	APOPANAL	APOPANAL.CBL
APOPEN	APOPEN.CBL	APOPFPA	APOPFPA.CBL
APOPFPR	APOPFPR.CBL	APOPMR	APOPMR.CBL
APOPTIMCH	APOPTIMCH.CBL	APOUPD1	APOUPD1.CBL
APPAR	APPAR.CBL	APPEIS	APPEIS.CBL
APPEL	APPEL.CBL	APPEL0	APPEL0.CBL
APPEL2	APPEL2.CBL	APPEL3	APPEL3.CBL
APPFPA	APPFPA.CBL	APPFPA1	APPFPA1.CBL
APPL5A	APPL5A.CBL	APPL5B	APPL5B.CBL
APPOL3	APPOL3.CBL	APPOP	APPOP.CBL
APPRICE	APPRICE.CBL	APPROFIT	APPROFIT.CBL
APPROM	APPROM.CBL	APPROM2	APPROM2.CBL
APPROM3	APPROM3.CBL	APPROM4	APPROM4.CBL
APPROM5	APPROM5.CBL	APRDOP	APRDOP.CBL
APRECOS1	APRECOS1.CBL	APRECOST	APRECOST.CBL
APRECSY1	APRECSY1.CBL	APRECSYN	APRECSYN.CBL
APRELATE	APRELATE.CBL	APRELEIS	APRELEIS.CBL
APRELOP	APRELOP.CBL	APREPHTL	APREPHTL.CBL
APREPORT	APREPORT.CBL	APREPSTK	APREPSTK.CBL
APROL3	APROL3.CBL	APSALES	APSALES.CBL
APSCLEAR	APSCLEAR.CBL	APSCREEN	APSCREEN.CBL
APSORT	APSORT.CBL	APSTAT	APSTAT.CBL
APSTAT1	APSTAT1.CBL	APSTATUS	APSTATUS.CBL
APSTKEID	APSTKEID.CBL	APSTKMET	APSTKMET.CBL
APSTKPER	APSTKPER.CBL	APSTOCK9	APSTOCK9.CBL
APSYG	APSYG.CBL	APSYNTEV	APSYNTEV.CBL
APTAMDEL	APTAMDEL.CBL	APTAMMAN	APTAMMAN.CBL
APTAMOV	APTAMOV.CBL	APTAMPR	APTAMPR.CBL
APTAMPR.STD	APTAMPR.STD	APTAMTIM	APTAMTIM.CBL
APTCHADD	APTCHADD.CBL	APTE1OP	APTE1OP.CBL
APTEAMOP	APTEAMOP.CBL	APTEAMS	APTEAMS.CBL
APTIMCH	APTIMCH.CBL	APTIMCHF	APTIMCHF.CBL
APTIMEID	APTIMEID.CBL	APTIMMEZ	APTIMMEZ.CBL
APTIMO	APTIMO.CBL	APTIMP0	APTIMP0.CBL
APTIMTAM	APTIMTAM.CBL	APTMEIS	APTMEIS.CBL
APTMENU	APTMENU.CBL	APTMEVR	APTMEVR.CBL
APTMFILL	APTMFILL.CBL	APTMJOIN	APTMJOIN.CBL
APTMOP	APTMOP.CBL	APTMXEIS	APTMXEIS.CBL
APTOTAL	APTOTAL.CBL	APTPR2MZ	APTPR2MZ.CBL
APTREIS	APTREIS.CBL	APTROP	APTROP.CBL
APTRWHOP	APTRWHOP.CBL	APTURN	APTURN.CBL
APUPDAT3	APUPDAT3.CBL	APUPDATE	APUPDATE.CBL
APUPDMV	APUPDMV.CBL	APUPDSAY	APUPDSAY.CBL
APWEEK	APWEEK.CBL	APWEEKST	APWEEKST.CBL
APYPOL	APYPOL.CBL	APYPOL0	APYPOL0.CBL

APYPOLB	APYPOLB.CBL	APZERO	APZERO.CBL
AP_CHK	ap_chk.cbl	ASC	ASC.CBL
ASC-AP	ASC-AP.CBL	ASC-AP0	ASC-AP0.CBL
ASC-AP1	ASC-AP1.CBL	ASC-AP2	ASC-AP2.CBL
ASC-APAN	ASC-APAN.CBL	ASC-CODE	ASC-CODE.CBL
ASC-LGAN	ASC-LGAN.CBL	ASC-PEL	ASC-PEL.CBL
ASC-PEL3	ASC-PEL3.CBL	ASC-PEL4	ASC-PEL4.CBL
ASC-PROM	ASC-PROM.CBL	ASC-PROM1	ASC-PROM1.CBL
ASC-PROM3	ASC-PROM3.CBL	ASC-TEAM	ASC-TEAM.CBL
ASC-TEAM2	ASC-TEAM2.CBL	ASUPDATE	ASUPDATE.CBL
BACKUP	BACKUP.CBL	BAR	BAR.CBL
BOXASK	boxask.cbl	CALC	calc.cbl
CALC.STD	calc.std	CALCTEST	calctest.cbl
CALL	CALL.CBL	CALL-EIS	CALL-EIS.CBL
CALL1	CALL1.CBL	CALLAPREL	CALLAPREL.CBL
CANCPRT	CANCPRT.CBL	COMPANY	COMPANY.CBL
COMPANY.ALX	COMPANY.ALX	COMPMEN	COMPMEN.CBL
DAY	DAY.CBL	DB	DB.CBL
DBEXEC	DBEXEC.CBL	DBMARK	DBMARK.CBL
DBPRNT	DBPRNT.CBL	DEDEIS	DEDEIS.CBL
EKTEIS	EKTEIS.CBL	EKTEPIL	EKTEPIL.CBL
EMP-AGO	EMP-AGO.CBL	EMP-APO	EMP-APO.CBL
EMP-PEL	EMP-PEL.CBL	EMP-PRO	EMP-PRO.CBL
EMP-TIM	EMP-TIM.CBL	ENTYPO	entypo.cbl
ENTYPO2	entypo2.cbl	EP2PROM	EP2PROM.CBL
EPDATE	EPDATE.CBL	EPDATE1	EPDATE1.CBL
EPDEL	EPDEL.CBL	EPDEL1	EPDEL1.CBL
EPDEL3	EPDEL3.CBL	EPDEL4	EPDEL4.CBL
EPESYN	EPESYN.CBL	EPIT1OPMR	EPIT1OPMR.CBL
EPITDEL2	EPITDEL2.CBL	EPITEIS	EPITEIS.CBL
EPITEIS1	EPITEIS1.CBL	EPITKAT	EPITKAT.CBL
EPITKAT1	EPITKAT1.CBL	EPITKEY	EPITKEY.CBL
EPITMEN	EPITMEN.CBL	EPITMEN1	EPITMEN1.CBL
EPITOPMR	EPITOPMR.CBL	EPITOPTRAN	EPITOPTRAN.CBL
EPITPEL	EPITPEL.CBL	EPITPROM	EPITPROM.CBL
EPITS	EPITS.CBL	EPITS1	EPITS1.CBL
EPITSYN	EPITSYN.CBL	EPMET	EPMET.CBL
EPMET1	EPMET1.CBL	EPSYN	EPSYN.CBL
EPSYN1	EPSYN1.CBL	EPTBL3	EPTBL3.CBL
EPTBL4	EPTBL4.CBL	EPTRAP	EPTRAP.CBL
EPTRAP1	EPTRAP1.CBL	EPWEIS	EPWEIS.CBL
EPWEIS1	EPWEIS1.CBL	EXEC	EXEC.CBL
FEANEW	FEANEW.CBL	FHANDLE	fhandle.cbl
FINIT	FINIT.CBL	FINIT1	FINIT1.CBL
FIXAPOLG	FIXAPOLG.CBL	FLAG0	FLAG0.CBL
FLAG1	FLAG1.CBL	FLAG2	FLAG2.CBL
GNFLCHK	GNFLCHK.CBL	GP	GP.CBL
GR1CNT	GR1CNT.CBL	GRADATE	GRADATE.CBL
GRADATE1	GRADATE1.CBL	GRADEL	GRADEL.CBL
GRADEL1	GRADEL1.CBL	GRAKAT1	GRAKAT1.CBL
GRAM1OPEN	GRAM1OPEN.CBL	GRAM1OPMR	GRAM1OPMR.CBL
GRAMDATE	GRAMDATE.CBL	GRAMDEL	GRAMDEL.CBL
GRAMEIS	GRAMEIS.CBL	GRAMEIS1	GRAMEIS1.CBL
GRAMEN	GRAMEN.CBL	GRAMEN1	GRAMEN1.CBL

GRAMET	GRAMET.CBL	GRAMET1	GRAMET1.CBL
GRAMKAT	GRAMKAT.CBL	GRAMOPEN	GRAMOPEN.CBL
GRAMOPMR	GRAMOPMR.CBL	GRAMPEL	GRAMPEL.CBL
GRAMPROM	GRAMPROM.CBL	GRAMTRAP	GRAMTRAP.CBL
GRATRAP	GRATRAP.CBL	GRATRAP1	GRATRAP1.CBL
GRCMENU	GRCMENU.CBL	GRCNT	GRCNT.CBL
GRCNT1	GRCNT1.CBL	GRDEL1	GRDEL1.CBL
GRDEL2	GRDEL2.CBL	GREPIN	GREPIN.CBL
GREPOPEN	GREPOPEN.CBL	GRIN	GRIN.CBL
GRTBEX2	GRTBEX2.CBL	GRTBL1	GRTBL1.CBL
GRTBL2	GRTBL2.CBL	ICL-ALL	ICL-ALL.CBL
ICL-CHK	ICL-CHK.CBL	ICL-CHR	ICL-CHR.CBL
ICL-IN	ICL-IN.CBL	ICL-OUT	ICL-OUT.CBL
ICLREAD	ICLREAD.CBL	ICLSALES	ICLSALES.CBL
K1NEW	K1NEW.CBL	K2NEW	K2NEW.CBL
KODACC	KODACC.CBL	KODE	KODE.CBL
KODEOP	KODEOP.CBL	LG	lg.cbl
LG-EKT	lg-ekt.cbl	LG-PL	LG-PL.CBL
LGAG	lgag.cbl	LGDEL	LGDEL.CBL
LGEMPUPD	lgempupd.cbl	LGPELUPD	lgpelupd.cbl
LGPROUPD	lgproupd.cbl	LINKMENU	LINKMENU.CBL
MAIN	MAIN.CBL	MAIN.ALX	MAIN.ALX
MAINDOS	MAINDOS.CBL	MAK+APAN	MAK+APAN.CBL
MAK+POLHS	MAK+POLHS.CBL	MAK-AP	MAK-AP.CBL mak-ap.cbl
MAK-APN	MAK-APN.CBL	MAK-APY	mak-apy.cbl
MAK-PEL	MAK-PEL.CBL	MAK-PR	MAK-PR.CBL
MAK999KE	MAK999KE.CBL	MAK999MA	MAK999MA.CBL
MAKAG	MAKAG.CBL makag.cbl	MAKAGD	MAKAGD.CBL
MAKAGD1	MAKAGD1.CBL	MAKAGDF	MAKAGDF.CBL
MAKAGP	MAKAGP.CBL MAKagp.CBL	MAKAGT	MAKagt.CBL
MAKAGT0-1	MAKAGT0-1.CBL	MAKANAL2	MAKANAL2.CBL
MAKAP	MAKAP.CBL	MAKAP-CS	MAKAP-CS.CBL
MAKAP0-1	MAKAP0-1.CBL	MAKAP0-11	MAKAP0-11.CBL
MAKAP1	MAKAP1.CBL	MAKAP1-2	MAKAP1-2.CBL
MAKAP2	MAKAP2.CBL	MAKAPAN	MAKAPAN.CBL
MAKAPAN0	MAKAPAN0.CBL	MAKAPAN1	MAKAPAN1.CBL
MAKAPAN2	MAKAPAN2.CBL	MAKAPAN3	MAKAPAN3.CBL
MAKAPAN4	MAKAPAN4.CBL	MAKAPANAL	MAKAPANAL.CBL
MAKAPD1	MAKAPD1.CBL	MAKAPD2	MAKAPD2.CBL
MAKAPDF	MAKAPDF.CBL	MAKAPDF.STD	MAKAPDF.STD
MAKAPF	MAKAPF.CBL	MAKAPFE	MAKAPFE.CBL
MAKAPFL	MAKAPFL.CBL	MAKAPGK	MAKAPGK.CBL
MAKAPGO	MAKAPGO.CBL	MAKAPLG	MAKAPLG.CBL
MAKAPMON	MAKAPMON.CBL	MAKAPMR	MAKAPMR.CBL makapmr.cbl
MAKAPMR1	MAKAPMR1.CBL	MAKAPO	makapo.cbl
MAKAPO1	MAKAPO1.CBL makapo1.cbl	MAKAPOGR	MAKAPOGR.CBL
MAKAPOM1	MAKAPOM1.CBL	MAKAPOMA	MAKAPOMA.CBL
MAKAPPO	MAKAPPO.CBL	MAKAPPOL	MAKAPPOL.CBL
MAKAPPOL1	MAKAPPOL1.CBL	MAKAPPOL2	MAKAPPOL2.CBL
MAKAPPR	MAKAPPR.CBL	MAKAPRD	MAKAPRD.CBL
MAKAPREL	MAKAPREL.CBL	MAKAPSEQ	makapseq.cbl
MAKAPSY	MAKAPSY.CBL	MAKAPSYN	MAKAPSYN.CBL

MAKAPT0-1	MAKAPT0-1.CBL	MAKAPT0-11	MAKAPT0-11.CBL
MAKAPTCH	MAKAPTCH.CBL	MAKAPTE	MAKAPTE.CBL
MAKAPTEAM	MAKAPTEAM.CBL	MAKAPTIMCH	MAKAPTIMCH.CBL
MAKAPTY2	MAKAPTY2.CBL	MAKAPTYP	MAKAPTYP.CBL
MAKAPYP	MAKAPYP.CBL	MAKAR	MAKAR.CBL
MAKARADD	MAKARADD.CBL	MAKARDEL	MAKARDEL.CBL
MAKARG	MAKARG.CBL	MAKART	MAKART.CBL MAKart.CBL
MAKARTHRO	MAKARTHRO.CBL	MAKCNT	MAKCNT.CBL
MAKCOUNT	MAKCOUNT.CBL	MAKDISC	MAKDISC.CBL
MAKDPCNT	MAKDPCNT.CBL	MAKDPD	MAKDPD.CBL
MAKEPIT	MAKEPIT.CBL	MAKEPIT1	MAKEPIT1.CBL
MAKGRAM	MAKGRAM.CBL	MAKGRAM1	MAKGRAM1.CBL
MAKGRX	MAKGRX.CBL	MAKINV	MAKINV.CBL
MAKLINK	MAKLINK.CBL	MAKMAIN	MAKMAIN.CBL
MAKMENU	MAKMENU.CBL	MAKOIKO	MAKOIKO.CBL
MAKOYBA	MAKOYBA.CBL	MAKPAR0-1	MAKPAR0-1.CBL
MAKPEL	MAKPEL.CBL MAKpel.CBL	MAKPL	MAKPL.CBL
MAKPL1	MAKPL1.CBL	MAKPL2	MAKPL2.CBL
MAKPLAN	MAKPLAN.CBL	MAKPLAN1	MAKPLAN1.CBL
MAKPLAN3	MAKPLAN3.CBL	MAKPLAN4	MAKPLAN4.CBL
MAKPLAN10	MAKPLAN10.CBL	MAKPLC-D	MAKPLC-D.CBL
MAKPLEVR	MAKPLEVR.CBL	MAKPLHELP	MAKPLHELP.CBL
MAKPLMR	MAKPLMR.CBL	MAKPLNPO	MAKPLNPO.CBL
MAKPLPR	MAKPLPR.CBL	MAKPOL	MAKPOL.CBL MAKpol.CBL makpol.cbl
MAKPOL1	MAKPOL1.CBL	MAKPOLD	MAKPOLD.CBL
MAKPOLD1	MAKPOLD1.CBL	MAKPOLD2	MAKPOLD2.CBL
MAKPOLDF	MAKPOLDF.CBL	MAKPOLHS	MAKPOLHS.CBL
MAKPR	MAKPR.CBL makpr.cbl	MAKPR-CS	MAKPR-CS.CBL
MAKPR8	MAKPR8.CBL	MAKPRALT	MAKPRALT.CBL
MAKPRAN	MAKPRAN.CBL	MAKPRAN3	MAKPRAN3.CBL
MAKPRMR	MAKPRMR.CBL	MAKPRO	MAKpro.CBL
MAKPROM	MAKPROM.CBL	MAKSYN1	MAKSYN1.CBL
MAKSYN2	MAKSYN2.CBL	MAKSYN3	MAKSYN3.CBL
MAKSYN5	MAKSYN5.CBL	MAKSYNT	MAKSYNT.CBL
MAKTIM	MAKtim.CBL	MAKTIM1	MAKTIM1.CBL
MAKTYPE	MAKTYPE.CBL	MAKUNIT2	MAKUNIT2.CBL
MAK_AGD	MAK_AGD.CBL mak_agd.cbl	MAK_AGDAT	MAK_AGDAT.CBL
MAK_AGT	mak_agt.cbl	MAK_AGT1	mak_agt1.cbl
MAK_APAN	MAK_APAN.CBL mak_apan.cbl	MAK_PLAN	MAK_PLAN.CBL
MAK_PLN1	mak_pln1.cbl	MAK_PLPOL	mak_plpol.cbl
MAK_POL	mak_pol.cbl	MAK_POL1	mak_pol1.cbl
MAK_POL2	mak_pol2.cbl	MAK_PRAN	MAK_PRAN.CBL mak_pran.cbl
MAK_PRN1	mak_prn1.cbl	MARK	MARK.CBL
MENCPL	MENCPL.CBL	MENEXT	MENEXT.CBL
MENFYI	MENFYI.CBL	MENRTS	MENRTS.CBL
MENRTS.LNX	MENRTS.LNX	MENRTS.OLD	MENRTS.OLD
MERSRT	MERSRT.CBL	MINIE-APO	MINIE-APO.CBL
MINIE-PEL	MINIE-PEL.CBL	MSG	msg.cbl
MSMAKAN	MSMAKAN.CBL	MSSEQ	MSSEQ.CBL
MYF	MYF.CBL	NEAXR	NEAXR.CBL

NEAXR.STD	NEAXR.STD	NEAXRDOS	NEAXRDOS.CBL
NEWANAL	NEWANAL.CBL	NEWPRG	NEWPRG.CBL
NOR-MENU	nor-menu.cbl	NUMDIS	numdis.cbl
NUMTEXT	numtext.cbl	OLDNUM	oldnum.cbl
OM-OM	OM-OM.CBL	ORDAPTM1	ORDAPTM1.CBL
ORDDATE2	ORDDATE2.CBL	ORDEID	ORDEID.CBL
ORDEIDDT	ORDEIDDT.CBL	ORDEIS	ORDEIS.CBL
ORDOP	ORDOP.CBL	ORDSCAN	ORDSCAN.CBL
PARACHK	PARACHK.CBL	PARAGOP	PARAGOP.CBL
PARDATE2	PARDATE2.CBL	PAREIS1	PAREIS1.CBL
PARM	PARM.CBL	PAROP	PAROP.CBL
PAROPCNT	PAROPCNT.CBL	PARPEL3	PARPEL3.CBL
PARPEL3.MARK	PARPEL3.MARK	PEL2HPL	PEL2HPL.CBL
PELDEL	PELDEL.CBL	PELDISC	PELDISC.CBL
PELFEIS	PELFEIS.CBL	PELFOP	PELFOP.CBL
PELIDX	PELIDX.CBL	PELINK	PELINK.CBL
PELNEW	PELNEW.CBL	PELOPEN	PELOPEN.CBL
PELPROD	PELPROD.CBL	PELSEQ	PELSEQ.CBL
PELTEAM	PELTEAM.CBL	PELYPEVR	PELYPEVR.CBL
PELYPOM	PELYPOM.CBL	PELYPOP	PELYPOP.CBL
PFORTEID	PFORTEID.CBL	PFORTEID2	PFORTEID2.CBL
PFORTSYN	PFORTSYN.CBL	PKMENU	pkmenu.cbl
PL-AFM	PL-AFM.CBL	PL-MOVE	PL-MOVE.CBL
PL-PL	PL-PL.CBL	PLAIM1	PLAIM1.CBL
PLAIM2	PLAIM2.CBL	PLANAL	PLANAL.CBL
PLANAL1	PLANAL1.CBL	PLANAL2	PLANAL2.CBL
PLANAL3	PLANAL3.CBL	PLANCHK	PLANCHK.CBL
PLANREAD	PLANREAD.CBL	PLANSRT	PLANSRT.CBL
PLAPOGR	PLAPOGR.CBL	PLCHK	PLCHK.CBL plchk.cbl
PLCHKAFM	PLCHKAFM.CBL	PLCOMMEN	PLCOMMEN.CBL
PLCRDB	PLCRDB.CBL	PLEKPT	PLEKPT.CBL
PLEPYPOL	PLEPYPOL.CBL	PLETIK	PLETIK.CBL
PLEVRET	PLEVRET.CBL	PLEVRET1	PLEVRET1.CBL
PLEVRET2	PLEVRET2.CBL	PLFAIM	PLFAIM.CBL
PLFAIMEV	PLFAIMEV.CBL	PLFAIML	PLFAIML.CBL
PLFILTER	PLFILTER.CBL	PLFLCHK	PLFLCHK.CBL
PLHSMHN	PLHSMHN.CBL	PLHSMHNP	PLHSMHNP.CBL
PLIN	PLIN.CBL	PLISOZ	PLISOZ.CBL
PLKATEL	PLKATEL.CBL	PLLOGI	PLLOGI.CBL
PLMHN	PLMHN.CBL	PLMHNP	PLMHNP.CBL
PLMHNPE	PLMHNPE.CBL	PLMHNPEP	PLMHNPEP.CBL
PLMHNSP	PLMHNSP.CBL	PLMHNSPP	PLMHNSPP.CBL
PLMHNSYN	PLMHNSYN.CBL	PLMREIS	PLMREIS.CBL
PLMYF	PLMYF.CBL	PLNMDT	PLNMDT.CBL
PLNME	PLNME.CBL	PLNMOP	PLNMOP.CBL
PLNS	PLNS.CBL	PLOPANAL	PLOPANAL.CBL
PLOPMR	PLOPMR.CBL	PLPELISOZ	PLPELISOZ.CBL
PLPEREVR	PLPEREVR.CBL	PLPERISOZ	PLPERISOZ.CBL
PLPERSYG	PLPERSYG.CBL	PLPERYPOL	PLPERYPOL.CBL
PLPOLREP	PLPOLREP.CBL	PLPREAD	PLPREAD.CBL
PLPRYP	PLPRYP.CBL	PLRANGE	PLRANGE.CBL
PLREAD	PLREAD.CBL	PLSALE1	PLSALE1.CBL
PLSCREEN	PLSCREEN.CBL	PLSTAT	PLSTAT.CBL
PLSYG	PLSYG.CBL	PLSYGALFA	PLSYGALFA.CBL

PLTREIS	PLTREIS.CBL	PLYPAGE	PLYPAGE.CBL
PLYPFAST	PLYPFAST.CBL	PLYPOL	PLYPOL.CBL
PLYPSEND	PLYPSEND.CBL	PLZERO	PLZERO.CBL
PL_CHK	pl_chk.cbl	POLREAD	POLREAD.CBL
PR-AFM	PR-AFM.CBL	PR-MOVE	PR-MOVE.CBL
PR-PR	PR-PR.CBL	PRAFMAK	PRAFMAK.CBL
PRANAL	PRANAL.CBL	PRANCHK	PRANCHK.CBL
PRANREAD	PRANREAD.CBL	PRAPOGR	PRAPOGR.CBL
PRCHK	PRCHK.CBL prchk.cbl	PRCHKAFM	PRCHKAFM.CBL
PRDEL	PRDEL.CBL	PREVRET	PREVRET.CBL
PREVRET1	PREVRET1.CBL	PREVRET6	PREVRET6.CBL
PRFEIS	PRFEIS.CBL	PRFLCHK	PRFLCHK.CBL
PRFOP	PRFOP.CBL	PRG-TIM	PRG-TIM.CBL
PRIN	PRIN.CBL	PRISOZ	PRISOZ.CBL
PRKATEL	PRKATEL.CBL	PRLOGI	PRLOGI.CBL
PRMERGE	PRMERGE.CBL	PRMHN	PRMHN.CBL
PRMHNSYN	PRMHNSYN.CBL	PRMREIS	PRMREIS.CBL
PRMYF	PRMYF.CBL	PROD-ASC	PROD-ASC.CBL
PROD-ASC.STD	PROD-ASC.STD	PROD-ASC1	PROD-ASC1.CBL
PROFF-OP	PROFF-OP.CBL	PROFFCHK	PROFFCHK.CBL
PROFFERS	PROFFERS.CBL	PROFFEV1	PROFFEV1.CBL
PROFFEV2	PROFFEV2.CBL	PROMX2PR	PROMX2PR.CBL
PROPANAL	PROPANAL.CBL	PROPEN	PROPEN.CBL
PROPMR	PROPMR.CBL	PRORD-OP	PRORD-OP.CBL
PRORDERS	PRORDERS.CBL	PRPEREVR	PRPEREVR.CBL
PRPERISOZ	PRPERISOZ.CBL	PRPERSYG	PRPERSYG.CBL
PRPERYPOL	PRPERYPOL.CBL	PRPLYP	PRPLYP.CBL
PRRANGE	PRRANGE.CBL	PRSCREEN	PRSCREEN.CBL
PRSTAT	PRSTAT.CBL	PRSYG	PRSYG.CBL
PRSYGALFA	PRSYGALFA.CBL	PRTEIS	PRTEIS.CBL
PRTPLYP	PRTPLYP.CBL	PRTREIS	PRTREIS.CBL
PRTSEL	PRTSEL.CBL	PRUPDATE	PRUPDATE.CBL
PRUPREAD	PRUPREAD.CBL	PRYPAGE	PRYPAGE.CBL
PRYPFAST	PRYPFAST.CBL	PRYPOL	PRYPOL.CBL
PRZERO	PRZERO.CBL	PR_CHK	pr_chk.cbl
PUZZLE	puzzle.cbl	RELPEL	RELPEL.CBL
RELPRO	RELPRO.CBL	REPLY	reply.cbl
RESTLIST	RESTLIST.CBL	RESTORE	RESTORE.CBL
RHANDLE	rhandle.cbl	RHANDLE.OLD	rhandle.old
SALES-ASC	SALES-ASC.CBL	SALES-ASC.STD	SALES-ASC.STD
SALES-ASC1	SALES-ASC1.CBL	SEQ-OTH	SEQ-OTH.CBL
SEQREAD	SEQREAD.CBL	SKT	SKT.CBL
STATAPS	STATAPS.CBL	STATBAR	STATBAR.CBL
STATBAR1	STATBAR1.CBL	STATBAR2	STATBAR2.CBL
STATBAR3	STATBAR3.CBL	STATPRT	STATPRT.CBL
STATPRT1	STATPRT1.CBL	STATPRT2	STATPRT2.CBL
STATPRT3	STATPRT3.CBL	STATPSAL	STATPSAL.CBL
STATPSAP	STATPSAP.CBL	STATPSAS	STATPSAS.CBL
STATPSSP	STATPSSP.CBL	STATSALE	STATSALE.CBL
STELIOS	STELIOS.CBL	STELIOS1	STELIOS1.CBL
STR-FCTS	str-fcts.cbl	SYNCHK	SYNCHK.CBL
SYNT	SYNT.CBL	SYNT00	SYNT00.CBL
SYNTOP	SYNTOP.CBL	SYNTPR	SYNTPR.CBL
SYSCALL	SYSCALL.CBL	SYSDESC	SYSDESC.CBL

SYSENV	SYSENV.CBL	TAMAPTM	TAMAPTM.CBL
TAPOMEIS	TAPOMEIS.CBL	TAPOMOP	TAPOMOP.CBL
TAPQNTY	TAPQNTY.CBL	TAPQNTYR	TAPQNTYR.CBL
TARTHROSEE	TARTHROSEE.CBL	TARUNL	TARUNL.CBL
TCNTGEN	TCNTGEN.CBL	TCOMEIS	TCOMEIS.CBL
TCOMOP	TCOMOP.CBL	TDEFPAR	TDEFPAR.CBL
TDMET	TDMET.CBL	TDPEKT	TDPEKT.CBL
TDPEKT1	TDPEKT1.CBL	TDPEKT2	TDPEKT2.CBL
TDPINF	TDPINF.CBL	TEST	TEST.CBL
TESTPRG	TESTPRG.CBL	TESTPRG.CBL,V	TESTPRG.CBL,v
TFORTEID	TFORTEID.CBL	TFORTEID2	TFORTEID2.CBL
TGAKYR	TGAKYR.CBL	TGEIS	TGEIS.CBL
TGEKD	TGEKD.CBL	TGEVR	TGEVR.CBL
TGEVR1	TGEVR1.CBL	TGGEN1	TGGEN1.CBL
TGGENRP1	TGGENRP1.CBL	TGGENRP2	TGGENRP2.CBL
TGMET	TGMET.CBL	TGSEE	TGSEE.CBL
THELPOP	THELPOP.CBL	TIMAGHLP	TIMAGHLP.CBL
TIMAKYR	TIMAKYR.CBL	TIMCHK	TIMCHK.CBL
TIMCNT	TIMCNT.CBL	TIMDEL	TIMDEL.CBL
TIMEBAR	TIMEBAR.CBL	TIMEIS	TIMEIS.CBL
TIMEKD	TIMEKD.CBL	TIMEVR	TIMEVR.CBL
TIMEVR1	TIMEVR1.CBL	TIMGEN	TIMGEN.CBL
TIMGEN1	TIMGEN1.CBL	TIMIN	TIMIN.CBL
TIMINF	TIMINF.CBL	TIMOPEN	TIMOPEN.CBL
TINVLIST	TINVLIST.CBL	TINVOP	TINVOP.CBL
TIPAR	TIPAR.CBL	TITLE	title.cbl
TITLEDEN	titleden.cbl	TITLEDET	titledet.cbl
TITLEKAR	titlekar.cbl	TITLENT	titlent.cbl
TITLEPKA	titlepka.cbl	TITLETER	titleter.cbl
TITLOP	TITLOP.CBL	TLINKCP	TLINKCP.CBL
TLINKOP	TLINKOP.CBL	TMETEIS	TMETEIS.CBL
TMETOP	TMETOP.CBL	TMFLCHK	TMFLCHK.CBL
TOMEIS	TOMEIS.CBL	TOMOP	TOMOP.CBL
TOPARTHR	TOPARTHR.CBL	TOPCNT	TOPCNT.CBL
TOPPOLHS	TOPPOLHS.CBL	TPLOMEIS	TPLOMEIS.CBL
TPLOMOP	TPLOMOP.CBL	TPOL2HMER	TPOL2HMER.CBL
TPOLDOP	TPOLDOP.CBL	TPOLHMEROL	TPOLHMEROL.CBL
TPOLSEE	TPOLSEE.CBL tpolsee.cbl	TPROMEIS	TPROMEIS.CBL
TPROMOP	TPROMOP.CBL	TRAPCMENU	TRAPCMENU.CBL
TRAPEIS	TRAPEIS.CBL	TRAPEVR	TRAPEVR.CBL
TRAPOP	TRAPOP.CBL	TSTEIS	TSTEIS.CBL
TSTEIS1	TSTEIS1.CBL	TTMETEIS	TTMETEIS.CBL
TTMETOP	TTMETOP.CBL	TVIEW	TVIEW.CBL
TVIEW1	TVIEW1.CBL	TYPOMEIS	TYPOMEIS.CBL
TYPOMOP	TYPOMOP.CBL	UPDTRS	UPDTRS.CBL
V3-AGO	V3-AGO.CBL	V3-EPGR	V3-EPGR.CBL
V3-EPGR1	V3-EPGR1.CBL	V3-PEL	V3-PEL.CBL
V3-PRO	V3-PRO.CBL	V3-TIM	V3-TIM.CBL
V3-XPERT	V3-XPERT.CBL	V5MAKUPD	V5MAKUPD.CBL
WINSEL	WINSEL.CBL	WINSEL1	WINSEL1.CBL
WINSEL2	WINSEL2.CBL	XP0-1	XP0-1.CBL
XP1-1	XP1-1.CBL	YEARMEN	YEARMEN.CBL
YP-YP	YP-YP.CBL	YPOK-APO	YPOK-APO.CBL
YPOK-PEL	YPOK-PEL.CBL	YPOK-PRO	YPOK-PRO.CBL

YPOK-TIM	YPOK-TIM.CBL	YPOMKNEW	YPOMKNEW.CBL
YPOMNEW	YPOMNEW.CBL	_ILO1CHK	_ILO1CHK.CBL

Appendix 15

This appendix presents the complete list of the source code files that contain datafile definitions. For each source file all the datafiles, which are defined in its source code, are listed in columns 2 & 3. Column 2 presents the names of the datafiles for which the source file contains a record description. Column 3 presents the names of the datafiles for which the source file contains a structure definition.

Source File	**Datafiles Defined Record Description**	**Datafiles Defined Structure Definition**
AGDREC.CBL	AGD-FILE	
AGFORTEID.CBL	PFORT-FILE PRINT-FILE	PFORT-FILE PRINT-FILE
AGFORTSYN.CBL	PFORT-FILE PRINT-FILE	PFORT-FILE PRINT-FILE
AGO.SEL		AGD-FILE AGP-FILE AGPCNT-FILE AGT-FILE COUNT-FILE INV-FILE LGLINK-FILE POLHS-FILE TDPR-FILE TDPRC-FILE THELP-FILE TMET-FILE TOM-FILE TYPOM-FILE
AGPCNTREC.CBL	AGPCNT-FILE	
AGPREC.CBL	AGP-FILE	
AGTREC.CBL	AGT-FILE	
AGVIEW1.CBL	WARTHRO-FILE	WARTHRO-FILE
AITEXAG.CBL	WAPTM-FILE	WAPTM-FILE
AITEXALL.CBL	WAPTM-FILE	WAPTM-FILE
AMUPD.SEL		AMUPD-FILE
AMUPD.REC	AMUPD-FILE	
AP_CHK.CBL		WAPANAL-FILE WAPO-FILE WK1-FILE WK2-FILE WYPOM-FILE WYPOMK-FILE
AP2HMER.CBL	HELP-FILE	HELP-FILE
apaddtrs.cbl	APUPD-FILE	APUPD-FILE
APAIT2EX.CBL	WAPTM-FILE	WAPTM-FILE
APALBRI.SEL		APALBRI-FILE
APALBRI.REC	APALBRI-FILE	
APAL.REC	APAL-FILE	
APANAL.SEL		APANAL-FILE
apanalho.rec	APANAL-FILE	
APANAL.REC	APANAL-FILE	
APANLOG.REC	APANLOG-FILE	
APANREC.CBL	APANAL-FILE	
AP-AP.CBL	APO-FILE WAPO-FILE	APO-FILE WAPO-FILE
APDGEN.CBL	WAPTM-FILE	WAPTM-FILE
APDPR.REC	APDPR-FILE	

APDPRREC.CBL	APDPR-FILE	
APD.REC	APD-FILE	
APDREC.CBL	APD-FILE	
APF1L.REC	APF1LNK-FILE	
APF1LREC.CBL	APF1LNK-FILE	
APF1.REC	APF1-FILE	
APF1REC.CBL	APF1-FILE	
APFEREC.CBL	APF-FILE	
APFL.REC	APFL-FILE	
APFLREC.CBL	APFL-FILE	
APFPA.REC	APFPA-FILE	
APFPR.SEL		APFPR-FILE
APFPAREC.CBL	APFPA-FILE	
APFPR.REC	APFPR-FILE	
APF.REC	APF-FILE	
APFREC.CBL	APF-FILE	
AP-GET.CBL	APO-FILE WAPO-FILE	APO-FILE WAPO-FILE
APGK.REC	APGK-FILE	
APGK.SEL		APGK-FILE
APHMEROL.CBL	HELP-FILE	HELP-FILE
APKATEL1.CBL	HELP-FILE	HELP-FILE
APKATEL.CBL	HELP-FILE	HELP-FILE
APKO2.REC	SYNT-KOS-FILE	
APKO2REC.CBL	SYNT-KOS-FILE	
APKODE.REC	KODE-FILE	
APKODEREC.CBL	KODE-FILE	
APKOS.REC	KOS-FILE	
APKOSREC.CBL	KOS-FILE	
APM2JOIN.CBL	WAPO1-FILE	WAPO1-FILE
APMOD.REC	APMOD-FILE	
APMODREC.CBL	APMOD-FILE	
APMON.REC	MON-FILE	
APMONREC.CBL	MON-FILE	
APNEW.CBL	WAPO-FILE	WAPO-FILE
APO.SEL		APAL-FILE APANLOG-FILE APD-FILE APDPR-FILE APF-FILE APF1-FILE APF1LNK-FILE APFL-FILE APFPA-FILE APMOD-FILE APP-FILE APRD-FILE APTEAM-FILE APTEAM1-FILE APTMX-FILE APTR-FILE KODE-FILE KOS-FILE MON-FILE SYNT-KOS-FILE TAPOM-FILE

APO1.SEL		APD-FILE APDPR-FILE APF-FILE APF1-FILE APF1LNK-FILE APFL-FILE APFPA-FILE APMOD-FILE APP-FILE APRD-FILE APTEAM-FILE APTEAM1-FILE APTMX-FILE APTR-FILE KODE-FILE KOS-FILE MON-FILE SYNT-KOS-FILE TAPOM-FILE
APOGRNEW.REC	APOGR-FILE	
APOGR.RE0	APOGR-FILE	
APOGR.REC	APOGR-FILE	
APOHO.SEL		APOGR-FILE
APOHO.SEL		APANAL-FILE APD-FILE APDPR-FILE APF-FILE APF1-FILE APF1LNK-FILE APFL-FILE APFPA-FILE APMOD-FILE APO-FILE APO1-FILE APOGR-FILE APP-FILE APRD-FILE APTEAM-FILE APTEAM1-FILE APTIMCH-FILE APTM-FILE APTMX-FILE APTR-FILE KODE-FILE KOS-FILE MON-FILE SYNT-FILE SYNT-KOS-FILE TAPOM-FILE
APOM1.REC	APO1-FILE	
APOM1.SEL		APO1-FILE
APOM1REC.CBL	APO1-FILE	
APOMANEW.CBL	APOMA-FILE WAPOMA-FILE	APOMA-FILE WAPOMA-FILE
APOMA.REC	APOMA-FILE	
APOMA.SEL		APOMA-FILE
APOMAST.REC	APO-FILE	
APOMAST.SEL		APO-FILE
APOMAST1.SEL		APO-FILE
APOMT.REC	TAPOM-FILE	
APOREC.CBL	APO-FILE	
APOPMR.CBL		APO-FILE APOMA-FILE APO1-FILE
APPOL3.CBL	HLP-FILE	HLP-FILE
APP.REC	APP-FILE	
APPREC.CBL	APP-FILE	

APRD.REC	APRD-FILE	
APRDREC.CBL	APRD-FILE	
APREL.REC	APREL-FILE	
APREL.SEL		APREL-FILE
APROL3.CBL	HLP-FILE	HLP-FILE
APSTAT1.CBL	HLP-FILE	HLP-FILE
APSTAT.CBL	HLP-FILE	HLP-FILE
APSTATUS.CBL	APSTAT-FILE	APSTAT-FILE
APSYN.REC	SYNT-FILE	
APSYN.SEL		SYNT-FILE
APSYNREC.CBL	SYNT-FILE	
APTE1.REC	APTEAM1-FILE	
APTE1REC.CBL	APTEAM1-FILE	
APTEAM.REC	APTEAM-FILE	
APTEAMREC.CBL	APTEAM-FILE	
APTIMCH.REC	APTIMCH-FILE	
APTIMCH.SEL		APTIMCH-FILE
APTM.SEL		APTM-FILE
APTIMCHREC.CBL	APTIMCH-FILE	
APTMJOIN.CBL	WAPTM-FILE	WAPTM-FILE
APTM.REC	APTM-FILE	
APTMX.REC	APTMX-FILE	
APTMXREC.CBL	APTMX-FILE	
APTRANS.CBL	WAPANAL-FILE WAPO-FILE WK1-FILE WK2-FILE WYPOM-FILE WYPOMK-FILE	WAPANAL-FILE WAPO-FILE WK1-FILE WK2-FILE WYPOM-FILE WYPOMK-FILE
APT.REC	APTR-FILE	
APTREC.CBL	APTR-FILE	
APTR.REC	WAPANAL-FILE WAPO-FILE WK1-FILE WK2-FILE WYPOM-FILE WYPOMK-FILE	
APTRREC.CBL	WAPANAL-FILE WAPO-FILE WK1-FILE WK2-FILE WYPOM-FILE WYPOMK-FILE	
APUPD.REC	APUPD-FILE	
APUPD.SEL		APUPD-FILE
APYPREC.CBL	YPOM-FILE	
ARTP.REC	ARTP-FILE	
ARTP.SEL		ARTP-FILE
ASC-AP.CBL		SEQ-FILE APO-FILE SYNT-FILE
ASC-AP0.CBL		SEQ-FILE APO-FILE
ASC-AP1.CBL	SEQ-FILE WAPO-FILE	SEQ-FILE WAPO-FILE
ASC-AP2.CBL	SEQ-FILE APO-FILE	SEQ-FILE APO-FILE
ASC-APAN.CBL		APANAL-FILE CNV-FILE SEQ-FILE

ASC-LGAN.CBL	SEQ-FILE LINK-FILE	SEQ-FILE LINK-FILE
ASC-PEL.CBL		SEQ-FILE PELAT-FILE
ASC-PEL1.CBL	SEQ-FILE LINK-FILE	SEQ-FILE PELAT-FILE LINK-FILE
ASC-PEL3.CBL	SEQ-FILE PELAT-FILE	SEQ-FILE PELAT-FILE
ASC-PEL4.CBL	SEQ-FILE PELAT-FILE	SEQ-FILE PELAT-FILE
ASC-PROM.CBL	SEQ-FILE CNV-FILE	SEQ-FILE PROM-FILE CNV-FILE
ASC-PEL.CBL	SEQ-FILE PELAT-FILE	
ASC-PROM1.CBL	SEQ-FILE LINK-FILE	SEQ-FILE LINK-FILE
ASC-PROM3.CBL	SEQ-FILE	SEQ-FILE PROM-FILE
ASC-TEAM2.CBL	SEQ-FILE APTEAM-FILE	SEQ-FILE APTEAM-FILE
ASC-TEAM.CBL	SEQ-FILE APTEAM-FILE	SEQ-FILE APTEAM-FILE
CANCPRT.CBL	PRTIND-FILE PRTLIST-FILE	PRTIND-FILE PRTLIST-FILE
CR-L.CBL		
CS-L.CBL		
DB.REC	DB-FILE	
DB.SEL		DB-FILE
DED.REC	DED-FILE	
DED.SEL		DED-FILE
EKTREC.CBL	EKT-FILE	
EKTEIS.CBL		EKT-FILE
EKTEPIL.CBL		EKT-FILE
EMP-AGO.CBL	AGD-FILE AGT-FILE TOM-FILE TYPOM-FILE WTOM-FILE WTYPOM-FILE WAGD-FILE WAGT-FILE	AGD-FILE AGT-FILE TOM-FILE TYPOM-FILE WTOM-FILE WTYPOM-FILE WAGD-FILE WAGT-FILE

EMP-APO.CBL	APANAL-FILE	APANAL-FILE
	APD-FILE	APD-FILE
	APFPA-FILE	APFPA-FILE
	APO-FILE	APO-FILE
	APTIMCH-FILE	APTIMCH-FILE
	K1-FILE	K1-FILE
	K2-FILE	K2-FILE
	MON-FILE	MON-FILE
	PRTIM-FILE	PRTIM-FILE
	WAPANAL-FILE	WAPANAL-FILE
	WAPD-FILE	WAPD-FILE
	WAPF-FILE	WAPF-FILE
	WAPFPA-FILE	WAPFPA-FILE
	WAPO-FILE	WAPO-FILE
	WAPO1-FILE	WAPO1-FILE
	WAPP-FILE	WAPP-FILE
	WAPTEAM-FILE	WAPTEAM-FILE
	WAPTEAM1-FILE	WAPTEAM1-FILE
	WAPTIMCH-FILE	WAPTIMCH-FILE
	WAPTM-FILE	WAPTM-FILE
	WKODE-FILE	WKODE-FILE
	WMON-FILE	WMON-FILE
	WTAPOM-FILE	WTAPOM-FILE
	YPOM-FILE	YPOM-FILE
EMP-PEL.CBL	ANAL-FILE	ANAL-FILE
	ARTHRO-FILE	ARTHRO-FILE
	EPAG-FILE	EPAG-FILE
	PELAT-FILE	PELAT-FILE
	PER-FILE	PER-FILE
	PLYP-FILE	PLYP-FILE
	POL-FILE	POL-FILE
	POLD-FILE	POLD-FILE
	WANAL-FILE	WANAL-FILE
	WARTHRO-FILE	WARTHRO-FILE
	WPELAT-FILE	WPELAT-FILE
	WPELF-FILE	WPELF-FILE
	WPLYP-FILE	WPLYP-FILE
	WPOLD-FILE	WPOLD-FILE
EMP-PRO.CBL	ANAL-FILE	ANAL-FILE
	ARTHRO-FILE	ARTHRO-FILE
	EPAG1-FILE	EPAG1-FILE
	PER1-FILE	PER1-FILE
	POLD-FILE	POLD-FILE
	PROM-FILE	PROM-FILE
	WANAL-FILE	WANAL-FILE
	WARTHRO-FILE	WARTHRO-FILE
	WPOLD-FILE	WPOLD-FILE
	WPRF-FILE	WPRF-FILE
	WPROM-FILE	WPROM-FILE
EMP-TIM.CBL	COMM-FILE	COMM-FILE
	COUNT-FILE	COUNT-FILE
	DELTEKD-FILE	DELTEKD-FILE
	EKPOLHS-FILE	EKPOLHS-FILE
	POLHS-FILE	POLHS-FILE
	TIM-FILE	TIM-FILE
	TIMCNT-FILE	TIMCNT-FILE
	TOM-FILE	TOM-FILE
	TYPOM-FILE	TYPOM-FILE
	WCOMM-FILE	WCOMM-FILE
	WCOUNT-FILE	WCOUNT-FILE
	WINV-FILE	WINV-FILE
	WPOLHS-FILE	WPOLHS-FILE
	WTOM-FILE	WTOM-FILE
	WTYPOM-FILE	WTYPOM-FILE
entypo2.cbl	TITLOS-FILE	TITLOS-FILE
	ENTYPA	ENTYPA
	ETER	ETER
	PRINT-FILE	PRINT-FILE
entypo.cbl	TITLOS-FILE	TITLOS-FILE
	ENTYPA	ENTYPA
	ETER	ETER
	PRINT-FILE	PRINT-FILE

EPAGREC1.CBL	EPAG1-FILE	
EPAGREC.CBL	EPAG-FILE	
EPESYN.CBL	EPIT1-FILE	EPIT1-FILE SORT-FILE
EPI.SEL		EPIT-FILE
EPI1.SEL		EPIT-FILE
EPIMASTER.CBL	EPIT-FILE	
EPIT1REC.CBL	EPIT-FILE	
EPITDEL2.CBL		EPIT-FILE
EPITKEY.CBL		EPIT-FILE
EPITOPTRAN.CBL		TRANS-FILE
EPITREC.CBL	EPIT-FILE	
EPITTRAN.CBL	TRANS-FILE	
EPITTRANSR.CBL	TRANS-FILE	
EPMET.CBL		PROM-FILE
FEANEW.CBL	POL-FILE PER-FILE EPAG-FILE PELF-FILE	POL-FILE PER-FILE EPAG-FILE PELF-FILE
fhandle.cbl	TXTFILE PRINTFILE	TXTFILE PRINTFILE
FLT.REC	FLT-FILE FLTHEAD-FILE	
FLT.SEL		FLT-FILE FLTHEAD-FILE
formvar.rec	VAR-FILE	
formvar.sel		VAR-FILE
GR1CNT.CBL	GRCNT1-FILE	GRCNT1-FILE
GRA1.REC	GRCNT1-FILE	
GRA1.SEL		GRAM-FILE GRCNT1-FILE
GRAM1OPTR.CBL		TRANS-FILE
GRAMOPTRAN.CBL		TRANS-FILE
GRAM1REC.CBL	GRAM-FILE	
GRAM1TREC.CBL	TRANS-FILE	
GRAMPLHR.CBL	GRAM-FILE	
GRAMREC.CBL	GRAM-FILE	
GRAMTRREC.CBL	TRANS-FILE	
GRA.REC	GRCNT-FILE	
GRA.SEL		GRAM-FILE GRCNT-FILE
GRCNT1REC.CBL	GRCNT1-FILE	
GRCNTREC.CBL	GRCNT-FILE	
GRL.REC	GRL-FILE	
GRL.SEL		GRL-FILE
GRPROMREC.CBL	PROM-FILE	
hlp.sel		TAPOM-FILE TPLOM-FILE
hlp1	PELAT-FILE	
HOSP.REC	HOSP-FILE	
HOSP.SEL		HOSP-FILE
HOTEL.REC	HOTEL-FILE	
HOTEL.SEL		HOTEL-FILE
HPL.REC	HPL-FILE	
HPL.SEL		HPL-FILE
HTLEXAG.CBL	WAPTM-FILE	WAPTM-FILE

ICL-IN.CBL	IDX-FILE	IDX-FILE
	SEQ-FILE	SEQ-FILE
ICLREAD.CBL	IDX-FILE	IDX-FILE
	SEQ-FILE	SEQ-FILE
K1NEW.CBL	APTEAM-FILE	APTEAM-FILE
	WK1-FILE	WK1-FILE
K2NEW.CBL	K2-FILE	K2-FILE
	WK2-FILE	WK2-FILE
KODE.REC	KODE-FILE	
KODE.SEL		KODE-FILE
LEM.REC	LINK-FILE	
LEM.SEL		LINK-FILE
LG-PL.CBL	LGEMPO	LGEMPO
	LGMAST	LGMAST
	WPELAT-FILE	WPELAT-FILE
LINKREC.CBL	LGLINK-FILE	
log1.rec	ALOG	
	ANALAA	
	ANALBB	
	ANALCC	
	FANT	
	LGDATE	
	LGEMPO	
	LGINFO	
	LGKATHG	
	LGKFPA	
	LGKKEID	
	LGMAST	
	LGMASTN	
	LGPARFPA	
	LGPFPA	
	SORTRAN	
	TRANCC	
log1.sel		ALOG
		ANALAA
		ANALBB
		ANALCC
		FANT
		LGDATE
		LGEMPO
		LGINFO
		LGKATHG
		LGKFPA
		LGKKEID
		LGMAST
		LGMASTN
		LGPARFPA
		LGPFPA
		SORTRAN
		TRANCC
log.rec	ANALAA	
	ANALBB	
	ANALCC	
	FANT	
	LGCNT	
	LGDATE	
	LGINFO	
	LGKATHG	
	LGKFPA	
	LGKKK	
	LGMAST	
	LGMASTN	
	LGPARFPA	
	LGPFPA	
	LGTADATE	
	LGTAM	
	SORTRAN	

log.sel		ANALAA ANALBB ANALCC FANT LGCNT LGDATE LGINFO LGKATHG LGKFPA LGKKK LGMAST LGMASTN LGPARFPA LGPFPA LGTADATE LGTAM SORTRAN
MAIN.REC	MAIN	
MAIN.SEL		MAIN
MAKAG.CBL		AGT-FILE ARTHRO-FILE
MAKAGD.CBL	AGD-FILE	AGD-FILE
MAKagt.CBL	ARTHRO-FILE WARTHRO-FILE	ARTHRO-FILE WARTHRO-FILE
MAKAP0-11.CBL	APO-FILE APO1-FILE	APO-FILE APO1-FILE
MAKAP0-1.CBL	APO-FILE WAPO-FILE	APO-FILE WAPO-FILE
MAKAP1-2.CBL	APO-FILE WAPO-FILE	APO-FILE WAPO-FILE
MAKAPAN2.CBL	WAPANAL-FILE	WAPANAL-FILE
MAK_APAN.CBL	APANAL-FILE WAPANAL-FILE	APANAL-FILE WAPANAL-FILE
mak_apan.cbl	APANAL-FILE WAPANAL-FILE	APANAL-FILE WAPANAL-FILE
MAK+APAN.CBL	APANAL-FILE WAPANAL-FILE	APANAL-FILE WAPANAL-FILE
MAKAPANAL.CBL		APANAL-FILE
MAK-AP.CBL	APO-FILE WAPO-FILE	APO-FILE WAPO-FILE
mak-ap.cbl	APO-FILE WAPO-FILE	APO-FILE WAPO-FILE
makap1.cbl		APO-FILE
MAKAP-CS.CBL	APO-FILE WAPO-FILE	APO-FILE WAPO-FILE
MAK-APN.CBL	APANAL-FILE WAPANAL-FILE	APANAL-FILE WAPANAL-FILE
makapo1.cbl	APO-FILE	APO-FILE
makapo.cbl	APO-FILE	APO-FILE
makapseq.cbl	WAPANAL-FILE HAPANAL-FILE	WAPANAL-FILE HAPANAL-FILE
MAKAPT0-11.CBL	APTEAM-FILE APTEAM1-FILE	APTEAM-FILE APTEAM1-FILE
MAKAPT0-1.CBL	APTEAM-FILE WAPTEAM-FILE	APTEAM-FILE WAPTEAM-FILE
mak-apy.cbl	APTEAM-FILE YPOM-FILE	APTEAM-FILE YPOM-FILE
MAKAR.CBL		ARTHRO-FILE
MAKARADD.CBL		ARTHRO-FILE
MAKARDEL.CBL		ARTHRO-FILE
MAKARG.CBL		ARTHRO-FILE

MAKart.CBL	ARTHRO-FILE WARTHRO-FILE	ARTHRO-FILE WARTHRO-FILE
MAKART.CBL		ARTHRO-FILE
MAKARTHRO.CBL		ARTHRO-FILE
MAKEPIT.CBL		EPIT-FILE
MAKEPIT1.CBL		EPIT-FILE
MAKGRAM.CBL		GRAM-FILE
MAKGRAM1.CBL		GRAM-FILE
MAKGRX.CBL	GRAM-FILE WGRAM-FILE	GRAM-FILE WGRAM-FILE
MAKINV.CBL	WINV-FILE	WINV-FILE
MAKOIKO.CBL	APANAL-FILE APD-FILE APFPA-FILE APO-FILE APTIMCH-FILE K1-FILE K2-FILE MON-FILE PRTIM-FILE WAPANAL-FILE WAPD-FILE WAPF-FILE WAPFPA-FILE WAPO-FILE WAPO1-FILE WAPP-FILE WAPTEAM-FILE WAPTEAM1-FILE WAPTIMCH-FILE WAPTM-FILE WKODE-FILE WMON-FILE WTAPOM-FILE YPOM	APANAL-FILE APD-FILE APFPA-FILE APO-FILE APTIMCH-FILE K1-FILE K2-FILE MON-FILE PRTIM-FILE WAPANAL-FILE WAPD-FILE WAPF-FILE WAPFPA-FILE WAPO-FILE WAPO1-FILE WAPP-FILE WAPTEAM-FILE WAPTEAM1-FILE WAPTIMCH-FILE WAPTM-FILE WKODE-FILE WMON-FILE WTAPOM-FILE YPOM
MAKPAR0-1.CBL	OLDPAR-FILE	OLDPAR-FILE PAR-FILE
MAK-PEL.CBL	PELAT-FILE WPELAT-FILE	PELAT-FILE WPELAT-FILE
MAKPLAN3.CBL	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE
MAK_PLAN.CBL	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE
mak_plan.cbl	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE
MAKPLCD.CBL	WPELAT-FILE	PELAT-FILE WPELAT-FILE
MAKPL-CS.CBL	WPELAT-FILE	PELAT-FILE WPELAT-FILE
mak_pln1.cbl	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE
MAKPLPR.CBL	PELAT-FILE PROM-FILE	PELAT-FILE PROM-FILE
MAKPRAN3.CBL	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE
MAK_PRAN.CBL	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE
mak_pran.cbl	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE
MAK-PR.CBL	PELAT-FILE WPELAT-FILE	PELAT-FILE WPELAT-FILE
MAKPR-CS.CBL	WPROM-FILE	PROM-FILE WPROM-FILE
mak_prn1.cbl	ANAL-FILE WANAL-FILE	ANAL-FILE WANAL-FILE

MAKPROM.CBL	PROM-FILE	PROM-FILE
	WPROM-FILE	WPROM-FILE
MAKTRAN.CBL	APANAL-FILE	APANAL-FILE
	ANAL-FILE	ANAL-FILE
MENCPL.REC	MENCPL-FILE	
	MENMDF-FILE	
MENCPL.SEL		MENCPL-FILE
		MENMDF-FILE
MEN.REC	MENU-FILE	
MEN.SEL		MENU-FILE
MERSRT.CBL	CANAL-FILE	SMANAL-FILE
	SANAL-FILE	CANAL-FILE
	SMOANAL-FILE	SANAL-FILE
		SMOANAL-FILE
MINIE-APO.CBL	MORFH-APO	MORFH-APO
	WAPD-FILE	WAPD-FILE
	WAPF-FILE	WAPF-FILE
	WAPFPA-FILE	WAPFPA-FILE
	WAPO-FILE	WAPO-FILE
	WAPO1-FILE	WAPO1-FILE
	WAPP-FILE	WAPP-FILE
	WAPTEAM-FILE	WAPTEAM-FILE
	WAPTEAM1-FILE	WAPTEAM1-FILE
	WAPTIMCH-FILE	WAPTIMCH-FILE
	WAPTM-FILE	WAPTM-FILE
	WKODE-FILE	WKODE-FILE
	WMON-FILE	WMON-FILE
	WTAPOM-FILE	WTAPOM-FILE
MINIE-PEL.CBL	MORFH-PEL	MORFH-PEL
	WPELAT-FILE	WPELAT-FILE
	WPELF-FILE	WPELF-FILE
	WANAL-FILE	WANAL-FILE
	WPOLD-FILE	WPOLD-FILE
	WARTHRO-FILE	WARTHRO-FILE
	WPLYP-FILE	WPLYP-FILE
MLTERM.REC	MLTERM-FILE	
	MLPOLHS-FILE	
MLTERM.SEL		MLTERM-FILE
		MLPOLHS-FILE
MNFYIMAS.REC	MNFYIMAS-FILE	
MNFYIMAS.SEL		MNFYIMAS-FILE
MNFYI.REC	MNFYI-FILE	
MNFYI.SEL		MNFYI-FILE
MNREFR.REC	MNREFR-FILE	
MNREFR.SEL		MNREFR-FILE
MNUSRPER.REC	MNUSRPER-FILE	
	TMPPER-FILE	
MNUSR.REC	MNUSR-FILE	
MNUSR.SEL		MNUSR-FILE
MNUSRPER.SEL		MNUSRPER-FILE
		TMPPER-FILE
MSMAKAN.CBL	MSANAL-FILE	MSANAL-FILE
	WMSANAL-FILE	WMSANAL-FILE
MSSEQ.CBL	MSANAL-FILE	MSANAL-FILE
	WMSANAL-FILE	WMSANAL-FILE
OFFH.REC	OFFH-FILE	
OFFH.SEL		OFFH-FILE
OFFI.REC	OFFI-FILE	
OFFI.SEL		OFFI-FILE
OM-OM.CBL	K1-FILE	K1-FILE
	APTEAM-FILE	APTEAM-FILE
ORDEID.CBL	PFORT-FILE	PFORT-FILE

ORDEIDDT.CBL	PFORT-FILE	PFORT-FILE
ORDER.REC	ORD-FILE	
ORPH.REC	ORPH-FILE	
ORPH.SEL		ORPH-FILE
ORPI.REC	ORPI-FILE	
ORPI.SEL		ORPI-FILE
PARAGREC.CBL	PAR-FILE	
PARCNTREC.CBL	PARCNT-FILE	
PARM.REC	PARM-FILE	
PARM.SEL		PARM-FILE
PEL.SEL		ANAL-FILE PELF-FILE POLD-FILE ARTHRO-FILE PLYP-FILE PLFAIM-FILE PELNME-FILE TPLOM-FILE
PEL1.SEL		ANAL-FILE PELF-FILE POLD-FILE ARTHRO-FILE PLYP-FILE PLFAIM-FILE PELNME-FILE TPLOM-FILE
PELANAL.REC	ANAL-FILE	
PELAT.REC	PELAT-FILE	
PELAT.SEL		PELAT-FILE
PELFREC.CBL	PELF-FILE	
PELIDX.CBL	SPELAT-FILE	SPELAT-FILE
PELMAST.REC	PELAT-FILE	
PELNEW.CBL	PELAT-FILE WPELAT-FILE	PELAT-FILE WPELAT-FILE
PELNMEREC.CBL	PELNME-FILE	
PELSEQ.CBL	PELAT-FILE SPELAT-FILE	PELAT-FILE SPELAT-FILE
PFORTEID2.CBL	PFORT-FILE PRINT-FILE	
PFORTEID.CBL	PFORT-FILE PRINT-FILE	PFORT-FILE PRINT-FILE
PFORTEID2.CBL		PFORT-FILE PRINT-FILE
PFORTSYN.CBL	PFORT-FILE PRINT-FILE	PFORT-FILE PRINT-FILE
PL-AFM.CBL	SEQ-FILE PELAT-FILE WPELAT-FILE	SEQ-FILE PELAT-FILE WPELAT-FILE
PLFAIMREC.CBL	PLFAIM-FILE	
PLOPMR.CBL		PELAT-FILE ANAL-FILE PELF-FILE POLD-FILE ARTHRO-FILE PLYP-FILE PLFAIM-FILE PELNME-FILE TPLOM-FILE
PL-PL.CBL	PELAT-FILE WPELAT-FILE	PELAT-FILE WPELAT-FILE
PLPOLREP.CBL	HLP-FILE PRINT-FILE	HLP-FILE PRINT-FILE

PLPREAD.CBL	PELAT-FILE PROM-FILE	PELAT-FILE PROM-FILE
PLSTAT.CBL	HLP-FILE	HLP-FILE
PLYPREC.CBL	PLYP-FILE	
POLREC.CBL	POL-FILE	
PR1.REC	PROM-FILE	
PR1.SEL		PROM-FILE
PR11.SEL		PROM-FILE
PR2.SEL		ANAL-FILE
PR3.SEL		POLD-FILE
PR4.SEL		ARTHRO-FILE
PR5.SEL		TPROM-FILE
PR6.SEL		PRF-FILE
PR-AFM.CBL	SEQ-FILE PROM-FILE WPROM-FILE	SEQ-FILE PROM-FILE WPROM-FILE
PRANALREC.CBL	ANAL-FILE	
PRFREC.CBL	PRF-FILE	
PRMASTREC.CBL	PROM-FILE	
PRORD.REC	PRORD-FILE PRORDCNT-FILE	
pro.rec	PROM-FILE	
pro.sel		PROM-FILE
PROPMR.CBL		PROM-FILE ANAL-FILE POLD-FILE ARTHRO-FILE TPROM-FILE PRF-FILE
PR-PR.CBL	PROM-FILE WPROM-FILE	PROM-FILE WPROM-FILE
PRORD.SEL		PRORD-FILE PRORDCNT-FILE
PRSTAT.CBL	HLP-FILE	HLP-FILE
PRSTAT.GEN	HLP-FILE	HLP-FILE
PRTEIS.CBL		EKT-FILE
PRTEIS.OLD		EKT-FILE
PRTSEL.CBL		EKT-FILE
PRTSEL.OLD		EKT-FILE
PRUPD.REC	PRUPD-FILE	
PRUPD.SEL		PRUPD-FILE
REC1.REC	WANAL-FILE WARTHRO-FILE	
REC1.SEL		WANAL-FILE WARTHRO-FILE
recpm.wor	EKT-FILE	
REC.REC	WAPANAL-FILE WANAL-FILE WARTHRO-FILE	
REC.SEL		WANAL-FILE WAPANAL-FILE WARTHRO-FILE
RELAPO.CBL	WAPO-FILE	APO-FILE WAPO-FILE
RELPEL.CBL		PELAT-FILE
RELPRO.CBL		PROM-FILE
rhandle.cbl	TXTFILE PRINTFILE	TXTFILE PRINTFILE

rhandle.old	TXTFILE PRINTFILE	TXTFILE PRINTFILE
SALES-ASC1.CBL	IDX-FILE SEQ-FILE SEQ-MS-FILE	IDX-FILE SEQ-FILE SEQ-MS-FILE
SALES-ASC.CBL	IDX-FILE SEQ-FILE SEQ-MS-FILE	IDX-FILE SEQ-FILE SEQ-MS-FILE
SALES-ASC.STD	IDX-FILE SEQ-FILE SEQ-MS-FILE	IDX-FILE SEQ-FILE SEQ-MS-FILE
SEQREAD.CBL	PRINT-FILE WAPANAL-FILE	PRINT-FILE WAPANAL-FILE
SYSENV.CBL	SYSCFG-FIE SYSENV-FILE SYSOUT-FILE	SYSCFG-FIE SYSENV-FILE SYSOUT-FILE
SYSENV.REC	SYSENV-FILE	
SYSENV.SEL		SYSENV-FILE
TAG1.REC	TIMAG-FILE	
TAG3.REC	TIMAG-FILE	
TAG.REC	TIMAG-FILE	
TAG.SEL		TIMAG-FILE
TAMDEL.REC	TAMDEL-FILE	
TAMDEL.SEL		TAMDEL-FILE
TAPOMREC.CBL	TAPOM-FILE	
TAPQNTY.REC	TAPQNTY-FILE	
TAPQNTY.SEL		TAPQNTY-FILE
TARTHREC.CBL	ARTHRO-FILE	
TARTHROREC.CBL	ARTHRO-FILE	
TCOMREC.CBL	COMM-FILE	
TCOUNTREC.CBL	COUNT-FILE	
TDPCREC.CBL	TDPC-FILE	
TDPEKT2.CBL	PFORT-FILE TMP-FILE	PFORT-FILE TMP-FILE
TDPREC.CBL	TDPR-FILE	
TAG2.REC	TIMAG-FILE	
TFORTEID2.CBL	PFORT-FILE	PFORT-FILE
TFORTEID.CBL	PFORT-FILE	PFORT-FILE
THELPREC.CBL	THELP-FILE	
TIMCNTREC.CBL	TIMCNT-FILE	
TIMREC.CBL	TIM-FILE	
TIM.SEL		COMM-FILE COUNT-FILE INV-FILE LGLINK-FILE ORD-FILE PAR-FILE PARCNT-FILE POLHS-FILE TIM-FILE TIMCNT-FILE TMET-FILE TOM-FILE TYPOM-FILE
TINVREC.CBL	INV-FILE	
title.cbl	ENTYPA ETER TITLOS-FILE	ENTYPA ETER TITLOS-FILE
titleden.cbl	ENTYPA	ENTYPA
titledet.cbl	ETER	ETER

titlekar.cbl	ENTYPA ETER TITLOS-FILE	ENTYPA ETER TITLOS-FILE
titlent.cbl	ENTYPA	ENTYPA
titlepka.cbl	ENTYPA ETER TITLOS-FILE	ENTYPA ETER TITLOS-FILE
titleter.cbl	ETER	ETER
TITLOP.CBL	TITLOS-FILE	TITLOS-FILE
TMETREC.CBL	TMET-FILE	
TMP.REC	HLP-FILE	
TMP.SEL		HLP-FILE
TOMREC.CBL	TOM-FILE	
TPLOMREC.CBL	TPLOM-FILE	
TPOLDREC.CBL	POLD-FILE	
TPOLHSREC.CBL	POLHS-FILE	
TPROMREC.CBL	TPROM-FILE	
TRA.SEL		TRA-FILE
TRAN.REC	WAPANAL-FILE WAPO-FILE WAPOMA-FILE WAPTEAM-FILE	
TRAN.SEL		WAPANAL-FILE WAPO-FILE WAPOMA-FILE WAPTEAM-FILE
TRAPREC.CBL	TRA-FILE	
TRAREC.CBL	TRA-FILE	
TRGETREC.CBL	WANAL-FILE WAPANAL-FILE WARTHRO-FILE	
TRWH.REC	TRWH-FILE	
TRWH.SEL		TRWH-FILE
TVIEW1.CBL	WARTHRO-FILE	WARTHRO-FILE
TYPOMREC.CBL	TYPOM-FILE	
VAR.REC	BIN-FILE VAR-FILE	
VAR.SEL		BIN-FILE VAR-FILE
WAPOM1.REC	WAPO1-FILE	
WAPO.REC	WAPO-FILE WAPO1-FILE WAPOMA-FILE	
WAPO.SEL		WAPO-FILE WAPO1-FILE WAPOMA-FILE
WAPO1.SEL		WAPO1-FILE
WINSEL.REC	WINSEL-FILE	
WINSEL.REC@	WINSEL-FILE	
WINSEL.SEL		WINSEL-FILE
WINSEL.SEL@		WINSEL-FILE
WPEL.REC	WPELAT-FILE	
WPEL.SEL		WPELAT-FILE
WPRO.REC	WPROM-FILE	
WPRO.SEL		WPROM-FILE

XPERTDB.CBL	AGART-FILE	AGART-FILE
	AGOD-FILE	AGOD-FILE
	APANAL-FILE	PRANAL-FILE
	PRANAL-FILE	
YPOK-APO.CBL	APD-FILE	APD-FILE
	APFPA-FILE	APFPA-FILE
	APO-FILE	APO-FILE
	APTM-FILE	APTM-FILE
	K1-FILE	K1-FILE
	K2-FILE	K2-FILE
	MEG-FILE	MEG-FILE
	MON-FILE	MON-FILE
	WAPD-FILE	WAPD-FILE
	WAPF-FILE	WAPF-FILE
	WAPFPA-FILE	WAPFPA-FILE
	WAPO-FILE	WAPO-FILE
	WAPO1-FILE	WAPO1-FILE
	WAPP-FILE	WAPP-FILE
	WAPTEAM-FILE	WAPTEAM-FILE
	WAPTEAM1-FILE	WAPTEAM1-FILE
	WAPTIMCH-FILE	WAPTIMCH-FILE
	WAPTM-FILE	WAPTM-FILE
	WKODE-FILE	WKODE-FILE
	WMON-FILE	WMON-FILE
	WTAPOM-FILE	WTAPOM-FILE
	YPOM-FILE	YPOM-FILE
	YPOMK-FILE	YPOMK-FILE
YPOKGET.CBL		WAPANAL-FILE
		WANAL-FILE
		WARTHRO-FILE
YPOK-PEL.CBL	EPAG-FILE	EPAG-FILE
	PELAT-FILE	PELAT-FILE
	PER-FILE	PER-FILE
	POL-FILE	POL-FILE
	POLD-FILE	POLD-FILE
	WANAL-FILE	WANAL-FILE
	WARTHRO-FILE	WARTHRO-FILE
	WPELAT-FILE	WPELAT-FILE
	WPELF-FILE	WPELF-FILE
	WPLYP-FILE	WPLYP-FILE
	WPOLD-FILE	WPOLD-FILE
YPOK-PRO.CBL	PROM-FILE	PROM-FILE
	PER1-FILE	PER1-FILE
	EPAG1-FILE	EPAG1-FILE
	POLD-FILE	POLD-FILE
	WPROM-FILE	WPROM-FILE
	WANAL-FILE	WANAL-FILE
	WPOLD-FILE	WPOLD-FILE
	WARTHRO-FILE	WARTHRO-FILE
	WPRF-FILE	WPRF-FILE
YPOK-TIM.CBL	COMM-FILE	COMM-FILE
	COUNT-FILE	COUNT-FILE
	TIM-FILE	TIM-FILE
	TIMCNT-FILE	TIMCNT-FILE
	TOM-FILE	TOM-FILE
	TYPOM-FILE	TYPOM-FILE
	WCOMM-FILE	WCOMM-FILE
	WCOUNT-FILE	WCOUNT-FILE
	WINV-FILE	WINV-FILE
	WPOLHS-FILE	WPOLHS-FILE
	WTOM-FILE	WTOM-FILE
	WTYPOM-FILE	WTYPOM-FILE
YPOMKNEW.CBL	APTEAM-FILE	APTEAM-FILE
	WYPOMK-FILE	WYPOMK-FILE
YPOMNEW.CBL	APTEAM-FILE	APTEAM-FILE
	WYPOM-FILE	WYPOM-FILE
YP-YP.CBL	APTEAM-FILE	APTEAM-FILE
	YPOM-FILE	YPOM-FILE

Appendix 16

The complete set of datafiles of the XPERT Hotel software application.

AGART-FILE	AGD-FILE	AGDXFILE	AGOD-FILE
AGP-FILE	AGPCNT-FILE	AGPXFILE	AGT-FILE
AGTXFILE	ALOG	ALPHA-FILE	AMUPD-FILE
AMUPDXFILE	ANAL	ANAL-FILE	ANALAA
ANALBB	ANALCC	ANALXFILE	APAL-FILE
APALBRI-FILE	APANAL-FILE	APANAL1-FILE	APANALXFILE
APANLOG-FILE	APD-FILE	APDPR-FILE	APDPRXFILE
APDXFILE	APF-FILE	APF1-FILE	APF1LNK-FILE
APFL-FILE	APFPA-FILE	APFPR-FILE	APGK-FILE
APGKXFILE	APMOD-FILE	APMODXFILE	APO-FILE
APO1-FILE	APO1XFILE	APOGR-FILE	APOGR1-FILE
APOGRXFILE	APOMA-FILE	APOMAX-FILE	APOX-FILE
APP-FILE	APRD-FILE	APRDXFILE	APREL-FILE
APSTAT-FILE	APTEAM-FILE	APTEAM1-FILE	APTIMCH-FILE
APTIMCHXFILE	APTM-FILE	APTMX-FILE	APTR-FILE
APTRXFILE	APUPD-FILE	APUPDXFILE	AREA-FILE
ARTHRO-FILE	ARTHROXFILE	ARTP-FILE	ARX
ASUPD-FILE	ASUPDXFILE	BAT-FILE	BIN-FILE
CANAL-FILE	CNV-FILE	COMM-FILE	COUNT-FILE
COUNTXFILE	DB-FILE	DED-FILE	DELTEKD-FILE
DOC-FILE	EKPOLHS-FILE	EKT-FILE	ENTYPA
EPAG-FILE	EPAG1-FILE	EPIT-FILE	EPIT1-FILE
EPITXFILE	ERROR-FILE	ETER	FANT
FLT-FILE	FLTHEAD-FILE	FORMPRT-FILE	GRAM-FILE
GRAMXFILE	GRCNT-FILE	GRCNT1-FILE	GRL-FILE
HAPANAL-FILE	HELP-FILE	HLP-FILE	HOSP-FILE
HOSPXFILE	HOTEL-FILE	HOTELXFILE	HPL-FILE
IDX-FILE	INV-FILE	IOSP-FILE	K1-FILE
K2-FILE	KODE-FILE	KOS-FILE	LGCNT
LGDATE	LGEMPO	LGINFO	LGKATHG
LGKFPA	LGKKEID	LGKKK	LGLINK-FILE
LGMAST	LGMASTN	LGPARFPA	LGPFPA
LGTADATE	LGTAM	LINK-FILE	MAIN
MEG-FILE	MENCPL-FILE	MENDOC-FILE	MENMDF-FILE
MENSPT-FILE	MENU-FILE	MLPOLHS-FILE	MLPOLHSXFILE
MLTERM-FILE	MNFYI-FILE	MNFYIMAS-FILE	MNREFR-FILE
MNUSR-FILE	MNUSRPER-FILE	MON-FILE	MORFH-APO
MORFH-PEL	MORFH-PRO	MSANAL-FILE	NEW-FILE
NOR-MENU-FILE	NOR-PELAT-FILE	NORFILT-FILE	OANAL-FILE
OFFH-FILE	OFFI-FILE	OLD-FILE	OLDPAR-FILE
ORD-FILE	ORDXFILE	ORPH-FILE	ORPI-FILE
PAR-FILE	PARCNT-FILE	PARM-FILE	PARXFILE
PELAT-FILE	PELATXFILE	PELF-FILE	PELFXFILE
PELNME-FILE	PER-FILE	PER1-FILE	PFORT-FILE
PLFAIM-FILE	PLHELP-FILE	PLYP-FILE	POL-FILE
POLD-FILE	POLDXFILE	POLHS-FILE	POLHSXFILE
POS-FILE	PRANAL-FILE	PRF-FILE	PRFXFILE
PRINT-FILE	PRINTFILE	PROM-FILE	PROMX-FILE

PROMXFILE	PRORD-FILE	PRORDCNT-FILE	PRTIM-FILE
PRTIND-FILE	PRTLIST-FILE	PRUPD-FILE	PRUPDXFILE
REL-FILE	SANAL-FILE	SEQ-FILE	SEQ-MIS-FILE
SMANAL-FILE	SMOANAL-FILE	SNF-FILE	SNFI-FILE
SORT-FILE	SORTED-FILE	SORTRAN	SPELAT-FILE
SYNT-FILE	SYNT-KOS-FILE	SYSCFG-FILE	SYSENV-FILE
SYSOUT-FILE	TAMDEL-FILE	TAPOM-FILE	TAPQNTY-FILE
TDPR-FILE	TDPRC-FILE	TDPRXFILE	TEMP-FILE
THELP-FILE	TIM-FILE	TIMAG-FILE	TIMCNT-FILE
TIMXFILE	TITLOS-FILE	TMET-FILE	TMP-FILE
TMPI-FILE	TMPPER-FILE	TOM-FILE	TOTAL-FILE
TPLOM-FILE	TPROM-FILE	TRA-FILE	TRANCC
TRANS-FILE	TRWH-FILE	TRWHXFILE	TXTFILE
TYPOM-FILE	VAR-FILE	WAGD-FILE	WAGT-FILE
WANAL-FILE	WAPANAL-FILE	WAPD-FILE	WAPF-FILE
WAPFPA-FILE	WAPO-FILE	WAPO1-FILE	WAPOMA-FILE
WAPP-FILE	WAPTEAM-FILE	WAPTEAM1-FILE	WAPTIMCH-FILE
WAPTM-FILE	WARTHRO-FILE	WCOMM-FILE	WCOUNT-FILE
WGRAM-FILE	WINSEL-FILE	WINV-FILE	WK1-FILE
WK2-FILE	WKODE-FILE	WMON-FILE	WMSANAL-FILE
WPELAT-FILE	WPELF-FILE	WPLYP-FILE	WPOLD-FILE
WPOLHS-FILE	WPRF-FILE	WPROM-FILE	WTAPOM-FILE
WTOM-FILE	WTYPOM-FILE	WYPOM-FILE	WYPOMK-FILE
YPOM-FILE	YPOMK-FILE	YPOMK-FILE	YPOMK-FILE

Appendix 17

This appendix presents the complete set of datafiles of the XPERT Hotel software application and their definition attributes. For each datafile all the source code files that contain a structure definition for the datafile are listed in column 2 and all the source code files that contain a record description for the datafile are listed in column 3. Any mismatches identified among those definitoins are reported in column 4. Column 5 reports the external name of the datafile, which is used in order to store the datafile in the physical storage media (filesystem).

Table Name	Select File	Define File	Mismatches	FileName
AGART-FILE	XPERTDB.CBL	XPERTDB.CBL	No	agart.dat
AGD-FILE	AGO.SEL EMP-AGO.CBL MAKAGD.CBL	AGDREC.CBL EMP-AGO.CBL MAKAGD.CBL	Define	agdelt.dat
AGDXFILE	V3-AGO.CBL	V3-AGO.CBL		
AGOD-FILE	XPERTDB.CBL	XPERTDB.CBL	No	
AGP-FILE	AGO.SEL	AGPREC.CBL	No	agpdel.dat
AGPXFILE	V3-AGO.CBL	V3-AGO.CBL		
AGPCNT-FILE	AGO.SEL	AGPCNTREC.CBL	No	agpcnt.dat
AGT-FILE	AGO.SEL EMP-AGO.CBL MAKAG.CBL	AGTREC.CBL EMP-AGO.CBL	Define	agtim.dat
AGTXFILE	V3-AGO.CBL	V3-AGO.CBL		
ALOG	log1.sel	log1.rec	No	
ALPHA-FILE	ALPHATEST.CBL ALPHATEST2.CBL	ALPHATEST.CBL ALPHATEST2.CBL		
AMUPD-FILE	AMUPD.SEL	AMUPD.REC	No	amupd.trs
ANAL	MAKEMP.CBL	MAKEMP.CBL	No	
ANAL-FILE	EMP-PEL.CBL EMP-PRO.CBL MAKPLAN3.CBL MAKPRAN3.CBL MAKTRAN.CBL mak_plan.cbl MAK_PLAN.CBL mak_pln1.cbl mak_pran.cbl MAK_PRAN.CBL mak_prn1.cbl **PEL.SEL** PEL1.SEL PLOPMR.CBL **PR2.SEL** PROPMR.CBL	EMP-PEL.CBL EMP-PRO.CBL MAKPLAN3.CBL MAKPRAN3.CBL MAKTRAN.CBL mak_plan.cbl MAK_PLAN.CBL mak_pln1.cbl mak_pran.cbl MAK_PRAN.CBL mak_prn1.cbl **PELANAL.REC** **PRANALREC.CBL**	Select & Define	pelanal.dat [PEL.SEL] promanal.dat [PR2.SEL]
ANALXFILE	V3-PEL.CBL V3-PRO.CBL	V3-PEL.CBL V3-PRO.CBL		
ANALAA	log.sel log1.sel	log.sel log1.sel	No	
ANALBB	log.sel log1.sel	log.rec log1.rec	No	
ANALCC	log.sel log1.sel	log.rec log1.rec	Select & Define	
APAL-FILE	APO.SEL	APAL.REC	No	apal.dat
APALBRI-FILE	APALBRI.SEL	APALBRI.REC	No	apalbri.dat

APANAL-FILE	**APANAL.SEL**	**APANAL.REC**	Select & Define	apanal.dat
	APOHO.SEL	apanalho.rec		
	ASC-APAN.CBL	APANREC.CBL		
	EMP-APO.CBL	EMP-APO.CBL		
	MAK+APAN.CBL	MAK+APAN.CBL		
	MAK-APN.CBL	MAK-APN.CBL		
	MAKAPANAL.CBL	MAKOIKO.CBL		
	MAKOIKO.CBL	MAKTRAN.CBL		
	MAKTRAN.CBL	mak_apan.cbl		
	mak_apan.cbl	XPERTDB.CBL		
	MAK_APAN.CBL			
APANALXFILE	V3-APO.CBL	V3-APO.CBL	No	
APANAL1-FILE	ANALMAKE.CBL	ANALMAKE.CBL	No	
APANLOG-FILE	APO.SEL	APANLOG.REC	No	apanlog.dat
APD-FILE	**APO.SEL**	**APD.REC**	Select & Define	apdate.dat
	APO1.SEL	APDREC.CBL		
	APOHO.SEL	EMP-APO.CBL		
	MAKOIKO.CBL	MAKOIKO.CBL		
	EMP-APO.CBL	YPOK-APO.CBL		
	YPOK-APO.CBL			
APDXFILE	V3-APO.CBL	V3-APO.CBL	No	
APDPR-FILE	**APO.SEL**	**APDPR.REC**	No	apdate.dat
	APO1.SEL	APDPRREC.CBL		
	APOHO.SEL			
APDPRXFILE	V3-APO.CBL	V3-APO.CBL	No	
APF-FILE	**APO.SEL**	**APF.REC**	No	apf.dat
	APO1.SEL	APFEREC.CBL		
	APOHO.SEL	APFREC.CBL		
APF1-FILE	**APO.SEL**	**APF1.REC**	No	apf1.dat
	APO1.SEL	APF1REC.CBL		
	APOHO.SEL			
APF1LNK-FILE	**APO.SEL**	**APF1L.REC**	No	apf1lnk.dat
	APO1.SEL	APF1LREC.CBL		
	APOHO.SEL			
APFL-FILE	**APO.SEL**	**APFL.REC**	No	apfl.dat
	APO1.SEL	APFLREC.CBL		
	APOHO.SEL			
APFPA-FILE	**APO.SEL**	**APFPA.REC**	Select & Define	apfpa.dat
	APO1.SEL	APFPAREC.CBL		
	APOHO.SEL	EMP-APO.CBL		
	EMP-APO.CBL	YPOK-APO.CBL		
	MAKOIKO.CBL	MAKOIKO.CBL		
	YPOK-APO.CBL			
APFPR-FILE	APFPR.SEL	APFPR.REC	No	apfpr.dat
APGK-FILE	APGK.SEL	APGK.REC	No	apgk.dat
APGKXFILE	V3-APO.CBL	V3-APO.CBL	No	
APMOD-FILE	**APO.SEL**	**APMOD.REC**	No	apmod.dat
	APO1.SEL	APMODREC.CBL		
	APOHO.SEL			
APMODXFILE	V3-APO.CBL	V3-APO.CBL	No	

APO-FILE	AP-AP.CBL AP-GET.CBL APOHO.SEL **APOMAST.SEL** APOMAST1.SEL APOPMR.CBL ASC-AP.CBL ASC-AP0.CBL ASC-AP2.CBL EMP-APO.CBL mak-ap.cbl MAK-AP.CBL MAKAP-CS.CBL MAKAP0-1.CBL MAKAP0-11.CBL MAKAP1-2.CBL makap1.cbl makapo.cbl makapo1.cbl MAKOIKO.CBL RELAPO.CBL YPOK-APO.CBL	AP-AP.CBL AP-GET.CBL **APOMAST.REC** APOREC.CBL ASC-AP2.CBL EMP-APO.CBL mak-ap.cbl MAK-AP.CBL MAKAP-CS.CBL MAKAP0-1.CBL MAKAP0-11.CBL MAKAP1-2.CBL makapo.cbl makapo1.cbl MAKOIKO.CBL YPOK-APO.CBL	Select & Define	apo.dat
APOX-FILE	APOX.SEL	APOX.REC	No	
APO1-FILE	APOHO.SEL **APOM1.SEL** APOPMR.CBL MAKAP0-11.CBL	**APOM1.REC** APOM1REC.CBL MAKAP0-11.CBL	Define	apo1.dat
APO1XFILE	V3-APO.CBL	V3-APO.CBL	No	
APOGR-FILE	**APOGR.SEL** APOHO.SEL	APOGR.RE0 **APOGR.REC** APOGRNEW.REC	Select & Define	apogr.dat
APOGRXFILE	V3-APO.CBL	V3-APO.CBL	No	
APOGR1-FILE	APGCONV1.CBL	APGCONV1.CBL	No	
APOMA-FILE	**APOMA.SEL** APOMANEW.CBL APOPMR.CBL	**APOMA.REC** APOMANEW.CBL	Select & Define	apoma.dat
APOMAX-FILE	APOMAX.SEL	APOMAX.REC	No	
APP-FILE	**APO.SEL** APO1.SEL APOHO.SEL	**APP.REC** APPREC.CBL	No	app.dat
APRD-FILE	**APO.SEL** APO1.SEL APOHO.SEL	**APRD.REC** APRDREC.CBL	No	aprd.dat
APRDXFILE	V3-APO.CBL	V3-APO.CBL	No	
APREL-FILE	APREL.SEL	APREL.REC	No	aprelate.dat
APSTAT-FILE	APSTATUS.CBL	APSTATUS.CBL	No	pstat.[time]
APTEAM-FILE	**APO.SEL** APO1.SEL APOHO.SEL ASC-TEAM.CBL ASC-TEAM2.CBL K1NEW.CBL mak-apy.cbl MAKAPT0-1.CBL MAKAPT0-11.CBL OM-OM.CBL YP-YP.CBL YPOMKNEW.CBL YPOMNEW.CBL	**APTEAM.REC** APTEAMREC.CBL ASC-TEAM.CBL ASC-TEAM2.CBL K1NEW.CBL mak-apy.cbl MAKAPT0-1.CBL MAKAPT0-11.CBL OM-OM.CBL YP-YP.CBL YPOMKNEW.CBL YPOMNEW.CBL	Select & Define	apteam.dat

APTEAM1-FILE	**APO.SEL** APO1.SEL APOHO.SEL MAKAPT0-11.CBL	**APTE1.REC** APTE1REC.CBL MAKAPT0-11.CBL	No	apteam1.dat
APTIMCH-FILE	APOHO.SEL **APTIMCH.SEL** EMP-APO.CBL MAKOIKO.CBL	**APTIMCH.REC** APTIMCHREC.CBL EMP-APO.CBL MAKOIKO.CBL	Select & Define	aptimch.dat
APTIMCHXFILE	V3-APO.CBL	V3-APO.CBL	No	
APTM-FILE	APOHO.SEL **APTM.SEL** YPOK-APO.CBL	**APTM.REC** YPOK-APO.CBL	Select & Define	aptm.dat
APTMX-FILE	**APO.SEL** APO1.SEL APOHO.SEL	**APTMX.REC** APTMXREC.CBL	Select & Define	aptmx.dat
APTR-FILE	**APO.SEL** APO1.SEL APOHO.SEL	**APT.REC** APTREC.CBL	No	aptr.dat
APTRXFILE	V3-APO.CBL	V3-APO.CBL	No	
apupd	**APUPD.SEL** apaddtrs.cbl	**APUPD.REC** apaddtrs.cbl	Select & Define	apupd.trs
APUPD-FILE	**APUPD.SEL** apaddtrs.cbl	**APUPD.REC** apaddtrs.cbl	Select & Define	apupd.trs
APUPDXFILE	V5MAKUPD.CBL	V5MAKUPD.CBL	Select & Define	
AREA-FILE	GWIN.SEL		No	
ARTHRO-FILE	EMP-PEL.CBL EMP-PRO.CBL MAKAG.CBL MAKagt.CBL MAKAR.CBL MAKARADD.CBL MAKARDEL.CBL MAKARG.CBL MAKart.CBL MAKART.CBL MAKARTHRO.CBL **PEL.SEL** PEL1.SEL PLOPMR.CBL **PR4.SEL** PROPMR.CBL	EMP-PEL.CBL EMP-PRO.CBL MAKagt.CBL MAKart.CBL TARTHREC.CBL **TARTHROREC.CBL**	Select & Define	arthro.dat [PEL.SEL] agart.dat [PR4.SEL]
ARTHROXFILE	V3-PEL.CBL V3-PRO.CBL	V3-PEL.CBL V3-PRO.CBL		
ARTP-FILE	ARTP.SEL	ARTP.REC	No	agartp.dat
ARX	LGDEL.CBL MAKLINK.CBL	LGDEL.CBL MAKLINK.CBL		
ASUPD-FILE	ASUPD.SEL	ASUPD.REC	No	
ASUPDXFILE	V5MAKUPD.CBL	V5MAKUPD.CBL	No	
BAT-FILE	ICLSALES.CBL	ICLSALES.CBL	No	
BIN-FILE	formbin.sel VAR.SEL	formbin.rec VAR.REC		
CANAL-FILE	MERSRT.CBL	MERSRT.CBL	No	
CNV-FILE	ASC-APAN.CBL ASC-PROM.CBL	ASC-APAN.CBL ASC-PROM.CBL		
COMM-FILE	EMP-TIM.CBL **TIM.SEL** YPOK-TIM.CBL	EMP-TIM.CBL **TCOMREC.CBL** YPOK-TIM.CBL	Select & Define	tcom.dat
COUNT-FILE	**AGO.SEL** EMP-TIM.CBL **TIM.SEL** YPOK-TIM.CBL	EMP-TIM.CBL **TCOUNTREC.CBL** YPOK-TIM.CBL	Select & Define	agcnt.dat [AGO.SEL] count.dat [TIM.SEL]

COUNTXFILE	V3-AGO.CBL V3-TIM.CBL	V3-AGO.CBL V3-TIM.CBL		
DB-FILE	DB.SEL	DB.REC	No	db.dat
DED-FILE	DED.SEL	DED.REC	No	ded.dat
DELTEKD-FILE	EMP-TIM.CBL	EMP-TIM.CBL	No	
DOC-FILE	DOC.SEL	DOC.REC	No	
EKPOLHS-FILE	EMP-TIM.CBL	EMP-TIM.CBL	No	
EKT-FILE	EKTEIS.CBL EKTEPIL.CBL PRTEIS.CBL PRTEIS.OLD **PRTSEL.CBL** PRTSEL.OLD	**recprn.wor** EKTREC.CBL		prt.dat *
ENTYPA	**entypo.cbl** entypo2.cbl title.cbl titleden.cbl titlekar.cbl titlent.cbl titlepka.cbl	**entypo.cbl** entypo2.cbl title.cbl titleden.cbl titlekar.cbl titlent.cbl titlepka.cbl	Select	ent.dat *
EPAG-FILE	EMP-PEL.CBL FEANEW.CBL YPOK-PEL.CBL	EMP-PEL.CBL EPAGREC.CBL FEANEW.CBL YPOK-PEL.CBL		peljob.dat
EPAG1-FLE	EMP-PRO.CBL YPOK-PRO.CBL	EMP-PRO.CBL EPAGREC1.CBL YPOK-PRO.CBL		prjob.dat
EPIT-FILE	**EPI.SEL** **EPI1.SEL** EPITDEL2.CBL EPITKEY.CBL MAKEPIT.CBL MAKEPIT1.CBL	EPIMASTER.CBL **EPIT1REC.CBL** **EPITREC.CBL**		epit.dat [EPI.SEL] epit1.dat [EPI1.SEL]
EPITXFILE	V3-EPGR.CBL V3-EPGR1.CBL	V3-EPGR.CBL V3-EPGR1.CBL		
EPIT1-FILE	EPESYN.CBL	EPESYN.CBL		epit1.dat
ERROR-FILE	ICL-CHK.CBL ICL-CHK.STD ICL-CHR.CBL	ICL-CHK.CBL ICL-CHK.STD ICL-CHR.CBL		
ETER	entypo.cbl entypo2.cbl title.cbl titledet.cbl titlekar.cbl titlepka.cbl **titleter.cbl**	entypo.cbl entypo2.cbl title.cbl titledet.cbl titlekar.cbl titlepka.cbl **titleter.cbl**	No	eter.dat
FANT	log.sel log1.sel	log.rec log1.rec		
FLT-FILE	FLT.SEL	FLT.REC		filter.dat
FLTHEAD-FILE	FLT.SEL	FLT.REC		flthead.dat
FORMPRT-FILE	prtlist.sel	prtlist.rec		
GRAM-FILE	**GRA.SEL** GRA1.SEL MAKGRAM.CBL MAKGRAM1.CBL MAKGRX.CBL	GRAM1REC.CBL GRAMPLHR.CBL **GRAMREC.CBL** MAKGRX.CBL	Define	gram.dat
GRAMXFILE	V3-EPGR.CBL V3-EPGR1.CBL	V3-EPGR.CBL V3-EPGR1.CBL		
GRCNT-FILE	**GRA.SEL**	**GRA.REC** GRCNTREC.CBL		grcount1.dat

GRCNT1-FILE	GR1CNT.CBL	GR1CNT.CBL		grcount1.dat
	GRA1.SEL	**GRA1.REC**		
		GRCNT1REC.CBL		
GRL-FILE	GRL.SEL	GRL.REC	No	grlink.dat
HAPANAL-FILE	makapseq.cbl	makapseq.cbl	No	
HELP-FILE	AP2HMER.CBL	AP2HMER.CBL	Select & Define	aphelp.dat
	APHMEROL.CBL	**APHMEROL.CBL**		
	APKATEL.CBL	APKATEL.CBL		
	APKATEL1.CBL	APKATEL1.CBL		
HLP-FILE	APPOL3.CBL	APPOL3.CBL	Select & Define	hlp.dat
	APROL3.CBL	APROL3.CBL		
	APSTAT.CBL	APSTAT.CBL		
	APSTAT1.CBL	APSTAT1.CBL		
	PLPOLREP.CBL	PLPOLREP.CBL		
	PLSTAT.CBL	PLSTAT.CBL		
	PRSTAT.CBL	PRSTAT.CBL		
	PRSTAT.GEN	PRSTAT.GEN		
	TMP.SEL	TMP.SEL		
HOSP-FILE	HOSP.SEL	HOSP.REC	No	hosp.dat
HOSPXFILE	V3-APO.CBL	V3-APO.CBL	No	
HOTEL-FILE	HOTEL.SEL	HOTEL.REC	No	hotel.dat
HOTELXFILE	V3-APO.CBL	V3-APO.CBL	No	
HPL-FILE	HPL.SEL	HPL.REC	No	hpl.dat
IDX-FILE	ICL-IN.CBL	ICL-IN.CBL	No	ts[date].idx
	ICLREAD.CBL	ICLREAD.CBL		
	SALES-ASC.CBL	SALES-ASC.CBL		
	SALES-ASC.STD	SALES-ASC.STD		
	SALES-ASC1.CBL	SALES-ASC1.CBL		
INV-FILE	AGO.SEL	TINVREC.CBL	No	agparmtr.dat [AGO.SEL]
	TIM.SEL			parametr.dat [TIM.SEL]
IOSP-FILE	IOSP.SEL	IOSP.REC	No	
K1-FILE	EMP-APO.CBL	APK1REC.CBL		
	MAKOIKO.CBL	EMP-APO.CBL		
	OM-OM.CBL	MAKOIKO.CBL		
	YPOK-APO.CBL	OM-OM.CBL		
		YPOK-APO.CBL		
K2-FILE	EMP-APO.CBL	EMP-APO.CBL		
	K2NEW.CBL	K2NEW.CBL		
	MAKOIKO.CBL	MAKOIKO.CBL		
	YPOK-APO.CBL	YPOK-APO.CBL		
KODE-FILE	**APO.SEL**	**APKODE.REC**	No	kode.dat
	APO1.SEL	APKODEREC.CBL		
	APOHO.SEL	KODE.REC		
	KODE.SEL			
KOS-FILE	**APO.SEL**	**APKOS.REC**	No	apkos.dat
	APO1.SEL	APKOSREC.CBL		
	APOHO.SEL			
LGCNT	log.sel	log.rec	No	
LGDATE	log.sel	log.rec		
	log1.sel	log1.rec		
LGEMPO	LG-PL.CBL	LG-PL.CBL		
	log1.sel	log1.rec		
LGINFO	log.sel	log.rec		
	log1.sel	log1.rec		
	NEAXR.STD	NEAXR.STD		
LGKATHG	log.sel	log.rec		
	log1.sel	log1.rec		
LGKFPA	log.sel	log.rec		
	log1.sel	log1.rec		
LGKKEID	log1.sel	log1.rec	No	l??kkp??.dat
LGKKK	log.sel	log.rec	No	

LGLINK-FILE	AGO.SEL **TIM.SEL**	LINKREC.CBL	No	lgaglink.dat
LGMAST	LG-PL.CBL log.sel **log1.sel**	LG-PL.CBL log.rec **log1.rec**	No	l??mas??.dat
LGMASTN	log.sel log1.sel	log.rec log1.rec		
LGPARFPA	log.sel log1.sel	log.rec log1.rec		
LGPFPA	log.sel log1.sel	log.rec log1.rec		
LGTADATE	log.sel	log.rec		
LGTAM	log.sel	log.rec		
LINK-FILE	ASC-LGAN.CBL ASC-PEL1.CBL ASC-PROM1.CBL LEM.SEL	ASC-LGAN.CBL ASC-PEL1.CBL ASC-PROM1.CBL LEM.REC	Select & Define	polfile.dat
MAIN	MAIN.SEL	MAIN.REC	No	maineter.dat
MEG-FILE	YPOK-APO.CBL	YPOK-APO.CBL	No	
MENCPL-FILE	MENCPL.SEL	MENCPL.REC	No	????????.???
MENDOC-FILE	MENDOC.SEL	MENDOC.REC	No	
MENMDF-FILE	MENCPL.SEL	MENCPL.REC	No	
MENSPT-FILE	MENSPT.SEL	MENSPT.REC	No	
MENU-FILE	MEN.SEL	MEN.REC	No	????????.???
MLPOLHS-FILE	MLTERM.SEL	MLTERM.REC	No	mlpolhs.dat
MLPOLHSXFILE	V3-AGO.CBL	V3-AGO.CBL	No	
MLTERM-FILE	MLTERM.SEL	MLTERM.REC	No	
MNFYI-FILE	MNFYI.SEL	MNFYI.REC	No	mnfyi.dat
MNFYIMAS-FILE	MNFYIMAS.SEL	MNFYIMAS.REC	No	mnfyimas.dat
MNREFR-FILE	MNREFR.SEL	MNREFR.REC	No	mnrefr.dat
MNUSR-FILE	MNUSR.SEL	MNUSR.REC	No	mnusr.dat
MNUSRPER-FILE	MNUSRPER.SEL	MNUSRPER.REC	No	mnusrper.dat
MON-FILE	**APO.SEL** APO1.SEL APOHO.SEL EMP-APO.CBL MAKOIKO.CBL YPOK-APO.CBL	**APMON.REC** APMONREC.CBL EMP-APO.CBL MAKOIKO.CBL YPOK-APO.CBL	Select & Define	apmon.dat
MORFH-APO	MINIE-APO.CBL NEOS-APO.SEL makap.cbl	MINIE-APO.CBL NEOS-APO.FIL makap.cbl		
MORFH-PEL	MINIE-PEL.CBL NEOS-PEL.SEL	MINIE-PEL.CBL NEOS-PEL.FIL		
MORFH-PRO	NEOS-PRO.SEL	NEOS-PRO.FIL	No	
MSANAL-FILE	**MSMAKAN.CBL** MSSEQ.CBL	**MSMAKAN.CBL** MSSEQ.CBL	Select & Define	msanal.dat msanal.seq
NEW-FILE	NEWANAL.CBL	NEWANAL.CBL	No	
NOR-MENU-FILE	nor-menu.sel	nor-menu.rec	No	
NOR-PELAT-FILE	nor-pelat.sel	nor-pelat.rec	No	
NORFILT-FILE	normark.cbl	normark.cbl	No	
OANAL-FILE	PLANSPT.CBL	PLANSPT.CBL	No	
OFFH-FILE	OFFH.SEL	OFFH.REC	No	offh.dat
OFFI-FILE	OFFI.SEL	OFFI.REC	No	offi.dat
OLD-FILE	APO-DESC.CBL APO-TR.CBL APOMA-TR.CBL APOMA2TR.CBL	APO-DESC.CBL APO-TR.CBL APOMA-TR.CBL APOMA2TR.CBL		
OLDPAR-FILE	MAKAP0-1.CBL	MAKAP0-1.CBL	No	
ORD-FILE	TIM.SEL	ORDER.REC	No	order.dat
ORDXFILE	V3-TIM.CBL	V3-TIM.CBL	No	
ORPH-FILE	ORPH.SEL	ORPH.REC	No	orph.dat
ORPI-FILE	ORPI.SEL	ORPI.REC	No	orpi.dat

PAR-FILE	TIM.SEL	PARAGREC.CBL	No	paragel.dat
	MAKPAR0-1.CBL			
PARXFILE	V3-TIM.CBL	V3-TIM.CBL		
PARCNT-FILE	TIM.SEL	PARCNTREC.CBL	No	parcnt.dat
PARM-FILE	PARM.SEL	PARM.REC	No	parm.dat
PELAT-FILE	ASC-PEL.CBL	ASC-PEL.CBL	Select & Define	pelat.dat
	ASC-PEL1.CBL	ASC-PEL3.CBL		
	ASC-PEL3.CBL	ASC-PEL4.CBL		
	ASC-PEL4.CBL	EMP-PEL.CBL		
	EMP-PEL.CBL	hlp1		
	MAK-PEL.CBL	MAK-PEL.CBL		
	MAK-PR.CBL	MAK-PR.CBL		
	MAKPL-CS.CBL	MAKPLPR.CBL		
	MAKPLCD.CBL	PELAT.REC		
	MAKPLPR.CBL	**PELMAST.REC**		
	PEL.SEL	PELNEW.CBL		
	PELAT.SEL	PELSEQ.CBL		
	PELNEW.CBL	PL-AFM.CBL		
	PELSEQ.CBL	PL-PL.CBL		
	PL-AFM.CBL	PLPREAD.CBL		
	PL-PL.CBL	YPOK-PEL.CBL		
	PLOPMR.CBL			
	PLPREAD.CBL			
	RELPEL.CBL			
	YPOK-PEL.CBL			
PELATXFILE	V3-PEL.CBL	V3-PEL.CBL	No	
PELF-FILE	FEANEW.CBL	FEANEW.CBL	No	pelf.dat
	PEL.SEL	**PELFREC.CBL**		
	PEL1.SEL			
	PLOPMR.CBL			
PELFXFILE	V3-PEL.CBL	V3-PEL.CBL		
PELNME-FILE	**PEL.SEL**	**PELNMEREC.CBL**	No	pelnme.dat
	PEL1.SEL			
	PLOPMR.CBL			
PER-FILE	EMP-PEL.CBL	EMP-PEL.CBL	Define	perioxh.dat
	FEANEW.CBL	FEANEW.CBL		perioxh.data
	YPOK-PEL.CBL	YPOK-PEL.CBL		
PER1-FILE	EMP-PRO.CBL	EMP-PRO.CBL	Define	perioxh1.dat
	YPOK-PRO.CBL	YPOK-PRO.CBL		perioxh1.data
PFORT-FILE	AGFORTEID.CBL	AGFORTEID.CBL	Select & Define	pf.[time]
	AGFORTSYN.CBL	AGFORTSYN.CBL		pfort.data
	ORDEID.CBL	ORDEID.CBL		
	ORDEIDDT.CBL	ORDEIDDT.CBL		
	PFORTEID.CBL	PFORTEID.CBL		
	PFORTEID2.CBL	PFORTEID2.CBL		
	PFORTSYN.CBL	PFORTSYN.CBL		
	TDPEKT2.CBL	TDPEKT2.CBL		
	TFORTEID.CBL	TFORTEID.CBL		
	TFORTEID2.CBL	TFORTEID2.CBL		
PLFAIM-FILE	**PEL.SEL**	**PLFAIMREC.CBL**	No	plfaim.dat
	PEL1.SEL			
	PLOPMR.CBL			
PLHELP-FILE	MAKPLHELP.CBL	MAKPLHELP.CBL		
PLYP-FILE	EMP-PEL.CBL	EMP-PEL.CBL	Define	pelypom.dat
	PEL.SEL	**PLYPREC.CBL**		
	PEL1.SEL			
	PLOPMR.CBL			
POL-FILE	EMP-PEL.CBL	EMP-PEL.CBL	No	polhth.dat
	FEANEW.CBL	FEANEW.CBL		polhth.data
	YPOK-PEL.CBL	POLREC.CBL		wpolhth.dat
		YPOK-PEL.CBL		

POLD-FILE	EMP-PEL.CBL EMP-PRO.CBL **PEL.SEL** PEL1.SEL PLOPMR.CBL **PR3.SEL** PROPMR.CBL YPOK-PEL.CBL YPOK-PRO.CBL	EMP-PEL.CBL EMP-PRO.CBL **TPOLDREC.CBL** YPOK-PEL.CBL YPOK-APO.CBL	Select & Define	agdate.dat [PR3.SEL] poldate.dat [PEL.SEL]
POLDXFILE	V3-PEL.CBL V3-PRO.CBL	V3-PEL.CBL V3-PRO.CBL		
POLHS-FILE	**AGO.SEL** EMP-TIM.CBL **TIM.SEL**	EMP-TIM.CBL **TPOLHSREC.CBL**	Define	agdp.dat [AGO.SEL] polhs.dat [TIM.SEL]
POLHSXFILE	V3-AGO.CBL V3-TIM.CBL	V3-AGO.CBL V3-TIM.CBL		
POS-FILE	POS.SEL	POS.REC		
PRANAL-FILE	XPERTDB.CBL	XPERTDB.CBL		
PRF-FILE	**PR6.SEL** PROPMR.CBL	**PRFREC.CBL**	No	prf.dat
PRFXFILE	V3-PRO.CBL	V3-PRO.CBL	No	
PRINT-FILE	AFM.CBL ag-ekt.cbl AGFORTEID.CBL AGFORTSYN.CBL APCHK.CBL PISOZ1.CBL APK1SYG.CBL APSYG.CBL entypo.cbl entypo2.cbl EPITSYN.CBL GRAMDATE.CBL GRAMTRAP.CBL lg-ekt.cbl MAKPLEVR.CBL MAKPR8.CBL PFORTEID.CBL PFORTEID2.CBL PFORTSYN.CBL PLCHK.CBL PLMHNP.CBL PLPOLREP.CBL PLYPFAST.CBL PRCHK.CBL PRT.SEL prt132.sel prt232.sel PRYPFAST.CBL SEQREAD.CBL SKT.CBL STATAPS.CBL TDMET.CBL TGMET.CBL	AFM.CBL ag-ekt.cbl AGFORTEID.CBL AGFORTSYN.CBL APCHK.CBL PISOZ1.CBL APK1SYG.CBL APSYG.CBL entypo.cbl entypo2.cbl EPITSYN.CBL GRAMDATE.CBL GRAMTRAP.CBL lg-ekt.cbl MAKPLEVR.CBL MAKPR8.CBL PFORTEID.CBL PFORTEID2.CBL PFORTSYN.CBL PLCHK.CBL PLMHNP.CBL PLPOLREP.CBL PLYPFAST.CBL PRCHK.CBL PRT.SEL prt132.sel prt232.sel PRYPFAST.CBL SEQREAD.CBL SKT.CBL STATAPS.CBL TDMET.CBL TGMET.CBL		
PRINTFILE	rhandle.cbl fhandle.cbl rhandle.old	rhandle.cbl fhandle.cbl rhandle.old		

PROM-FILE	ASC-PROM.CBL ASC-PROM1.CBL ASC-PROM3.CBL EMP-PRO.CBL EPMET.CBL MAKPLPR.CBL MAKPR-CS.CBL MAKPROM.CBL PLPREAD.CBL PR-AFM.CBL PR-PR.CBL **PR1.SEL** PR11.SEL pro.sel PROPMR.CBL RELPRO.CBL YPOK-PRO.CBL	EMP-PRO.CBL GRPROMREC.CBL MAKPLPR.CBL MAKPROM.CBL PLPREAD.CBL PR-AFM.CBL PR-PR.CBL PR1.REC **PRMASTREC.CBL** pro.rec YPOK-APO.CBL	Select & Define	prom.dat
PROMX-FILE	PROMX.SEL	PROMX.REC	No	
PROMXFILE	V3-PRO.CBL	V3-PRO.CBL	No	
PRORD-FILE	PRORD.SEL	PRORD.REC	No	prord.dat
PRORDCNT-FILE	PRORD.SEL	PRORD.REC	No	prordcnt.dat
PRTIM-FILE	EMP-APO.CBL MAKOIKO.CBL	EMP-APO.CBL MAKOIKO.CBL		
PRTIND-FILE	CANCPRT.CBL	CANCPRT.CBL	No	prtind.dat
PRTLIST-FILE	CANCPRT.CBL	CANCPRT.CBL	No	prtlist.dat
PRUPD-FILE	PRUPD.SEL	PRUPD.REC	No	prupd.trs
PRUPDXFILE	V5MAKUPD.CBL	V5MAKUPD.CBL	No	
REL-FILE	rel300.sel	rel300.rec	No	
SANAL-FILE	MERSRT.CBL PLANSRT.CBL	MERSRT.CBL		

SEQ-FILE	AFM.CBL	AFM.CBL		
	AGFLCHK.CBL	AGFLCHK.CBL		
	APFLCHK.CBL	APFLCHK.CBL		
	ASC-AP.CBL	ASC-AP.CBL		
	ASC-AP0.CBL	ASC-AP0.CBL		
	ASC-AP1.CBL	ASC-AP1.CBL		
	ASC-AP2.CBL	ASC-AP2.CBL		
	ASC-APAN.CBL	ASC-APAN.CBL		
	ASC-LGAN.CBL	ASC-LGAN.CBL		
	ASC-PEL.CBL	ASC-PEL.CBL		
	ASC-PEL1.CBL	ASC-PEL1.CBL		
	ASC-PEL3.CBL	ASC-PEL3.CBL		
	ASC-PEL4.CBL	ASC-PEL4.CBL		
	ASC-PROM.CBL	ASC-PROM.CBL		
	ASC-PROM1.CBL	ASC-PROM1.CBL		
	ASC-PROM3.CBL	ASC-PROM3.CBL		
	ASC-TEAM.CBL	ASC-TEAM.CBL		
	ASC-TEAM2.CBL	ASC-TEAM2.CBL		
	ASC.CBL	ASC.CBL		
	GNFLCHK.CBL	GNFLCHK.CBL		
	GP.CBL	GP.CBL		
	ICL-ALL.CBL	ICL-ALL.CBL		
	ICL-CHK.CBL	ICL-CHK.CBL		
	ICL-CHK.STD	ICL-CHK.STD		
	ICL-CHR.CBL	ICL-CHR.CBL		
	ICL-IN.CBL	ICL-IN.CBL		
	ICL-OUT.CBL	ICL-OUT.CBL		
	ICLREAD.CBL	icl-seq.rec		
	ICLSALES.CBL	ICLREAD.CBL		
	MYMAIN.CBL	ICLSALES.CBL		
	PL-AFM.CBL	MYMAIN.CBL		
	PLFLCHK.CBL	PL-AFM.CBL		
	PR-AFM.CBL	PLFLCHK.CBL		
	PRFLCHK.CBL	PR-AFM.CBL		
	PROD-ASC.CBL	PRFLCHK.CBL		
	PROD-ASC.STD	PROD-ASC.CBL		
	PROD-ASC1.CBL	PROD-ASC.STD		
	SALES-ASC.CBL	PROD-ASC1.CBL		
	SALES-ASC.STD	SALES-ASC.CBL		
	SALES-ASC1.CBL	SALES-ASC.STD		
	SEQ-OTH.CBL	SALES-ASC1.CBL		
	TMFLCHK.CBL	SEQ-OTH.CBL		
	V3-XPERT.CBL	TMFLCHK.CBL		
	V5MAKUPD.CBL	V3-XPERT.CBL		
	_IL01.CBL	V5MAKUPD.CBL		
	_IL01CHK.CBL	_IL01.CBL		
	_IL01TEST.CBL	_IL01CHK.CBL		
		_IL01TEST.CBL		
SEQ-MIS-FILE	SALES-ASC.CBL	SALES-ASC.CBL	No	ts[date].txt
	SALES-ASC.STD	SALES-ASC.STD		
	SALES-ASC1.CBL	SALES-ASC1.CBL		
SMANAL-FILE	MERSRT.CBL			
SMOANAL-FILE	MERSRT.CBL	MERSRT.CBL	No	smoanal.[time]
SNF-FILE	SNF.SEL	SNF.REC	No	
SNFI-FILE	SNFI.SEL	SNFI.REC	No	

SORT-FILE	AG2HMER.CBL			
	AGHMEROL.CBL			
	APK1EIDA.CBL			
	APLGEIDOS.CBL			
	APLGEVR.CBL			
	APPRICE.CBL			
	EPESYN.CBL			
	EPITS.CBL			
	EPITS1.CBL			
	MAKPLEVR.CBL			
	PLNS.CBL			
	PRMERGE.CBL			
	SORT.SEL			
	TINVLIST.CBL			
	TPOL2HMER.CBL			
	TPOL3HMER.CBL			
	TPOLHMEROL.CBL			
SORTED-FILE	GRTBEX2.CBL	GRTBEX2.CBL		
	PELINK.CBL	PELINK.CBL		
	PLANAL1.CBL	PLANAL1.CBL		
	PLANAL2.CBL	PLANAL2.CBL		
	PLANAL3.CBL	PLANAL3.CBL		
	PLTREIS.CBL	PLTREIS.CBL		
SORTRAN	log.sel	log.sel		
	log1.sel	log1.sel		
SPELAT-FILE	PELIDX.CBL	PELIDX.CBL	No	pelat.seq
	PELSEQ.CBL	PELSEQ.CBL		
SYNT-FILE	APOHO.SEL	**APSYN.REC**	Select & Define	apsynt.dat
	APSYN.SEL	APSYNREC.CBL		
	ASC-AP.CBL			
SYNT-KOS-FILE	**APO.SEL**	**APKO2.REC**	Define	apkost2.dat
	APO1.SEL	APKO2REC.CBL		
	APOHO.SEL			
SYSCFG-FILE	SYSENV.CBL	SYSENV.CBL	No	sysenv.cfg
SYSENV-FILE	SYSENV.CBL	SYSENV.CBL	No	sysenv.dat
	SYSENV.SEL	SYSENV.REC		
SYSOUT-FILE	SYSENV.CBL	SYSENV.CBL	No	
TAMDEL-FILE	TAMDEL.SEL	TAMDEL.REC	No	tamdel.dat
TAPOM-FILE	**APO.SEL**	**APOMT.REC**	No	tapom.dat
	APO1.SEL	TAPOMREC.CBL		
	APOHO.SEL			
	hlp.sel			
TAPQNTY-FILE	TAPQNTY.SEL	TAPQNTY.REC	No	tapqnty
TDPR-FILE	AGO.SEL	TDPREC.CBL	No	tdpr.???
TDPRC-FILE	AGO.SEL	TDPCREC.CBL	No	tdprc.???
TDPRXFILE	V3-AGO.CBL	V3-AGO.CBL	No	
TEMP-FILE	APANFL2.CBL	APANFL2.CBL		
	APPEL3.CBL	APPEL3.CBL		
	APPROM3.CBL	APPROM3.CBL		
	PELTEAM.CBL	PELTEAM.CBL		
	PLPRANAL.CBL	PLPRANAL.CBL		
	PLPRYP.CBL	PLPRYP.CBL		
	PRPLYP.CBL	PRPLYP.CBL		
	TEMP.SEL	TEMP.REC		
	TPOL3HMER.CBL	TPOL3HMER.CBL		
THELP-FILE	AGO.SEL	THELPREC.CBL	No	thelp.dat
TIM-FILE	EMP-TIM.CBL	EMP-TIM.CBL	Define	tim???
	TIM.SEL	**TIMREC.CBL**		
	YPOK-TIM.CBL	YPOK-TIM.CBL		

TIMAG-FILE	**TAG.SEL**	**TAG.REC**	Define	timag.dat
		TAG1.REC		
		TAG2.REC		
		TAG3.REC		
TIMCNT-FILE	**TIM.SEL**	EMP-TIM.CBL	No	timcnt.???
		TIMCNTREC.CBL		
		YPOK-TIM.CBL		
TIMXFILE	V3-TIM.CBL	V3-TIM.CBL	No	
TITLOS-FILE	V3-TIM.CBL	V3-TIM.CBL	Define	tit.data
	TITLOP.CBL	TITLOP.CBL		
	entypo.cbl	entypo.cbl		
	entypo2.cbl	entypo2.cbl		
	title.cbl	**title.cbl**		
	titlekar.cbl	titlekar.cbl		
	titlepka.cbl	titlepka.cbl		
TMET-FILE	**AGO.SEL**	**TMETREC.CBL**	No	tmet.dat
	TIM.SEL			ttmet.dat
TMP-FILE	APANALY.CBL	APANALY.CBL	Select & Define	
	APDGEN1.CBL	APDGEN1.CBL		
	APGCNLTA.CBL	APGCNLTA.CBL		
	APGDIAF1.CBL	APGDIAF1.CBL		
	APGLIST2.CBL	APGLIST2.CBL		
	APGLIST2.RST	APGLIST2.RST		
	APGLIST2.STD	APGLIST2.STD		
	APGTACNL.CBL	APGTACNL.CBL		
	APGTAKT.CBL	APGTAKT.CBL		
	APGTAKT3.CBL	APGTAKT3.CBL		
	APMHNST.CBL	PLCRDB.CBL		
	APMHNSTF.CBL	PRMERGE.CBL		
	APPL5A.CBL	TDPEKT2.CBL		
	APPL5B.CBL	TEMPA.REC		
	PLCRDB.CBL	TEMPB.REC		
	PRMERGE.CBL			
	TDPEKT2.CBL			
TMPI-FILE	PRORDERS.CBL	PRORDERS.CBL	No	
TMPPER-FILE	MNUSRPER.SEL	MNUSRPER.REC	No	TMPPER.DAT
TOM-FILE	**AGO.SEL**	EMP-AGO.CBL	Define	tom.dat
	EMP-AGO.CBL	EMP-TIM.CBL		agtom.dat
	EMP-TIM.CBL	**TOMREC.CBL**		
	TIM.SEL	YPOK-TIM.CBL		
	YPOK-TIM.CBL			
TOTAL-FILE	TDPEKT.CBL	TDPEKT.CBL		
	TDPEKT1.CBL	TDPEKT1.CBL		
TPLOM-FILE	**PEL.SEL**	**TPLOMREC.CBL**	No	tplom.dat
	PEL1.SEL			
	PLOPMR.CBL			
	hlp.sel			
TPROM-FILE	**PR5.SEL**	**TPROMREC.CBL**	No	tprom.dat
	PROPMR.CBL			
TRA-FILE	**TRA.SEL**	TRAPREC.CBL	Define	trapeza.dat
		TRAREC.CBL		
TRANCC	log1.sel	log1.rec	No	
TRANS-FILE	EPITOPTRAN.CBL	EPITTRAN.CBL	Select & Define	epittran.data
	GRAM1OPTR.CBL	EPITTRANSR.CBL		gramtran.data
	GRAMOPTRAN.CBL	GRAM1TREC.CBL		gramtran1.data
		GRAMTRREC.CBL		
TRWH-FILE	TRWH.SEL	TRWH.REC	No	trwh.dat
TRWHXFILE	V3-APO.CBL	V3-APO.CBL	No	
TXTFILE	rhandle.cbl	rhandle.cbl	No	
	rhandle.old	rhandle.old		
	fhandle.cbl	fhandle.cbl		

TYPOM-FILE	**AGO.SEL** EMP-AGO.CBL EMP-TIM.CBL TIM.SEL YPOK-TIM.CBL	EMP-AGO.CBL EMP-TIM.CBL **TYPOMREC.CBL** YPOK-TIM.CBL	No	agtypom.dat typom.dat
VAR-FILE	VAR.SEL formvar.sel	VAR.SEL formvar.sel	Select & Define	
WAGD-FILE	EMP-AGO.CBL	EMP-AGO.CBL	No	agdelt.dat
WAGT-FILE	EMP-AGO.CBL	EMP-AGO.CBL	No	agtim.dat
WANAL-FILE	EMP-PEL.CBL EMP-PRO.CBL MAKPLAN3.CBL MAKPRAN3.CBL mak_plan.cbl MAK_PLAN.CBL mak_pln1.cbl mak_pran.cbl MAK_PRAN.CBL mak_prn1.cbl MINIE-PEL.CBL REC.SEL REC1.SEL YPOK-PEL.CBL YPOK-PRO.CBL YPOKGET.CBL	EMP-PEL.CBL EMP-PRO.CBL MAKPLAN3.CBL MAKPRAN3.CBL mak_plan.cbl MAK_PLAN.CBL mak_pln1.cbl mak_pran.cbl MAK_PRAN.CBL mak_prn1.cbl MINIE-PEL.CBL REC.REC REC1.REC TRGETREC.CBL YPOK-PEL.CBL YPOK-PRO.CBL	Select & Define	pelanal.dat
WAPANAL-FILE	APTRANS.CBL AP_CHK.CBL EMP-APO.CBL MAK+APAN.CBL MAK-APN.CBL MAKAPAN2.CBL makapseq.cbl MAKOIKO.CBL mak_apan.cbl MAK_APAN.CBL REC.SEL SEQREAD.CBL TRAN.SEL YPOKGET.CBL	APTRANS.CBL APTR.REC APTRREC.CBL CR-L.CBL CS-L.CBL MAK+APAN.CBL MAK-APN.CBL MAKAPAN2.CBL makapseq.cbl MAKOIKO.CBL mak_apan.cbl MAK_APAN.CBL EMP-APO.CBL SEQREAD.CBL YPOKGET.CBL	Select & Define	apanal.dat
WAPD-FILE	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	No	apdate.dat
WAPF-FILE	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	No	apf.dat
WAPFPA-FILE	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	No	apfpa.dat

WAPO-FILE	AP-AP.CBL	AP-AP.CBL	Select & Define	apo.dat
	AP-GET.CBL	AP-GET.CBL		
	APNEW.CBL	APNEW.CBL		
	APTRANS.CBL	APTRANS.CBL		
	AP_CHK.CBL	AP_CHK.CBL		
	ASC-AP1.CBL	ASC-AP1.CBL		
	EMP-APO.CBL	EMP-APO.CBL		
	mak-ap.cbl	mak-ap.cbl		
	MAK-AP.CBL	MAK-AP.CBL		
	MAKAP-CS.CBL	MAKAP-CS.CBL		
	MAKAP0-1.CBL	MAKAP0-1.CBL		
	MAKAP1-2.CBL	MAKAP1-2.CBL		
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	RELAPO.CBL	RELAPO.CBL		
	TRAN.SEL	TRAN.REC		
	WAPO.SEL	WAPO.REC		
	YPOK-APO.CBL	YPOK-APO.CBL		
WAPO1-FILE	APM2JOIN.CBL	APM2JOIN.CBL	Define	apo1.dat
	EMP-APO.CBL	EMP-APO.CBL		
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	WAPO.SEL	WAPO.REC		
	YPOK-APO.CBL	YPOK-APO.CBL		
	WAPO1.SEL	WAPOM1.REC		
WAPOMA-FILE	APOMANEW.CBL	APOMANEW.CBL	Define	apoma.dat
	TRAN.SEL	TRAN.SEL		wapoma.dat
	WAPO.SEL	WAPO.REC		
WAPP-FILE	EMP-APO.CBL	EMP-APO.CBL	No	app.dat
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	YPOK-APO.CBL	YPOK-APO.CBL		
WAPTEAM-FILE	EMP-APO.CBL	EMP-APO.CBL	Select	apteam.dat
	MAKAPT0-1.CBL	MAKAPT0-1.CBL		apteam.seq
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	TRAN.SEL	TRAN.REC		
	YPOK-APO.CBL	YPOK-APO.CBL		
WAPTEAM1-FILE	EMP-APO.CBL	EMP-APO.CBL		No affects!!! - No file
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	YPOK-APO.CBL	YPOK-APO.CBL		
WAPTIMCH-FILE	EMP-APO.CBL	EMP-APO.CBL		No affects!!! - No file
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	YPOK-APO.CBL	YPOK-APO.CBL		
WAPTM-FILE	AITEXAG.CBL	AITEXAG.CBL	Select & Define	aptm.dat
	AITEXALL.CBL	AITEXALL.CBL		
	APAIT2EX.CBL	APAIT2EX.CBL		
	APDGEN.CBL	APDGEN.CBL		
	APTMJOIN.CBL	APTMJOIN.CBL		
	EMP-APO.CBL	EMP-APO.CBL		
	HTLEXAG.CBL	HTLEXAG.CBL		
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	YPOK-APO.CBL	YPOK-APO.CBL		

WARTHRO-FILE	AGVIEW1.CBL	AGVIEW1.CBL	Select & Define	agart.dat
	EMP-PEL.CBL	EMP-PEL.CBL		arthro.dat
	EMP-PRO.CBL	EMP-PRO.CBL		
	MAKagt.CBL	MAKagt.CBL		
	MAKart.CBL	MAKart.CBL		
	MINIE-PEL.CBL	MINIE-PEL.CBL		
	REC.SEL	REC.REC		
	REC1.SEL	REC1.REC		
	TVIEW1.CBL	TRGETREC.CBL		
	YPOK-PEL.CBL	TVIEW1.CBL		
	YPOK-PRO.CBL	YPOK-PEL.CBL		
	YPOKGET.CBL	YPOK-PRO.CBL		
WCOMM-FILE	EMP-TIM.CBL	EMP-TIM.CBL	No	tcom.dat
	YPOK-TIM.CBL	YPOK-TIM.CBL		
WCOUNT-FILE	EMP-TIM.CBL	EMP-TIM.CBL	No	count.dat
	YPOK-TIM.CBL	YPOK-TIM.CBL		
WGRAM-FILE	MAKGRX.CBL	MAKGRX.CBL	No	
WINSEL-FILE	WINSEL.SEL	WINSEL.REC	No	[time].tmp
	WINSEL.SEL@	WINSEL.REC@		
WINV-FILE	EMP-TIM.CBL	EMP-TIM.CBL	No	parametr.dat
	MAKINV.CBL	MAKINV.CBL		
	YPOK-TIM.CBL	YPOK-TIM.CBL		
WK1-FILE	APTRANS.CBL	APTR.REC	Select & Define	k1.seq [APTRANS.CBL]
	AP_CHK.CBL	APTRREC.CBL		wk1.data [K1NEW.CBL]
	K1NEW.CBL	K1NEW.CBL		
WK2-FILE	APTRANS.CBL	APTR.REC	Select & Define	k2.seq [APTRANS.CBL]
	AP_CHK.CBL	APTRREC.CBL		wk2.data [K2NEW.CBL]
	K2NEW.CBL	K2NEW.CBL		
WKODE-FILE	EMP-APO.CBL	EMP-APO.CBL	No	kode.dat
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	YPOK-APO.CBL	YPOK-APO.CBL		
WMON-FILE	EMP-APO.CBL	EMP-APO.CBL	No	apmon.dat
	MAKOIKO.CBL	MAKOIKO.CBL		
	MINIE-APO.CBL	MINIE-APO.CBL		
	YPOK-APO.CBL	YPOK-APO.CBL		
WMSANAL-FILE	MSMAKAN.CBL	MSMAKAN.CBL	No	wmsanal.dat
	MSSEQ.CBL	MSSEQ.CBL		
WPELAT-FILE	EMP-PEL.CBL	EMP-PEL.CBL	Select & Define	pelat.dat
	LG-PL.CBL	LG-PL.CBL		
	MAK-PEL.CBL	MAK-PEL.CBL		
	MAK-PR.CBL	MAK-PR.CBL		
	MAKPL-CS.CBL	MAKPL-CS.CBL		
	MAKPLCD.CBL	MAKPLCD.CBL		
	MINIE-PEL.CBL	MINIE-PEL.CBL		
	PELNEW.CBL	PELNEW.CBL		
	PL-AFM.CBL	PL-AFM.CBL		
	PL-PL.CBL	PL-PL.CBL		
	WPEL.SEL	WPEL.SEL		
	YPOK-PEL.CBL	YPOK-PEL.CBL		
WPELF-FILE	EMP-PEL.CBL	EMP-PEL.CBL	No	pelf.dat
	MINIE-PEL.CBL	MINIE-PEL.CBL		
	YPOK-PEL.CBL	YPOK-PEL.CBL		
WPLYP-FILE	EMP-PEL.CBL	EMP-PEL.CBL	No	pelypom.dat
	MINIE-PEL.CBL	MINIE-PEL.CBL		
	YPOK-PEL.CBL	YPOK-PEL.CBL		
WPOLD-FILE	EMP-PEL.CBL	EMP-PEL.CBL	No	poldate.dat
	EMP-PRO.CBL	EMP-PRO.CBL		agdate.dat
	MINIE-PEL.CBL	MINIE-PEL.CBL		
	YPOK-PEL.CBL	YPOK-PEL.CBL		
	YPOK-PRO.CBL	YPOK-PRO.CBL		

WPOLHS-FILE	EMP-TIM.CBL YPOK-TIM.CBL	EMP-TIM.CBL YPOK-TIM.CBL	No	polhs.dat
WPRF-FILE	EMP-PRO.CBL YPOK-PRO.CBL	EMP-PRO.CBL YPOK-PRO.CBL	No	prf.dat
WPROM-FILE	EMP-PRO.CBL MAKPR-CS.CBL MAKPROM.CBL PL-AFM.CBL PR-PR.CBL WPRO.SEL YPOK-PRO.CBL	EMP-PRO.CBL MAKPR-CS.CBL MAKPROM.CBL PL-AFM.CBL PR-PR.CBL WPRO.REC YPOK-PRO.CBL	Define	prom.dat
WTAPOM-FILE	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	EMP-APO.CBL MAKOIKO.CBL MINIE-APO.CBL YPOK-APO.CBL	No	tapom.dat
WTOM-FILE	EMP-AGO.CBL EMP-TIM.CBL YPOK-TIM.CBL	EMP-AGO.CBL EMP-TIM.CBL YPOK-TIM.CBL	No	tom.dat agtom.dat
WTYPOM-FILE	EMP-AGO.CBL EMP-TIM.CBL YPOK-TIM.CBL	EMP-AGO.CBL EMP-TIM.CBL YPOK-TIM.CBL	No	typom.dat agtypom.dat
WYPOM-FILE	APTRANS.CBL AP_CHK.CBL YPOMNEW.CBL	APTR.REC APTRREC.CBL YPOMNEW.CBL	Select & Define	apypom.seq wapypom.data
WYPOMK-FILE	APTRANS.CBL AP_CHK.CBL YPOMKNEW.CBL	APTR.REC APTRREC.CBL YPOMKNEW.CBL	Select & Define	apypomk.seq wypomk.data
YPOM-FILE	EMP-APO.CBL mak-apy.cbl MAKOIKO.CBL YP-YP.CBL YPOK-APO.CBL	APYPREC.CBL EMP-APO.CBL mak-apy.cbl MAKOIKO.CBL YP-YP.CBL YPOK-APO.CBL	Select & Define	apypom.dat apypom.data
YPOMK-FILE	YPOK-APO.CBL	YPOK-APO.CBL	No	apypomk.data

Appendix 18

The next figure presents the graph produced by the gain browser after executing the 'call tree' graphical query to the PRSTAT program. 53 software objects, which are additional libraries used at compilation time or external programs called at run time, as well as the way that they are related with each other are presented in the graph.

The next figure presents the graph produced by the gain browser after executing the 'affect tree' graphical query to the PRSTAT program. 61 software objects, which are additional libraries used at compilation time, external programs called at run time or datafiles affected at run time, as well as the way that they are related with each other are presented in the graph.