CodeRedo

Program ReEngineering Tool

Version: 1.2
Date: July, 2010

Harry M. Sneed

Harry.Sneed@anecon.com

Software Design und Beratung G.m.b.H.
Alserstraße 4 / Hof 1
A-1090 Wien

1. PURPOSE OF THE CODEREDO TOOLS 4

1.1. IMPROVING CODE READABILITY 5
1.2. REDUCING CODE COMPLEXITY 6
1.3. ENABLING CODE COMPREHENSION 6
1.4. REMOVING INCOMPATIBLE DATA TYPES 6
1.5. ENHANCING CODE FLEXIBILITY 7
1.6. ACHIEVING HIGHER PORTABILITY 7
1.7. REDUCING CONTROL FLOW COMPLEXITY 7
1.8. REFACTORING DEEPLY NESTED PROCEDURES 8

2. FUNCTIONS OF THE CODEREDO TOOLS 9

2.1. REFORMATING OF SOURCE CODE 9
2.2. REFINING THE CODE 10
2.3. RENAMING OF VARIABLES 10
2.4. REMOVING INCOMPATIBLE DATA TYPES 11
2.5. REPLACING HARD-WIRED DATA 12
2.6. RELOCATING I-O AND ACCESS OPERATIONS 12
2.7. RESTRUCTURING THE CODE 13
2.8. REFACTORING THE CODE 13

3. INPUTS TO THE CODEREDO TOOLS 15

3.1. ASSEMBLER SOURCE FILES 15
3.1.1 ASSEMBLER CSECTS 15
3.1.2 ASSEMBLER COPIES AND MACROS 16
3.2. PL/I SOURCE FILES 16
3.2.1 PLI EXTERNAL PROCEDURES 16
3.2.2 PLI INCLUDES 16
3.3. COBOL SOURCE FILES 16
3.3.1 COBOL EXTERNAL PROCEDURES 17
3.3.2 COBOL COPIES 17
3.4. C++ SOURCE FILES 17
3.4.1 C++ CLASSES AND PROCEDURES 17
3.4.2 C++ HEADER FILES 17
3.5 MACRO/FUNCTION TABLE 18
3.6 DATA NAME TABLES 19
3.7 PROCEDURAL COPIES AND INCLUDES 19
3.8 GUI PARAMETERS 19
3.8.2 CONTROL PARAMETERS 19
3.8.3 REENGINEERING OPTIONS 20

4 OUTPUTS FROM THE CODEREDO TOOLS 21

4.4 ASSEMBLER CSECTS & COPIES 21
4.5 PLI PROCEDURES & INCLUDES 22
4.6 COBOL PROGRAMS & COPIES 23

4.7 C++ CLASSES & HEADER FILES 24

5 CODEREDO USAGE 25

5.1 GENERAL SELECTION BAR 26
5.1.1 FILE OPERATIONS 26
5.1.2 ACTIONS 26
5.1.3 VIEWERS 27
5.1.4 LOGS 27
5.1.5 HELP 27
5.2 SPECIFIC SELECTION BAR 28
5.3 MAIN PANEL 28
5.3.1 LANGUAGE SELECTION 28
5.3.2 NAMES 28
5.3.3 OUTPUT DIRECTORY 29
5.3.4 REENGINEERING FUNCTION SELECTION LIST 29
5.3.5 SELECTING FILES 29
5.3.6 STARTING A JOB 30
5.3.7 DISPLAYING SELECTED FILES 30
5.4 EDITING MACRO PANEL 31
5.5 SCANNING FOR MACROS AND IO FUNCTIONS 32
5.6 RUNNING A REENGINEERING JOB 33
5.7 VIEWING THE REENGINEERING RESULTS 33

6 CODEREDO REENGINEERING SAMPLES 35

6.1 SAMPLE MACRO TABLE 35
6.2 SAMPLE FUNCTION TABLE 35
6.3 SAMPLE NAME FILE 36
6.4 SAMPLE BATCH PROCESS INTERFACE 38
6.5 SAMPLE OF A RENOVATED ASSEMBLER CSECT 38
6.6 SAMPLE OF A RENOVATED PLI PROCEDURE 45
6.7 SAMPLE OF A RENOVATED COBOL PROGRAM 61
6.7 SAMPLE OF A RENOVATED C++ CLASS 68

1. Purpose of the CodeRedo Tools

CodeRedo has a twofold purpose. On the one hand it is intended to support the maintenance engineer in renovating old code. On the other hand it is also intended to prepare old code for migration. Old code is understood here as procedural code written in a classical procedural language. For this reason CodeRedo is restricted to the four languages – Assembler, PL/I, COBOL and C/C++. The idea here is to improve the quality of the code without reimplementing it or before migrating it to another language. The traditional goal of software reengineering was to bring up the quality of the software in order to reduce the costs of maintenance and evolution. This is exactly the goal of CodeRedo. It is designed to make different, measurable improvements to the code which will lead to a higher code quality and a higher maintenance productivity. It can also be used to clean up code prior to migration in accordance with the motto "first renovate, then convert". It should be clear to all that clean, well structured and refactored code is easier to migrate than dirty, entangled code. Problematic code is difficult to maintain and error prone. Problematic code in COBOL or PL/I migrated to Java will only result in problematic Java code. Problematic code wrapped as web services will result in problematic web services. One is only moving the problems from one environment to another. Code problems cost time and effort and have to be solved sooner or later. The sooner they are solved the greater the savings. With the help of CodeRedo many such problems can be resolved in a quick and inexpensive manner.

The question which comes up here, is what types of problems can be resolved by code reengineering and, in particular, by automated code reengineering. There are many problems associated with old code, most of which have been there from the very beginning and others which have been introduced as a result of continual change. It would be impossible for a tool to address all of these problems, but some of them can be alleviated by means of a tool like CodeRedo. It goes without saying that, for the sake of readability, code should be broken up into no more than one statement per line and nested code should be indented. That makes the code easier to follow and to process. There are many dubious code constructs which render the code either incomprehensible or error prone. They need to be eliminated. Readability also requires that the code contains understandable names. Menomic names carried over from the Assembler programming times when data names were restricted to 8 characters make it very difficult if not impossible to understand the code, especially where comments are lacking. What is worse, if the code is reverse engineered, then these names are carried over into the model of the system rendering the model practically useless. It is therefore essential to replace the old names with new ones, especially if the code is going to be migrated.

Another particularly costly problem is that of using hard-wired data in statements. Rather than create variable names, programmers insert numbers and texts into their procedural code. This is a very bad practice since if a constant or literal is used in many places, it has to be changed in many places. This is often a major cost driver in evolving software. Another problem comes up with the data when programmers use incompatible data types, i.e. data types that are not standardized. Examples of this are bit data, binary data, packed data and floating point data. Such data cannot be readily converted. Besides it is error prone. In COBOL and PL/I all data should be either in character or decimal numeric format so they can be stored as ASCII character data. A problem that occurs often in migrations is the mixing of database accesses and user interface operations with business logic. That makes it costly to replace the existing database or to use another user interface driver. It also makes it more

difficult to test. That is why all interactions with the environment should be kept separate from the core functionality.

Finally, there is the problem of unstructured code on the one hand and too deeply structured code on the other. Unstructured code is a result of using GO TO branches to connect different code blocks. It becomes very difficult to follow the control flow and to make changes to it without undesired side effects. Besides it reduces reusability, since it is impossible to extract code blocks for reuse as long as they are intertwined with the code blocks around them. Conversely, if procedural code is too deeply nested it becomes difficult to understand and to handle. Code building blocks – procedures or methods – need to be limited in size and in complexity. This entails breaking them up into smaller code units.

It is true that there are many other problems as well, but much can be gained by resolving those problems mentioned above. The purpose of CodeRedo is to reduce maintenance and migration costs by:

- Improving readability by formatting the code
- Eliminating problematic code constructions
- Raising portability by removing incompatible data types
- Enabling comprehension by renaming data variables
- Enhancing flexibility by extracting hard-wired data
- Promoting reuse by relocating database, file and user interface operations
- Reducing control flow complexity by restructuring the control flow
- Increasing modularity by refactoring oversized procedures

(*see figure 1: The Goals of CodeRedo*)

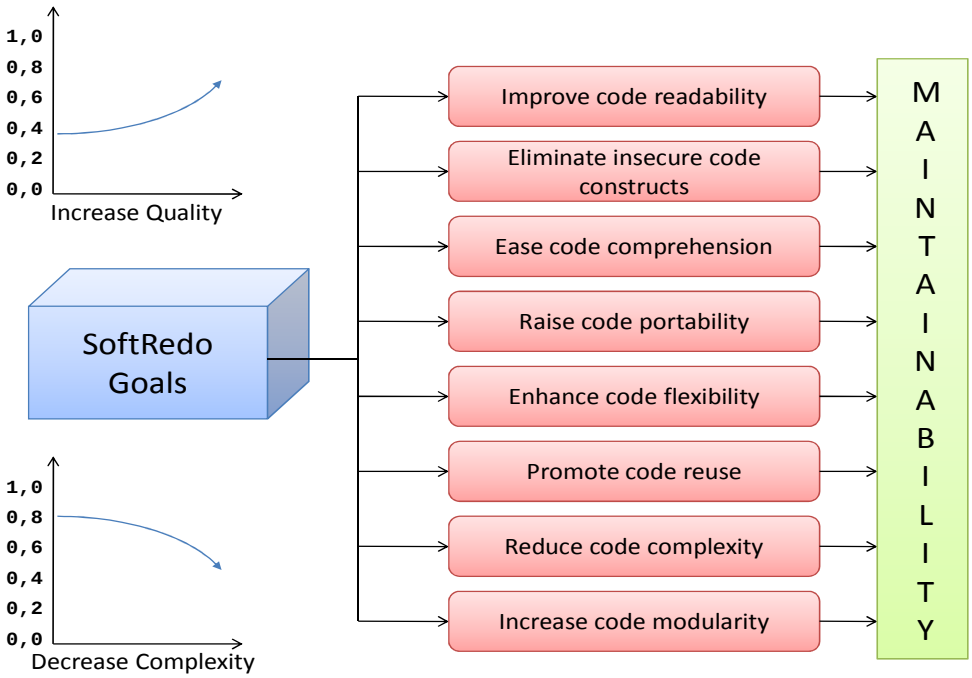


Figure 1: The Goals of CodeRedo

1.1. Improving Code Readability

The first goal of CodeRedo is to improve code readability so that the maintenance programmer can more quickly gain an overview of the code and locate what he is looking for. For this the lines should be limited in length and contain no more than one statement or declaration. Furthermore, procedurally nested code should be indented and easily recognizable as such. The format of the code should resemble that of a well structured document.

1.2. Reducing Code Complexity

The second goal of CodeRedo is to improve code quality by replacing complicated or error prone code constructs. In the world of program analysis these are referred to as code smells. Typical code smells are in Assembler branches to physical addresses, in PLI label variables, in COBOL Alters, Perform Thru and GoTo Depending on, and in C conditional assignments, casting and implied data types. There are many such statement types that should never have been used, but programmers still do because they are either lazy or malicious. Such coding constructs are recognized by code auditing tools as code deficiencies. They affect the conformity of the code to the rules of good programming style, thus reducing the code quality. By eliminating them, the code quality is raised.

1.3. Enabling Code Comprehension

The third goal of CodeRedo is to enable code comprehension so that persons can associate the code to the business function it is performing. This is not only an essential prerequisite to reimplementing the code, but also an important factor in extracting the business rules from the code. It is senseless to reverse engineer the code without having meaningful names. The reverse engineering only creates a structure with no meaningful content. The names of the data need to be cleaned up before either reverse engineering or reengineering the code. If not, one will never achieve the goals of those actions. This renaming is especially important if the goal is to convert the data.

1.4. Removing incompatible Data Types

The fourth goal of CodeRedo is to remove incompatible data types. This is primarily a problem for mainframe programs which use bits, packed, floating point and binary data fields. On the mainframe this numeric data is represented in EBCDIC format. On Unix, Linux and PC machines this data is stored in ASCII character format. In relational data bases there are no bit and packed attributes. There are integer and double fields for binary types but there layout is different and they are not readable to a human. The idea is to put all data in a human readable form so as to make the system testable. Therefore it is better if these data types are removed. Bit data, as in Assembler and PL/I, should become a single character or byte. Packed, floating point and binary data should be converted to decimal character format. This will allow this numeric data to be converted both to XML and to SQL. Then the tester will be able to read them without having to interpret their content. This makes a big difference in the testability of the system. Testability is now a big issue and needs to be considered when defining data. The removal of these data types also makes the system more portable since then the data can be ported 1:1 from one machine to another without having to be converted.

1.5. Enhancing Code Flexibility

The fifth goal of CodeRedo is to enhance the flexibility of the code by extracting the hard-coded data. Text literals and numeric constants in code are a cause of unnecessary maintenance effort and a source of errors. Besides being a problem for compilers and source analyzers, they are also a problem for the programmers. If a programmer wants to change a given constant, he must identify all locations where that constant occurs. In changing a certain number or text he may also change numbers and texts which he does not want to change. This practice of hard-wiring data into procedural code is a very bad practice, but unfortunately, it is also wide spread. Reducing or constraining it will lead to less maintenance costs.

1.6. Achieving higher Portability

The sixth goal of CodeRedo is to raise the portability of the code. Portability is determined by the degree to which the code is dependent on a particular database system, file system, operating system or teleprocessing monitor. This dependency is a factor of the number of database accesses, file operations, macros and teleprocessing operations contained within the code. When these instructions are mixed together with the business logic instructions, the module becomes very difficult to migrate. To ease migration it is absolutely essential to separate the environment dependent statements from the other. This also enhances maintainability since the business logic can then be changed without impacting the system interfaces. These should be in a special access or user interface shell. One goal of any reengineering tool should be to collect these non-portable operations and to gather them together in separate procedures where they can be more readily replaced without endangering the business logic.

1.7. Reducing Control Flow Complexity

The seventh goal of CodeRedo is to reduce control flow complexity by restructuring the code while at the same time creating smaller code blocks. Restructuring eliminates the GoTo connections between separate blocks of code and makes it possible to reuse them in another context. Remodularization creates more and smaller blocks of code. Modularity is understood as the degree to which code is split up into small, separately performable blocks of code with a single entry and a single exit. By splitting the code up into digestible chunks it is easier for the maintenance programmer to understand them and to reuse them. It is also possible to alter a chunk without an immediate effect on the others. Each code block should also have only one entry and one exit so that it can be reused in another context. There should be no direct connection from one code block to another. Of course the code blocks are still linked by the common data they use. To really separate them from one another, one would have to assign each code block its own data area and to copy over the data each time the block is invoked, but this would cause significant performance problems. Therefore, for the time being it is sufficient to separate the procedural code blocks and to cap all direct links between them. Since the internal procedures of a PLI program can be located anywhere within the main procedure, it is also better to move them to the end of the source after the main procedure, where they are all together. In this way the individual code blocks are not only easier to comprehend and change, but it is also possible to reuse them in another context, such as subroutines of a web service.

1.8. Refactoring deeply nested Procedures

The final goal of CodeRedo is to refactor deeply nested control logic. Deeply nested procedures are a problem for those responsible for maintaining them. This comes up when selections and loops are built into other selections and loops making it difficult to reach the inner conditions without fulfilling the outer ones. The testability drops because the probability of reaching the inner branches decreases with each additional nesting level. In addition, it becomes difficult to alter the conditional logic without affecting the lower level conditions. For these reasons, procedures should not exceed a certain nesting level. One of the goals of refactoring is to refactor out such deeply nested code and to make a new independent procedure out of it. It should be invoked from the location where it is removed. This is a refactoring measure that can be carried out automatically by a tool such as CodeRedo.

2. Functions of the CodeRedo Tools

To fulfill the goals outlined above, CodeRedo offers eight individual, selectable functions, one for each purpose:

- it reformats the source code (Realign),
- it refines and cleanses the code (Refine)
- it renames the poorly named variables (Rename),
- it recognizes and removes incompatible data types (Remove),
- it replaces hard-coded data with symbolic constants (Replace),
- it relocates database, file and teleprocessing operations (Relocate)
- it breaks the code up into smaller independent units with a central control (Restuc)
- it refactors deeply nested procedures (Refact).

(*see Figure 2: Code Renewal Functions*)

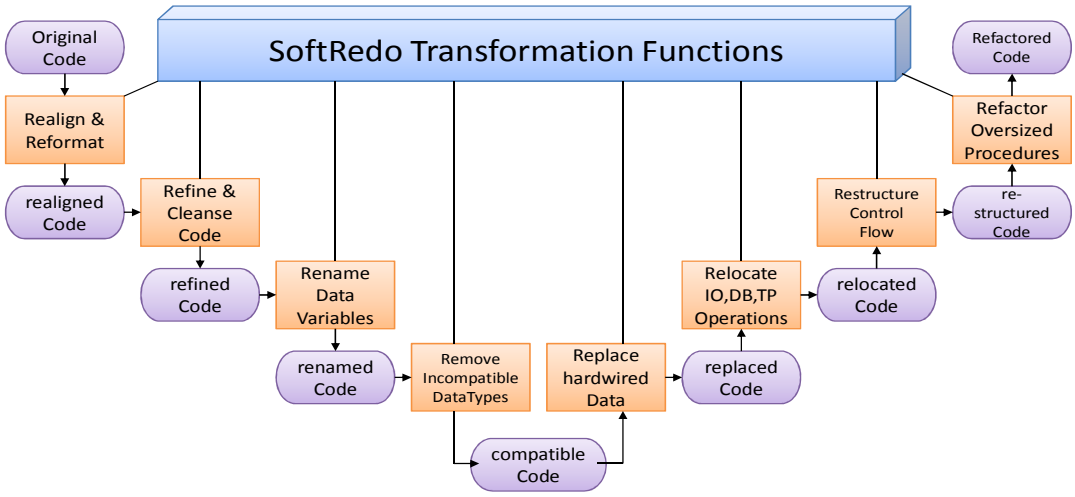


Figure 2: Code Renewal Functions

2.1. Reformating of Source Code

In reformatting the source code CodeRedo splits code lines in which there are more than one statement per line and creates a line per statement. If a line exceeds 100 characters, it is cut into two lines. This not only makes it easier to analyze and read the code, but it also helps to close the gap between the number of statements and the number of lines of code. In addition, CodeRedo indents nested code, i.e. code included in code blocks, loops and if conditions. This indentation makes the code more readable. Many of the older source texts were created with an old unreliable editor. Special characters may be represented with a non-standard code and blank characters, carriage control and line end markers may be missing. It is here that the source text is cleaned up for further processing. Since not only the program

code but also the macros, copies, includes and header files can be poorly formatted with multiple data definitions on a single line, these sources should also be reformatted.

With the code of PL/I and C++ programs comments that are in the same lines as the code are taken out and put in a separate line proceeding the line where they were originally located. This is done to make it possible to distinguish between code lines and comment lines as is the case in COBOL. It also makes it easier to reengineer and to convert the code into another language.

2.2. Refining the Code

There are many code constructs in legacy programs which should never have been used. In PLI it is the use of external data, Begin Blocks and pointers. In COBOL it is the Perform-Thru, the Alter, and the Next Sentence statements. Besides there is the use of the period to terminate if else conditions. In C++ it is the use of casting, conditional assignments and nested assignments as well as missing default statements. The refinement of code is, thus, very language specific.

PLIRedo makes the following refinement adjustments to the code:

- It converts external data blocks to parameters
- It converts Begin blocks to internal procedures
- It changes all On conditions to call an internal procedure
- It replaces Like references by Include references
- It replaces Fetch commands with standard Call statements
- It removes label variables
- It inserts the Other clause into Select statements
- It separates the code blocks with comment lines

COBRedo makes the following refinement adjustments:

- It replaces Perform Thru statements with a sequence of performs
- It replaces Alter statements with an Evaluate statement
- It replaces examine statements with inspect statements
- It replaces Next Sentence statements with If statements
- It places an End-If after each If statement
- It ensures that there is an Other Case in each Evaluate statement
- It embeds the comments in the proper code block
- It separates the code blocks with comment lines

CPPRedo makes the following refinement adjustments:

- It replaces casting statements with function calls
- It replaces conditional assignments with a condition and an assignment
- It splits up nested assignments into several assignment statements
- It inserts missing break statements
- It inserts result checks after external function calls.

2.3. Renaming of Variables

In the past it was common to assign short mnemonic names to data variables. This form of short hand writing was intended to save typing time. In Assembler it was not possible to use names longer than 8 characters and most Assembler programmers carried this practice over into higher level languages like COBOL and PL/I even though it was no longer necessary. Those languages allow up to 30 character names. Unfortunately no one other than the original author or someone very familiar with the data is able to make anything out of the crippled names. They make the code incomprehensible. If the code is redocumented, the documents are still incomprehensible because the short names remain. If the code is migrated, then the names are carried over into the new language making the new code equally incomprehensible. There is no other solution but to replace these names.

To replace the names, the original programmers must create a name substitution table with two columns. In the one column are the old short names, in the other the new long names. CodeRedo reads this table into a direct access database using the short name as a key. It then processes the sources and replaces all old names found with the new names. If a line becomes too long, the statement is continued in a continuation line. This way the source text becomes longer, but the content of the text becomes more understandable.

Speaking names are the key to program maintainability. Without converting the data names to an understandable form the programs will never be understandable to a stranger unfamiliar with the data. The renaming of variables is therefore of immense importance. If the code is to live on and be maintained then the old assembler like names should be replaced.

2.4. Removing incompatible data types

In migrating programs and their data to new environments there are always problems with incompatible data. Certain data types, in particular bit fields, binary fields and packed-decimal fields are stored differently on different machines. The solution is to put them in character or zoned decimal format. CodeRedo contains a function which will do this automatically. In COBOL it cuts off the USAGE clause. In PL/I it replaces the data type. In the case of packed and binary data this has no effect on the processing statements since the data values are converted before any computational operation. The Boolean operations in PL/I are changed automatically to address a character field.

The consequences of this field conversion are that the data structure to which the converted fields belong becomes longer. If there is a string or substring operation within that structure, the length variable has to be adjusted. If the structure in which the converted fields are contained is redefined then the redefining field will be automatically padded to match the length of the redefined field. In either case a comment is inserted in the data definition to give the programmer the length of the new extended data structure. Overlay data structures using redefinition is one of the worst features of legacy programs. Removing them is one of the most important functions in reengineering.

Should the user want to retest the adapted programs in their old environment, he will have to adjust the record lengths of the files according to the new lengths. This will require a data conversion. Therefore, it is better to convert the data types to a standard format before porting the programs to another platform or converting the code to another language.

In C and C++ it is not data types which are removed but environment specific macros such as EXEC CICS and EXEC SQL. They may also be user-defined macros. The macros are placed

into comments so as not to affect the compilation. It is up to the user to replace them by equivalent macros for the new environment. The goal here is to produce a portable version of the code.

2.5. Replacing hard-wired Data

It has been pointed out that the use of numeric constants and text literals in procedural statements is a bad practice which not only reduces the changeability of the code but also reduces the performance. Those are the reasons for forbidding this practice in almost all of the books on programming. The fact that it is so widely spread only confirms the fact that most practicing programmers could care less about the maintainability of their code. Unfortunately, this has dire consequences for the costs of maintaining that code.

It is, of course, difficult to assign meaningful names to symbolic constants after the fact but CodeRedo can at least remove the hard-wired data from the procedural statements replacing them with generated symbolic constant names. The symbolic constants are collected together either in a separate copy or include member or in a resource header file. These new source member files are added to the source library where the original source is located. There they can be included in the original source code at compile time. This makes it possible to change constants and text literal independently of the procedural code. It is even possible for non-programming staff to edit them, thus reducing the costs of change. Afterwards a text editor can scan through the code and replace the generated names with the speaking ones.

With this feature, external resource files, i.e. text or code tables, can be created which are exchangeable. This can be important when it comes to the localization of code, e.g. replacing German texts with English texts or changing numeric codes like postal codes. Such an evolution can be achieved without going into the actual source code. One need only alter the tables or resource files.

2.6. Relocating I-O and Access Operations

Programs which mix the technical I-O and database access operations with the business logic code are neither portable nor reusable since their business logic is totally dependent on a particular user interface and database driver. It is not possible to reuse them in another environment which uses another database system or another teleprocessing monitor. This is a particularly strong handicap when it comes to wrapping the code for reuse, since only those code units can be reused, i.e. wrapped, which are independent of the technical environment. With many old applications this is less than half of the code. For this reason, it is essential to separate the interactions with the environment from the functional operations.

To overcome this handicap, CodeRedo relocates all such environment dependent operations such as file operations, database access operations and user dialog operations into separate code blocks of an access shell, either in Assembler Csects, in PL/I internal procedures, in COBOL sections or in C++ dedicated IO-functions. Each and every such operation becomes a performable function, which is then invoked from the location where it previously was. In this way, the business rule logic becomes free of dependencies on the environment. If there are many such IO and access operations, e.g. opens, writes, reads, closes, EXEC-CICS, EXEC-DLI, EXEC-ADABAS, send, receive, usw., this comes to a major code revision. CodeRedo is able to make this revision automatically in that it relocates all such interactions with the

environment to an I/O/Access shell, replacing them by standard procedure calls. By separating the business logic from the input/output and access logic, the portability and reusability of the code is vastly improved.

2.7. Restructuring the Code

It has long been known that GO TO branches are harmful to the code. They make it hard to follow and difficult to change. Nevertheless, most old code is full of such wild branches. In Assembler it is the only way to get from one location to another. In PL/I it should never have been used but it was allowed, so that many programmers used it to steer the control flow. In later versions of structured COBOL there was no need for it, yet many programmers continued to use it since they were not able to think in another way. Oddly enough it is still allowed in the C/C++ language and still programmers use it. The moral of the story is that if there is any opportunity to make a mess of their code, programmers will use it. This is due in part to the lack of developer qualification und in part to the lack of supervision as well as to the absence of any kind of coding convention. The problem is that code blocks connected by GO TOs cannot be reused. The GOTO connections have to be capped if the code is to be converted or wrapped.

CodeRedo does this by including a label variable in the code. This label variable points to the next block of code to be executed. The GOTO statements are placed in comments so that the programmers may see where they are. In their place a label variable is set to the label to which the GOTO jumps. All of the following statements up to the end of that code block are then masked by a nested if. In the end there are no more branches out of the individual code blocks. In PLI the code blocks are internal procedures, in COBOL paragraphs, in C++ functions. Should the code have to be converted into an object-oriented language without GOTO statements, then the prerequisites for such a conversion are fulfilled here.

Finally, the restructuring function collects all of the internal procedures and moves them to after the main procedure. This is where they belong as subroutines. In so doing, all of the data declarations pertaining to the main procedure are automatically filtered up to within the main procedure. This is an important prerequisite for the later conversion of the procedures to classes. Each internal procedure becomes a separate class whereas the main procedure is broken up into several classes, one for each global data structure within the main procedure.

2.8. Refactoring the Code

Code complexity increases to the degree that the procedural logic is nested. The more ifs, cases and loops are embedded within each other the higher the code complexity becomes and the more difficult it is to test the code. There are even metrics such as the Plauger nesting level metric which measure program complexity based on this feature. Therefore, after a certain nesting level is exceeded, the code beyond that level should be factored out and moved to a separate subroutine. In PLI it becomes a new internal procedure attached to the end of the code. In COBOL it becomes a separate paragraph and is relocated to a subroutine section at the end of the program. In C++ it becomes a sub function which is attached to the end of the source. This breaking up of complex logic expressions into small partial expressions is derived from Algebra where equations are factored. Therefore the term “refactoring”.

The maximum procedural nesting level is language and context dependent. Therefore it is left to the user to select the appropriate nesting level, over which the code will be factored out. It

may range from 3 to 9. Refactoring increases the length of the source text but it reduces the width. Not only does this decrease the complexity of the code, but it also increases the modularity, since now the code is broken up into more and smaller units.

3. Inputs to the CodeRedo Tools

The inputs to the CodeRedo tool are the original source code members.

- Assembler CSects, makros and copies,
- PLI external procedures and Include members,
- COBOL74/85 programs and copy members,
- C/CPP classes and header files
- Name tables for replacing data names
- Copy/Include library with all procedural copies and includes.

In addition to these source inputs, CodeRedo accepts a number of parameters from the user interface including the step selection and the options for the selected steps. The user has a wide range of choice in determining which reengineering actions should be performed and to what degree. (*see Figure 3: CodeRedo Inputs*)

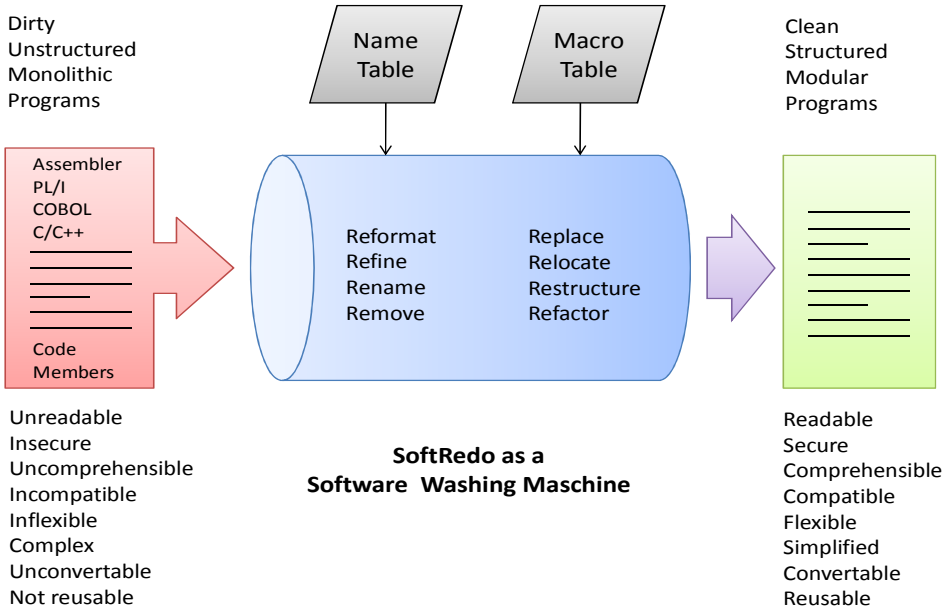


Figure 3: CodeRedo Inputs

3.1. Assembler Source Files

There are two types of Assembler source files, on the one hand, Assembler CSects and on the other Assembler copies and macros. Of course, in Assembler data types cannot be changed and names should not be replaced.

3.1.1 Assembler CSects

On Assembler Csects only six of the eight reengineering functions can be performed:

- the code can be formatted (REALIGN)

- the code can be refined by replacing certain command types (REFINE)
- the text literals can be replaced by variable names (REPLACE)
- the IO operations can be relocated to separate subroutines (RELOCATE)
- the code can be restructured, i.e. each label denotes a separate subroutine which is invoked from a central control loop (RESTRUC)

3.1.2 Assembler Copies and Macros

On Assembler Copys and Macros only two of the eight reengineering functions can be performed:

- the code can be formatted (REALIGN)
- the code can be refined by replacing certain data definitions (REFINE)

3.2. PL/I Source Files

There are two types of PLI source files – PLI external procedures and PLI include members.

3.2.1 PLI External Procedures

On PLI external procedures all eight of the reengineering functions can be performed:

- the code can be formatted (REALIGN)
- the code can be refined by replacing certain command types (REFINE)
- the data short names can be replaced by long names (RENAME)
- the PL/I specific data types can be removed (REMOVE)
- the text literals and numeric constants can be reassigned to variable names (REPLAC)
- the IO operations can be relocated to separate procedures (RELOCATE)
- the code can be restructured, i.e. each code block becomes a separate internal procedure invoked from a central control loop (RESTRUC)
- the procedures can be refactored, i.e. code beyond a specified nesting level is factored out to a separate internal procedure (REFACTOR).

3.2.2 PLI Includes

On PLI include data definitions three of the eight reengineering functions can be performed:

- the code can be formatted (REALIGN)
- the data short names can be replaced by long names (RENAME)
- the PL/I specific data types can be removed (REMOVE).

3.3. COBOL Source Files

There are two types of COBOL source files – COBOL programs and COBOL copy members for defining common data structures.

3.3.1 COBOL External Procedures

On COBOL programs all eight of the reengineering functions can be performed:

- the code can be formatted (REALIGN)
- the code can be refined by replacing obsolete command types and restructuring the remaining commands (REFINE)
- the data short names can be replaced by long names (RENAME)
- the packed and binary data types can be converted (REMOVE)
- the text literals and numeric constants can be replaced by symbolic constant names (REPLAC)
- the IO and database access operations can be relocated to a separate IO section (RELOCATE)
- the code can be restructured, i.e. each paragraph becomes a separate method invoked from a central control loop (RESTRUC)
- the paragraphs can be refactored, i.e. code beyond a specified nesting level is factored out to a separate paragraph (REFACTOR).

3.3.2 COBOL Copies

On COBOL copies three of the eight reengineering functions can be performed:

- the code can be reformatted (REALIGN)
- the data short names can be replaced by long names (RENAME)
- the COBOL specific data types can be removed (REMOVE).

3.4. C++ Source Files

There are two types of C++ source files – C++ classes or procedures and C header files for defining data structures and interfaces.

3.4.1 C++ Classes and Procedures

On C++ classes and procedures six of the reengineering functions can be performed:

- the code can be formatted (REALIGN)
- the code can be improved by removing undesired command types (REFINE)
- the text literals and numeric constants can be replaced by defines symbolic constant names (REPLAC)
- the IO operations can be relocated to separate IO methods (RELOCATE)
- the code can be restructured, i.e. gotos are converted to method calls (RESTRUC)
- the methods can be refactored, i.e. code beyond a specified nesting level is factored out to a separate method at the end of the class (REFACTOR).

3.4.2 C++ Header Files

On C++ header files only two of the eight reengineering functions can be performed:

- the code can be formatted (REALIGN)
- friend methods are removed and method interfaces assigned an explicit scope (REFINE).

- Short names can be replaced with longer ones (RENAME)
- Incompatible data types can be removed (REMOVE)

3.5 Macro/Function Table

The macro/function table contains a row for each macro or function with 3 columns. For the procedural languages Assembler, PL/I and COBOL the rows are macro operations. For the languages C and C++ the rows are standard functions for I/O operations.

The first column is a four letter code for the type of macro or functions. The second column is an 8 or 40 character string giving the name of the macro or function. The third column is used only by the tool ***SoftAudit***. It is the number of function–points associated with each I/O operation or file type. It can range from 3 to 6 for Input operations, from 4 to 7 for Output operations, from 5 to 10 for Files and from 7 to 15 for Databases.

In the case of macros the name of the macro is only 8 characters, followed by the number of parameters for the macro and the position of the parameter with the name of the object being processed by the macro, i.e. the file, record datagroup or map.

READ GETNEXT, PARAMS = 3, OBJECT = 1

The 25 valid macro or function types are:

- CALL = A procedure or method is called
- CLOS = A file or database ir opened
- DB = A database is declared
- DECL = A datatype is declared
- DELT = A file or database record is deleted
- ENTR = An entry to the module
- EXIT = An exit from the module
- FILE = A file is declared
- FUNC = A function to compute a value
- GETS = A data storage is allocated
- INCL = Another source is copied in
- INPT = An input operation
- ISRT = A database, record is inserted
- LIST = A report is printed
- MASK = A user interface is declared
- MESS = A message is declared
- OPEN = A file or database is opened
- OUTP = An output operation
- PROC = A procedure is declared
- RECV = A user interface or message is received
- READ = A file record is read
- SELT = A database record is queried
- UPDT = A database record is updated
- WRIT = A file record is written
- SEND = A user interface or message is sent.

A sample macro table for Assembler and a sample function table for C++ are given in *Sample 6.1 – Macro-Table und Sample 6.2 – Function-Table*. As can be seen, it is impossible to identify the input/output, DB and TP operations without the help of such a table. CodeRedo needs it to identify and relocate the IO, DB and TP operations. In Assembler and C these operations are user defined. Therefore it is up to the user to define them also for the reengineering tool. PL/I and COBOL have standard IO operations, so unless the user is using a special macro language, the table is not required for processing those languages.

3.6 Data Name Tables

If the user wants to rename his data variables, i.e. changing short names to long names, then he must set up the renaming tables. These tables can be made with Excel. They contain two comma separated columns. In the first column is the original short name of a variable. It may be no longer than 20 characters long. In the second column is the new long name. It can be up to 30 characters in COBOL, 32 in PLI and 36 in C/C++. The format of the table is as follows:

<shortName>;<longName>
<shortName>;<longName> (*Sample 6.3 – Name-Table)*

There can be several tables for each set of programs and data structures to be renamed. In any one table there can be any number of rows. Creating the tables is a responsibility of the user, but he may use the comment lines attached to the data declarations as a basis for creating the new long names.

3.7 Procedural Copies and Includes

Since the whole program with the exception of the external data structure definitions is being reengineered and the program parts are interconnected with each other, it is necessary to build the procedural copies or include members into the code before reengineering begins. It would be possible to reengineer the procedural copies and includes separately, but there is no rule as to what these copies can contain. They are simply text excerpts which can be copied in anywhere. Therefore it is necessary to build them into the code before the reengineering begins. For this the user must assign the procedural copy library. The very first step in the reengineering process is to resolve the procedural copies or includes.

3.8 GUI Parameters

The ***CodeRedo*** user interface gives the user the opportunity to set the control parameters and to select which reengineering functions should be performed.

3.8.2 Control Parameters

The Control parameters to be submitted by the user include the following names and directories:

- Language Selection (Assembler, PL/I, COBOL74, COBOL85 or C/C++)

- Name of the product being reengineered
- Name of the system being reengineered
- Output directory for storing the reengineered sources
- List of source files to be processed
- List of name tables to be used when renaming the data
- List of Options for the reengineering actions.

3.8.3 Reengineering Options

The reengineering options are:

- Add procedural code copies
- Reformat code text
- Refine code features
- Rename data variables
- Remove data types
- Replace text literals
- Replace numeric constants
- Relocate File and IO operations
- Relocate database access operations
- Relocate teleprocessing operations
- Restructure program logic
- Refactor the procedures

If the user selects the refactoring function he is requested to give the nesting level above which the code will be factored out.

The input parameters are collected together in an XML interface message which acts as a link to the batch processes. Should the user want to run CodeRedo in another environment other than Windows, he can create his own driver and still use the underlying batch processes by setting up this XML interface and invoking the Redo processes. In this way it is even possible to run CodeRedo on the mainframe. (*Sample 6.4 – Batch Process Interface)*

4 Outputs from the CodeRedo Tools

The outputs from CodeRedo are the renovated sources plus the resource files.

- Assembler Programs & Copies
- PLI Programs & Includes
- COBOL Programs & Copies
- C++ Classes & Header Files
- Copy, include, header files for Constants & Literals

(*see figure 4: CodeRedo Outputs*)

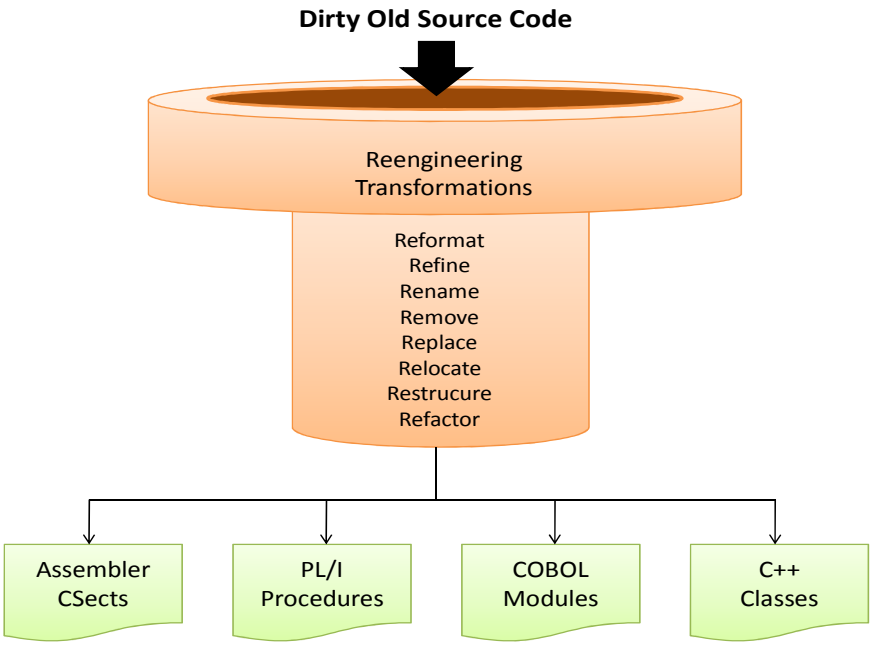


Figure 4: CodeRedo Outputs

4.4 Assembler CSects & Copies

The renovated Assembler sources will have been changed in the following ways:

- The source text has been reformatted with three columns. The first column contains the labels. The second column contains the operations. The third column contains the comments. All columns are aligned to start on the same position.
- Entry and Exit operations such as BR 15 will have been replaced with standard macros and physical addresses will have been replaced by logical ones using symbolic names in place of physical locations with displacements
- Text literals longer than 4 characters in Assembler operations have been replaced with a symbolic constant LIT999 and the symbolic constant inserted as a defined constant at the end of the source.
- Should there have been any file IO operations or database accesses (DLI, DB2) or teleprocessing operations (CICS, IMSDC), these will have been relocated to subroutines in an IO section at the end of the source.

- Forward branches have been converted to IF..ELSE macros and backward branches to WHILE loops. All branches are marked with comments. In this respect the code has been both restructured and annotated.

The renovated Assembler programs may not be directly compiled and executed. They must first be manually adjusted. On the other hand, they may be directly converted to either PLI or COBOL. The main sense of the renovation measures is to prepare the Assembler sources for conversion to a higher level language. *(see Sample 6.5: Reengineered Assembler CSect)*

In the case of Assembler copies or macros, they will only have been reformatted and some data definitions refined by placing labels in front of empty DC definitions and marking the structures of DS definitions. *(see Sample 6.6: Reengineered Assembler Copy)*

4.5 PLI Procedures & Includes

The renovated PLI procedures will have been altered as follows:

- The source has been realigned so that nested code is indented by two columns and lines with two or more statements are split. This makes the code easier for the maintainer to read and easier to process.
- The comments have been removed from the code lines and placed in separate lines immediately before the code lines to which they apply.
- If the user has selected the refine function the external data declarations will have become parameters to the procedure, the Begin blocks will have been converted to Do blocks, the label variables removed and the Other clause will have been inserted into the Select statements. Besides the code blocks will be separated by comment lines and the Leave statement marked.
- If the user has selected the renaming function, the short names will have been replaced by long names.
- If the user has selected the remove function, the packed data types will have been converted to decimal numeric and the bit types to a byte.
- If the user has selected the replacement function, the text literals and numeric constants will have been replaced by symbolic constant names and an include member with these constant data values will have been created. The Include statement for it is inserted right after the procedure declaration.
- If the user has selected the relocate function all IO operations, all database accesses (IMS-DB, DB2 and ADABAS) and all the teleprocessing operations (IMS-DC and CICS) will have been relocated to internal procedures after the main procedure. They are invoked by a call from their former location.
- If the user has selected the restructuring function the GOTO branches will have been replaced by Calls to emulate the GOTO path, the labeled code blocks will have become internal procedures and all of the internal procedures will have been placed after the main procedure and the global data declarations.
- If the user has selected the refactoring function, the nested code beyond the maximum nesting level specified will have been extracted and moved to a sub procedure at the end of the source. It is called from its original location.

The renovated PLI procedures are now in a state from which they can be more easily converted to an object-oriented language while their data structures are compatible with any SQL-type database. *(see Sample 6.7: Renovated PLI Procedure)*

PLI include members will have been realigned, the data variables renamed and the packed and binary data types converted if the user has selected those functions. Their data structures can now be converted to SQL tables or XML documents. *(see Sample 6.8: Renovated PLI Include)*

4.6 COBOL Programs & Copies

The renovated COBOL programs will reflect the following changes:

- Should the COBOL Procedure Division contain references to Copy members, these procedural copy texts will have been inserted into the main line COBOL code. For this reason the user must assign the procedural copy library.
- The source text has been realigned so that nested code is indented by two columns and lines with two or more statements are split. This makes the code easier for the maintainer to read. In addition, comments between paragraphs are now placed inside the paragraphs to enable the paragraphs to be relocated without losing the comments.
- If the user has selected the refine function the obsolete COBOL-74 statements such as Exhibit, Examine, Alter, Note, etc. will have been replaced, the Next Sentence commands will have become IF statements, the GOTO depending on constructs will be Evaluate statements, the Perform Thrus will be expressed as a sequence of Performs and all If statements will be closed out by End_Ifs. In addition, the code blocks, i.e. paragraphs, will be separated by comment lines.
- If the user has selected the renaming function, the short names will have been replaced by long names.
- If the user has selected the remove function, the packed data types and the binary data types will have been converted to decimal numeric.
- If the user has selected the replacement function, the text literals and numeric constants will have been replaced by symbolic constant names and a copy member with these constant data values will have been created. The copy statement is inserted at the end of the Working-Storage Section.
- If the user has selected the relocate function all IO operations, all database accesses (IMS-DB, DB2 and ADABAS) and all the teleprocessing operations (IMS-DC and CICS) will have been relocated to an IO Section at the end of the program. They are invoked by a Perform from their former location.
- If the user has selected the restructuring function a Next_Label variable will be declared, all GoTo statements will be replaced by a Next_Label to the GoTo target label and a central control loop will have been placed at the beginning of the Procedure Divsion which calls the paragraphs indicated by the next label until a program exit or Goback is reached.
- If the user has selected the refactoring function, the nested code of a paragraph beyond the maximum nesting level specified will have been extracted and moved to a separate Section at the end of the source. It is performed from its original location.

The renovated COBOL programs are now in a state from which they can be more readily converted to OO-COBOL or JAVA while their data records can be converted to SQL tables. *(see Sample 6.9: Renovated COBOL Program)*

COBOL Copy members will have been realigned, the data variables renamed and the packed and binary data types converted if the user has selected those functions. Their data structures can now be converted to SQL tables or XML documents. *(see Sample 6.10: Renovated COBOL Copy)*

4.7 C++ Classes & Header Files

The renovated CPP classes will have been changed in the following ways:

- The source text has been reformatted so that nested code is indented by two columns and lines with two or more statements are split. This makes the code easier for the maintainer to read.
- The comments have been removed from the code lines and placed in separate lines immediately before the code lines to which they apply.
- If the user has selected the refine function the casting constructs will have been replaced by function invocations, conditional assignments will now be a condition followed by an assignment, nested assignments will be split up into several individual assignment statements, breaks will have been inserted into the switch statements and asserts will have been inserted after every foreign function call to check the result returned.
- If the user has selected the renaming function, the short names will have been replaced by long names.
- There is no change to the data types.
- If the user has selected the replacement function, the text literals and numeric constants will have been replaced by symbolic constant names and a header file with these DEFINE constant data values will have been created. An Include statement is inserted before the first method or function.
- If the user has selected the relocate function all IO operations, all database accesses (IMS-DB, DB2 and ADABAS) and all the teleprocessing operations (IMS-DC and CICS) will have been relocated to special IO Functions at the end of the class. They are invoked from their former location.
- If the user has selected the restructuring function the GoTo branches will be replaced with a function invocation and a break statement, the labels will have been removed, the friend function calls will have been converted and returns will have been inserted after every method.
- If the user has selected the refactoring function, the nested code of methods and procedures beyond the maximum nesting level specified will have been extracted and moved to a separate Method at the end of the source. This method will then be invoked by a function call from its original location. The refactored methods will have been flattened.

After reengineering the renovated CPP classes should be in a more maintainable state than what they had been in before. *(see Sample 6.11: Renovated C++ Class)*

CPP Header Files will have been realigned, the data variables renamed and the methods given an explicit Scope = Private, Public, Protected – provided the user has selected those functions. The new data structures and interface definitions should now be safer and easier to comprehend. *(see Sample 6.12: Renovated CPP Header File)*

5 CodeRedo Usage

The tool ***CodeRedo*** offers the user a graphical user interface from which he can select which language he wants to process and which reengineering steps he wants to execute. The user interface comes up when the CodeRedo.exe is clicked on. Prior to starting the tool, the user should have collected the sources he wants to reengineer together in a common directory. The Assembler Csects have the extension .asm. The PLI procedures have the extension .pli. and the PLI include members the extension .inc. The COBOL programs have the extension .cob or .cbl while the COBOL copy members have the extension .cpy. The C procedures have the extension .c. The CPP classes have the extension .cpp. The C header files must have the extension .h. Source files without the proper extension will be ignored. Should the user want to rename data, then he must have created a data name table and stored it in a common directory. The name of this directory must be selected prior to starting the reengineering job.

The CodeRedo user interface is a single all inclusive panel. It contains the language selection and all the possible actions which can be carried out. There is a general selection bar at the top. Under it the panel is divided into two halves. In the left half is the language selection, the system and product names, the output directory selection and the list of possible actions. In the right half at the top are the buttons for selecting files and starting jobs. Under that is a displace place for showing either the name files selected or the source files selected. The user determines which files to display by clicking the appropriate button for the files selected for reengineering. At the bottom of the panel is a bar for displaying error messages.

The languages which can be selected are with a simple check button are:

- IBM Basic Assembler
- PL/I
- COBOL-74 = COBOL-1
- COBOL-85 = COBOL-2
- Standard C
- C++

The user may select only one language to process at a time. If a system contains different languages then the reengineering job must be repeated for each language using another output directory.

The content of the user panel depends on the selections made but in general there are the following seven parts of the main panel:

- The general selection bar
- The specific selection bar
- The language selection buttons
- The parameter input boxes
- The action list
- The action buttons and
- The display field

In addition, there is a separate panel for editing the macro tables. It contains an action menu bar, an edit bar and two display fields, one for displaying the macro names discovered by the scanner and one for assigning the macros to macro types.

5.1 General Selection Bar

In the general selection bar the user can choose between the general purpose functions

- File
- Actions
- Viewers
- Logs
- Help.

5.1.1 File Operations

In selecting the menu item „Files", a pulldown menu appears with 7 standard functions

- open existing job
- new job
- save job
- save job as
- import a word list
- export a word list
- exit.

The „open existing job" option causes a pop–up window to appear in which the user can search for a file containing the redo job parameters of a previous run in a standard windows directory.

The „new job" option brings up the user panel for setting the redo job parameters.

The „save job" option causes the current job parameters to be restored in the file where they had been before.

The „save job as" option causes a pop–up window to appear with a standard windows directory in which the user can find a place to store the current redo job parameters. They are stored in a file whose name is composed from the product and the system name with the extension .jsrt.

The „exit" option will cause the CodeRedo tool to be closed.

5.1.2 Actions

In selecting the menu item „Actions" a pulldown menu comes down with the options:

- Add files to the job
- Add complete directories
- Add Copy/Include libraries
- Add csv files for renaming
- Start prescanner of sources
- Start reengineering

- Skip rest of processing.

The option „add files to the job" gives the user a view of the windows directories from which he can select individual source members to be reengineered.

The option „add complete directories" gives the user a view of the windows directories from which he can select whole subdirectories of source files to be reengineered.

The option "add copy libraries" gives the user a view of the windows directories from which he can select one or more copy directories. One should know that only the procedural copies will be resolved here. The data copies remain out.

The option „add CSV files for renaming" gives the user a view of the windows directories from which he can select one or more csv files for renaming data.

The option „start prescanner" triggers the process to scan thru the selected source files for unknown statements and to store them in the macro file for the user to define.

The option „start–reengineering" triggers the reengineering process for that set of source members selected.

The option „skip rest of processing" causes the currently running reengineering process to be interrupted and the results to be reset.

5.1.3Viewers

The selection of the menu item „Viewer" results in a pull–down menu with the options:
- Open a file with the general viewer
- Call external editor.

The first option displays the contents of the output directory and allows the user to select any file in the output directory for viewing. The output directory should contain the reengineered programs and include, copy or header files.

The second option allows the user to use any other external editor to view the results of the reengineering, e.g. notepad, wordpad, etc.

5.1.4Logs

When the menu item „Logs" is selected, the log of the last reengineering run will be displayed with all of the messages from the reengineering steps. The log file will contain messages on any errors that may have occurred.

5.1.5Help

By selecting the menu item „Help" the user is shown the table of contents of this user documentation and can select the procedures or methods he wants to read. With the search topic option he can submit a key word and see where this word is used in the documentation.

5.2 Specific Selection Bar

The specific selection bar contains only two buttons:

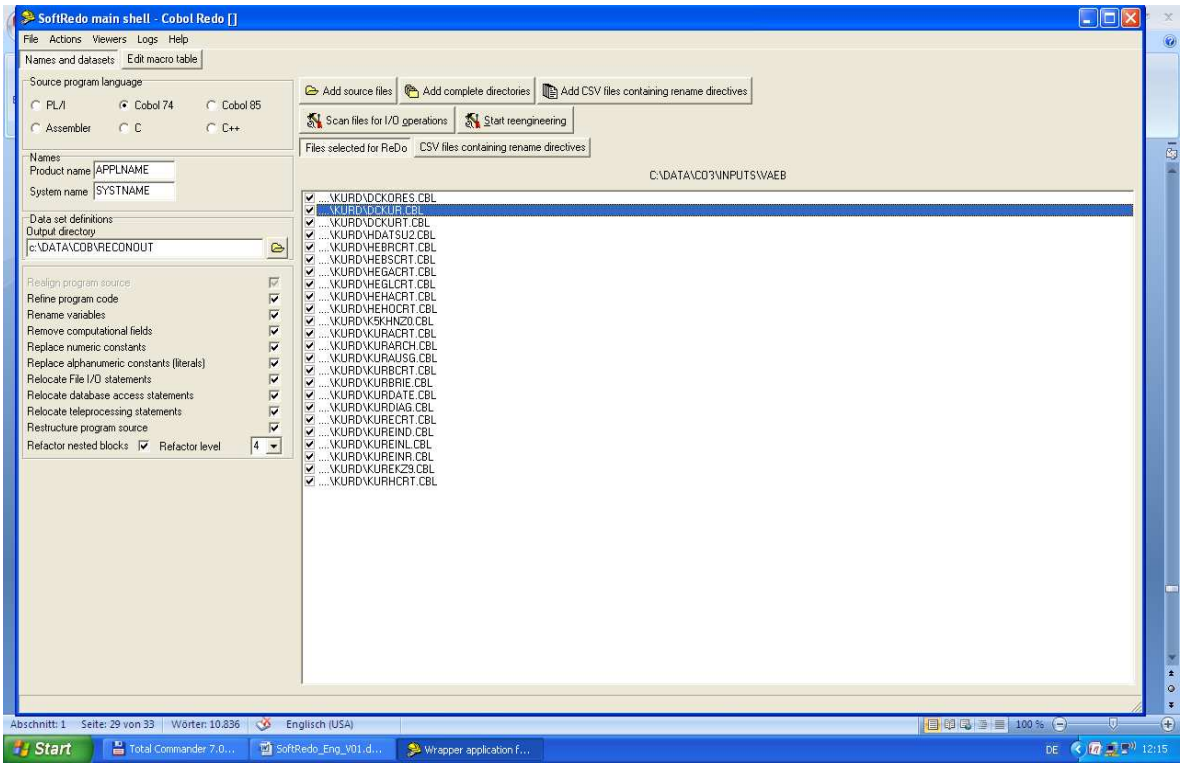
- One for selecting the main panel and
- One for editing the macro list.

5.3 Main Panel

The main panel contains the fields for:

- selecting the language
- giving in the product and system names
- assigning an output directory
- determining function steps
- selecting files and
- starting either a scanning or a reengineering job

(see Screenshot 1: REDO Main Panel)



5.3.1Language Selection

The language selection field contains six buttons, one for each of the possible languages to be reengineered – Assembler, PL/I, COBOL-74, COBOL-85, C, C++. The user may check only one per reengineering job.

5.3.2Names

Here the user is requested to submit a product and a system name for the reengineered sources. The product and system names are 8 character strings intended to uniquely identify the product and system being reengineered. A product may have several subsystems. In this case there would be several audit runs for one product each with a different system name. These names are used for logging purposes and also for separating the reengineered sources into different libraries. Each system corresponds to a separate library.

5.3.3Output Directory

Here the user is requested to select an output directory in which to store the reengineered sources. He is given a view of the windows directory tree in which he can navigate to find the name of the designated output directory. He may also create a new directory for this purpose. As pointed out above, sub directories will be allocated in the output directory, one for each subsystem.

5.3.4Reengineering Function Selection List

Here the user is presented with a list of action steps. There are altogether 12 possible reengineering functions, whereby only the second – realignment - is mandatory. All of the rest are optional:

(1) Insert procedural copies
(2) Reformat program source code
(3) Refine program source code
(4) Rename data variables
(5) Remove computation fields
(6) Replace numeric constants
(7) Replace literal constants
(8) Relocate file operations
(9) Relocate database accesses
(10) Relocate teleprocessing operations
(11) Restructure program logic
(12) Refactor program source code

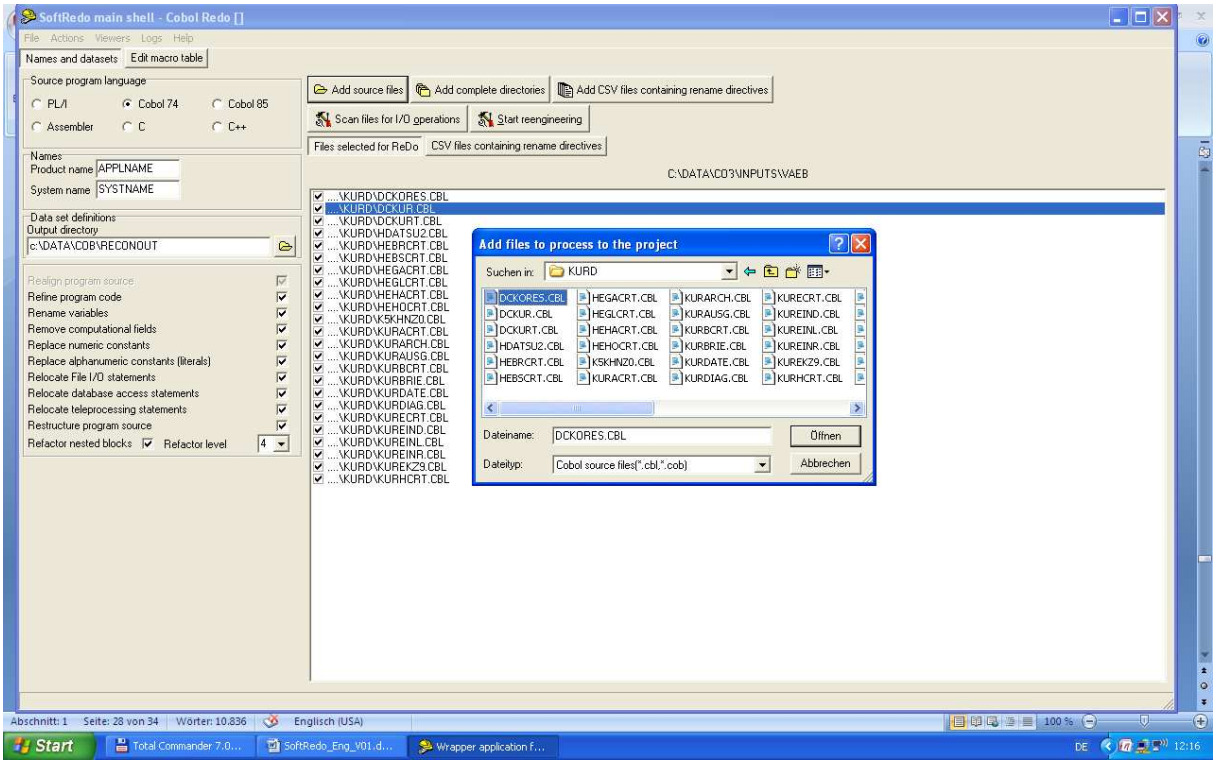
The user can click on any combination of these reengineering functions before starting the job. If he selects the last function – refactoring – he is required to give a level number. This is the nesting level of the procedural logic, above which the code will be factored out into separate procedures or methods.

5.3.5Selecting Files

At the top of the right half of the main panel, are two rows of selection buttons. In the first row are the buttons for:

- Adding source files
- Adding complete directories
- Adding copy directories
- Adding csv files for renaming

When the first button is clicked the user is given a window to the windows directories which he may scan through to select one or more source files to be reengineered. Here he must mark the files and click the open button. *(see Screenshot 2: REDO File Selection)*



When the second button is clicked the user is also given a window to the windows directories. Here he marks the name of a directory and opens it. Upon opening, all of the files of that directory will be selected for reengineering.

When the third button is clicked the user is given yet another window to the windows directories. Here he may select a copy directory containing the procedural copies to be inserted into the source code.

When the fourth button is clicked the user is given a window to the windows directory which he may navigate through to select name files. These are the csv files with the short and long name pairs. The files he selects will be concatenated into a single large name table for renaming the data in the program sources selected.

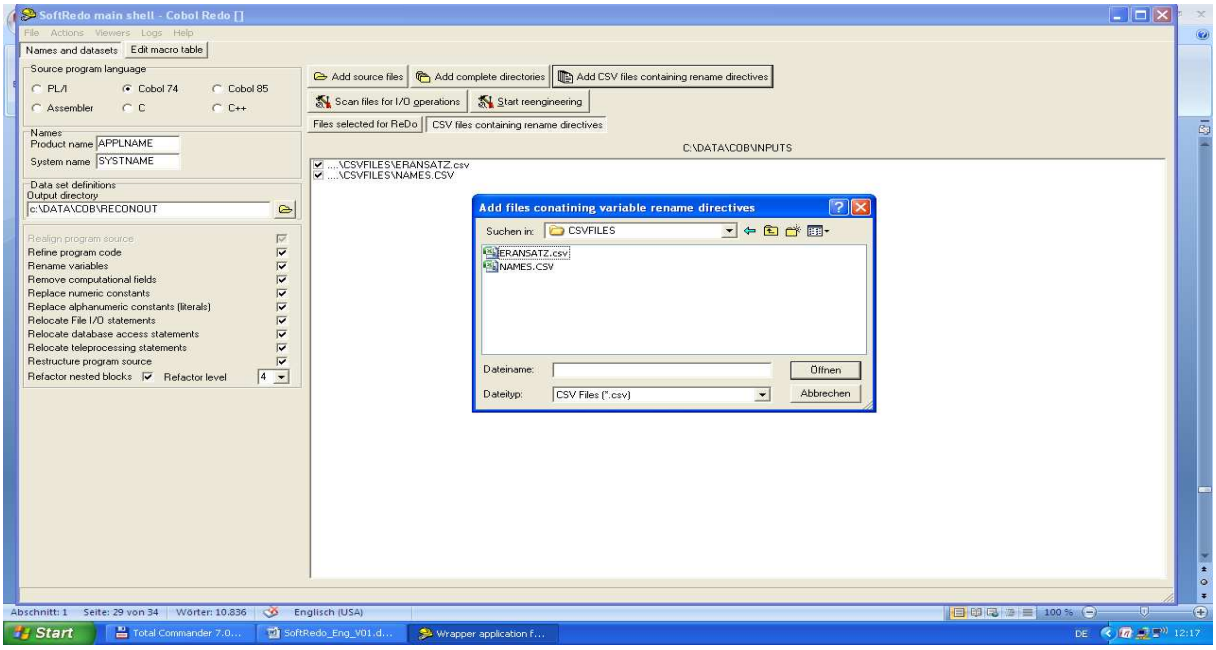
5.3.6Starting a Job

In the second row of the top right half of the main panel one finds two buttons for starting jobs. The first button is to start a job for scanning the macros in the source code. The second button is for starting the reengineering job.

5.3.7Displaying selected files

The bottom part of the right half of the main panel is reserved for displaying files. There are two buttons for selecting which files to display. The first button displays the source files

to be reengineered. The second button displays the name files that were selected. The user must take care to click the right button for the file display. Within the file display he may uncheck files and remove them from the list. *(see Screenshot 3: REDO Name File Display)*



5.4 Editing Macro Panel

The macro editing panel displays the current macro or function table for the target language. For Assember the table will contain all of the non-standard assembler statements which are possible macros. For C/C++ the table will contain all of the external functions called which are possible framework functions. It is up to the user to identify and classify them.

The option „Edit Word List" will cause the current macro or function table for the target language to be displayed in two columns

- internal type and
- macro / function name.

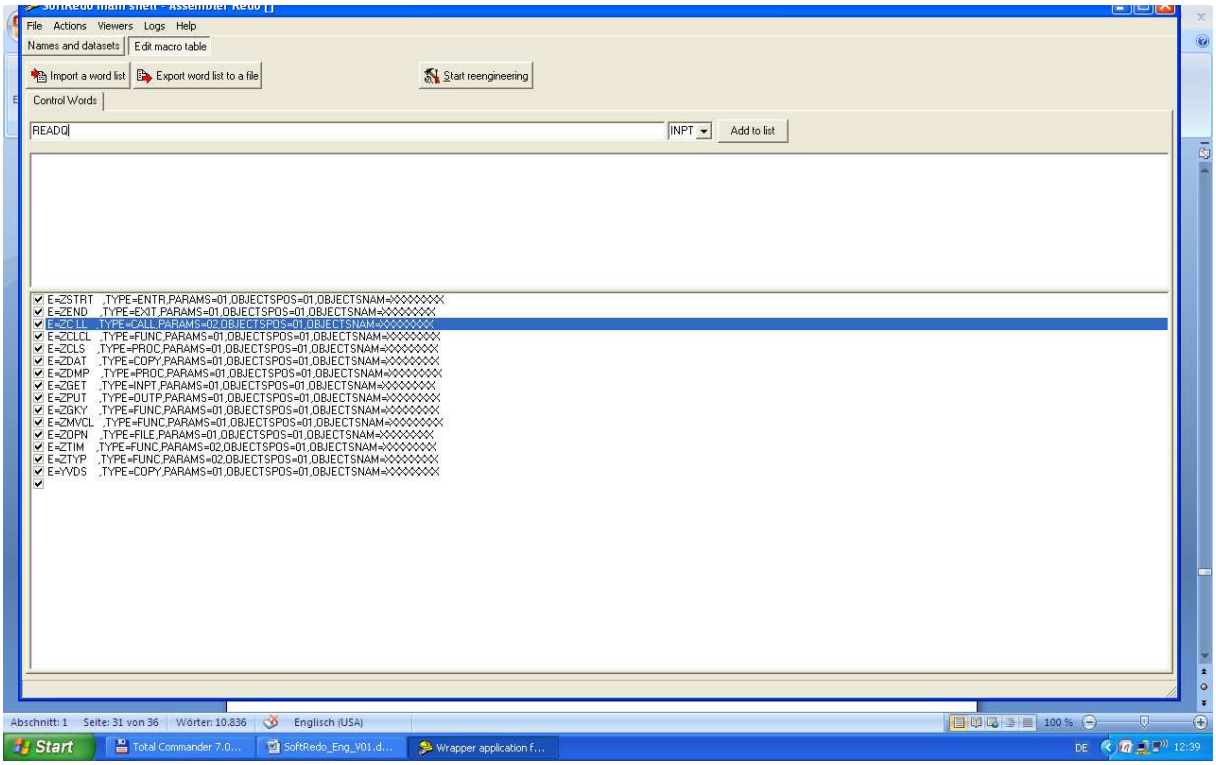
After a prescanning run, the macros and functions are displayed but their internal type will be unknown. The names of all macros will be listed. It will be up to the user to assign them one of the 30 predefined internal types.

Otherwise the contents of the existing standard table of functions will be displayed. The user can overwrite the macro or function names. He may also add additional macros or functions and assign them one of the predefined internal types. If the user wants to use an existing table from a previous analysis he may import it using the import function „Import Word List".

When the table is ready the user can export it by selecting the button „Export Word List". He will then be given a view of the windows directory and a box for assigning the name. The default name is FuncTab. It is recommended to store the FuncTab in the input directory together with the sources to be audited.

Later this table can be read in again by clicking the button „ Import a word list”. Otherwise the existing standard table of functions will be loaded.

The check box to the left of the table entries allows the user to remove macros and functions by unchecking them and clicking the right mouse button to remove them from the list. Care should be taken in keeping the tables of macros and functions consistent with the sources they refer to. *(See Screenshot 4: Setting up the Macro/Function table)*



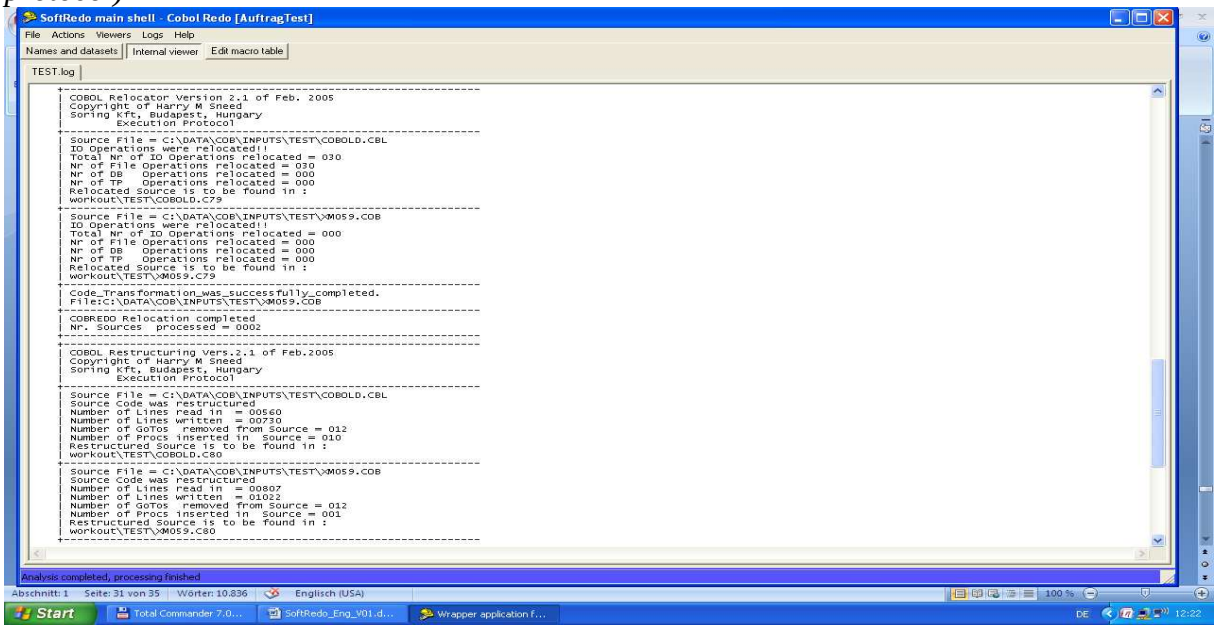
5.5 Scanning for Macros and IO Functions

If the user is not sure what user specific macros or functions are contained in the source code of the programs, he should run the sources through the prescanner before beginning a reengineering job. This is done by selecting either individual source files or complete source file directories. The selected files will be listed out in the text box of the names and datasets panel. Files can be removed by unchecking them.

The user then clicks the button „Start Prescanner”. A background task in a black–box will be started which goes through all of the sources of the target language type and inserts the unknown statements into the macro table for the user to define. When the job is finished the log file is displayed. The freshly created macro table is now the current one. The internal types are marked with question marks. They should be replaced by standard codes set by the user prior to running the reengineering job. The purpose of this scan is to identify the I/O and database access operations for the interface documentation. If the source contains special macros or methods for these operations then they must be predefined so that they can be recognized as such. The unrecognized statement types will be listed out in the macro table or function table, but it remains for the user to assign them to the internal statement types like ISRT, UPDT, READ, WRIT, etc. This must be done prior to starting the reengineering process, in particular for Assembler und C, since in these languages the IO and database access operations are almost always macros or special functions.

5.6 Running a Reengineering Job

To run a reengineering job the user must select the language, give in the product and system names, select an output directory, select the sources to be reengineered and click on the reengineering functions to be performed. He then activates the start reengineering button. The job will run in the background. After each and every reengineering function the user will be requested to confirm the action has been completed. He does this by pressing the enter key. After the last function has been completed a protocol of the job will be displayed showing what functions have been performed upon which programs. *(see Screenshot 5: CodeRedo Job protocol)*



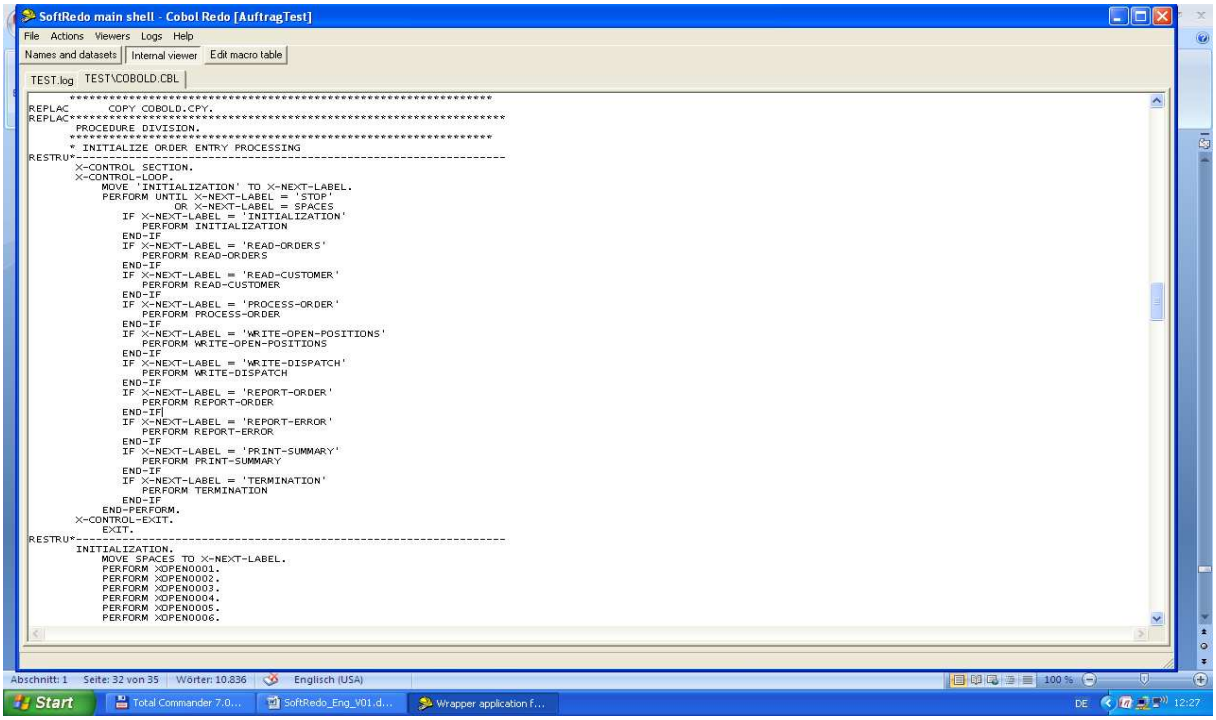
5.7 Viewing the Reengineering Results

The last step in using CodeRedo is to view the source files reengineered. All of them are stored in the output directory assigned by the user. The user can view them by using his own editor or by using the standard CodeRedo viewer.

To view the results via the standard CodeRedo viewer, the user should click the option „Viewers" in the upper menu bar. A pulldown menu will appear which will offer two options

- Open a file with the internal viewer
- Call an external editor.

The internal viewer will display all of the source files in the output directory. The user needs only to click on a file to view its contents. It is recommended to take a close look at the results before going on to use them.


SoftRedo main shell - Cobol Redo [AuftragTest]
File Actions Viewers Logs Help
Names and datasets
Internal viewer
Edit macro table
TEST.log
TEST\COBOLD.CBL
REPLAC COPY COBOLD.CPY.
PROCEDURE DIVISION.
* INITIALIZE ORDER ENTRY PROCESSING
X-CONTROL SECTION.
X-CONTROL-LOOP.
MOVE 'INITIALIZATION' TO X-NEXT-LABEL.
PERFORM UNTIL X-NEXT-LABEL = 'STOP'
OR X-NEXT-LABEL = SPACES
IF X-NEXT-LABEL = 'INITIALIZATION'
PERFORM INITIALIZATION
END-IF
IF X-NEXT-LABEL = 'READ-ORDERS'
PERFORM READ-ORDERS
END-IF
IF X-NEXT-LABEL = 'READ-CUSTOMER'
PERFORM READ-CUSTOMER
END-IF
IF X-NEXT-LABEL = 'PROCESS-ORDER'
PERFORM PROCESS-ORDER
END-IF
IF X-NEXT-LABEL = 'WRITE-OPEN-POSITIONS'
PERFORM WRITE-OPEN-POSITIONS
END-IF
IF X-NEXT-LABEL = 'WRITE-DISPATCH'
PERFORM WRITE-DISPATCH
END-IF
IF X-NEXT-LABEL = 'REPORT-ORDER'
PERFORM REPORT-ORDER
END-IF
IF X-NEXT-LABEL = 'REPORT-ERROR'
PERFORM REPORT-ERROR
END-IF
IF X-NEXT-LABEL = 'PRINT-SUMMARY'
PERFORM PRINT-SUMMARY
END-IF
IF X-NEXT-LABEL = 'TERMINATION'
PERFORM TERMINATION
END-IF
END-PERFORM.
X-CONTROL-EXIT.
EXIT.
INITIALIZATION.
MOVE SPACES TO X-NEXT-LABEL.
PERFORM XOPEN0001.
PERFORM XOPEN0002.
PERFORM XOPEN0003.
PERFORM XOPEN0004.
PERFORM XOPEN0005.
PERFORM XOPEN0006.
Abschnitt: 1 Seite: 32 von 35 Wörter: 10.836 Englisch (USA)
100 %
Start
Total Commander 7.0...
SoftRedo_Eng_V01.d...
Wrapper application f...
DE
12:27

6 CodeRedo Reengineering Samples

6.1 Sample Macro Table

Lnr Lng Code Name
0001 04 PROC NOPR
0002 07 LINK OBJNAME
0003 06 FUNC YABVAN
0004 07 COPY YABZBER
0005 07 PROC YAD1401
0006 08 FUNC YADRESS1
0007 08 FUNC YADRESSE
0008 08 LINK YAGRCALL
0009 08 LINK YAGRCALT
0010 05 DB YAIBD
0011 04 FUNC YAKV
0012 05 CNTR YALIN
0013 07 FUNC YANREDE
0014 04 FUNC YAUS
0015 05 FUNC YBAUS
0016 05 FUNC YBEIN
0017 06 FUNC YBEIN1
0018 05 PROC YBMIX
0019 04 FUNC YBOP
0020 05 FUNC YBPPZ
0021 07 PROC YBTCALL
0022 05 CALL YCALL
0023 05 FUNC YCAUS
0024 05 FUNC YCEIN
0025 04 FUNC YCLC
0026 05 FUNC YCLEN
0027 05 FILE YCLOS
0028 05 FUNC YCMIX
0029 07 LINK YCOCALL
0030 06 COPY YCOMRG
0031 07 COPY YCOMRG2
0032 06 ENTR YCSANF
0033 06 EXIT YCSEND
0034 07 FUNC YCTIME0
0035 07 FUNC YCTIME1
0036 05 FUNC YDATE
0037 06 ENTR YDCANF
0038 04 FILE YDCB
0039 06 EXIT YDCEND
0040 07 FUNC YDFTRAC
0041 04 FUNC YDIV
0042 07 MASK YDLCALL
0043 06 DB YDLPCB

6.2 Sample Function Table

Lnr Lng Code Name
0001 10 FILE open
0002 10 FILE Open
0003 10 FILE Connect
0004 10 FILE FileStream
0005 10 FILE OpenWrite
0006 10 FILE OpenFile

0007 10 FILE BhBelegSAPFile
0008 10 FILE BhBeleg
0009 00 FILE Disconnect
0010 00 FILE close
0011 00 FILE Close
0012 00 FILE CloseFile
0013 06 READ Read
0014 06 READ ExecuteXmlReader
0015 06 READ ExecuteXmlTextReader
0016 06 READ ReadXml
0017 06 READ ReadLine
0018 06 READ ReadByte
0019 06 READ ReadtoEnd
0020 06 READ StreamReader
0021 04 INPT GetData
0022 04 INPT Seek
0023 04 INPT Load
0024 07 OUTP Transform
0025 07 OUTP SaveAs
0026 07 WRIT makeProtokoll
0027 07 WRIT StreamWriter
0028 07 WRIT Write
0029 07 WRIT WriteLine
0030 07 WRIT WriteXmlSchema
0031 07 WRIT WriteThru
0032 07 WRIT WriteXml
0033 07 WRIT writeSap
0034 07 OUTP Copy
0035 07 OUTP Flush
0036 07 OUTP AppendText
0037 07 OUTP Create
0038 07 OUTP Move
0039 07 OUTP Info
0040 07 OUTP Warn
0041 07 OUTP Error
0042 07 OUTP Fatal
0043 06 GET ReadMitglied

6.3 Sample Name File

OldName;NewName;
CRT-EIN;CRT-EINGABE;
CRT-AUS;CRT-AUSGABE;
RPARA;R-PARAMETER;
RPARA-ZE;R-PARAMETER-ZENTRALEINHEIT
KO;KASSENORDNUNG;
AP;APPLIKATION;
AR;ABRECHNUNG;
HB;HAUPTBEREICH;
PS;POSTESTELLE;
KOAN;KOSTENANNAHME;
KOAB;KOSTENABRECHNUNG;
KOPR;KOSTENPROGRAMM;
KOFR;KOSTENVERANSCHLAGUNG;
APAN;APPLIKATIONSANNAHME;
APAB;APPLIKATIONSABNAHME;
APFR;APPLIKATIONSVERANSCHLAGUNG;
APPR;APPLIKATIONSPROGRAMM;
ARAB;ABRECHNUNGSABNAHME;
ARAN;ABRECHNUNGSANNAHME;

ARFR;ABRECHNUNGSVERANSCHLAGUNG;
ARPR;ABRECHNUNGSPROGRAMM;
HBAN;HAUPTBEREICHSANNAHME;
HBAB;HAUPTBEREICHSABNAHME;
HBPR;HAUPTBEREICHSPROGRAMM;
HBFR;HAUPTBEREICHSVERANSCHLAGUNG;
PSAN;POSTSTELLEANNAHME;
PSAB;POSTSTELLEABNAHME;
PSFR;POSTSTELLEVERANSCHLAGUNG;
PSPR;POSTSTELLEPROGRAMM;
C-ORD1;C-ORDNUNG-1;
C-ORD2;C-ORDNUNG-2;
C-ANZ;C-ANZAHL;
C-ANZB;C-ANZAHL-BEZAHLT;
C-GEB;C-GEBUEHR;
C-GEB6;C-GEBUEHR-6;
C-MKZ;C-MEDIANKZ;
C-IND;C-INDEX;
C-LFNR;C-LAUFENDE-NUMMER;
C-MLDG-OK;C-MELDUNG-OK;
C-MLDG-WEITER;C-MELDUNG-WEITER;
C-LNFNR9;C-LAUFNR-9;
ZKTBL;ZENTRALKOSTENTABELLE;
ZKTNRB;ZENTRALKOSTENNUMMER;
ZPNAM;ZENTRALPOSTNAME;
ZPADR;ZENTRALPOSTADDRESSE;
ZK-UST;ZENTRALKASSE-UMSATZSTEUER;
ZK-BRF;ZENTRALKASSE-BERECHNUNGSFORMEL;
ZFUNK2;ZENTRALFUNKTION-2;
ZFUNK19;ZENTRALFUNKTION-19;
ZK-FRI;ZENTRALKASSE-FREITAG;
ZK-SKO;ZENTRALKASSE-ORDNUNG;
ZI-FAM;ZI-FAMILIENNAME;
ZI-VOR;ZI-VORNAME;
ZI-SUCH;ZI-SUCHAKTION;
ZPNR-TEL;ZP-TELEFONNR;
ZPNR-UNT;ZP-UNTERNEHMERNUMMER;
ZP05-SI;ZP-05-SIGNATUR;
ZP05-SIA;ZP-05-SIGNALE;
ZTB-ANF;ZTB-ANFANG;
K-KVNR;KUNDEN-KRANKENVERSICHERUNGSNUMMER;
K-GEBDAT;KUNDEN-GEBURTSDATUM;
K-GEBT;KUNDEN-GEBURTSTAG;
K-GEBM;KUNDEN-GEBURTSMONAT;
K-GEBJ;KUNDEN-GEBURTSJAHR;
K-MEHRL;KUNDEN-MEHRLEISTUNG;
K-FAMNAM;KUNDEN-FAMILIENNAME;
K-VORNAM;KUNDEN-VORNAME;
K-TITELKZ;KUNDEN-TITEL;
K-PLZ;KUNDENPOSTLEITZAHL;
K-STR;KUNDENSTRASSE;
K-GESCHL;KUNDENGESCHLECT;

6.4 Sample Batch Process Interface

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--DOCTYPE CODEREDO SYSTEM "Redo.dtd"-->
<REDO>
 <Names
  ProductName = "AUFTRAG"
  SystemName  = "TEST"
  Version     = "COB74"  />
 <Parameters
  RefineCode = “Y”
  ReplaceCon  = "Y"
  ReplaceLit  = "Y"
  RelocateFile = "Y"
  RelocateDB   = "Y"
  RelocateTP   = "Y"
  RestructCode = "Y"
  RefactorLev  = "5"  />
 <Directories>
  <Orig_Output_Dir>C:\DATA\COB\RECONOUT</Orig_Output_Dir>
  <Input_Dir>workinp</Input_Dir>
  <Output_Dir>workout</Output_Dir>
  <Name_Tab>names.csv</Name_Tab>
  <Work_Dir>work</Work_Dir>
  <Func_Tab>functab.wrl</Func_Tab>
 </Directories>
 <Files>
  <Orig_File_0000>C:\DATA\COB\INPUTS\TEST\COBOLD.CBL</Orig_File_0000>
  <File_0000>workout\FIL0000.C80</File_0000>
  <Orig_File_0001>C:\DATA\COB\INPUTS\TEST\XM059.COB</Orig_File_0001>
  <File_0001>workout\FIL0001.C80</File_0001>
 </Files>
</REDO>
```

6.5 Sample of a Renovated Assembler CSect

```
******** Section=KV0226  *****************
KV0226   ZSTRT BASIS=R11,REGS=2
         L     R3,0(R1)
         USING ARB,R3
         L     R10,SAVA            SAVE-BEREICH T025
         L     R5,WT025A
         USING T025F01,R5
         L     R4,WT010A
         USING T010F01,R4
*
         IF   (T010F01,EQ,SBEH)
         B     Z2261
*=====>* GOTO branch forward to Z2261
         ELSE
         MVC   SBEH,T010F01
****     perform subroutine
         BAL   R14,Z226U1           EINLESEN BEST-TAB
         ENDIF
*
         IF   (T010F471,NE,LIT001)               ?
         B     Z22692               NEIN
*=====>* GOTO branch forward to Z22692
         ELSE
         SPACE
         ENDIF
```

```
******** Label=Z2261   *****************
Z2261    DS    0H
         SPACE
*
         IF   (T025F021,NE,LIT002)
         B     Z2269               ENDE,KEINE STROM GEB.SAETZE
*=====>* GOTO branch forward to Z2269
         ELSE
         ENDIF
*
         IF   (T025F86,NE,LIT003)                R
         B     Z2268               ERH.UM 1 GEB.SATZ
*=====>* GOTO branch forward to Z2268
         ELSE
         ENDIF
*
         IF   (T025F36,NE,LIT004)                SATZ GEWUENSCHT
         B     Z2263               ABER KEIN TARIFWECHSEL
*=====>* GOTO branch forward to Z2263
         ELSE
         MVI   T025F36,LIT005
         B     Z2268               JA
*=====>* GOTO branch forward to Z2268
         ENDIF
******** Label=Z2263   *****************
Z2263    DS    0H
*
         IF   (T025F36,NE,LIT006)                SATZ GEWUENSCHT,
         B     Z2262               ABER KEIN VERBR.- + GRUNDPREIS,
*=====>* GOTO branch forward to Z2262
         ELSE
*        MVC   FEHLTEXT(L'Z22FEHL8),Z22FEHL8  ODER KEIN TARIFWECHSEL
*        MVC   FEHLTEXT+L'Z22FEHL8-4(L'T025F03),T025F03
*        BAL   R14,Z22PROT
         MVI   T025F36,LIT007
         B     Z2268               JA
*=====>* GOTO branch forward to Z2268
         ENDIF
******** Label=Z2262   *****************
Z2262    DS    0H
*
         IF   (T025F36,EQ,LIT008)                .
         B     Z2268               JA
*=====>* GOTO branch forward to Z2268
         ELSE
         ENDIF
*
         IF   (T025F36,EQ,LIT002)                 BESTABGER.
         B     Z2268               JA
*=====>* GOTO branch forward to Z2268
         ELSE
         ENDIF
*
         IF   (T025F36,EQ,LIT007)                . DA FEHLER
         B     Z2268               JA
*=====>* GOTO branch forward to Z2268
         ELSE
         ENDIF
*
         IF   (T025F36,EQ,LIT009)                HNE LIN.KOMPO.
         B     Z2268               JA
*=====>* GOTO branch forward to Z2268
         ELSE
```

```
         ENDIF
*
         IF   (T025F36,EQ,LIT005)                . OHNE LIN.KOMPO.
         B     Z2268                JA
*=====>* GOTO branch forward to Z2268
         ELSE
         SPACE
* LOESCHEN  FELDER PRO TARIF (ALLE GEB.SAETZE)
*        XC    GEBA1,GEBA1
         XC    GEBA2,GEBA2
         MVI   ZUSCHL1,LIT010
         MVI   KZLIN0,LIT010
         MVC   ZUSCHL1+1(L'BZAEHL1-1),ZUSCHL1
         ZAP   KZZUS,KP0
         ZAP   ZUBEL,KP0
         ZAP   REFTA1,KP0
         ZAP   REFTA2,KP0
         ZAP   REFTA3,KP0
         ZAP   REFTA4,KP0
         ZAP   REFLI,KP0
         ZAP   KZTAA,KP0
         ZAP   KZTAN,KP0
         ZAP   SMENG,KP0
         ZAP   LMENG,KP0
         XC    KZLK,KZLK
         XC    KZBM,KZBM
         ZAP   RF1,KP0
         ZAP   SPREI,KP0
         XC    BKZE,BKZE
         SPACE
****     perform subroutine
         BAL   R14,Z226U2           ERMITTELN SUM ALLER TARIFE
****     perform subroutine
         BAL   R14,Z226U3          UP VEGL. 3 TARIFE
         L     R5,WT025A
         L     R10,SAVA
         B     Z22681
*=====>* GOTO branch forward to Z22681
         ENDIF
******** Label=Z2268   *****************
Z2268    DS    0H
         A     R5,WLT025
         A     R10,WLT025
******** Label=Z22681  *****************
Z22681   DS    0H
*
         IF   (R5,NL,WT025)
         B     Z2269
*=====>* GOTO branch forward to Z2269
         ELSE
         ENDIF
******** Label=Z2267   *****************
Z2267    DS    0H
         B     Z2261
*=====>* GOTO branch forward to Z2261
******** Label=Z2269   *****************
Z2269    DS    0H
         SPACE
**       LOESCHUNG       ALLER       INTERNEN       KENNUNGEN       (T035F36)
03390000
         L     R5,WT025A
******** Label=Z22693  *****************
Z22693   DS    0H
```

```
*
         IF   (T025F021,NE,LIT002)              ?
         B     Z22692                     NEIN
*=====>* GOTO branch forward to Z22692
         ELSE
         ENDIF
*
         IF   (T025F36,EQ,LIT008)               BESTABR. ?
         B     Z22691                     JA
*=====>* GOTO branch forward to Z22691
         ELSE
         ENDIF
*
         IF   (T025F86,NE,LIT003)               CHNET ?
         B     Z22691                     NEIN
*=====>* GOTO branch forward to Z22691
         ELSE
         AP    WACC3,KP1                  ZAEHLEN SAETZE F.ACCOUNT
         ENDIF
*
         IF   (T025F36,EQ,LIT005)               BESTABR. ?
         B     Z22696                     JA
*=====>* GOTO branch forward to Z22696
         ELSE
         ENDIF
*
         IF   (T025F36,NE,LIT009)               LIN.KOMPO. ?
         B     Z22694                     NEIN
*=====>* GOTO branch forward to Z22694
         ELSE
         ENDIF
******** Label=Z22696  *****************
Z22696   DS    0H
         MVI   T025F36,LIT011             KEINE LIN.KOMPO.
         B     Z22691
*=====>* GOTO branch forward to Z22691
******** Label=Z22694  *****************
Z22694   DS    0H
         XC    T025F36,T025F36            LOESCHEN INT.KZ.
******** Label=Z22691  *****************
Z22691   DS    0H
         A     R5,WLT025
*
         IF   (R5,NL,WT025)
         B     Z22692
*=====>* GOTO branch forward to Z22692
         ELSE
         B     Z22693
*=====>* GOTO branch forward to Z22693
         ENDIF
******** Label=Z22692  *****************
Z22692   DS    0H
         ZEND
         EJECT
*   E   I   N   L   E   S   E   N     D   E   R       B   E   S   T   -   T   A   B
DS       F
******** Label=Z226U1  *****************
Z226U1   ST    R14,Z226U1-4
         PACK  RF1,T010F01
         MP    RF1,KP10
         MVC   BHNR,RF1+L'RF1-L'BHNR-1
         ZGKY  SYS030,BQU,WRAB,BHNR,BHNR
*
```

```
         IF   (BQU,EQ,LIT012)
         B     Z226U12             VORHANDEN
*=====>* GOTO branch forward to Z226U12
         ELSE
         XC    T010F471,T010F471        LOESCHEN KZ-BEST
         MVC   FEHLTEXT(L'Z22FEHL0),Z22FEHL0
****     perform subroutine
         BAL   R14,Z22PROT
         MVI   BKZE,LIT001              KZ=KEINE BESTABR
         B     Z226U19
*=====>* GOTO branch forward to Z226U19
         ENDIF
******** Label=Z226U12 *****************
Z226U12  DS    0H
         L     R14,WRAB
         A     R14,LBST
         ST    R14,WREB
******** Label=Z226U19 *****************
Z226U19  DS    0H
         L     R14,Z226U1-4
****     return to caller
         BR    R14
         EJECT
* E R M I T T E L N   S U M M E N   A L L E R  G E B . S A E T Z E
* ( P R O   T A R I F )
         DS    F
******** Label=Z226U2  *****************
Z226U2   ST    R14,Z226U2-4
         L     R6,WRAB
         USING BESTSA,R6           TABELLE BEST-ABR.
         LA    R7,BZAEHL1
         USING BTF,R7              TARIF
         L     R15,SAVA
         L     R14,WT025A
         ZMVCL R15,SAV,R14             SICHERN  T025
         B     Z226U254
*<=====* GOTO branch backward to Z226U254
         SPACE
*   BERECHNEN   DER   1.BEMESSUNGSZAHL(TARIF   I   -   KLEINSTVERBRAUCHER)
04010000
*
04020000
******** Label=Z226U253*****************
Z226U253 DS    0H
****     perform subroutine
         BAL   R14,Z22T025      ZURUECKUEBERTRAGEN BEM. ZAHL 1 T025
******** Label=Z226U254*****************
Z226U254 DS    0H
         MVI   SBEMZ1,LIT010
         MVI   SBEMZ2,LIT010
         MVI   SBEMZ3,LIT010
         MVI   KZLIN1,LIT010
         MVI   KZLIN2,LIT010
         MVI   KZLIN3,LIT010
         ZAP   KZTAA1,KP0
         ZAP   KZTAA2,KP0
         ZAP   KZTAA3,KP0
         L     R6,WRAB
*
         IF   (T025F27,EQ,LIT013)                NR. BELEGT?
         B     Z226U25                    NEIN
*=====>* GOTO branch forward to Z226U25
         ELSE
```

```
         LA    R15,0
         LA    R7,L'BESTSA1
         IC    R15,T025F27                 ZEILEN NR.
*
         DOWHILE(T025F27,EQ,LIT013)              NR. BELEGT?
         BR    R15
         ENDDO
*<====>* GOTO branch to register R15
*
         MR    R14,R7
         AR    R6,R15
         B     Z226U251
*=====>* GOTO branch forward to Z226U251
         ENDIF
******** Label=Z226U25 *****************
Z226U25  DS    0H
*
         IF   (R6,NL,WREB)
         B     Z226U26
*=====>* GOTO branch forward to Z226U26
         ELSE
         ENDIF
*
         IF   (T025F042,NE,KP0)                 ELEGT ?
         B     Z226U252                  JA
*=====>* GOTO branch forward to Z226U252
         ELSE
         DROP  R5
         USING T025F01,R10                SAVE-BEREICH
         XC    T025F041,T025F041          LOESCHEN ZEI.NR. BEM.ZAHL
         XC    T025F27,T025F27            LOESCHEN ZEI.NR. BEM.ZAHL
         DROP  R10
         USING T025F01,R5                 BEST-BEREICH
         B     Z226U26                    NEIN
*=====>* GOTO branch forward to Z226U26
         ENDIF
******** Label=Z226U252*****************
Z226U252 DS    0H
*
         IF   (BZAEHL1(L'BZAEHL1),EQ,LEER)       IN BESTAB NICHT BELEGT
         B     Z226U28
*=====>* GOTO branch forward to Z226U28
         ELSE
         ENDIF
******** Label=Z226U251*****************
Z226U251 DS    0H
         LA    R7,BZAEHL1
         MVC   BT025+3(1),T025F041        UEBERTRAGEN
         MVC   BT025+4(1),T025F08
         MVC   BT025+5(2),T025F40       AUS T025  FUER
         MVC   BT025(3),T025F03         1.GEB.SATZ
         MVC   GEBA1,T025F03            SICHERN TARIF SCHLUESSEL
         LA    R14,4                    TARIF SUCHEN (AUCH LIN.KOMPO)
******** Label=Z226U27 *****************
Z226U27  DS    0H
         ST    R14,SR14
*
         IF   (BTF00,NE,BT025)
         B     Z226U24
*=====>* GOTO branch forward to Z226U24
         ELSE
         MVC   KZLIN1,BLIKZ             LIN.KOMPO.1.BEMZ.
         AP    KZTAA1,KP1               KZ TARIF ALT (1.BEMZ.)
```

```
         LA    R7,BZAEHL1
         ST    R7,SBANF                 1. TARIF /1.GEB.SATZ
         LA    R9,REFTA1                RECHF. TARIF I
         LA    R14,3                   3 TARIFE
         ZAP   WACC1,KP1               FUER SAVE UNTERZ.TAB.
         ENDIF
******** Label=Z226U21 *****************
Z226U21  DS    0H
         ST    R14,SR14
         XC    T025F07,T025F07         MESSPREIS NICHT FUER BEST
         XC    T025F051,T025F051       LOESCHEN
         XC    T025F061,T025F061       LOESCHEN
         MVC   T025F08,BTF04           PR.EINH.
         MVC   T025F041,BTF01          UEBERTR. FUER
         SPACE
*
         IF   (T025F08,EQ,0)                    EIT ?
         B     Z226U215
*=====>* GOTO branch forward to Z226U215
         ELSE
         ENDIF
*
         IF   (T025F023,EQ,CON001)              ER ?
         B     Z226U219                JA
*=====>* GOTO branch forward to Z226U219
         ELSE
         ZAP   WACC2,KP0               NEIN UNTERZAEHLER
         B     Z226U220
*=====>* GOTO branch forward to Z226U220
         ENDIF
******** Label=Z226U219*****************
Z226U219 DS    0H
         AP    WACC2,KP1               FUER SAVE UNTERZ.TAB.(ALLE 3
******** Label=Z226U220*****************
Z226U220 DS    0H                      TARIFE VERBR.UZ.ABZIEHEN)
         MVI   QHI,LIT010              FUER   KV0213
         ST    R5,AFUEB                FUER   KV0213
         ZCALL KV0213,/ARB             VERBRPR. ERMITTELN
         AP    0(L'REFTA1,R9),T025F72  VERBRAUSPREIS
         ZAP   WACC1,KP0               FUER SAVE UNTERZ.TAB.
******** Label=Z226U212*****************
Z226U212 DS    0H
*
……..……..…
……..……..
         PRINT GEN
         YVDS  DSECT,ARB,ARB
         YVDS  DSECT,T01A,T01
         YVDS  DSECT,T02A,T02
*        END   KV0226
*********************************************************************
         TITLE ' ASSEMBLER IO ROUTINES '
XIORETRN DC    F                IO Return Address
*********************************************************************
*********************************************************************
*RECON   TABLE OF LITERAL CONSTANTS
LIT001   EQU   C'1'
LIT002   EQU   C'E'
LIT003   EQU   C'3'
LIT004   EQU   C'D'
LIT005   EQU   C'G'
LIT006   EQU   C'A'
LIT007   EQU   C'C'
```

```
LIT008   EQU   C'B'
LIT009   EQU   C'F'
LIT010   EQU   C' '
LIT011   EQU   C'L'
LIT012   EQU   C'0'
LIT013   EQU   X'00'
CON001   EQU   240
LIT014   EQU   C'2'
LIT015   EQU   X'F0'
*RECON   END OF LITERAL CONSTANTS
*********************************************************************
         END   KV0226
```

6.6 Sample of a Renovated PLI Procedure

```
DLITPLI: PROC(PCB_BE,PCB_TAA,
              PCB_TA,PCB_VA,
              COIBM,PLIXOPT,PLIXHD)
              OPTIONS(MAIN)  REORDER ;
 %INCLUDE P2715; /* SYMBOLIC CONSTANTS & LITERALS  */

 /*******************************************************************/
 /*                                                                 */
 /* UPD.-#:    DATUM   UPDATE-BEZEICHNUNG UND/ODER -GRUND           */
 /* -------  --------  ------------------------------------------- */
 /*                                                                 */
 /* BVO001   20.10.83  SKIP AUF NEUE SEITE FEHLT (ON ENDPAGE)      */
 /*                                                                 */
 /*                                                                 */
 /*******************************************************************/
 /*************************************************************/
 /*                                                           */
 /*       PARAMETER - DECLARATIONEN:                          */
 /*       --------------------------                          */
 /*                                                           */
 /*************************************************************/
 DCL COIBM CHAR(76);
 /* DECLARES   */
 DCL SUCHKEY         CHAR(04)        INIT('') ;
 DCL LINECOUNTER      DECFIXED(04,0) INIT('') ;          /*CONVERTED*/
 DCL COUNTER         PIC '9(15)'     INIT('') ;          /*CONVERTED*/
 DCL TAWERT_Z        PIC 'Z.ZZZ.ZZZ.ZZ9V,99' ;
 DCL TAWERT_NEU      CHAR(16) ;
    DCL BSTKNOM         PIC '9(16)' INIT(0) ;           /*CONVERTED*/
    DCL BNOMSFRS        PIC '9(16)' INIT(0) ;           /*CONVERTED*/
    DCL HFLOAT_1        PIC '9(16)' INIT(0) ;           /*CONVERTED*/
    DCL HFLOAT_2        PIC '9(16)' INIT(0) ;           /*CONVERTED*/
    DCL HFLOAT_3        PIC '9(16)' INIT(0) ;           /*CONVERTED*/
 DCL STEUER_VALUE    PIC '(1)9'  INIT('') ;
 DCL LOOP_ERROR_COUNT       DECFIXED (04,0) INIT(0) ;   /*CONVERTED*/
 DCL AKTUELLE_AKTSTUFE     CHAR(1) INIT('') ;
 /* OPERATOR FUER DLI-ZUGRIFFE           */
 DCL GLOBAL_OPERATOR CHAR(2) INIT(' =') ;
 /* ZAEHLERFELDER FUER LISTEN_OUTPUT     */
 DCL VALOREN_I       PIC '9(05)'   INIT(0) ;           /*CONVERTED*/
 DCL VALOREN_O       PIC '9(05)'   INIT(0) ;           /*CONVERTED*/
 DCL BESTAND_I       PIC '9(05)'   INIT(0) ;           /*CONVERTED*/
 DCL BESTAND_O       PIC '9(05)'   INIT(0) ;           /*CONVERTED*/
 DCL TABELLEN_I      PIC '9(05)'   INIT(0) ;           /*CONVERTED*/
 DCL TABELLEN_O      PIC '9(05)'   INIT(0) ;           /*CONVERTED*/
```

```
DCL MARCHZINSEN             PIC '9(11.2)'    ;           /*CONVERTED*/
DCL MARCHZ_MELDUNG           CHAR(1)          INIT('0') ;;
DCL PARM_FAELL              CHAR    (10)     INIT('') ;
DCL PARM_SICHERST           PIC     '(8)9'   INIT('') ;
DCL PARM_LZPERB             CHAR    (10)     INIT('') ;
DCL PARM_LZPERV             CHAR    (10)     INIT('') ;
DCL PARM_EVERF              CHAR    (10)     INIT('') ;
DCL PARM_STKNOM             PIC '9(13.2)'             ;  /*CONVERTED*/
DCL PARM_ZSATZ              PIC '9(07.5)'             ;  /*CONVERTED*/
DCL PARM_WAEK               PIC     '999V9999'        ;
DCL PARM_WAEE               PIC     '999'             ;
DCL PARM_USANZ              CHAR    (1)      INIT('') ;
DCL INDEX_GEFUNDEN           CHAR(1)          INIT('0') ;;
DCL I_COUNTER                DECFIXED (04)   INIT(0) ;   /*CONVERTED*/
DCL AUSWAHL_PTR PTR ;
DCL CONVERSIONS_FELD_1      CHAR(11) INIT('') ;
DCL CONVERSIONS_FELD_2      CHAR(11) INIT('') ;
DCL CONVERSIONS_FELD_3      CHAR(12) INIT('') ;
DCL KEY_FELD                CHAR(33) INIT('') ;
DCL (ONE,TWO,THREE,FOUR,FIVE,SIX)  DECFIXED (08,0) ;      /*CONVERTED*/
    ONE   = 1 ;
    TWO   = 2 ;
    THREE = XCON3 ;
    FOUR  = XCON4 ;
    FIVE  = XCON5 ;
    SIX   = XCON6 ;
DCL  BESTAND_DB         PIC '9(8).9'  INIT (1) ;       /*CONVERTED*/
DCL  TABELLEN_DBA       PIC '9(8).9'  INIT (2) ;       /*CONVERTED*/
DCL  TABELLEN_DB        PIC '9(8).9'  INIT (3) ;       /*CONVERTED*/
DCL  VALOREN_DB         PIC '9(8).9'  INIT (4) ;       /*CONVERTED*/
DCL  1 SSA_BE      ,
       3  SEGMENT CHAR  (  8)  INIT ('BESTAND') ,
       3  KEYNAME CHAR  (  9)   INIT ('(BESKEY') ,
       3  OP      CHAR  (  2)   INIT (' =') ,
       3  KEY     CHAR  ( 32)   INIT ('')  ,
       3  FILLER  CHAR  (  1)  INIT(')') ;
DCL  1 SSA_TA      ,
       3  SEGMENT CHAR  (  8)  INIT ('TABEL') ,
       3  KEYNAME CHAR  (  9)   INIT ('(TABKEY') ,
       3  OP      CHAR  (  2)   INIT (' =') ,
       3  KEY     CHAR  ( 15)   INIT ('')  ,
       3  FILLER  CHAR  (  1)  INIT(')') ;
DCL  1 SSA_TAA     ,
       3  SEGMENT CHAR  (  8)  INIT ('TABEL') ,
       3  KEYNAME CHAR  (  9)   INIT ('(TABKEY') ,
       3  OP      CHAR  (  2)   INIT (' =') ,
       3  KEY     CHAR  ( 15)   INIT ('')  ,
       3  FILLER  CHAR  (  1)  INIT(')') ;
DCL  1 SSA_VA      ,
       3  SEGMENT CHAR  (  8)  INIT ('VALOR') ,
       3  KEYNAME CHAR  (  9)   INIT ('(VALKEY') ,
       3  OP      CHAR  (  2)   INIT (' =') ,
       3  KEY     CHAR  ( 32)   INIT ('')  ,
       3  FILLER  CHAR  (  1)  INIT(')') ;
DCL  1 SSA_BEU     ,
       3  SEGMENT CHAR  (  9)  INIT ('BESTAND') ;
DCL  1 SSA_TAU     ,
       3  SEGMENT CHAR  (  9)  INIT ('TABEL  ') ;
DCL  1 SSA_TAAU    ,
       3  SEGMENT CHAR  (  9)  INIT ('TABEL  ') ;
DCL  1 SSA_VAU     ,
       3  SEGMENT CHAR  (  9)  INIT ('VALOR  ') ;
/*************************************************************/
```

```
/*                                                            */
/*       PLIXOPT:    SPEZIFIKATION EXECUTION-TIME-OPTION      */
/*                                                            */
/*       PLIXHD:     IDENTIFIZIEREN DES COUNT/REPORT-OUTPUTS */
/*                                                            */
/*                   ====>  LAENGE ABHAENGIG VON INIT-STRING */
/************************************************************/
        DCL      PLIXOPT  CHAR(12) VAR STATIC;
        DCL      PLIXHD   CHAR(50) VAR STATIC;
/************************************************************/
/*                                                          */
/*       CALL'S:                                            */
/*                                                          */
/************************************************************/
        DCL      DCDATA  ENTRY(PIC 9(5),PIC 9( 7)) RETURNS(PIC S9(5)) ;
        DCL      DCDATB  ENTRY(PIC 9(5),PIC S9(5)) RETURNS(PIC 9( 7)) ;
        DCL      DCDATC  ENTRY(PIC 9(5),PIC 9( 7)) RETURNS(PIC S9(5)) ;
        DCL      DCDATD  ENTRY(PIC 9(5),PIC S9(5)) RETURNS(PIC S9(5)) ;
        DCL      DCDT1   ENTRY(PIC 9(7),PIC 9( 7)) RETURNS(PIC 9( 7)) ;
        DCL      DCDT2   ENTRY(PIC 9(7),PIC 9( 7)) RETURNS(PIC 9( 7)) ;
        DCL      DCDT3   ENTRY(PIC 9(7))           RETURNS(PIC S9(5)) ;
        DCL      DCDT4   ENTRY(PIC 9(7))           RETURNS(PIC 9( 7)) ;
        DCL      DCDT5   ENTRY(PIC 9(7))           RETURNS(PIC 9( 7)) ;
        DCL      DCDT6   ENTRY(PIC 9(7))           RETURNS(PIC 9( 7)) ;
        DCL      DCTABA  ENTRY RETURNS(PIC S9(5)) ;
        DCL      DCTABB  ENTRY RETURNS(PIC S9(5)) ;
        DCL      DCTABC  ENTRY RETURNS(PIC S9(5)) ;
        DCL      PLITDLI    ENTRY ;
        /************************************************************/
        /*                                                          */
        /*       FILE - DECLARATIONEN:                              */
        /*       ---------------------                              */
        /*                                                          */
        /************************************************************/
 DCL    LISTE       FILE    OUTPUT    PRINT
           ENV(BUFFERS(2)) ;
        DCL    SYSPRINT    FILE    OUTPUT    PRINT
                           ENV(BUFFERS(1)) ;
        DCL    SYSIN       FILE    INPUT
                           ENV(BUFFERS(1)) ;
/************************************************************/
/*                                                          */
/*      %INCLUDE:   DATENBANK - DEKLARATIONEN               */
/*                                                          */
/************************************************************/
        %NOPRINT ;
        %INCLUDE  P2700BE  ;                /*       BESTAND       */
        %INCLUDE  P2700DAT ;                /*       DATUM-RECORD  */
        %INCLUDE  P2700VA  ;                /*       VALOR         */
                                            /* TABELLEN-DECLARES   */
                                            /* -----------------   */

        /************************************************************/
        /*                                                          */
        /*    DECLARATIONEN:           HILFSFELDER                  */
        /*    --------------           TABELLEN                     */
        /*                             BUILTIN-FUNCTIONS            */
        /*                                                          */
        /************************************************************/
        %PRINT ;
        DCL  (PBEST,PDAT,PMARCH,PVAL,PVRM,PTAB,PREP,PKON) PTR ;
        PBEST = ADDR(IO_BE) ;
        PVAL  = ADDR(IO_VA) ;
```

```
        PTAB  = ADDR(IO_TA) ;
        DCL    FETAB(10) CHAR(30) INIT('AAAA','BBBB','CCCC','DDDD',
                                       'EEEE','FFFF','GGGG','HHHH',
                                       'IIII','JJJJ');
        DCL    EOF      CHAR    ( 1)  INIT  (' ') ;
        DCL    I        PIC '9(04)'                ;      /*CONVERTED*/
        DCL    J        PIC '9(04)'                ;      /*CONVERTED*/
        DCL    IFE      PIC '9(04)'                ;      /*CONVERTED*/
        DCL    ABEND    ENTRY   (       PIC S9(9) ) ;
        DCL    (ADDR,TRANSLATE,VERIFY,INDEX,LOW,HIGH,PLIRETC,
                PLISRTD,SUBSTR,PLIDUMP,ANY,ALL,LENGTH,
                DATE,TIME,ONCODE,STRING,ONCHAR,ONSOURCE,
                SUM)             BUILTIN ;
DCL HERKUNFT         CHAR(10) INIT('') ;
DCL 1 F_AKTUELL (5) ,
      2 F_DATE       CHAR (6) INIT('') ,
      2 F_TIME       CHAR (9) INIT('') ;
DCL BEDINGUNG  CHAR15) INIT('000011111111111') ;;         /*CONVERTED*/
DCL AKTUELLES_DATUM  CHAR ( 8) INIT('') ;
/*******************************************************/
/*                                                     */
/*    SCHLUSSBLATT - ANGABEN                           */
/*                                                     */
/*******************************************************/
        DCL    CT(1:20)          PIC '9(05)'   ;          /*CONVERTED*/
        DCL    CTT(1:20)         CHAR    ( 23)   INIT
             ('ANZAHL EINGELESENE     ','SEGMENT               ',
              '                       ','VALOR                 ',
              '                       ','TABELLEN              ',
              '                       ',' '                    ',
              '                       ',' '                    ',
              'ANZAHL MUTIERTE        ','VALOREN               ',
              '                       ','TABELLEN              ',
              '                       ',' '                    ',
              '                       ',' '                    ',
              'ANZAHL SEITEN          ','ANZAHL ZEILEN         ') ;
        DCL    DATUM             CHAR    ( 8) ,
               ZEIT              CHAR    (12) ;
/*******************************************************/
/*                                                     */
/*    IMS / VS  DEFINITIONEN:                          */
/*                                                     */
/*    ======>   WAREA (WORKAREA) MUSS SOLANGE GEWAEHLT */
/*              WERDEN, DASS DER LAENGSTMOEGLICHE PATH */
/*              (SEGMENTLEVEL 1 BIS X (MAX. 15)) INKL. */
/*              RESERVE DARIN PLATZ FINDET.            */
/*                                                     */
/*******************************************************/
        DCL      DB_NR       DECFIXED (04) ;             /*CONVERTED*/
        DCL      IO_BE       CHAR   (2000) ,
                 IO_VA       CHAR   (2000) ,
                 IO_TA       CHAR   (2000) ;
DCL IO_TAA       CHAR(2000)  BASED(PTAB) ;
DCL DATUM_I            CHAR(8) INIT('') ;
                                          /**********************/
                                          /*  PCB - MASKE       */
                                          /**********************/
        DCL    PCB_PTR    PTR ;
        DCL  1   PCB      BASED (PCB_PTR) ,
               3 DBNAME   CHAR  (   8) ,     /*  DB NAME            */
               3 SEGL     CHAR  (   2) ,     /*  SEGMENT-LEVEL      */
               3 STATUS   CHAR  (   2) ,     /*  STATUS-CODE        */
               3 F1       CHAR  (   4) ,     /*  IRRELEVANT IMS-INT */
```

```
              3 F2        PIC '9(08)',      /*  IMS-INTERN         */
              3 SNAME    CHAR  (   8) ,     /*  SEGMENT-NAME       */
              3 DKEYL     PIC '9(08)',      /*  KEY-LAENGE         */
              3 F3        PIC '9(08)',      /*  IMS-INTERN         */
              3 KEY      CHAR  (  32) ;     /*  KEY-FEEDBACKAREA   */
        DCL  1   PCBX     DEFINED PCB,
              3 XNAME    CHAR  (   8) ,     /*  DB NAME            */
              3 XREST    CHAR  (  60) ;     /*  DB DATA            */
                                            /*   UNTERTEILBAR      */
        /*************************************************************/
        /*                                                           */
        /*      DL/I - ANGABEN:                                      */
        /*                                                           */
        /*************************************************************/
                                            /***********************/
                                            /*  DL/I-FUNKTIONEN    */
                                            /***********************/
           3  FILLER               CHAR(12);
/* REDEFINING GROUP LENGTH DOES NOT MATCH REDEFINED!*/
/* REDEFINED  GROUP LENGTH = 0080 */
/* REDEFINING GROUP LENGTH = 0068 */
/* DIFFERENCE IS           = 12   */
/* GENERATED FILLER MUST PAD THE SHORTER GROUP!     */
        DCL  1   FUNC ,
              3 GU       CHAR  (   4)    INIT('GU  ') ,
              3 GHU      CHAR  (   4)    INIT('GHU ') ,
              3 GN       CHAR  (   4)    INIT('GN  ') ,
              3 GHN      CHAR  (   4)    INIT('GHN ') ,
              3 GNP      CHAR  (   4)    INIT('GNP ') ,
              3 GHNP     CHAR  (   4)    INIT('GHNP') ,
              3 DLET     CHAR  (   4)    INIT('DLET') ,
              3 REPL     CHAR  (   4)    INIT('REPL') ,
              3 ISRT     CHAR  (   4)    INIT('ISRT') ;
                                          /************************/
                                          /* DL/I-PARAMETER COUNT */
                                          /************************/
        DCL  1   PARC ,
              3 C3        PIC '9(08)'    INIT(3) ,       /*CONVERTED*/
              3 C4        PIC '9(08)'    INIT(4) ,       /*CONVERTED*/
              3 C5        PIC '9(08)'    INIT(5) ,       /*CONVERTED*/
              3 C6        PIC '9(08)'    INIT(6) ;       /*CONVERTED*/
        /*************************************************************/
        /*                                                           */
        /*        O N - E R R O R                                    */
        /*                                                           */
        /*************************************************************/
DCL PLIRETCODE    DECFIXED (08) INIT (0) ;               /*CONVERTED*/
ON  ERROR
CALL P2715_002;
/*-----------------------------------------------*/
 P2715_002: PROC;
  ON ERROR SYSTEM ;
/* ON ERROR CALL ABEND(999) */     /**** ERROR IN ON-ERROR **/
  LOOP_ERROR_COUNT = LOOP_ERROR_COUNT + 1 ;
  IF LOOP_ERROR_COUNT > XCON5
  THEN
  DO ;
     CALL X_PUT_001; /* RELOCATED */
    STOP ;
  END ;
   CALL X_CLOS_002; /* RELOCATED */
  CALL PLIRETC(PLIRETCODE) ;
  IF PLIRETCODE > XCON4
```

```
  THEN
  DO ;
    IFE = 0 ;
    CALL FEHLER(IFE,PCB_PTR) ;
  END ;
  /* GO TO PROGRAMM_ENDE REMOVED !! */
   RETURN;
  CALL   PLIDUMP('TFHCBA','P2715') ;
/* CALL   ABEND(2714)      */
END ;
ON CONVERSION
CALL P2715_003;
/*-----------------------------------------------------*/
 P2715_003: PROC;
  ON ERROR SYSTEM ;
  IF ONCODE ^= XCON612
  THEN
  DO ;
     CALL X_PUT_003; /* RELOCATED */
    PLIRETCODE = XCON12 ;
    SIGNAL ERROR ;
  END ;
  ELSE
  DO ;
     CALL X_PUT_004; /* RELOCATED */
    ONSOURCE = SUBSTR(XLIT007,1,
    LENGTH(ONSOURCE)) ;
  END ;
END ;
DCL FIXED_ERROR_COUNT    DECFIXED (04) INIT(0) ;          /*CONVERTED*/
ON FIXEDOVERFLOW
CALL P2715_004;
/*-----------------------------------------------------*/
 P2715_004: PROC;
  ON ERROR SYSTEM ;
  IF FIXED_ERROR_COUNT > XCON200
  THEN
  DO ;
     CALL X_PUT_005; /* RELOCATED */
    PLIRETCODE = XCON12 ;
    SIGNAL ERROR ;
  END ;
  FIXED_ERROR_COUNT = FIXED_ERROR_COUNT + 1 ;
  ONSOURCE = 0 ;
END ;
/*********************************************************/
/*                                                       */
/*       UEBERSCHRIFT - LISTE                            */
/*                                                       */
/*********************************************************/
ON  ENDPAGE(LISTE)
CALL P2715_005;
/*-----------------------------------------------------*/
 P2715_005: PROC;
  CT(19)  =  CT(19)  +  1 ;
  CT(20)  =  CT(20)  +  XCON3 ;
   CALL X_PUT_006; /* RELOCATED */
   CALL X_PUT_007; /* RELOCATED */
   CALL X_PUT_008; /* RELOCATED */
   CALL X_PUT_009; /* RELOCATED */
  LINECOUNTER = XCON8 ;
END ;
/*********************************************************/
```

```
/*                                                          */
/*                  INITIALISIERUNG                         */
/*                                                          */
/************************************************************/
 CALL P;
 CALL LOOP;
 CALL PROGRAMM_ENDE;
 RETURN;
 /*********** END OF MAIN PROCEDURE ************/
PGM_START: PROCEDURE;
                ;
        DATUM  =  XLIT010 ;
        ZEIT   =  TIME ;
        ZEIT   =  TRANSLATE(XLIT011,ZEIT,XLIT012) ;
        PLIRETCODE = 0 ;
         CALL X_OPEN_010; /* RELOCATED */
/************************************************************/
/*                                                          */
/*             V E R A R B E I T U  N G                     */
/*                                                          */
/************************************************************/
    /*   FESTSTELLEN OB JOB UEBERHAUPT LAUFEN SOLL             */
KEY_FELD = LOW(32) ;
CALL DB_AKTION(TABELLEN_DB,'GU',KEY_FELD,'') ;
 IF PCB.STATUS ^= ' '
 THEN
DO ;
   CALL X_PUT_011; /* RELOCATED */
  PLIRETCODE = XCON12 ;
  SIGNAL ERROR ;
END ;
 ELSE
 DO ;
   PDAT = ADDR(IO_TA) ;
   AKTUELLE_AKTSTUFE = DATUMREC.AKTSTUFE ;
END ;
KEY_FELD = XLIT013 ;
GLOBAL_OPERATOR = '>=' ;
CALL DB_AKTION(TABELLEN_DBA,'GU',KEY_FELD,'') ;
 IF PCB.STATUS ^= ' ' ! SUBSTR(IO_TA,1,4) ^= KEY_FELD
 THEN
DO ;
   CALL X_PUT_012; /* RELOCATED */
  PLIRETCODE = XCON12 ;
  SIGNAL ERROR ;
END ;
 ELSE
 DO ;
   PDAT = ADDR(IO_TA) ;
   DATUM = TRANSLATE(XLIT014,ABR9900.MASCHOV,XLIT015) ;
END ;
SIGNAL ENDPAGE(LISTE) ;
 IF AKTUELLE_AKTSTUFE = 'F'
 THEN
DO ;
  KEY_FELD = XLIT016 !! LOW(15) ;
  SUCHKEY  = XLIT016 ;
END ;
 ELSE
 DO ;
   KEY_FELD = XLIT017 !! LOW(15) ;
   SUCHKEY  = XLIT017 ;
END ;
```

```
IO_TA     = SUCHKEY ;
GLOBAL_OPERATOR = '>=' ;
CALL DB_AKTION(TABELLEN_DBA,'GHU',KEY_FELD,'') ;
 IF PCB.STATUS ^= ' ' ! SUBSTR(IO_TA,1,4) ^= SUCHKEY
 THEN
DO ;
   CALL X_PUT_013; /* RELOCATED */
  PLIRETCODE = XCON4 ;
  SIGNAL ERROR ;
END ;
/*  DATUMSRECORD DER BESTANDS_DB HOLEN ZUR BESETZUNG PARM_SICHERST*/
KEY_FELD = LOW(32) ;
GLOBAL_OPERATOR = '>=' ;
CALL DB_AKTION(BESTAND_DB,'GU',KEY_FELD,'') ;
 IF PCB.STATUS ^= ' ' ! SUBSTR(IO_BE,1,15) ^= LOW(15)
 THEN
DO ;
  CALL FEHLERMELDUNG(1) ;
  PLIRETCODE = XCON8 ;
  SIGNAL ERROR ;
END ;
 ELSE
 DO ;
   PDAT = ADDR(IO_BE) ;
   PARM_SICHERST = TRANSLATE(XLIT018,DATUMREC.SICHERST,
   XLIT019) ;
END ;
/*  BESTAND       LESEN, SOLANGE DL/I-STATUS 'OK'                */
END PGM_START;
LOOP: PROCEDURE;
           CB.STATUS = ' ' ) ;
  CALL DB_AKTION(BESTAND_DB,'GHN','','U') ;
  IF PCB.STATUS ^= ' '
  THEN
  DO ;
    LEAVE LOOP ;
  END ;
  ELSE
  DO ;
    /* VERARBEITUNG NUR WENN FLAG GESETZT         */
    IF B_ABRTAGW = '1'
    THEN
    DO ;
      KEY_FELD = B_VALNR !!
      B_VALZUS !!
      B_ZUGEH  !!
      B_EIGENT !!
      B_NL     !!
      B_RESKEYBE ;
      CALL DB_AKTION(VALOREN_DB,'GU',KEY_FELD,'') ;
       CALL LOOP_0001(); /* Refactored */
      HERKUNFT = XLIT020 ;
       CALL LOOP_0002(); /* Refactored */
       CALL LOOP_0003(); /* Refactored */
      /* OBLIGATION ODER AKTIE ? */
       CALL LOOP_0004(); /* Refactored */
        /* OBLIGATIONEN GEFUNDEN    */
       CALL LOOP_0005(); /* Refactored */
        /*  AKTIEN GEFUNDEN    */
      /*  B_TAGEFA BERECHNEN              */
      HERKUNFT = XLIT023 ;
      BNOMSFRS = B_NOMSFR ;
      HFLOAT_1 = B_RENDTA ;
```

```
       B_TAGEFA = ROUND1(BNOMSFRS,HFLOAT_1) ;
       /*  FLAG RUECKSETZEN                  */
       B_ABRTAGW = '' ;
       /*  REWRITE BESTANDES_RECORD          */
       CALL DB_AKTION(BESTAND_DB,'REPL','','R') ;
        CALL LOOP_0006(); /* Refactored */
       /*  MELDUNG AUF LISTE AUSGEBEN        */
       TAWERT_Z  = B_TAWERT ;
       /*  CHANGE PUNKTE IN HOCHKOMMA   */
       TAWERT_NEU = TRANSLATE(TAWERT_Z,'''','.') ;
       LINECOUNTER = LINECOUNTER + 2 ;
        CALL LOOP_0007(); /* Refactored */
        CALL X_PUT_014; /* RELOCATED */
        CALL X_PUT_015; /* RELOCATED */
       COUNTER = COUNTER + 1 ;
     END ;            /* IF FLAG GESETZT                          */
   END ;               /* IF BESTANDS_RECORD FOUND                 */
*/* Refactored nested Procedures */
*/* These subroutines were factored out of mainline */
*-------------------------------------------------------------------------
-----
-/* Refactored PLI Procedure */
 PROCEDURE LOOP_0001();
     IF PCB.STATUS ^= ' '
     THEN
     DO ;
       CALL FEHLERMELDUNG(2) ;
       PLIRETCODE = XCON8 ;
       SIGNAL ERROR ;
     END ;
 END LOOP_0001;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE LOOP_0002();
     IF S_SNCD = 'S'
     THEN
     DO ;
       B_VPFWET = 0 ;
     END ;
 END LOOP_0002;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE LOOP_0003();
     ELSE
     DO ;
       HFLOAT_1 = B_TAWERT ;
       HFLOAT_2 = S_VPFANS / XCON100 ;
       B_VPFWET = ROUND1(HFLOAT_1,HFLOAT_2) ;
     END ;
 END LOOP_0003;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE LOOP_0004();
     IF VERIFY(SUBSTR(S_VALART,2,1),XLIT022) = 0
     THEN
     DO ;
       CALL RENDBER ;
     END ;
 END LOOP_0004;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE LOOP_0005();
     ELSE
```

```
    DO ;
      BSTKNOM = B_STKNOM ;
      SELECT (S_SNCD) ;
        WHEN ('N')
        DO ;
          KEY_FELD = '01' !! VALOR.C_OWAEC ;
          CALL DB_AKTION(TABELLEN_DB,'GU',KEY_FELD,'') ;
          IF PCB.STATUS ^= ' '
          THEN
          DO ;
            CALL FEHLERMELDUNG(3) ;
            PLIRETCODE = XCON8 ;
            SIGNAL ERROR ;
          END ;
          IF D_WAEE ^= 0 & B_TAWERT ^= 0
          THEN
          DO ;
            HFLOAT_1 =         (D_WAEK / D_WAEE) ;
            HFLOAT_2 = BSTKNOM * S_DIV / B_TAWERT ;
            B_RENDTA = ROUND2(HFLOAT_1,HFLOAT_2) ;
          END ;
          ELSE
          DO ;
            B_TAWERT = 0 ;
          END ;
        END ;
        WHEN ('S')
        DO ;
          KEY_FELD = '01' !! VALOR.C_HWAEC ;
          CALL DB_AKTION(TABELLEN_DB,'GU',KEY_FELD,'') ;
          IF PCB.STATUS ^= ' '
          THEN
          DO ;
            KEY_FELD = '01' !! VALOR.C_OWAEC ;
            CALL DB_AKTION(TABELLEN_DB,'GU',KEY_FELD,'') ;
            IF PCB.STATUS ^= ' '
            THEN
            DO ;
              CALL FEHLERMELDUNG(3) ;
              PLIRETCODE = XCON8 ;
              SIGNAL ERROR ;
            END ;
          END ;
          IF D_WAEE ^= 0 & B_TAWERT ^= 0
          THEN
          DO ;
            HFLOAT_1 =         (D_WAEK / D_WAEE) ;
            HFLOAT_2 = BSTKNOM * S_DIV * XCON100 / B_TAWERT ;
            B_RENDTA = ROUND2(HFLOAT_1,HFLOAT_2) ;
          END ;
          ELSE
          DO ;
            B_TAWERT = 0 ;
          END ;
        END ;  /*  'WHEN ('S')    */
        OTHERWISE
        DO ;
          B_RENDTA = 0 ;
        END ;
      END ;     /* SELECT S_SNCD */
    END ;        /* OBLI / AKTIE  */
 END LOOP_0005;
-/*   End of new Procedure   */
```

```
-/* Refactored PLI Procedure */
 PROCEDURE LOOP_0006();
     IF PCB.STATUS ^= ' '
     THEN
     DO ;
       CALL FEHLERMELDUNG(4) ;
       PLIRETCODE = XCON12 ;
       SIGNAL ERROR ;
     END ;
 END LOOP_0006;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE LOOP_0007();
     IF LINECOUNTER > XCON60
     THEN
     DO ;
       SIGNAL ENDPAGE(LISTE) ;
     END ;
 END LOOP_0007;
-/*   End of new Procedure   */
 END ;                   /* DO WHILE BESTANDS_RECORDS VORHANDEN     */
 /* LOESCHEN DES ABRUFRECORDS NACH VERARBEITUNG                     */
  IF AKTUELLE_AKTSTUFE = 'F'
  THEN
 DO ;
   KEY_FELD = XLIT016 !! LOW(15) ;
   SUCHKEY  = XLIT016 ;
 END ;
  ELSE
  DO ;
    KEY_FELD = XLIT017 !! LOW(15) ;
    SUCHKEY  = XLIT017 ;
 END ;
 IO_TA     = SUCHKEY ;
 GLOBAL_OPERATOR = '>=' ;
 CALL DB_AKTION(TABELLEN_DBA,'GHU',KEY_FELD,'') ;
  IF PCB.STATUS ^= ' ' ! SUBSTR(IO_TA,1,4) ^= SUCHKEY
  THEN
 DO ;
    CALL X_PUT_016; /* RELOCATED */
   PLIRETCODE = XCON4 ;
   SIGNAL ERROR ;
 END ;
 CALL DB_AKTION(TABELLEN_DBA,'DLET','','R') ;
  IF PCB.STATUS ^= ' '
  THEN
 DO ;
   PLIRETCODE = XCON12 ;
    CALL X_PUT_017; /* RELOCATED */
   PLIRETCODE = XCON12 ;
   SIGNAL ERROR ;
 END ;
 /*  ENDSUMME AUSGEBEN              */
  CALL X_PUT_018; /* RELOCATED */
  CALL X_PUT_019; /* RELOCATED */
  IF UPDDAT(PCB_BE) ^= 0
  THEN
 DO;
   CALL FEHLERMELDUNG(5);
   PCB_PTR = PCB_BE;
   PLIRETCODE = XCON12;
   SIGNAL ERROR;
 END;
```

```
/* IF UPDDAT(PCB_TAA) ^= 0 THEN
DO;
  CALL FEHLERMELDUNG(5);
  PCB_PTR = PCB_TAA;
  PLIRETCODE = 12;
  SIGNAL ERROR;
END; */
  /*   FEHLER-BEHANDLUNG      */
 FEHLER:    PROC(JFE,PCB0) ;                    /*  FEHLER BEHANDLUNG  *
           DCL  PCB0      PTR       ;
           DCL  JFE        DEC15,0) ;                    /*CONVERTED*/
            CALL X_PUT_020; /* RELOCATED */
END  FEHLER ;
 FEHLERMELDUNG:  PROC(FEHLER_INDEX) ;
DCL FEHLER_INDEX     DECFIXED (04)  ;                    /*CONVERTED*/
DCL MAX_INDEX        DECFIXED (04) INIT (30) ;           /*CONVERTED*/
DCL FEHLERTEXT (30) CHAR(60) INIT (
    'FEHLERHAFTER START AUF BESTANDS_DB',
    'VALOREN-RECORD NICHT VORHANDEN (MUSS)',
    'TABELLEN_RECORD (ORIGIN.-WAEH.) NICHT GEFUNDEN' ,
    'REPLACE-ERROR IN BESTANDS_DB',
    'FEHLER  5',
    'FEHLER  6',
    'FEHLER  7',
    'FEHLER  8',
    'FEHLER  9',
    'PGM_FEHLER: FALSCHER SELCT DB_NR',
    'FEHLER 11',
    'FEHLER 12',
    'FEHLER 13',
    'FEHLER 14',
    'FEHLER 15',
    'FEHLER 16',
    'FEHLER 17',
    'FEHLER 18',
    'FEHLER 19',
    'FEHLER 20',
    'FEHLER 21',
    'FEHLER 22',
    'FEHLER 23',
    'FEHLER 24',
    'FEHLER 25',
    'FEHLER 26',
    'FEHLER 27',
    'FEHLER 28',
    'FEHLER 29',
    'FEHLER 30'     ) ;
IF (FEHLER_INDEX > MAX_INDEX) !
   (FEHLER_INDEX < 1)
   THEN
DO ;
   CALL X_PUT_021; /* RELOCATED */
  /* GO TO FEHLERM_END REMOVED !! */
   RETURN;
END ;
 CALL X_PUT_022; /* RELOCATED */
 CALL X_PUT_023; /* RELOCATED */
 DB_AKTION: PROC (DB_NR,DB_FUNKTION,DB_SCHLUESSEL,AKTION) ;
   DCL DB_NR               DECFIXED (04) ;              /*CONVERTED*/
   DCL SSA_PTR            PTR           ;
   DCL DB_FUNKTION        CHAR (4)      ;
   DCL DB_SCHLUESSEL      CHAR (*)    ;
   DCL AKTION             CHAR (1)    ;
```

```
    DCL FOUR                 DECFIXED (08) INIT (4) ;      /*CONVERTED*/
    SELECT (DB_NR) ;
      WHEN (BESTAND_DB)
       CALL DB_AKTION_0008(); /* Refactored */
      WHEN (TABELLEN_DBA)
       CALL DB_AKTION_0009(); /* Refactored */
      WHEN (TABELLEN_DB)
       CALL DB_AKTION_0010(); /* Refactored */
      WHEN (VALOREN_DB)
       CALL DB_AKTION_0011(); /* Refactored */
      OTHERWISE
       CALL DB_AKTION_0012(); /* Refactored */
    END ;
    GLOBAL_OPERATOR = ' =' ;
*/* Refactored nested Procedures */
*/* These subroutines were factored out of mainline */
*-------------------------------------------------------------------------
-----
-/* Refactored PLI Procedure */
 PROCEDURE DB_AKTION_0008();
     DO ;
       SSA_BE.OP  = GLOBAL_OPERATOR ;
       SSA_BE.KEY = DB_SCHLUESSEL ;
       SELECT (AKTION) ;
         WHEN ('U')
           CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_BE,IO_BE,SSA_BEU) ;
         WHEN ('R')
           CALL  PLITDLI(THREE,DB_FUNKTION,PCB_BE,IO_BE) ;
         OTHERWISE
           CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_BE,IO_BE,SSA_BE) ;
       END ;
       PCB_PTR = PCB_BE ;
     END ;
 END DB_AKTION_0008;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE DB_AKTION_0009();
     DO ;
       SSA_TAA.OP  = GLOBAL_OPERATOR ;
       SSA_TAA.KEY = DB_SCHLUESSEL ;
       SELECT (AKTION) ;
         WHEN ('U')
           CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_TAA,IO_TA,SSA_TAAU) ;
         WHEN ('R')
           CALL  PLITDLI(THREE,DB_FUNKTION,PCB_TAA,IO_TAA) ;
         OTHERWISE
           CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_TAA,IO_TA,SSA_TAA) ;
       END ;
       PCB_PTR = PCB_TAA  ;
     END ;
 END DB_AKTION_0009;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE DB_AKTION_0010();
     DO ;
       SSA_TA.OP  = GLOBAL_OPERATOR ;
       SSA_TA.KEY = DB_SCHLUESSEL ;
       SELECT (AKTION) ;
         WHEN ('U')
           CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_TA,IO_TA,SSA_TAU) ;
         WHEN ('R')
           CALL  PLITDLI(THREE,DB_FUNKTION,PCB_TA,IO_TA) ;
         OTHERWISE
```

```
           CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_TA,IO_TA,SSA_TA) ;
         END ;
         PCB_PTR = PCB_TA ;
       END ;
 END DB_AKTION_0010;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE DB_AKTION_0011();
       DO ;
         SSA_VA.OP  = GLOBAL_OPERATOR ;
         SSA_VA.KEY = DB_SCHLUESSEL ;
         SELECT (AKTION) ;
           WHEN ('U')
             CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_VA,IO_VA,SSA_VAU) ;
           WHEN ('R')
             CALL  PLITDLI(THREE,DB_FUNKTION,PCB_VA,IO_VA) ;
           OTHERWISE
             CALL  PLITDLI(FOUR,DB_FUNKTION,PCB_VA,IO_VA,SSA_VA) ;
         END ;
         PCB_PTR = PCB_VA   ;
       END ;
 END DB_AKTION_0011;
-/*   End of new Procedure   */
-/* Refactored PLI Procedure */
 PROCEDURE DB_AKTION_0012();
       DO ;
         CALL FEHLERMELDUNG(10) ;
         PLIRETCODE = XCON12 ;
         SIGNAL ERROR ;
       END ;
 END DB_AKTION_0012;
-/*   End of new Procedure   */
 END DB_AKTION ;
 %INCLUDE RENDITE ;
 %INCLUDE RENDBER ;
 %INCLUDE DAYDIFF ;
 %INCLUDE ROUND1 ;
 %INCLUDE ROUND2 ;
 %INCLUDE UPDDAT ;
 /*****************************************************/
 /******* INPUT-OUTPUT PROCEDURE *******/
  X_PUT_001: PROCEDURE;
     PUT FILE(LISTE) SKIP EDIT
     ('LOOP INNERHALB ON ERROR BLOCK') (A) ;
    RETURN;
 END X_PUT_001;
 /********* END OF IO PROCEDURE ********/
 /******* INPUT-OUTPUT PROCEDURE *******/
  X_CLOS_002: PROCEDURE;
    CLOSE  FILE(LISTE)       ;
    RETURN;
 END X_CLOS_002;
 /********* END OF IO PROCEDURE ********/
 /******* INPUT-OUTPUT PROCEDURE *******/
  X_PUT_003: PROCEDURE;
     PUT SKIP EDIT ('CONVERSION-ERROR, ONCODE: ',ONCODE) (A,P'ZZZ9');
    RETURN;
 END X_PUT_003;
 /********* END OF IO PROCEDURE ********/
 /******* INPUT-OUTPUT PROCEDURE *******/
  X_PUT_004: PROCEDURE;
     PUT SKIP EDIT ('CONVERSION-ERROR 612   ',
     HERKUNFT,':>',ONSOURCE,'< ',KEY_FELD)
```

```
    (A,A,A,A,A,A) ;
  RETURN;
END X_PUT_004;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_005: PROCEDURE;
    PUT SKIP EDIT('FIXED_ERROR_COUNT > 200 ',
    'TEST - TERMINATON       ') (A,A) ;
  RETURN;
END X_PUT_005;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_006: PROCEDURE;
  PUT FILE(LISTE) EDIT
  ('SBG-WAB','MANUELL GEAENDERTE TAGESWERTE PER',DATUM,'LISTE',
  '4',AKTUELLE_AKTSTUFE,'SEITE :',CT(19))
  (COL(01),A,COL(16),A,COL(50),A,COL(70),A,COL(76),A,
  COL(77),A,COL(100),A,COL(108),P'ZZZZ9') PAGE;      /* BVO001 */
  RETURN;
END X_PUT_006;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_007: PROCEDURE;
  PUT FILE(LISTE) SKIP(2) EDIT
  ('FUER FOLGENDE POSITIONEN WURDEN B_RENDTA, B_TAGEFA,',
  ' B_VPFWET NEU GERECHNET:')
  (COL(1),A,COL(52),A) ;
  RETURN;
END X_PUT_007;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_008: PROCEDURE;
  PUT FILE(LISTE) SKIP(3) EDIT
  ('VALNR','VALZUS','ZUGEH','EIGENT','NL','TITKUT',
  'TAGESWERT NEU')
  (COL(01),A,COL(16),A,COL(26),A,COL(35),A,COL(46),A,COL(53),A,
  COL(82),A) ;
  RETURN;
END X_PUT_008;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_009: PROCEDURE;
  PUT FILE(LISTE) SKIP(2) EDIT ('   ') (COL(1),A) ;
  RETURN;
END X_PUT_009;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_OPEN_010: PROCEDURE;
        OPEN  FILE(LISTE)             PAGESIZE(42) ;
  RETURN;
END X_OPEN_010;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_011: PROCEDURE;
  PUT FILE(LISTE) SKIP EDIT
  ('DATUMRECORD IN TABELLEN_DB NICHT VORHANDEN') (A) ;
  RETURN;
END X_PUT_011;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_012: PROCEDURE;
  PUT FILE(LISTE) SKIP EDIT
  ('DATUMRECORD IN TABELLEN_DB NICHT VORHANDEN') (A) ;
```

```
  RETURN;
END X_PUT_012;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_013: PROCEDURE;
  PUT FILE(LISTE) SKIP EDIT
  ('KEIN ABRUF') (A) ;
  RETURN;
END X_PUT_013;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_014: PROCEDURE;
      PUT FILE(LISTE) SKIP  EDIT
      (S_VALNR,S_VALZUS,S_ZUGEH,S_EIGENT,S_NL,S_TIKUT,
      TAWERT_NEU)
      (COL(1),A,COL(18),A,COL(26),P'99999',COL(37),P'9',
      COL(45),P'9999',COL(53),A,COL(82),A) ;
  RETURN;
END X_PUT_014;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_015: PROCEDURE;
      PUT FILE(LISTE) SKIP EDIT ('      ') (COL(1),A) ;
  RETURN;
END X_PUT_015;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_016: PROCEDURE;
  PUT FILE(LISTE) SKIP EDIT
  ('KEIN ABRUF') (A) ;
  RETURN;
END X_PUT_016;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_017: PROCEDURE;
  PUT FILE(LISTE) SKIP EDIT (
  'DELETE ERROR, TABELLEN (ABRUF_RECORD) ') (A) ;
  RETURN;
END X_PUT_017;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_018: PROCEDURE;
PUT FILE(LISTE) SKIP EDIT ('          ') (COL(2),A) ;
  RETURN;
END X_PUT_018;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_019: PROCEDURE;
PUT FILE(LISTE) SKIP EDIT
   ('********** TOTAL ',COUNTER,'POSITIONEN NEU GERECHNET **********')
   (COL(30),A,COL(47),P'ZZZZ9',COL(53),A) ;
  RETURN;
END X_PUT_019;
/********* END OF IO PROCEDURE *******/
/******* INPUT-OUTPUT PROCEDURE *******/
 X_PUT_020: PROCEDURE;
           PUT  FILE(LISTE)   EDIT
               ('DBNAME    = >',PCB_PTR->PCB.DBNAME ,'<',
                '  STATUS  = >',PCB_PTR->PCB.STATUS ,'<',
                '  SEGMENT = >',PCB_PTR->PCB.SNAME  ,'<',
                '  S.LEVEL =  ',PCB_PTR->PCB.SEGL   ,
                '  KEY     = >',PCB_PTR->PCB.KEY    ,'<',
                '  PLIRETC =  ',PLIRETCODE          )
```

```
               (SKIP(1),A,A,A,
                SKIP(1),A,A,A,
                SKIP(1),A,A,A,
                SKIP(1),A,A,
                SKIP(1),A,A,A,
                SKIP(1),A,A) ;
  RETURN;
 END X_PUT_020;
 /********* END OF IO PROCEDURE ********/
 /******* INPUT-OUTPUT PROCEDURE *******/
  X_PUT_021: PROCEDURE;
   PUT FILE(LISTE) SKIP(3) EDIT
   ('FEHLER_INDEX OUT OF RANGE, VALUE: ',FEHLER_INDEX)
   (COL(40),A,P'ZZZZZZZ9',SKIP(3)) ;
   RETURN;
 END X_PUT_021;
 /********* END OF IO PROCEDURE ********/
 /******* INPUT-OUTPUT PROCEDURE *******/
  X_PUT_022: PROCEDURE;
 PUT FILE(LISTE) SKIP EDIT (FEHLERTEXT(FEHLER_INDEX))
    (COL(01),A);
   RETURN;
 END X_PUT_022;
 /********* END OF IO PROCEDURE ********/
 /******* INPUT-OUTPUT PROCEDURE *******/
  X_PUT_023: PROCEDURE;
 PUT FILE(LISTE) SKIP EDIT
     ('LAST FILESTATUS: ',PCB.STATUS) (A,A) ;
   RETURN;
 END X_PUT_023;
 /********* END OF IO PROCEDURE ********/
 /*********** END OF DATA ACCESS PROCEDURES **********/
 PROGRAMM_ENDE_ENDE:
 END  DLITPLI ;                      /*     PROGRAMM ENDE       */
```

6.7 Sample of a Renovated COBOL Program

```
      ***************************************************************
REPLAC      COPY COBOLD.CPY.
REPLAC*****************************************************************
       PROCEDURE DIVISION.
      ***************************************************************
       * INITIALIZE ORDER ENTRY PROCESSING
RESTRU*---------------------------------------------------------------
       X-CONTROL SECTION.
       X-CONTROL-LOOP.
           MOVE 'INITIALIZATION' TO X-NEXT-LABEL.
           PERFORM UNTIL X-NEXT-LABEL = 'STOP'
                      OR X-NEXT-LABEL = SPACES
              IF X-NEXT-LABEL = 'INITIALIZATION'
                 PERFORM INITIALIZATION
              END-IF
              IF X-NEXT-LABEL = 'READ-ORDERS'
                 PERFORM READ-ORDERS
              END-IF
              IF X-NEXT-LABEL = 'READ-CUSTOMER'
                 PERFORM READ-CUSTOMER
              END-IF
              IF X-NEXT-LABEL = 'PROCESS-ORDER'
                 PERFORM PROCESS-ORDER
              END-IF
              IF X-NEXT-LABEL = 'WRITE-OPEN-POSITIONS'
```

```
                PERFORM WRITE-OPEN-POSITIONS
             END-IF
             IF X-NEXT-LABEL = 'WRITE-DISPATCH'
                PERFORM WRITE-DISPATCH
             END-IF
             IF X-NEXT-LABEL = 'REPORT-ORDER'
                PERFORM REPORT-ORDER
             END-IF
             IF X-NEXT-LABEL = 'REPORT-ERROR'
                PERFORM REPORT-ERROR
             END-IF
             IF X-NEXT-LABEL = 'PRINT-SUMMARY'
                PERFORM PRINT-SUMMARY
             END-IF
             IF X-NEXT-LABEL = 'TERMINATION'
                PERFORM TERMINATION
             END-IF
          END-PERFORM.
       X-CONTROL-EXIT.
          EXIT.
RESTRU*-----------------------------------------------------------------
       INITIALIZATION.
          MOVE SPACES TO X-NEXT-LABEL.
          PERFORM XOPEN0001.
          PERFORM XOPEN0002.
          PERFORM XOPEN0003.
          PERFORM XOPEN0004.
          PERFORM XOPEN0005.
          PERFORM XOPEN0006.
          MOVE 1 TO PAGE-COUNT.
          MOVE PAGE-COUNT TO PAGE-COUNT-NO.
          MOVE HEADER-PRT-LINE TO PRT-LINE.
          PERFORM XWRIT0007.
          IF X-NEXT-LABEL = SPACES
             MOVE 'READ-ORDERS' TO X-NEXT-LABEL
          END-IF.
RESTRU*-----------------------------------------------------------------
      *
      ****************************************************************
      * READ ORDERS UNTIL END OF ORDER-FILE
      *----------------------------------------------------------------
       READ-ORDERS.
          MOVE SPACES TO X-NEXT-LABEL.
          PERFORM XREAD0008
          IF X-RETCODE NOT = ZEROS
            MOVE 'TERMINATION' TO X-NEXT-LABEL
          END-IF
          IF X-NEXT-LABEL = SPACES
             MOVE 'READ-CUSTOMER' TO X-NEXT-LABEL
          END-IF.
RESTRU*-----------------------------------------------------------------
      ****************************************************************
      * READ CUSTOMER-DATA WITH KEY = CUST-NO
      * ERROR CORRECTED, ERROR-TYPE INITIALIZED
      *----------------------------------------------------------------
       READ-CUSTOMER.
          MOVE SPACES TO X-NEXT-LABEL.
          MOVE ZERO TO ERROR-TYPE.
          MOVE CUST-NO IN ORDER-RECORD TO CUST-KEY.
          PERFORM XREAD0009
          IF X-RETCODE NOT = ZEROS
            MOVE 1 TO ERROR-TYPE
            MOVE 'REPORT-ERROR' TO X-NEXT-LABEL
```

```
           END-IF
           IF X-NEXT-LABEL = SPACES
             MOVE 0 TO POS
             MOVE ZERO TO TOTAL-ITEMS-FULFILLED
             MOVE ZERO TO TOTAL-CUST-PRICE
           END-IF
           IF X-NEXT-LABEL = SPACES
              MOVE 'PROCESS-ORDER' TO X-NEXT-LABEL
           END-IF.
RESTRU*-----------------------------------------------------------------
      ******************************************************************
      * PROCESS ORDER ITEMS FROM 1 TO 9
      * OR UNTIL ITEM-NO = 999
      * ERROR CORRECTED, ERROR-TYPE INITIALIZED
      *-----------------------------------------------------------------
       PROCESS-ORDER.
           MOVE SPACES TO X-NEXT-LABEL.
           MOVE ZERO TO ERROR-TYPE.
           ADD 1 TO POS.
           IF POS > XCON9
             OR ITEM-NO IN ORDER-RECORD (POS) = XCON999
REFINE*               GO TO PRINT-SUMMARY
             SUBTRACT 1 FROM POS
             MOVE 'PRINT-SUMMARY' TO X-NEXT-LABEL
           END-IF
           IF X-NEXT-LABEL = SPACES
             PERFORM XREAD0010
             IF X-RETCODE NOT = ZEROS
               MOVE 2 TO ERROR-TYPE
               MOVE 'REPORT-ERROR' TO X-NEXT-LABEL
             END-IF
           END-IF
           IF X-NEXT-LABEL = SPACES
             IF ITEM-QUAN IN ORDER-RECORD (POS) >
               ART-QUAN IN ARTICLE-RECORD
      *                GO TO WRITE-OPEN-POSITIONS
               MOVE 3 TO ERROR-TYPE
               MOVE 'WRITE-OPEN-POSITIONS' TO X-NEXT-LABEL
             END-IF
           END-IF
           IF X-NEXT-LABEL = SPACES
             SUBTRACT ITEM-QUAN IN ORDER-RECORD (POS) FROM
             ART-QUAN IN ARTICLE-RECORD
REFINE*                GO TO WRITE-DISPATCH
             PERFORM XREWR0011
             MOVE 'WRITE-DISPATCH' TO X-NEXT-LABEL
           END-IF
           IF X-NEXT-LABEL = SPACES
              MOVE 'WRITE-OPEN-POSITIONS' TO X-NEXT-LABEL
           END-IF.
RESTRU*-----------------------------------------------------------------
      * ERROR OCCURED HERE BY SKIPPING OVER LAST ORDER-ITEM **********
      * IF POS = 9 SHOULD BE IF POS > 9
      * IF ITEM-NO (POS) > 999 SHOULD BE ITEM-NO (POS) = 999
      * ERROR OCCURED HERE BECAUSE POS IS 1 MORE THAN FINAL ITEM
      * READ ARTICLE DATA WITH KEY = ART-NO
      * CHECK IF QUANTITY IS SUFFICIENT AND DEDUCT ORDER FROM STOCK
      ******************************************************************
      * WRITE OPEN POSITION
      *-----------------------------------------------------------------
       WRITE-OPEN-POSITIONS.
           MOVE SPACES TO X-NEXT-LABEL.
           MOVE CUST-NO IN ORDER-RECORD TO CUST-NO IN POSITION-RECORD.
```

```
           MOVE ORDER-NO IN ORDER-RECORD TO ORDER-NO IN POSITION-RECORD.
           MOVE ORDER-ITEM IN ORDER-RECORD (POS) TO DISPATCH-ITEM
           IN POSITION-RECORD.
           MOVE "P " TO REC-TYPE IN POSITION-RECORD.
           PERFORM XWRIT0012.
           MOVE 'REPORT-ORDER' TO X-NEXT-LABEL
           IF X-NEXT-LABEL = SPACES
              MOVE 'WRITE-DISPATCH' TO X-NEXT-LABEL
           END-IF.
RESTRU*-----------------------------------------------------------------
      ***************************************************************
      * WRITE DISPATCH RECORD FOR SHIPMENT
      *-----------------------------------------------------------------
       WRITE-DISPATCH.
           MOVE SPACES TO X-NEXT-LABEL.
           ADD 1 TO TOTAL-ITEMS-FULFILLED.
           MOVE ORDER-NO IN ORDER-RECORD TO ORDER-NO IN DISPATCH-RECORD
           MOVE CUST-NO IN ORDER-RECORD TO CUST-NO IN DISPATCH-RECORD
           MOVE CUST-NAME IN CUSTOMER-RECORD TO CUST-NAME IN
           DISPATCH-RECORD.
           MOVE CUST-ADDRESS IN CUSTOMER-RECORD TO
           CUST-ADDRESS IN DISPATCH-RECORD.
           MOVE ORDER-ITEM IN ORDER-RECORD (POS) TO DISPATCH-ITEM
           IN DISPATCH-RECORD.
           IF CUST-CREDIT > XCON5
             MOVE "N" TO PAYMENT
           ELSE
             MOVE "C" TO PAYMENT
           END-IF.
           MOVE "D " TO REC-TYPE IN DISPATCH-RECORD.
           PERFORM XWRIT0013.
           IF X-NEXT-LABEL = SPACES
              MOVE 'REPORT-ORDER' TO X-NEXT-LABEL
           END-IF.
RESTRU*-----------------------------------------------------------------
      * PRINT OUT ORDER DATA WITH TOTALED PRICES
      *-----------------------------------------------------------------
       REPORT-ORDER.
           MOVE SPACES TO X-NEXT-LABEL.
REFINE*       GO TO REPORT-ERROR
           IF ERROR-TYPE > 0
             MOVE 'REPORT-ERROR' TO X-NEXT-LABEL
           END-IF
           IF X-NEXT-LABEL = SPACES
             MOVE SPACES TO DATA-PRT-LINE
             MOVE CUST-NO IN CUSTOMER-RECORD TO CUST-NO IN
            DATA-PRT-LINE
             MOVE CUST-NAME IN CUSTOMER-RECORD TO CUST-NAME IN
             DATA-PRT-LINE
             MOVE ART-NO IN ARTICLE-RECORD TO ART-NO IN DATA-PRT-LINE
             MOVE ART-NAME IN ARTICLE-RECORD TO ART-NAME IN
             DATA-PRT-LINE
             MOVE ITEM-QUAN IN ORDER-RECORD (POS) TO
             ART-QUAN IN DATA-PRT-LINE
             MOVE ART-PRICE IN ARTICLE-RECORD TO
             ART-PRICE IN DATA-PRT-LINE
             MULTIPLY ART-PRICE IN ARTICLE-RECORD BY ITEM-QUAN
             IN ORDER-RECORD (POS) GIVING PRICE
             MOVE PRICE TO TOTAL-PRICE
             ADD PRICE TO TOTAL-CUST-PRICE
             MOVE DATA-PRT-LINE TO PRT-LINE
             PERFORM XWRIT0014
             ADD 2 TO LINE-COUNT
```

```
            IF LINE-COUNT > XCON50
              ADD 1 TO PAGE-COUNT
              MOVE PAGE-COUNT TO PAGE-COUNT-NO
              MOVE HEADER-PRT-LINE TO PRT-LINE
              PERFORM XWRIT0015
              MOVE UNDER-LINE TO PRT-LINE
              PERFORM XWRIT0016
              MOVE 0 TO LINE-COUNT
            END-IF
            IF X-NEXT-LABEL = SPACES
              MOVE 'PROCESS-ORDER' TO X-NEXT-LABEL
            END-IF
          END-IF
          IF X-NEXT-LABEL = SPACES
             MOVE 'REPORT-ERROR' TO X-NEXT-LABEL
          END-IF.
RESTRU*-----------------------------------------------------------------
      * PRINT ERROR MESSAGE
      *-----------------------------------------------------------------
       REPORT-ERROR.
          MOVE SPACES TO X-NEXT-LABEL.
          MOVE SPACES TO DATA-PRT-LINE.
          MOVE SPACES TO ERROR-PRT-LINE.
          MOVE ORDER-NO IN ORDER-RECORD TO ORDER-NO IN DATA-PRT-LINE.
          MOVE CUST-NO IN ORDER-RECORD TO CUST-NO IN DATA-PRT-LINE.
          IF ERROR-TYPE > 1
            MOVE ART-NO IN ORDER-RECORD (POS) TO
            ART-NO IN DATA-PRT-LINE
            IF ERROR-TYPE > 2
              MOVE ART-NAME IN ARTICLE-RECORD TO
              ART-NAME IN DATA-PRT-LINE
              MOVE ART-QUAN IN ARTICLE-RECORD TO
              ART-QUAN IN DATA-PRT-LINE
            END-IF
          END-IF.
          MOVE DATA-PRT-LINE TO PRT-LINE.
          PERFORM XWRIT0017.
          IF ERROR-TYPE = 1
            MOVE XLIT005 TO ERROR-MESSAGE
          END-IF.
          IF ERROR-TYPE = 2
            MOVE XLIT006 TO ERROR-MESSAGE
          END-IF.
          MOVE ERROR-PRT-LINE TO PRT-LINE.
          PERFORM XWRIT0018.
          IF ERROR-TYPE = 3
            MOVE XLIT007 TO ERROR-MESSAGE
          END-IF.
          MOVE ERROR-PRT-LINE TO PRT-LINE.
          PERFORM XWRIT0019.
          ADD 1 TO LINE-COUNT.
          IF LINE-COUNT > XCON50
            ADD 1 TO PAGE-COUNT
            MOVE PAGE-COUNT TO PAGE-COUNT-NO
            MOVE HEADER-PRT-LINE TO PRT-LINE
            PERFORM XWRIT0020
            MOVE UNDER-LINE TO PRT-LINE
            PERFORM XWRIT0021
            MOVE 0 TO LINE-COUNT
          END-IF.
      *       GO TO PROCESS-ORDER
          IF ERROR-TYPE > 1
            MOVE 'PROCESS-ORDER' TO X-NEXT-LABEL
```

```
           END-IF
REFINE*        GO TO READ-ORDERS
           IF X-NEXT-LABEL = SPACES
             MOVE 'READ-ORDERS' TO X-NEXT-LABEL
           END-IF
           IF X-NEXT-LABEL = SPACES
              MOVE 'PRINT-SUMMARY' TO X-NEXT-LABEL
           END-IF.
RESTRU*-------------------------------------------------------------------
      * ERROR OCCURED HERE BECAUSE DATA-PRT-LINE INSTEAD OFF ERROR
      * PRINT CUSTOMER SUMMARY
      *-------------------------------------------------------------------
       PRINT-SUMMARY.
           MOVE SPACES TO X-NEXT-LABEL.
           MOVE CUST-NO IN CUSTOMER-RECORD TO CUST-NO IN
           SUMMARY-PRT-LINE.
           MOVE CUST-NAME IN CUSTOMER-RECORD TO CUST-NAME IN
           SUMMARY-PRT-LINE.
           MOVE POS TO NO-ITEMS-ORDERED.
           MOVE TOTAL-ITEMS-FULFILLED TO NO-ITEMS-FULFILLED.
           MOVE TOTAL-CUST-PRICE TO TOTAL-PRICE-FOR-CUST.
           MOVE SUMMARY-PRT-LINE TO PRT-LINE.
           PERFORM XWRIT0022
           ADD 3 TO LINE-COUNT.
           IF LINE-COUNT > XCON50
             ADD 1 TO PAGE-COUNT
             MOVE PAGE-COUNT TO PAGE-COUNT-NO
             MOVE HEADER-PRT-LINE TO PRT-LINE
             PERFORM XWRIT0023
             MOVE UNDER-LINE TO PRT-LINE
             PERFORM XWRIT0024
             MOVE 0 TO LINE-COUNT
           END-IF.
           MOVE 'READ-ORDERS' TO X-NEXT-LABEL
           IF X-NEXT-LABEL = SPACES
              MOVE 'TERMINATION' TO X-NEXT-LABEL
           END-IF.
RESTRU*-------------------------------------------------------------------
      ***************************************************************
      * TERMINATE ORDER ENTRY PROCESSING
      *-------------------------------------------------------------------
       TERMINATION.
           MOVE SPACES TO X-NEXT-LABEL.
           DISPLAY XLIT008.
           PERFORM XCLOS0025.
           PERFORM XCLOS0026.
           PERFORM XCLOS0027.
           PERFORM XCLOS0028.
           PERFORM XCLOS0029.
           PERFORM XCLOS0030.
           MOVE 'STOP' TO X-NEXT-LABEL
           STOP RUN.
           IF X-NEXT-LABEL = SPACES
              MOVE 'TERMINATION' TO X-NEXT-LABEL
           END-IF.
RESTRU*-------------------------------------------------------------------
      ********************** end of source ****************************
*****************************************************************************
REMOVE*
REMOVE*  IO Section for File and Database Access
REMOVE*
*****************************************************************************
```

```
       X-DATA-ACCESS SECTION.
           MOVE SPACES TO X-NEXT-LABEL.
       XOPEN0001.
           OPEN INPUT  ORDER-FILE.
*****************************************************************************
       XOPEN0002.
           OPEN INPUT  CUSTOMER-FILE.
*****************************************************************************
       XOPEN0003.
           OPEN I-O    ARTICLE-FILE.
*****************************************************************************
       XOPEN0004.
           OPEN OUTPUT DISPATCH-FILE.
*****************************************************************************
       XOPEN0005.
           OPEN OUTPUT OPEN-POSITIONS.
*****************************************************************************
       XOPEN0006.
           OPEN OUTPUT CUSTOMER-REPORT.
*****************************************************************************
       XWRIT0007.
           WRITE PRT-LINE AFTER ADVANCING PAGE.
*****************************************************************************
       XREAD0008.
           MOVE ZEROES TO X-RETCODE
           READ ORDER-FILE
                AT END
                  MOVE "99" TO X-RETCODE
           END-READ.
*****************************************************************************
       XREAD0009.
           MOVE ZEROES TO X-RETCODE
           READ CUSTOMER-FILE
                INVALID KEY
                  MOVE "99" TO X-RETCODE
           END-READ.
*****************************************************************************
       XREAD0010.
           MOVE ZEROES TO X-RETCODE
           READ ARTICLE-FILE
                INVALID KEY
                  MOVE "99" TO X-RETCODE
           END-READ.
*****************************************************************************
       XREWR0011.
              REWRITE ARTICLE-RECORD.
*****************************************************************************
       XWRIT0012.
           WRITE POSITION-RECORD.
*****************************************************************************
       XWRIT0013.
           WRITE DISPATCH-RECORD.
*****************************************************************************
       XWRIT0014.
           WRITE PRT-LINE AFTER ADVANCING 2 LINES.
*****************************************************************************
       XWRIT0015.
              WRITE PRT-LINE AFTER ADVANCING PAGE-COUNT.
*****************************************************************************
       XWRIT0016.
              WRITE PRT-LINE AFTER ADVANCING 1 LINES.
*****************************************************************************
       XWRIT0017.
```

```
           WRITE PRT-LINE AFTER ADVANCING 2 LINES.
*****************************************************************************
       XWRIT0018.
           WRITE PRT-LINE AFTER ADVANCING 2 LINES.
*****************************************************************************
       XWRIT0019.
           WRITE PRT-LINE AFTER ADVANCING 1 LINES.
*****************************************************************************
       XWRIT0020.
                WRITE PRT-LINE AFTER ADVANCING PAGE.
*****************************************************************************
       XWRIT0021.
                WRITE PRT-LINE AFTER ADVANCING 1 LINES.
*****************************************************************************
       XWRIT0022.
           WRITE PRT-LINE AFTER ADVANCING 3 LINES.
*****************************************************************************
       XWRIT0023.
                WRITE PRT-LINE AFTER ADVANCING PAGE.
*****************************************************************************
       XWRIT0024.
                WRITE PRT-LINE AFTER ADVANCING 1 LINES.
*****************************************************************************
       XCLOS0025.
           CLOSE ORDER-FILE.
*****************************************************************************
       XCLOS0026.
           CLOSE CUSTOMER-FILE.
*****************************************************************************
       XCLOS0027.
           CLOSE ARTICLE-FILE.
*****************************************************************************
       XCLOS0028.
           CLOSE DISPATCH-FILE.
*****************************************************************************
       XCLOS0029.
           CLOSE OPEN-POSITIONS.
*****************************************************************************
       XCLOS0030.
           CLOSE CUSTOMER-REPORT.
CLOS  ****** End of Data Access Section *******
*****************************************************************************
       EXIT.
*****************************************************************************
```

6.7 Sample of a Renovated C++ Class

```
 ((APushbutton *) (GetControl (PB_CANCEL)))->SetMlsID (RC_VERLASSEN);  //
Ressource muess geaendert werden.

     return TRUE;
  } // NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: Command

// ---------------------------------------------------------------------
// NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: ListboxSelect
// Author: TH
//
// Description
//
// Listbox Handle Funktion.
//
```

```
// Parameter                Description
//
// usLb                     Id der Listbox
//
// Return Value
//
// keiner
// ---------------------------------------------------------------------

  void  NNMNzkto_Funktionen_Zum_ErgebnisblattDlg  ::  ListboxSelect  (USHORT
usLb)
  {
     switch (usLb)
     {
        //                      ------------------------------------------
Ergebnisblattuebersicht
        case TB_NZKTO_EBLATTUEBERSICHT:
        {
           SelectTabListbox        (*pTb,        dbcNERG,        *pdbmcNERG,
TB_NZKTO_EBLATTUEBERSICHT);

           if (bGetFromDB)
           {
              EnableControl  (PB_NZKTO_EBLATT_NACHFOLGER,  (pTb->GetCount  ()
> 0));
              EnableControl (PB_NZKTO_EBLATT_AENDERN,     (pTb->GetCount ()
> 0));
           } // if

           if (pTb->GetCount () <= 0)
           {
  /* Nested Code has been factored out !! */
              NNMFUEB1.NNMFUEB1_03();
           } // if
           else
           {
                // wenn kein Vorgaenger vorhanden ist
              if (pdbsNERG->fszNERG_Id0.IsDefault ())
              {
  /* Nested Code has been factored out !! */
                 NNMFUEB1.NNMFUEB1_04();
              } // if
              else
              {
                 if (bGetFromDB)
                 {
                    EnableControl (PB_NZKTO_EBLATT_ABTRENNEN, TRUE);
                    EnableControl (PB_NZKTO_EBLATT_ANHAENGEN, FALSE);
                 } // if
              } // else
           } // else
            // Focus auf anderes Feld setzen, damit nach dem disablen des
            // Bearbeiten-Pushbuttons dieser nicht mehr den Focus hat.
           SetFocus (TB_NZKTO_EBLATTUEBERSICHT);
           break;
        } // case
     } // switch

  } // NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: ListboxSelect

// ---------------------------------------------------------------------
// NNMNzkto_Funktionen_Zum_ErgebnisblattDlg::GetKey
// Author: HL
```

```
//
// Description
//
// fuer die Anzeige von Daten in der Titelzeile
// ------------------------------------------------------------------------
--
  FString NNMNzkto_Funktionen_Zum_ErgebnisblattDlg::GetKey (void) const
  {

     return (dbsFUEBL.fkTyp == "ZIZA" ?
     GetMlsString (RC_NZKTO_EBAUM_WARTEN_KUPON_ZINSABGRENZUNG) :
     dbsFUEBL.fkTyp.ToString ());

  } // NNMNzkto_Funktionen_Zum_ErgebnisblattDlg::GetKey

// ------------------------------------------------------------------------
// NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: CallDlgErgebnisblatt
// Author: TH
//
// Description
//
// Aufruf weiterer Funktionen.
//
// Parameter               Description
//
// fsPushbutton            gedrueckter Pushbutton (Input)
//
// Return Value
//
// keiner
// ------------------------------------------------------------------------
----

  void  NNMNzkto_Funktionen_Zum_ErgebnisblattDlg  ::  CallDlgErgebnisblatt
(const FAnzahl_Klein_0 fsPushbutton)
  {
     SNERG   dbsNERG_temp;

     NNFerg *pBase = new NNFerg (this);
    //BOOL  bRc  =  pBase->CallDlgErgblattTyp  (dbsFUEBL.fkTyp,  pdbsNERG-
>fszNERG_Id,   dbcBAUM,   dbcNERG.xdbsDbRef,   fsPushbutton,   dbsNERG_temp,
pParent->IsDialog ());
     BOOL  bRc  =  pBase->CallDlgErgblattTyp  (dbsFUEBL.fkTyp,  pdbsNERG-
>fszNERG_Id, xdbcmNERG,  fsPushbutton, dbsNERG_temp, pParent->IsDialog (),
m_rhcMbaum);

     delete pBase;


     if (bRc)
     {
        BOOL bDurchfuehren = TRUE;

        *pdbsNERG = dbsNERG_temp; // Daten geaendert/geloescht/neu

        if ((fsPushbutton != PUSH_ANLEGEN)
        &&
        (fsPushbutton != PUSH_NACHFOLGER_ANLEGEN))
        {
  /* Nested Code has been factored out !! */
           NNMFUEB1.NNMFUEB1_05();
        } // if
        else
```

```
        {
           dbcNERG.dbsNOSTBAUM.SetDefaults   ();   //   Daten   noch   nicht
vorhanden
            // WDSSJ 7.6.2001: Neu wegen Mangel Leitgeb 17:
            // Defaultsetzen der NERG_Id unter Kommentar gesetzt:
//            dbcNERG.xdbsNerg.fszNERG_Id.SetDefault ();
            // Daten noch nicht vorhanden
                           //  Notwendig,  damit  das  'ja'  fuer  im  Baum
verwendet
                           //   vom   aktuell   markierten   Datensatz   nicht
ueber-
                           // nommen wird.

           ActionTabListbox (*pTb, dbcNERG, *pdbmcNERG, ACT_ADD);
        } // else

        if (bDurchfuehren)  // Bei Delete unnoetig
        {
           NNFerg *pBase = new NNFerg (this);
            //pBase->IsBaumUsed  (dbcBAUM,  phcMbaum);  //  Daten  ergaenzen
(fuer Sortierung)
           pBase->IsBaumUsed  (xdbcmNERG,  m_rhcMbaum);  //  Daten  ergaenzen
(fuer Sortierung)

           delete pBase;

           ActionTabListbox (*pTb, dbcNERG, *pdbmcNERG); // neu darstellen
        } // if

        ListboxSelect (TB_NZKTO_EBLATTUEBERSICHT); // f. Inaktivierung von
Controls
     } // if


  } // NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: CallDlgErgebnisblatt


//  ------------------------------------------------------------------------
----
// NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: CallDlgAuswaehlen
// Author: TH
//
// Description
//
// Aufruf weiterer Funktionen.
//
// Parameter               Description
//
// keine                   keine
//
// Return Value
//
// keiner
//  ------------------------------------------------------------------------
----

  void NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: CallDlgAuswaehlen (void)
  {
     if (!Get ())
     return;

     SFUAUS  dbsFUAUS;
     dbsFUAUS.fszNERG_Id = pdbsNERG->fszNERG_Id;
```

```
     dbsFUAUS.fkTyp      = dbsFUEBL.fkTyp;

    // Aufruf DLG-NZKTO/Ergebnisblatt_Auswaehlen
    //NNFuaus   *pDlg   =   new   NNFuaus   (this,   dbsFUAUS,   dbcBAUM,
dbcNERG.xdbsDbRef, m_rhcMbaum);
     NNFuaus *pDlg = new NNFuaus (this, dbsFUAUS, xdbcmNERG,  m_rhcMbaum);


     BOOL bRc = pDlg->DLG_NZKTO_Ergebnisblatt_Auswaehlen (pParent->IsDialog
());

     if (bRc && pDlg->IsOutput ())
     {
        BOOL bDurchfuehren = TRUE;

        *pdbsNERG  =  pDlg->GetSNERG  ();  //  Daten  wurden  vom  Dialog
geaendert.

  /* Nested Code has been factored out !! */
        NNMFUEB1.NNMFUEB1_06();

  /* Nested Code has been factored out !! */
        NNMFUEB1.NNMFUEB1_07();

        ListboxSelect (TB_NZKTO_EBLATTUEBERSICHT); // f. Inaktivierung von
Controls
     } // if

     delete pDlg;

  } // NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: CallDlgAuswaehlen

// NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: MarkiereUebergebenenSatz
// Author: HL
//
// Description
//
// Setzt den im Entscheidungsbaum ausgewaehlten Satz als Gesetzt
//
// Parameter               Description
//
// keine                   keine
//
// Return Value
//
// keiner
// ----------------------------------------------------------------------

  void NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: MarkiereUebergebenenSatz
(FSurrogat & rfszNERG_Id)
  {
     int iMultIdx;
     NNFerg NNFerg (this);
     NNFerg.GetAktNergIdx (xdbcmNERG, rfszNERG_Id, iMultIdx);
    //  in  ActionTabListbox  wurden  Tabboxeintraege  mit  Multiple-index
initialisiert
    // (und der 1. Satz markiert).
    //  Sie  koennen  also  hier  damit  wieder  geholt  werden  um  den  im
Entscheidungsbaum
    // markierten Satz auch hier zu markieren.
                // gefunden...

     if (iMultIdx >= 0)
```

```
     {
        for (int iLBIdx = 0; iLBIdx < pTb->GetCount (); iLBIdx++)
        {
  /* Nested Code has been factored out !! */
           NNMFUEB1.NNMFUEB1_08();
        } // for
     } // if

  } // NNMNzkto_Funktionen_Zum_ErgebnisblattDlg :: MarkiereUebergebenenSatz

 /*-------------------------------------------------*/
 /* Refactored CPP Methods and C Procedures         */
 /*-------------------------------------------------*/
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_01()
  {
    if ((pParent_In->GetDialogID () == DID_NZKTO_HOR) || (FALSE == xbBez))
    {
       EnableControl  (TR_NZKTO_ZWEIGBEZEICHNUNG,       FALSE);  //  Control
disablen
       EnableControl  (EF_NZKTO_ZWEIGBEZEICHNUNG_36,   FALSE);  //  Control
disablen
    } // if
  };
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_02()
  {
    if (pdbsFUEBL_BAUM_In && (FALSE == fszNERG_Id_in.IsDefault ()))
    MarkiereUebergebenenSatz (fszNERG_Id_in);


    Put (TRUE);             // Anzeigen der Attribute

    }         //         NNMNzkto_Funktionen_Zum_ErgebnisblattDlg         ::
NNMNzkto_Funktionen_Zum_ErgebnisblattDlg
  };
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_03()
  {
    if (bGetFromDB)
    {
       EnableControl   (PB_NZKTO_EBLATT_ABTRENNEN,   FALSE);   //   Control
disablen
       EnableControl   (PB_NZKTO_EBLATT_ANHAENGEN,   FALSE);   //   Control
disablen
    } // if
  };
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_04()
  {
    if (bGetFromDB)
    {
       EnableControl   (PB_NZKTO_EBLATT_ABTRENNEN,   FALSE);   //   Control
disablen
       EnableControl (PB_NZKTO_EBLATT_ANHAENGEN, TRUE);  // Control enablen
    } //
  };
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_05()
  {
    if (pdbsNERG->GetDB () != SQL_DELETE) // wenn Daten nicht geloescht
    ActionTabListbox (*pTb, dbcNERG, *pdbmcNERG, ACT_CHANGE); // aendern
    else
```

```
    {
       ActionTabListbox   (*pTb,   dbcNERG,   *pdbmcNERG,   ACT_REMOVE);   //
loeschen
       bDurchfuehren = FALSE;
    } // else
  };
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_06()
  {
    if (pdbsNERG->GetDB () != SQL_DELETE) // wenn Daten nicht geloescht
    ActionTabListbox (*pTb, dbcNERG, *pdbmcNERG, ACT_CHANGE); // aendern
    else
    {
       ActionTabListbox   (*pTb,   dbcNERG,   *pdbmcNERG,   ACT_REMOVE);   //
loeschen
       bDurchfuehren = FALSE;
    } // else
  };
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_07()
  {
    if (bDurchfuehren)
    {
       NNFerg *pBase = new NNFerg (this);
            //pBase->IsBaumUsed  (dbcBAUM,  phcMbaum);  //  Daten  ergaenzen
(fuer Sortierung)
       pBase->IsBaumUsed (xdbcmNERG, m_rhcMbaum); // Daten ergaenzen (fuer
Sortierung)
       delete pBase;

       ActionTabListbox (*pTb, dbcNERG, *pdbmcNERG); // neu darstellen
    } // if
  };
 /* New Method created from refactoring             */
  void NNMFUEB1::NNMFUEB1_08()
  {
    if (iMultIdx == pTb->GetIndex (iLBIdx))
    {
       pTb->ChangeCurrentIndex (iLBIdx);
       pTb->SetFocus ();       // (damit Enter als 1. Aktion  funktioniert)
       break; // Verlassen der Schleife
    } // if
  };
```
