

Press Release blue river software GmbH

Issued: February 2009

Author: Productmanager

blue river software GmbH
Aeussere Sulzbacher Str. 159-161
90491 Nuremberg Germany

Phone.: ++49-911 - 206 26 - 0
Fax.: ++49-911 - 206 26 - 18

Homepage: <http://www.blue-river-software.com>

E32: All software life cycle phases in one tool

General Development Environment

Software development is becoming increasingly important in all areas of engineering. The procedures in virtually every technical product are meanwhile controlled by software and the scope of this control is growing constantly. Where the scope of software previously tended to be low due to high hardware costs, more extensive product functions are now also solved with more extensive software.

Software engineers are therefore faced with a variety of problems:

- Efficient software is to be produced in spite of a drop in hardware costs.
- The implementation language changes due to the requirements, e.g. from Assembler to "C" or from "C" to C/C++.
- Increasingly higher quality demands are also placed on software development, which is expressed in the requirement for understandable development phases and standardized documentation.
- Old and well-proven techniques are to be included in the new development of a product. Not every software code of the past can be simply ignored. Reuse is necessary for efficiency reasons.

The use of tools in software development is essential to enable a product to fulfil these requirements. This results in a requirement for tools for the following purposes:

- Support in analysis, specification and design to permit more efficient development.
- Support in the implementation phase for creating the source code.
- Compiling the source code into an executable program.
- Testing and debugging the created program.
- Support for maintenance and fault clearance.
- Handling subsequent specification or design changes in the implementation phase (reengineering).
- Support for the integration of existing well-proven code into the new project (reverse engineering).
- Documentation tool for creating standardized development documentation.

Tools are naturally available for each of these phases. However, if separate special tools are used for each requirement, this results in new problems such as a non-standard user interface for these tools and various interface problems. The probability that the analysis tool and implementation tool then provide a compatible and standard documentation is relatively low.

This creates the requirement that all subtasks in the development process are carried out with one integrated tool if possible.

A tool which meets this requirement is "E32 - the modern life cycle" from *blue river software* GmbH in Nuremberg, which we now present below.

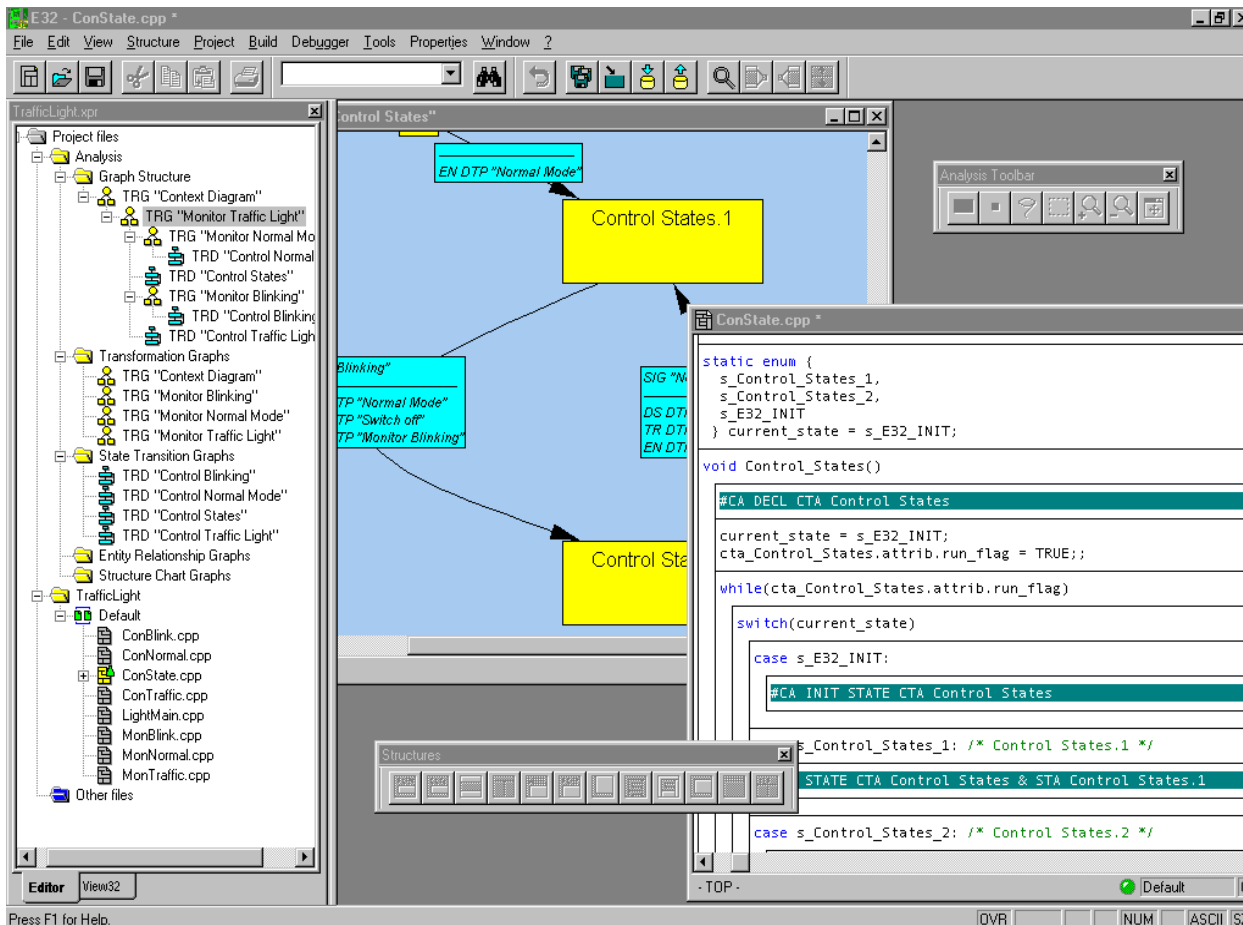


Fig. 1: E32 User Interface

From Analysis to Design

Various software development methods are used to perform out a comprehensible project analysis.

E32 uses a structured analysis approach with the SA/RT method according to Ward & Mellor. The application starts with the feasibility study and continues via the system analysis to the specification and software design. The Ward&Mellor method is based on the Yourdon/DeMarco approach. The system to be developed is described by three types of graph: the data flow diagrams (or transformation graphs), the state transition diagrams and the entity relationship diagrams.

These graphs have become accepted as a working method in the early project phases.

E32 is designed so that software developers with only a slight knowledge of the method can also use it successfully. A clear graphic specification can be created which acts as a basis for further development. Such a specification is very important for communication between the developers (also between developers of different disciplines such as software developer <--> hardware developer or software developer <--> mechanical engineer) and is also adequate without considering all the method requirements. A knowledge of the Ward&Mellor method is, however, essential if the facilities offered by E32 are to be fully utilized or code is to be generated automatically.

The different graphs are created using the graphic method editor. Windows-based operation with mouse, context menus and toolbars goes without saying.

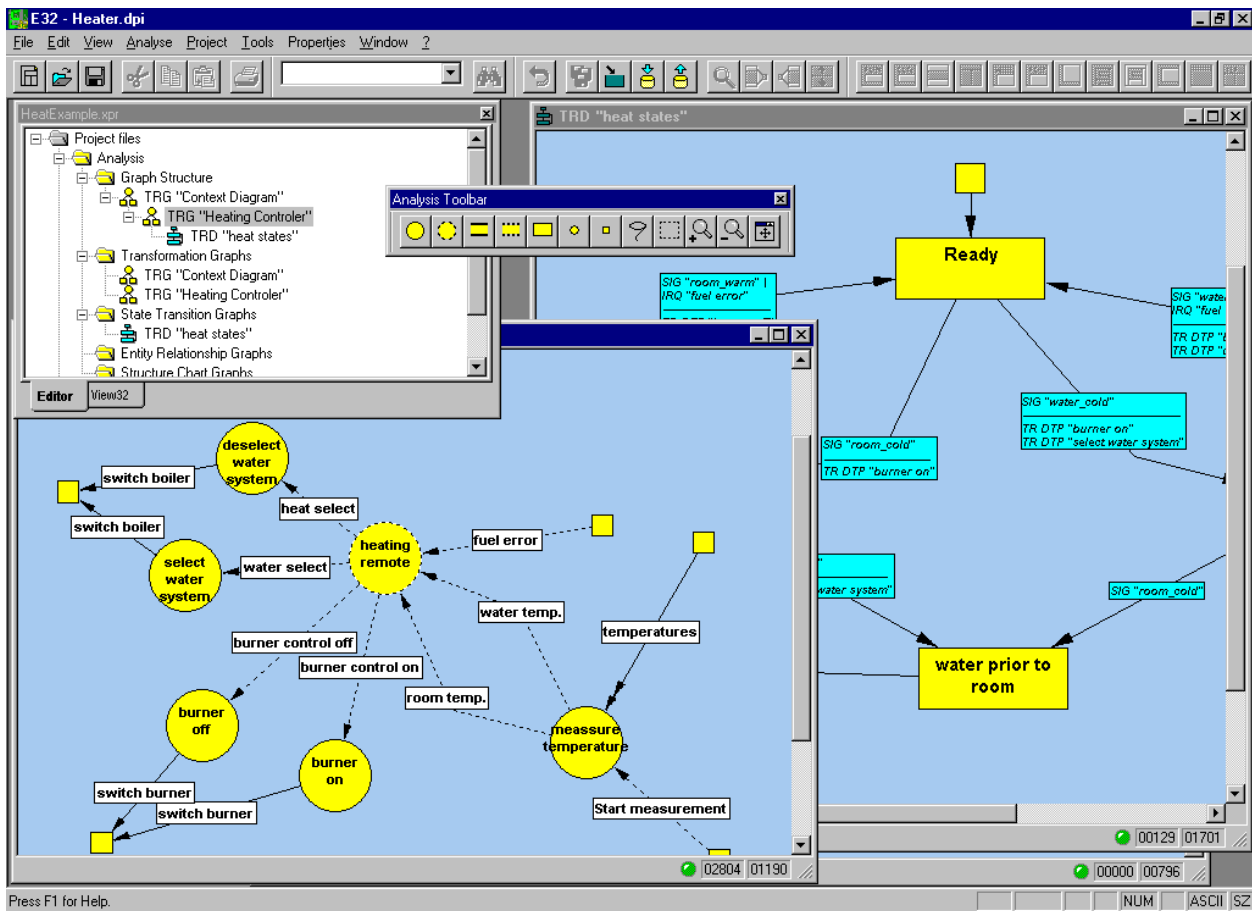


Fig. 2: User Interface of E32 Analysis Module

The zoom function permits an overview and accurate detail work.
The graph hierarchy can be seen in the project tree.

The analysis tool can be used to carry out a global analysis of all objects of the project and a detailed analysis of individual graphs.
Faults can be cleared directly and on-line.

Code Generation: From Design to Implementation

If a complete implementation model based on the Ward & Mellor method is available, this is also a programming specification. It contains considerations which reflect the subsequent software architecture. This also means that it contains all information which recurs later in the program. As both the Ward & Mellor method and the software design rules are mathematically proved and consistent models, it is possible to create programs automatically from the implementation model.

The E32 code generator is an intelligent, rule-based tool which can create an operational 'C' program from the generated specification. The result is a program proposal, whose completeness depends on the structure of the specification. The generation rate can reach 100%. Even lower generation rates offer the invaluable advantage that it is no longer necessary to start programming again from the very beginning. The work already carried out in the specification need not be repeated. This saves a lot of time, avoids many faults and thus creates economic advantages.

Another advantage not to be underestimated is the fact that the specification, design and program are matched and fully consistent.

The starting point for code generation is provided by the Ward & Mellor diagrams. These describe the process behaviour in the real world using the state transition diagrams. The transformations are derived from this and describe the actual information processing. Provided the graphs are consistent, code can be generated from them. In the generated program, the state transition graphs form

control tasks, which manage the associated transformations as a task system. The control tasks are usually completely programmed out and the transformations are program frames with the following content:

- system description from the design system
- data agreements and declarations
- coding areas.

The system description is taken from the design system via the description screen belonging to the task. All the data agreements for the generated program part are defined automatically in the declaration part. The coding areas contain the entries from the primitive process specifications in the form of comments.

In general: A compilable unit is created in structogram form for each graph. The content of a structogram mapped from a transformation graph differs according to whether the graph has been defined at task or module level. It contains task or procedure declarations depending on its origin and represents a prepared program frame whose body must be programmed out.

A structogram that has been mapped from a state transition diagram contains the complete, compilable and working code of a control task or procedure.

The structure of the generated structograms is clearly arranged and identical for all projects. This layout is supported by appropriate designations of coding areas and fixed masks. The designations from the specifications are used again in the generated programs. The developed software actually conforms to the specification and documentation derived from it. This considerably simplifies incorporation in projects.

Implementation in Structogram Form

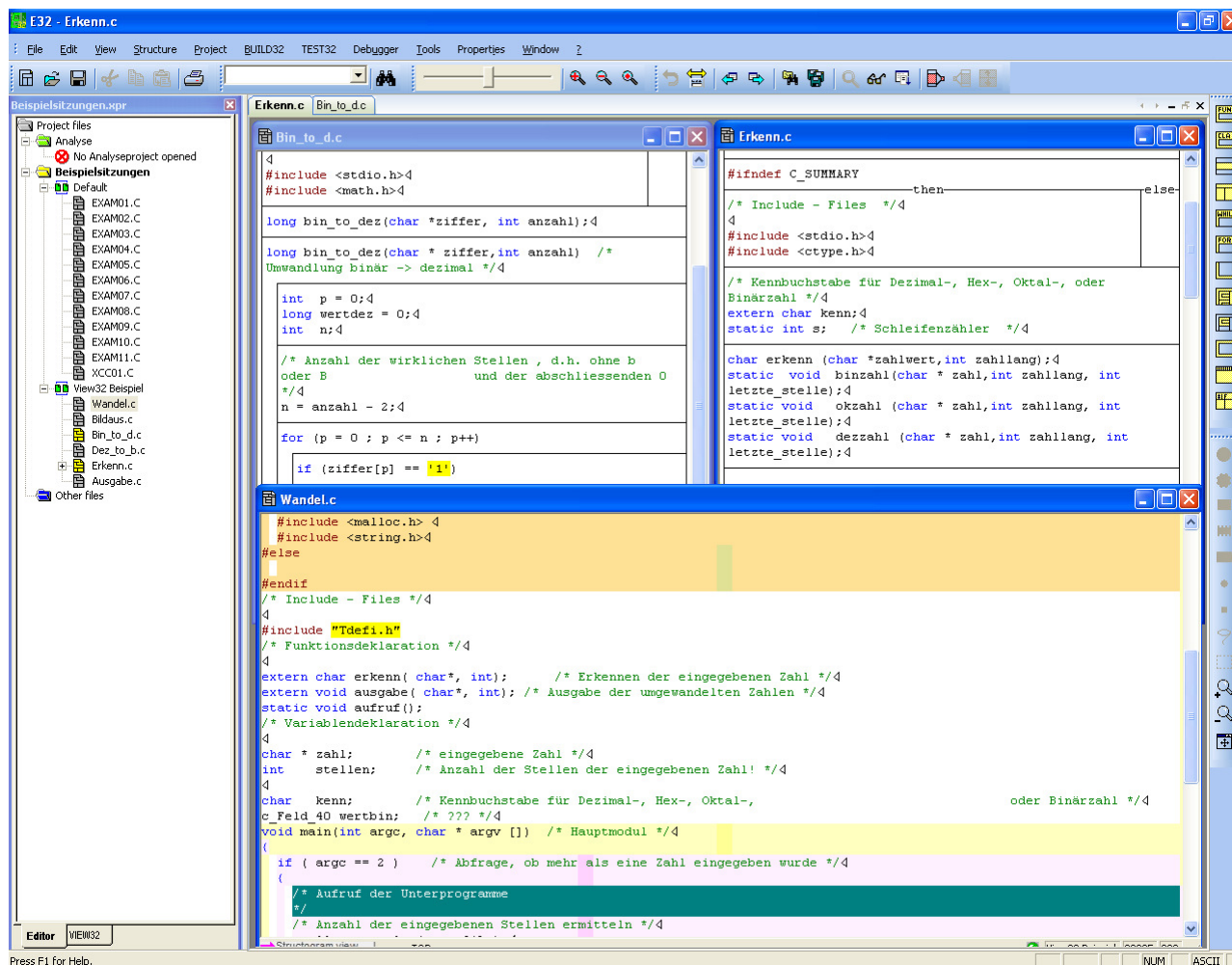


Fig. 3: E32 Interface with Project Tree, Structogram and Docking Output Window

The E32 structogram editor is a convenient tool for recording source code in structograms. It is equipped with all the functions expected of a modern editor, with the subtle difference that the source code can be entered directly into the graphically displayed structure. The adaptable syntax colouring offers extra support.

As the key words marking the block limits are set automatically as source code on saving the structogram, these do not have to be entered. This eliminates programming errors due to forgotten block ends or incorrectly interleaved blocks.

The clear graphic layout of the structograms together with the possibility of masking complex program parts ensures that even extensive software modules and functions always remain manageable. This mask structure is integrated into the project tree, which simplifies fast orientation in the complete project.

Source Generator

A corresponding source file is automatically created from a structogram by the integrated source generator. The source generator converts the input structograms into a source code that can be understood by the compiler. The XSF structogram format used in E32 saves the structogram information directly in source text. Each target language naturally has a specific source generator. The format of the source code to be generated can be conveniently defined by the developer.

Syntax-check Control System

Development with E32 is based on any target compiler as the choice of the compiler in project development is influenced by many factors and cannot be solved in a general way.

E32 offers you an open interface via which the usual compilers can be activated. Parameter files for many compilers are already included in the scope of delivery. You can easily configure your own special parameters.

The compiler error messages are evaluated and displayed directly in the structogram by positioning the cursor in the line with the error. Batch or make files can also be edited, e.g. if several files are to be translated together. If errors are reported in such a case in a file which is not currently loaded in E32, its structogram is then loaded automatically so that the error message can be displayed. This enables errors to be cleared quickly and the compiler started again.

Debugger Interface

A similar situation applies to using the debugger. On the one hand, special debuggers must be used according to the requirements, on the other hand, the developer would also like to use the familiar tools in the debug phase and not have to get used to another user interface.

A debugger interface which is already supported by debugger manufacturers is therefore integrated into E32.

Project Information System

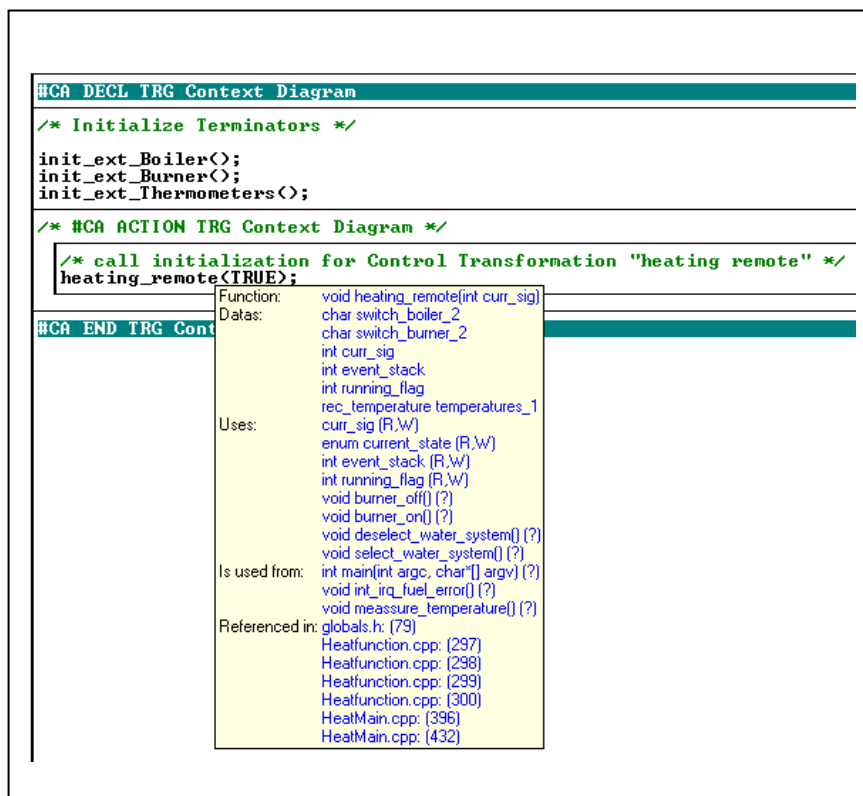
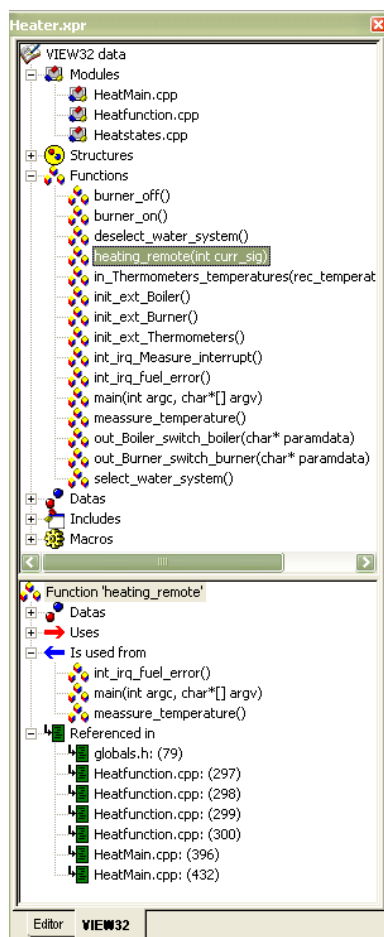


Fig. 4/5: Overview with E32 Project Information System

Project work is supported by a project information system and a source browser. Besides data, types, functions, etc., the classes, methods and attributes of the project are also listed in separate windows. You can move directly to the corresponding definition in the structogram by double-clicking an entry.

This provides a fast overview of the data, functions, class definitions, etc. used in the project. It is also possible to conveniently navigate through the source code.

The sources generated for compiling are linked to the document database by the document compiler and document linker.

The generation of the project database can also be adapted to the different language dialects without the need for source changes.

Reengineering

It is shown in practice that the system to be developed is subject to new and/or changed requirements in all phases during the project duration. It is therefore also possible to carry out changes at the specification level during the implementation or test phase. These changes are then incorporated in the program without the need to completely develop the manually created program parts again.

The close connection between the specification graphs and the implementation structograms means that changes in the structograms are also immediately visible in the analysis module.

Reengineering/Reverse Engineering Function

Using the **source reengineering** function, existing source code can be converted fully automatically to structograms which can be processed by the E32 editor. These structograms can then be used, for example, to create structogram print-outs, which are necessary for obtaining documentation that is clear and easy to understand. The sources generated in this way can naturally be handled in exactly the same way as sources directly edited with E32.

A reverse engineering function for generating an SA/RT specification from existing old sources is under preparation.

Documentation

The project documentation in E32 is provided by files in Rich Text Format.

This initially comprises

Analysis documentation

Product analysis and specification is rounded off by generating the necessary report. The major task is the graphic representation and modelling of a system. Apart from creating the model, one of the most important tasks in a project is the model documentation, which records the working result. This can be handed over to the client or stored together with the remaining project documents. The printed version of the model is usually the basis for acceptance by the client. The optimum presentation of the working results is therefore all the more important for the project manager.

The report generator documents the graphs and the associated information. All the facilities are naturally available for designing the report yourself, i.e. both the layout and the scope of contents can be changed.

Each graph is obviously included in the report as a graphic representation and all the entities such as memory, processes, flows, etc. contained in each graph are listed. The report also contains all the other information of the entity descriptions.

It is therefore possible to assign the associated data to each data flow and each process on paper and thus study the design at your desk. This is particularly advantageous if the analysis is to be discussed with the client or a department.

The screenshot displays the E32 software interface. On the left, a 'Transformation Graph: Context Diagram' is shown, featuring a central 'control heating' entity connected to 'Thermometers' and 'water prior to room'. Below the graph is a caption: 'Figure 1: TRG "Context Diagram"'. On the right, a report titled 'Report on the Heater project:' is open, showing sections for 'Transformation', 'Control Flows', and 'Relations to other entities'. The report text includes details about the heating system's control logic and data flows.

The implemented source code is shown in the

Implementation documentation

E32 creates documentation for project modules in the clear and space-saving structogram form. The structogram print-outs can be designed to meet user requirements using the extensive layout parameters. It is possible to print out the structograms directly or to incorporate them as part of the complete documentation or in other documents via the integrated output in Rich Text Format.

The screenshot displays the E32 software interface with several panels. On the left, there are sections for function documentation, including '2.3 Functions', '2.3.4 Function: "heating_remote"', '2.3.3 Function: "deselect_water"', and '2.3.5 Function: "int_irq_fuel_error"'. Each section includes a table with columns for 'Data', 'uses', and 'Is used by'. The 'heating_remote' table lists data like 'curr_sig', 'event_stack', 'running_flag', 'switch_boiler_2', 'switch_burner_2', 'temperatures_1', and 'select_water_system'. The 'deselect_water' table lists 'switch_boiler'. The 'int_irq_fuel_error' table lists 'paramdata'. In the center, there is a 'call trees' panel showing a hierarchical tree structure of function calls, including 'Measure_interrupt', 'measure_temperature', 'heating_remote', 'burner_off', 'burner_on', 'deselect_water_system', 'select_water_system', and 'in_Thermometers_temperature'. On the right, there is a panel titled '3.2.1 Function: "heating_remote"' showing a detailed view of the function's implementation, including code snippets like 'running_flag++;' and 'select_water_system(...)'. The interface is overlaid with a large, stylized graphic of a rolled-up document corner.

Integrated Work Environment

E32 users have direct access to their project files for all project phases via a project tree. In spite of different programming languages, compilers and debuggers, a common standard user interface is available with all the controls you are accustomed to with modern 32-bit tools. Frequently used actions can be initiated by double-clicking with the mouse and the context menus of the right mouse button offer support for local commands. The freely defined functions and positions of the toolbars enable an individual work environment to be created.

All the necessary actions in the development process can be executed directly from the E32 interface via the many connection options for tools such as compilers, debuggers and project management systems.

E32 therefore meets the requirement for an integrated software development tool.

E32 can run under 32-bit Windows systems and is available as stand-alone terminal and network version.

The implementation function integrated into E32 is also compatible with the familiar X32 implementation tool from *blue river software GmbH*.