case/4/0

The Introductory Version

microTOOL

case/4/0

The Introductory Version

Welcome to the world of **case/4/0**. Is professional software development your daily business? Then take a look at our CASE-Tool **case/4/0**. We have developed it for you.

Discover case/4/0 ...

... and convince yourself just how simple and effective this tool is. Take a good look at every aspect. Try structuring, animating, printing and generating—from Analysis through to source code.

If you want to know more about case/4/0 ...

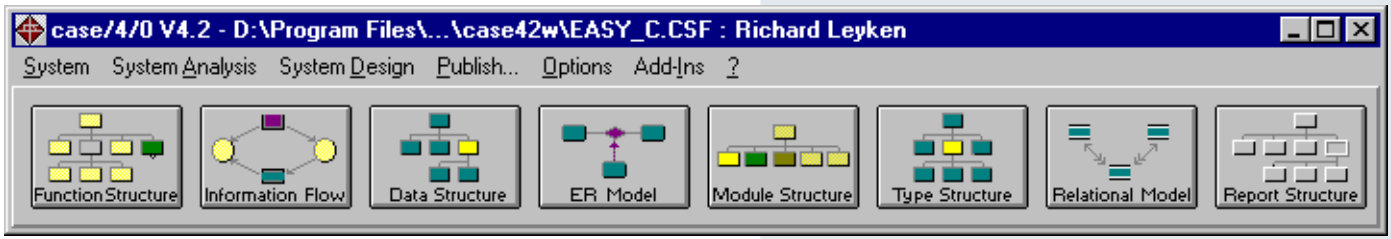
... familiarize yourself with the overview of the product and comprehensive information on the following pages, with which we hope to stimulate your journey through **case/4/0**.

Copyright

case/4/0—Analysis, Design and Programming Tools for LAN Operation

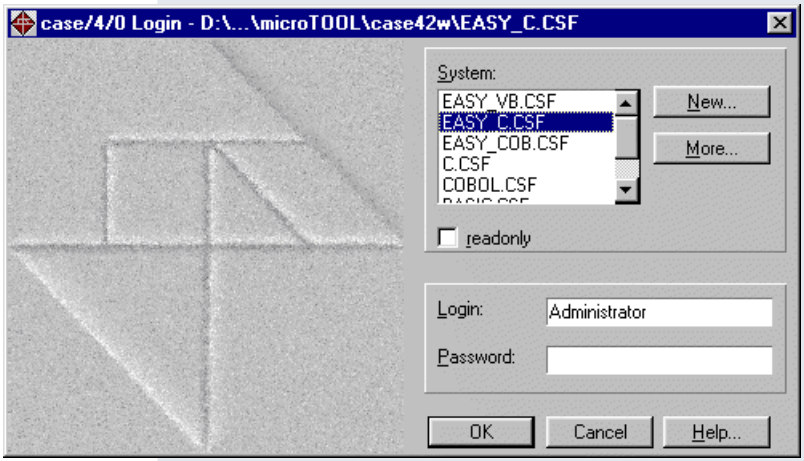
What is **case/4/0**? What can it do for you? A look at the menu provides an answer to this question:
case/4/0 integrates

- Tools for ***System Analysis***
- Tools for ***System Design*** and
- A Publisher for the creation of target group documentation



Whether you are developing commercial or technical applications, regardless of whether you are specifically concerned with dialog processing, database communication or real-time applications, **case/4/0** offers you comprehensive tool support. You can enlarge the scope of possible uses even further through your own add-ins and by using the integrated Script language. From the toolbar you can see that **case/4/0** offers a number of graphical outlines for System Analysis and System Design. This is for good reason: **case/4/0** is conceived for task-share projects in which complexity should be reduced, where progress is made according to plan and architectural decisions are reached consciously. Integrated ***Version*** and ***Configuration Managers*** enable the effective steering of the software engineering process—especially in large IT organizations.
case/4/0 is ready for any project size: The model-driven approach that **case/4/0** consequently follows makes sure of that. The concentrated, graphically structured knowledge in the models is actively employed by **case/4/0** for quality control and for the generation of results that lead you on toward your goal.

Ready for major projects



From systems analysis to programming, from data administration to quality control—the spectrum of **case/4/0** uses is broad indeed. The **case/4/0** Manager in **case/4/0** assists in the unambiguous definition and delineation of the tasks and rights of project members which is especially necessary in large projects.
Practically speaking, it works like this: Every **case/4/0** user must register in the login dialog for the system to be edited. The **case/4/0** administrator then defines which results a user can edit by means of an administration component.
It helps the administrator define individual users and user groups for each **case/4/0** system. Each group or individual user can be given individual rights to edit certain types of results. The distribution of rights can be quite complex, in that the states that the results pass through in the process of the software life cycle are taken into consideration as rights are granted.
The whole procedure can be done by simple Drag & Drop: Users or user groups are dragged to the states for which they should have rights. Detailed editing rights can be "clicked together" in a dialog field.
The result for the user looks like this: All menu functions or command buttons which start functions the user is not entitled to are disabled and cannot be selected.

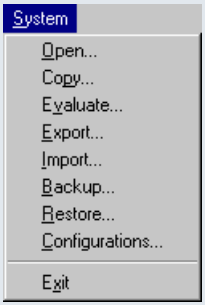
Central Repository for Multi-User Operation

As soon as you begin constructing a model, **case/4/0** automatically makes sure that your results fit in with those from all other project members without contradictions. This is possible because **case/4/0** was conceived especially for multi-user-operation. It includes an integrated LAN-Repository which actively secures the consistency of all the results.

***case/4/0** includes three sample systems for you to edit as you choose:*
*Choose **EASY_C**, if you are interested in developing with Microsoft C/C++, Microsoft Foundation Class Library (MFC) and ODBC-Interface, **EASY_VB**, if you develop Microsoft Visual Basic applications, and **EASY_COB**, if you would like to follow the development of a Micro Focus COBOL application with Micro Focus Dialog System.*
The results available for you to edit depend on the login ID you use. In the sample system, for example, the project leader Richard Leyken (login: RL) and the data administrator Gunnar Holl (login: GH) did the modeling. Go ahead and step into their roles if you want to see how the effects of user administration feel. In order to follow all the work steps shown here, sign in as "Administrator". You don't need a password to edit the system.

Open architecture—flexible interfaces

case/4/0 conforms flexibly to the most widely varying forms of project organization. **case/4/0** is componentware, as defined by Microsoft COM. This means that a part of the internal architecture is published in the form of exposed classes and is described in a type library. **case/4/0** is thereby open for user-specific, object-based communication with other COM-ready tools.
Additionally, the user can access the integrated LAN repository via a C-programmer interface and **case/4/0**'s own script language—for example in order to evaluate development results (***System/Evaluate***). We will show the opportunities for need-based configuration presented by open architecture in an example later on.
By the way: besides ensuring consistency, **case/4/0** also guarantees the formal correctness of your results, because it has ...



*The **case/4/0** repository guarantees secure multi-user operation. With its **Import** and **Export** functions, **case/4/0** supports the division of a system into parts, as well as the logically consistent integration of decentralized engineering results.*

... a firm foundation: Structured Methods in case/4/0

case/4/0 offers two mutually coordinated methods for **System Analysis** and **System Design**. These combine the well-tried concepts of

- *Structured Analysis* from DeMarco with Real-Time Extensions from Ward and Mellor,
- *Entity Analysis* from Chen,
- *Structured Systems Design* based on Page-Jones and Yourdon,
- *Relational Data Modeling* from Codd

into a continual process. Strong practical impulses have lead to continuous development updates of the structured concepts. Therefore, for application development in C, COBOL or Visual Basic on relational data bases they are, now more than ever, the ideal structuring aid. They are perfect for planning large projects—without strategic risk.
With the help of its script language, **case/4/0** can also be programmed for other target environments with no problems.

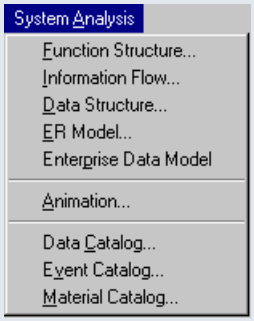
System Analysis Highlights

What does **System Analysis** mean in **case/4/0**? Let's take a quick glance at the menu element of the same name. Functions, data and behavior of an application system are modeled from a problem domain point of view by the graphical System Analysis tools. For this purpose, **case/4/0** offers five diagram types:

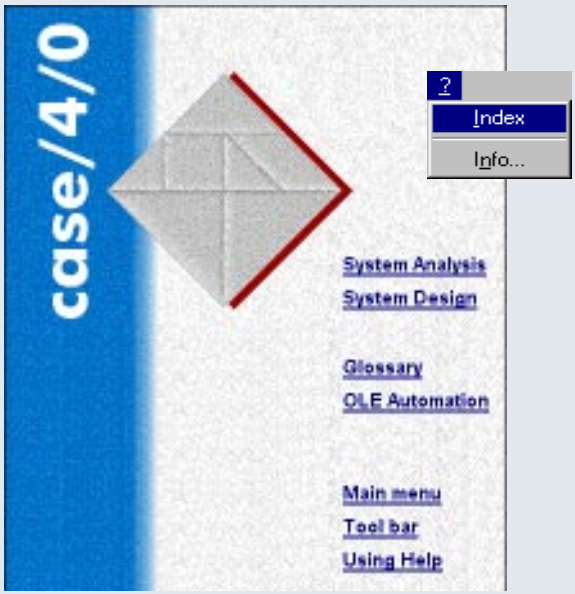
- The ***Function Structure***: a simple tree diagram for the functional decomposition of a system,
- The ***Information Flow***: a data flow diagram extended by the presentation of control and material flows,
- The ***Data Structure***: it structures data hierarchically and describes views of entity types,
- The ***Entity Relationship Model (ER Model)***: it illustrates data objects and their relationships,
- The ***Enterprise Data Model***: it offers graphic editing functions, navigation and zoom aids in order to integrate entity relationship models into a complete model without redundancy.

Data, ***event*** and ***material catalogs*** complement the graphical models by means of formal and textual descriptions.
What the menu doesn't show you is that **case/4/0** provides three further tool components when developing information flows:

- The ***State Transition Diagram***: for system control specification,
- A ***Dialog Designer***: for the graphical design of the user interfaces,
- A syntax and context sensitive ***Editor***: for the description of the so-called *transformation behavior*, i.e. the principle working process of the processing functions.



Choosing EASY_C, EASY_VB or EASY_COB lands you in the middle of a project team's work. The team has the task of improving the processing order of the—fictitious, of course—"Easy Furniture Company". Some results are already available, so that you have at least one example of every possible type of systems analysis results.



The **case/4/0** *Online Help can answer your questions on how to use the tool. You can learn more about the Help under the question mark in the main menu.*

Once the functionality, control and user interface of a system have been structured from a problem domain point of view, then the organization of the specified enterprise processes can be dynamically simulated and verified. **case/4/0** supplies you with a true to life impression of the new system under the menu point ***Animation***.
With the aid of System Analysis **case/4/0** offers you a system specification, which

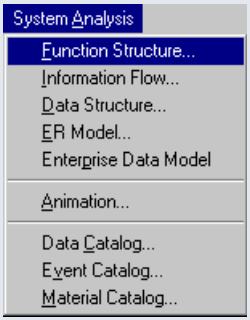
- consists, in the sense of modularization, of components with tight functional binding and loose data coupling—a necessary requirement for easy maintenance of a system,
- already works, that is to say it describes the organizational control of the functions found from a problem domain point of view,
- establishes the redundancy-free, logical order of data and thereby the strategic potential of the new system.

The Function Structure

A function structure provides a structured overview of all of the problem domain functions in a system. With **case/4/0** you are able to distinguish between several function types within a function structure because...

... control, presentation, processing—function does not equal function

case/4/0 knows:
Processing functions (light yellow): these change the contents or structure of data or material.
Dialog functions (green): these give the future users of the system the chance to intervene in the system process by means of menus and dialog boxes.
Control functions (yellow): these coordinate the activities of the processing and dialog functions and react to events in the system.
The reuse of system components can be planned and modeled with *Recurrent Structures* (white).



*If you choose **Function Structure**, then you can select a diagram to edit from the list of all diagrams of this type available or set up a new one. In the sample system you will find the EASY Furniture Company's function structure.*

This is the toolbar for editing a function structure:

Function

Recurrent Structure

Part: Separate Subtree

Block: Structuring Aid

Condition

Action: Link elements

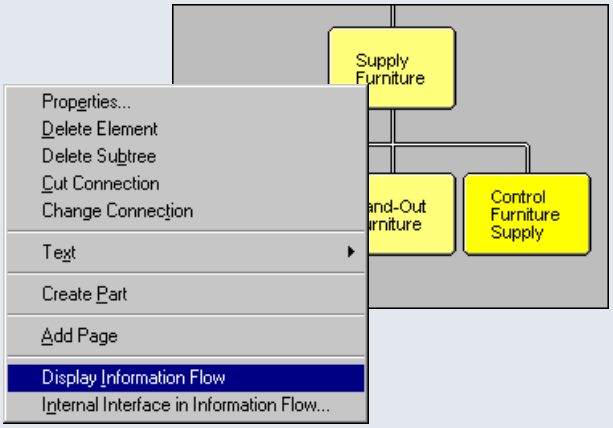
For the software developer, the function structure marks the path through the system specifications, for the project manager it forms the basis of task distribution, project planning, and project control.

The Information Flow

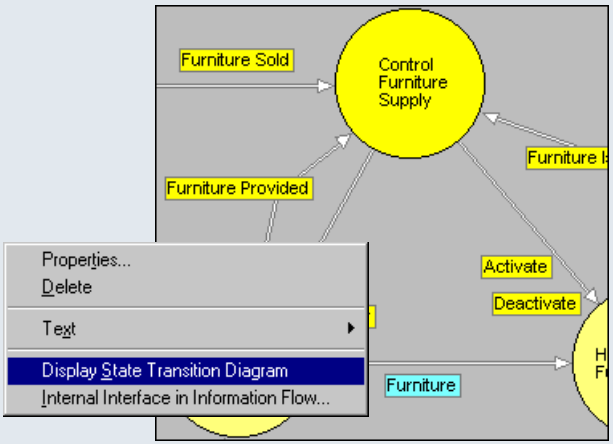
Data Flow Charts are the most flexible and best-loved element in Structured Analysis. There's a good reason for this: their notation is very simple and they are extremely suited for communications—even for the "method laity". Therefore it is no great surprise that event-oriented extensions to the information flows were suggested by various authors at the end of the eighties, so that they could be implemented even further—for example for the specification of real-time applications.

Structuring a Business Process clearly

We have gone one step further and have integrated the development of dialog applications with graphical user interfaces in the world of structured methods. Besides the exchange of data, material and events between functions, interfaces and stores, **case/4/0** also specifies the "inner life" of functions.
Processing functions can be divided into sub-functions for which the interactions are illustrated in an information flow. The consistency of the resulting information flow hierarchy is taken care of by **case/4/0** top down.
Dialog functions are specified via the graphic design of the corresponding dialog fields and menus. Furthermore, the definition of callbacks determines how the dialog function should react to the user's actions. This is taken care of by **case/4/0**'s own *Dialog Designer*, initially unconnected to the future system.
Control functions are specified in *state transition diagrams*. They illustrate how control functions react to events dependent on the current state of the system.



If you click on an element with the right mouse button, it tells you what you can do with it. Try it out: Click on Supply Furniture *and choose* ***Display Information Flow*** *in order to branch off into the information flow for the function. A double click on* Supply Furniture*, by the way, would have the same effect.*



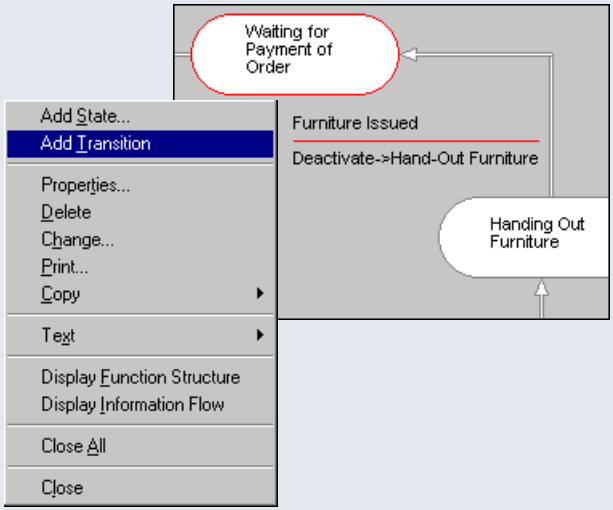
There's a state transition diagram for the function Control Furniture Supply. *Go ahead and take a look at it.*

The notation of the state transition diagram is quite simple: oval elements represent the system states and arrows represent the transitions. A transition is labelled with the *condition* which must be fulfilled in order for the transition to take place. Under it, the *actions* carried out by a system in order to reach the new state are entered.

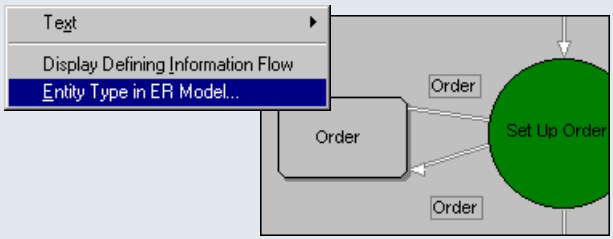
The Entity Relationship Model

Information flows show which data is exchanged, processed or created by the functions of a system. Data from the problem domain must first be structured before detailed information flows can be created. **case/4/0** supplies you with the entity relationship model (ER Model for short) to take over the task of semantic data modeling. The name "entity relationship model" expresses exactly the elements which are contained in a diagram of this type, namely:

- *Entity types*, symbolized by rectangles. These represent collections of similar problem domain objects and consist of attributes which refer to uniquely defined data elements in the **case/4/0** data catalog,
- *Relationships*, symbolized by diamonds and connected to entity types, represent the problem domain relationships of the entity types. They are labeled with their cardinality and semantic type,
- *Associative entity types*, i.e. elements, which can be interpreted as relationships on the one hand, but also as entity types on the other, because they possess self-describing characteristics, i.e. attributes. They are represented by rectangles which are connected to diamonds by arrows,
- *Sub and super relationships* which illustrate specialization or generalization of entity types and which can be recognized by a blue triangle.

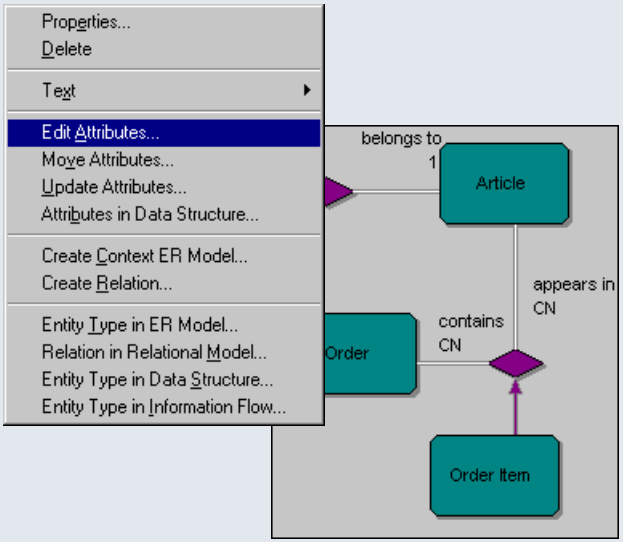


If you want to set up a new transition, then the name of the condition can only be the name of an event which in the information flow is connected to the control with an incoming arrow. This type of dependency between diagrams is taken care of by **case/4/0** *automatically.*



In the Sell Furniture *information flow you will find an inconspicuous, grey box—the* Order *entity type. If you click on this with the right hand mouse key, it shows you the path to the ER Model from which this entity type originates.*

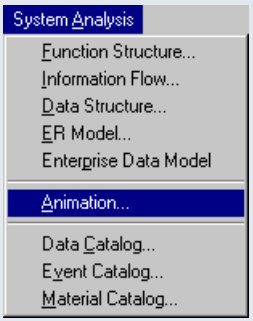
Entity types from the ER Models can be inserted in information flows as data stores. Later updates to an entity type are consistently passed on by **case/4/0** to all information flows affected.
How do you structure an ER Model? In **case/4/0**, we suggest that you orient your ER Models to the subject matter. Redundancies between ER Models are completely permissible, and are automatically taken care of by **case/4/0**. If you want to know in what problem domain relationships an entity-type is involved, in all ER Models, then you can generate a *context ER Model* which shows this very aspect.
All entity types at once? No problem for **case/4/0**. ER Models can be integrated, redundancy free, into an *Enterprise Data Model (EDM)*.
The design of ER Models in **case/4/0** entailsof not only conceptual clarity, but also a very practical advantage for software design and implementation: From an ER Model you can derive a relational data base design in the form of a relational model at the click of the mouse. (We will deal with relational models comprehensively later). And from it—once again, at the push of a button—you can generate SQL-Data Base description.



This is where the Order *entity type comes from: The* Order Transaction *ER Model. With the* ***Edit Attributes*** *command you can see what's behind the box.*

Your Specification Learns to Operate

Once system control, user interfaces and data storage are correctly specified within the context of an information flow as concretely as shown above, then it is just a small step to the ***Animation*** of the system behavior. By this we mean the systematic run-through of the business processes which are described by one or more information flows.
But there is still something left to do! First of all the following question must be answered: How do processing functions principally react to input data, events or incoming material? The answer to this is given by the definition of the *transformation behavior* of the elementary processing functions in the informa-



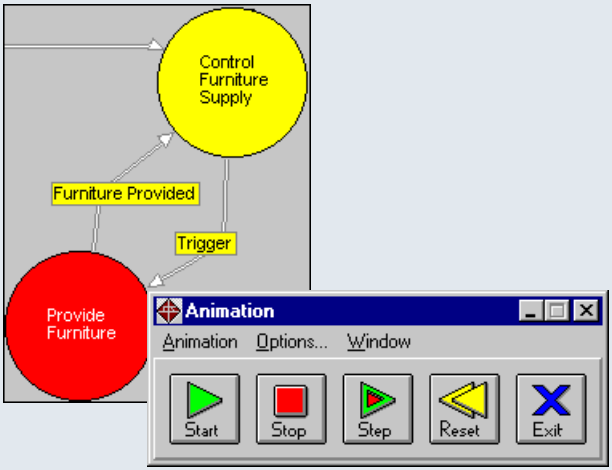
Go ahead and try out animation! The menu item for it is in the main menu under ***System Analysis****.*

tion flow. In the form of simplified rules and with active support from a context and syntax sensitive editor, the inputs from which the function generates its outputs can determined. In the animation, the transformation rules appear in place of the detailed algorithmic logic of the functions. If this requirement is met, then **case/4/0** only needs information concerning which *scenario* you would like to dynamically examine in order to start. A scenario consists of one or more information flows which, taken together, describe a business procedure, and is extended by outline conditions. An outline condition is, for example, the number of external events which should happen in the process of the business procedure.

And this is how the animation looks: For the purpose of demonstrating a business procedure, **case/4/0** marks the use of information paths, and the activation of functions in information flows by moving colored highlighting. Even more, if a dialog function is activated during the process of an animation, then the user interface designed for it appears on the screen. Now you can intervene in the business process—just as the user will later—by clicking on a dialog element, for example. In the information flow you can then follow the highlighting and see which functions are set in motion by your actions. If another dialog function is activated, then a new dialog box will appear on the screen. The business process can therefore be followed abstractly by information flows in the form of moving highlights and concretely, as dialog flow.

If the animation was successful, then the result of the System Analysis with **case/4/0** is a complete and functional specification from the problem domain point of view. An optimal basis for software design is therefore created by **case/4/0**.

We suggest that you animate the scenario Furniture Sales and Deliveries. *For this scenario, the EASY Furniture Company project team has already set up the outline conditions. With* **Store Values** *for example, it has defined the* Furniture Warehouse *as having 100 pieces of furniture (material stores) available. Select* **Start** *to begin the animation.*



The animation runs automatically with **Start**. *Don't forget to step into the shoes of the future user.* **case/4/0** *will wait for your reaction when a dialog field is shown.*

System Design Highlights

System Design with **case/4/0** means

- *Large-scale design*, i.e. data base design and the development of software architecture based on the results of System Analysis,
- *Small-scale design*, i.e. detailed specification of the algorithms of all complex functions as a precondition for implementation.

For these steps—as can be seen in a glance at the main menu item ***System Design***—**case/4/0** offers three types of diagrams:

- The ***Module Structure***: a tree diagram that, on the one hand, presents a graphical contents list for a module, and on the other, acts as a sort of "container" for the source code. Who calls who—this aspect is also displayed by module structures,
- The ***Type Structure***: the technical counterpart to the logical data structure, and
- The ***Relational Model***, for the design of the relational data base.

A fourth type of system design diagram is also accessible, not from the main menu, but via module structures:

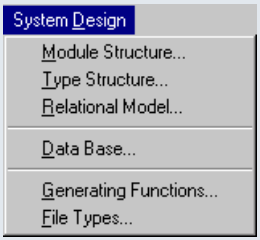
- The ***Implementation Tree***: it illustrates the algorithmic logic of complex functions of the module structure in the form of graphical pseudo code.

From these graphic results, the standard **case/4/0** package generates code for the following target technologies:

- Micro Focus COBOL with Micro Focus dialog system and Embedded SQL for IBM Database/2, as well as
- Microsoft C/C++ with Microsoft Foundation Class Library and Microsoft ODBC-Interface,
- Microsoft Visual Basic with forms and recordsets.

That won't work with your target environment? With your own ***Generating Functions***, found under the menu item of the same name, you can expand the generating capacity of **case/4/0** to other target systems.

Data Bases are generated on the basis of relational models.



Before you check out what's hiding behind these menu items in detail, let **case/4/0** *do some hard work and generate the design results mentioned in the menu from the System Analysis results. Because ...*

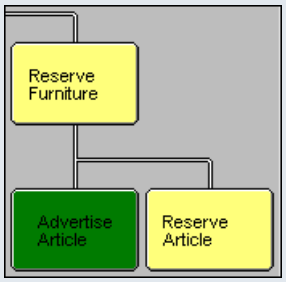
System Design's primary objective is to transfer the results of analysis into software design without loss of problem domain knowledge and architectural characteristics. To reach this objective, **case/4/0** offers three different procedure alternatives:

- *Explicit module development:* This means setting up module structures and transfering the required functions and data from system analysis to your module design, with the help of **case/4/0**.
- *Automated recurrent design steps:* If you realize that you often require modules of a similar structure, then you should design a generally valid setup—we call this a *Module Template*—for these steps. Besides the graphical structure, a module template contains so-called *Generating functions*, which are formulated in a simple script language and give you a chance to influence the results of code generation. **case/4/0** creates concrete module structures from module templates. To do this you only have to inform the tool as to which up-to-date results should be used in the module structure. Module templates are therefore a suitable instrument for illustrating and carrying out enterprise-specific structure and programming standards. This is due to the fact that **case/4/0** takes care of compliance to them on its own.
- *Leave the work of modularization to* **case/4/0** *and go ...*

... from Analysis to Design by mouse click

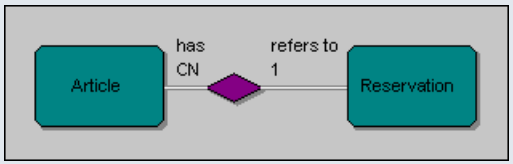
The **case/4/0** tool component which relieves you of the design work is called the ***Design Assistant***. The Design Assistant generates module structures from the information flows of System Analysis, relational models with attributes and primary and foreign keys from ER Models, and type structures from data structures.
Two things are required to design software "at the push of a button":

... Software design by mouseclick is definitely worth trying out. Best of all with your own piece of "System Analysis". We suggest:



Step 1: Expand the EASY Furniture Company function structure by adding a third function, Reserve Furniture *to* Sell Furniture *and* Hand-Out Furniture. *Then divide this function in a dialog function* Advertise Article *and a processing function* Reserve Article. *Finally, connect* Reserve Furniture *with* Self Pick-up Furniture Sales.

Step 2: Set up a new ER Model with the name Reservations, *that looks something like this:*



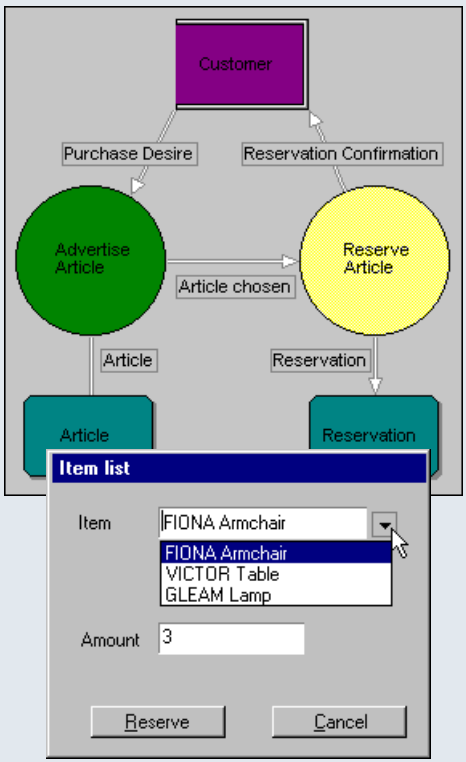
- a fundamental architectural concept for structuring a software system,
- detailed construction plans for software structures, i.e. module templates.

The Design Assistant works on the basis of a distributable *layer architecture*, which can also be used for the development of Client/Server applications. It is equipped with standard templates for the user interface, processing and data access modules, which can be modified, extended or replaced specific to your application at any time.

This is how the Design Assistant works: Each entity type in the information flow is checked as to whether it has already been transferred into a relation, i.e. the design of a data base table. If this is not the case then the Design Assistant makes up for this design step and automatically creates data base designs from the ER Models in the form of relational models. After this preparation, the Design Assistant is ready to generate *one* module structure *for each* entity type for access to the corresponding relation. All data structures which describe the input/output data of the information flow functions are transferred into type structures, which are subsequently used by the Design Assistant to create data definitions in module structures. *All* processing functions which appear in an information flow are summarized in the module structure of *one* processing module. For *each* dialog function, *one* module structure of an interface module is created. Finally, the relationships used between interface, processing and access modules are derived from the information flow and illustrated in the module structures.

Come on, let's take a look at some of the results generated.

Step 3: Click in the Function Structure on Reserve Furniture *with the right mouse button, in order to open the function's context menu. Choose* ***Show Information flow*** *and create this diagram:*



Customer *is an external interface,* Article *and* Reservation *are the entity types of the ER Model you just set up.*

Make a dialog field for Show Article *and determine (with* ***Assign/Callback****) for the* Reserve *control button, that the user's click will cause* Reserve Article *to be carried out. If you like, you can add sample values as shown above. That finishes your piece of System Analysis. Now leave the software design to* **case/4/0**.

The Relational Model

The relational model serves to design a data base. From its elements—the *relations*—**case/4/0** generates SQL table definitions. Remember the menu point ***Data Base*** in the ***System Design*** menu?
Relations can be described by their attributes, primary keys and indices. Apart from the rectangular symbols for relations, the relational model also shows arrows. These represent the relationships between the relations which should be used as access paths. These are described by foreign keys.
By the way, you can also derive a relational model from an ER Model directly—without the Design Assistant. And what if there are later corrections to be made to an entity type? No problem! A mutual update of the relational and ER Models can be made at any time.

The Module Structure

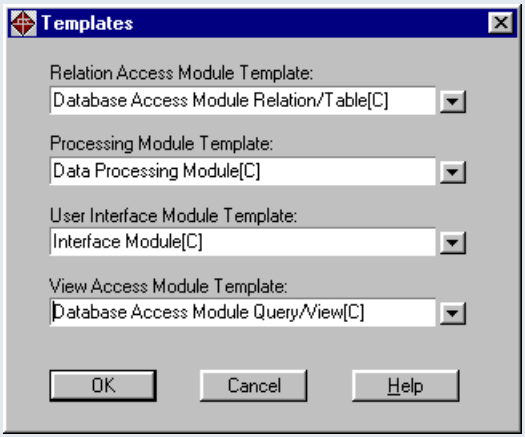
Module structures make the design of software architecture into a conscious, creative process. A *module structure* is the graphical presentation of a C-Module or of a COBOL- or Visual Basic-Program and shows:

- each function or dataset which realize a common problem domain or technical task or which should be coupled together, and
- all foreign modules or called programs which are used in the module.

A *function* in a C-module structure or a *section* in a COBOL-module structure can be traced back from a function structure to a problem domain function. A special role is played here by the dialog functions. These are denoted by the abbreviation "Dlg". If they have been taken over from a function structure, then they will bring the design of the user interface created in System Analysis with them. Of course, menus and dialog boxes can also be worked on within a module structure. User interface elements which the **case/4/0** dialog designer does not contain as standard can be included in this process.

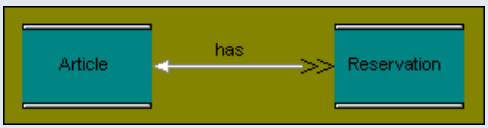
Step 4: In the Information Flow's context menu you will find the menu item ***Design Assistant****.*

Step 5: The Design Assistant requires you to give it the blueprints it should use for the module design. Select the **Templates** *control button.*



Once the templates are defined (like for Microsoft C above), the Design Assistant shows what it's going to generate: a relational model Reservations, *a processing module* Reserve Furniture, *a user interface module* Advertise Article, *two access modules ... see for yourself.*

Step 6: When you end the dialog with the Design Assistant by clicking **OK**, *it creates this Relational Model, among other things:*



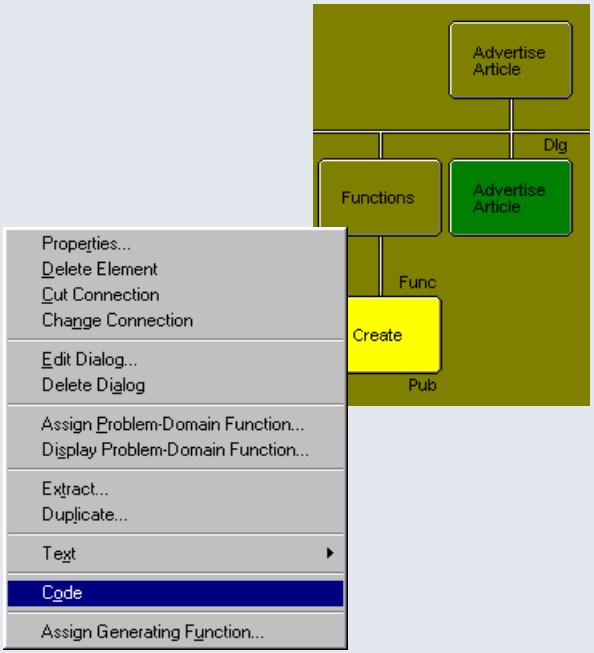
If you are working with MFC-Interface Classes or with the Micro Focus Dialog System, then you no longer need to code the interface—this is done automatically by **case/4/0**. The key to code generation—and not only for user interfaces—are the generating functions, which can be assigned to the elements of a module structure.

Let case/4/0 do the programming for you ...

... by developing reusable generating functions for all common recurrent tasks within a software system and assigning them to the elements of a module template or module structure. As already mentioned, generating functions are formulated in **case/4/0**'s own generating language. This BASIC-like language consists of simple constructs and accesses to the repository, with which **case/4/0** ascertains the knowledge required in the current processing context. As opposed to macro expansion, **case/4/0** does not insert a static piece of code into the source, but carries out a generating function whenever you wish to see or work on the code in a module structure. In doing this, **case/4/0** reverts back to the current repository contents so that all updates in the specification automatically take effect in code.

If you use generating functions in module templates then you will receive complete construction plans for modules. **case/4/0** is supplied with a whole set of these. Their effect: Design and programming of standard tasks such as interface treatment or data access is taken over *completely* by **case/4/0**.

Where the development of complex processing logic is concerned, then the responsibility is, as before, yours. First of all you can specify processing functions from module structures with graphical pseudo code in the form of implementation trees to be filled with code subsequently. It is thus not necessary to exit **case/4/0** to write code. The objective of this procedure is to link the specification and code so closely that they both stay up-to-date and consistent with each other for the



The Design Assistant has done even more for you and generated a large part of the source code. Let's take a look, for example, at the newly created Module Structure Advertise Article. *Use the menu function* ***Code*** *to look behind the boxes of this diagram and find out what* **case/4/0** *has done for you. Or create a* ***Listing*** *with the menu function of the same name, which you find in the context menu for the Module Structure's header box.*

whole software life cycle. Code updates outside of **case/4/0** are also possible and can be subsequently taken over into the tool. Maintenance in **case/4/0** therefore always builds upon the up-to-date, easily readable, graphical documentation.

Integrated Version and Configuration Management

Clarity and understandability are of central importance in daily work on a project.
With its Version and Configuration Managers, **case/4/0** offers you two elements that, if rigorously followed, lead to contribute significantly to the success in the software development process.

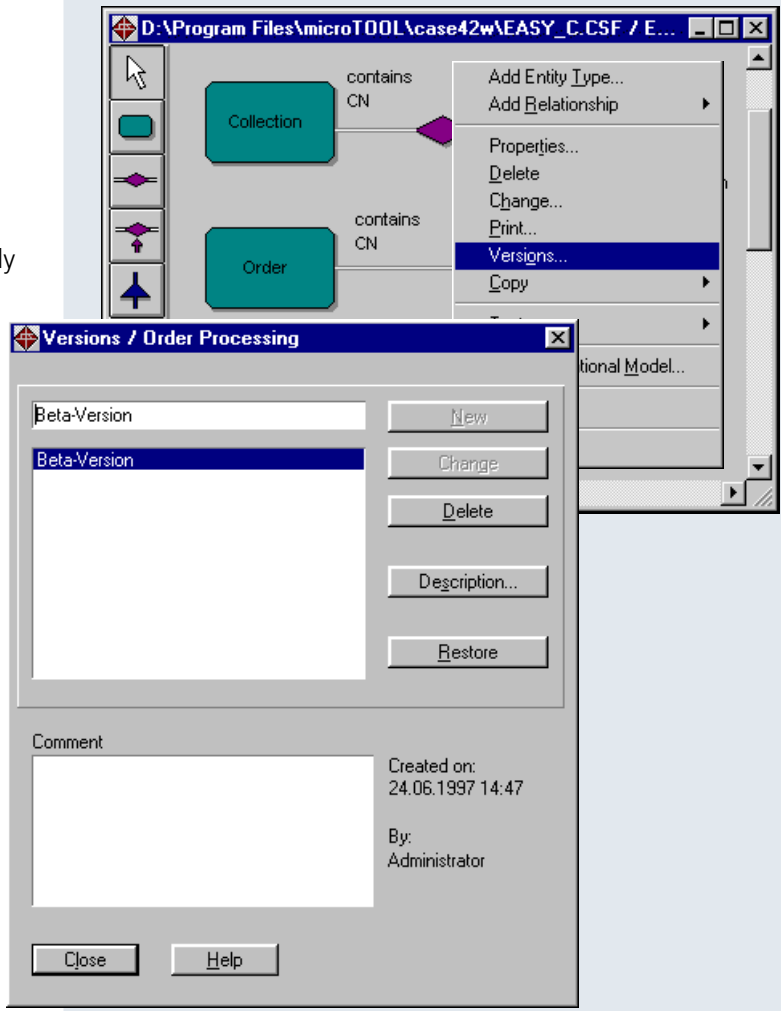
Versioning the Results

In **case/4/0,** both diagrams, such as ER Models or module structures, as well as results without graphic representation, such as data elements or collections of generating functions, can be versioned. A version is a snapshot of the current development state and cannot be changed later. **case/4/0** makes the versioned results available in a new system separately, if need be. Should an older version be further developed in the context of other results, which may have been created later, it is necessary to make sure that all the results fit together without contradiction. The import function in **case/4/0**, that brings together the results, can take over this task for you.

Making Configurations

By "configuration" we mean a user-defined part of a **case/4/0** system. When it is set up, it only receives a name. Later, at any time you choose, you can assign diagrams or other results from the current **case/4/0** system to it. A configuration serves to collect results which together represent the building blocks of a system or milestones achieved. If the results collected in a

This is what version management looks like:



If you want to discard the current state of results and replace it in the process of development, with an earlier version, then you just have to activate the desired version with the function **Restore**.
Go ahead and give it a try!

configuration have reached a state of development that you wish to hold on to, then you can "freeze" a configuration with a mouseclick. This process is similar to versioning single results: A frozen configuration cannot be edited. It maintains its elements' current state of development.
case/4/0 makes sure that the frozen partial system is internally consistent, that it does not contain, for example, references to a void.
The import function in **case/4/0** once again takes care of the consistent combination of a recreated configuration with the current development results.

Expanding case/4/0 as needed

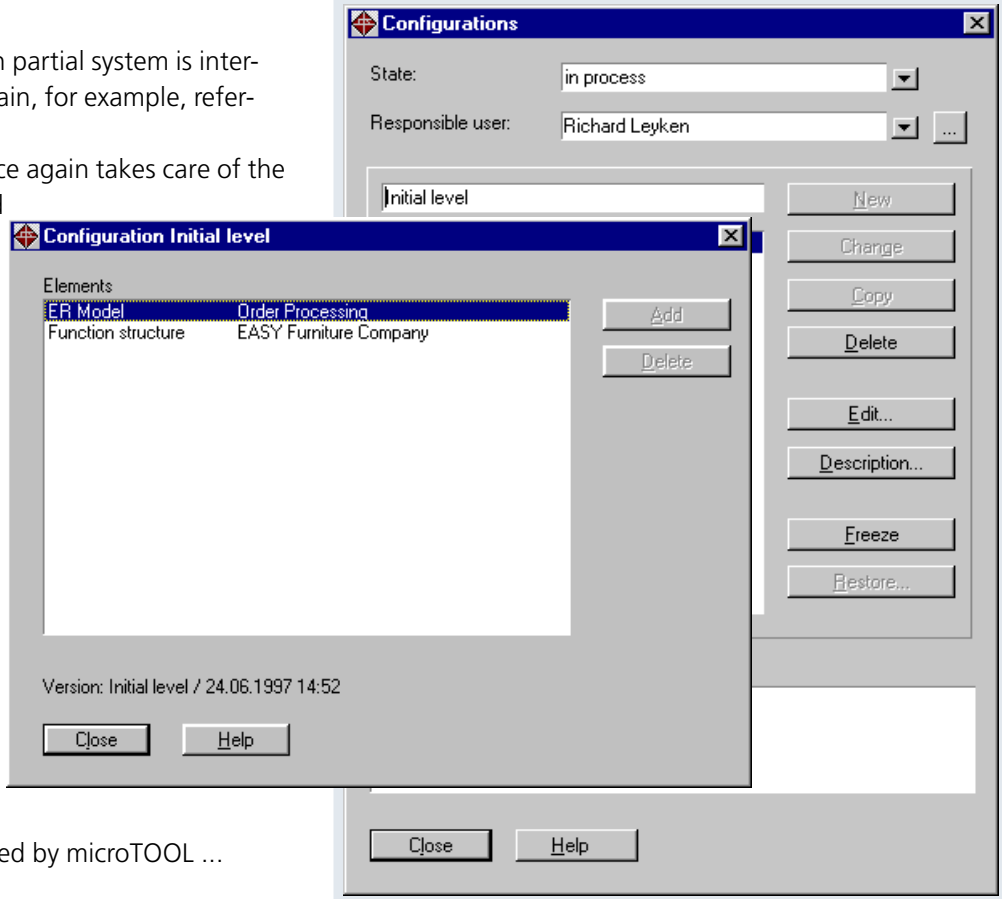
You've gotten a first glimpse of **case/4/0**'s capabilities. We would also like to show you how to expand **case/4/0**'s functionality according to your indiviuell needs by the way of add-ins. You have two options:

- You use the open architecture of **case/4/0** and expand the set of functions through add-ins you have developed yourself, or
- You lean back on the add-ins offered by microTOOL ...

... for Example, with Reverse Engineering for COBOL and PL/1

Many of the COBOL or PL/1 systems in use today require a major service commitment. The decision to improve the internal implementation while leaving external functionalities unchanged is often unavoidable.
A requirement for cleaning and reorganizing legacies is the "rediscovery" of the original software design on the basis of the existing source code. This step is supported by the Reverse

*You come to configuration management by activating the function of the same name in the **System** menu.*



*If a frozen configuration should be restored to the development state, then the user doesn't have to do any more than tell **case/4/0** to recreate the configuration. All results of a configuration are then made available for editing with a new **case/4/0** system.*

Engineering functions available for **case/4/0** as add-ins. They discover the structure of the programs and produce clear graphic documentation in **case/4/0**, which makes it easier to understand the programs and their restructuring. We show you how this works in a COBOL example.
The add-in for COBOL Reverse Engineering has a parser which analyzes COBOL programs and copy elements. Its use requires syntactically correct code. After analyzing COBOL programs and copy elements the following elements are created and deposited in the **case/4/0** repository:

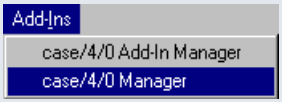
- Module structures with sections/paragraphs, variables, type structure references and references to the programs called,
- Type structures with data groups and datafields, repeat groups and exclusive alternatives for redefined data,
- Data elements in the data catalog with formatting informations, default values, and enumeration sets.

The COBOL source code is also taken up into the **case/4/0** repository and assigned to the graphic elements which were created. The result is a clear graphic description of the COBOL programs, which is mechanically connected to the source code.
Results which were developed in **case/4/0** are not formally different from those created by the add-in. This means that the many evaluative functions that **case/4/0** offers for quality control are also available for reverse-engineering results. The editing and restructuring of modules and type structures are not in any way differentiated from primary software engineering in **case/4/0**. An add-in for Reverse Engineering is available for SQL, in addition to the COBOL and PL/1 add-in.

From Analysis to Maintenance: A well-travelled Road in case/4/0

This is where our quick tour through **case/4/0** ends. We haven't been able to present anywhere near all the functions in this short piece. There is much more to discover, but also one thing to remember:

case/4/0 means ...



Regardless of whether you extend **case/4/0** *with ready-made add-Ins or with your own, the Add-In Manager (which, incidentally, is itself an add-In) makes sure that you work on a single interface consistently. You can use it to integrate new tool functions in* **case/4/0**.

... great advantages for your projects

Because during system analysis, **case/4/0**'s graphic methods lead to better communication of all participants, and thereby to quick progress in the project. They entice higher precision in the problem domain field and give you strategic security beyond the boundaries of a single project. The verification of the system behavior, before a single line of code is written, brings the assurance that what is conceived of will also function. Software design and implementation with **case/4/0** mean standardization of the results and palpable reduction in the amount of effort required, measurable at the latest by the number of statements generated.

Convince yourself that **case/4/0** delivers your project everything promised by this quick tour. We would be glad to discuss the development of an enterprise-specific **case/4/0** implementation with you.
Just give us a call:

microTOOL GmbH

Voltastrasse 5
D-13355 Berlin

Phone: (+49 30) 467 0860
Fax: (+49 30) 464 4714
e-Mail: info@microTOOL.de

microTOOL Sp. z o.o.

ul. Wołodyjowskiego 64
PL-02-724 Warszawa

Phone: (+48 22) 648 32 99
Fax: (+48 22) 43 81 01
e-Mail: info@microTOOL.com.pl

*One more thing that you should definitely try out: How to create and update for example, job specifications by using **Publish** with OLE and Microsoft Word.*