objectiF®

The Introductory Version

microTOOL

Copyrights

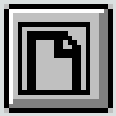
Quotes have be taken from the following:
Christopher Alexander, The Timeless Way of Building*, Oxford University Press, New York, 1979.*
Edward Yourdon, Decline & Fall of the American Programmer*, Yourdon Press, New Jersey, 1993.*

Reading the Documentation

This part of the ***objectiF*** description should be read first to ensure a good start with the Introductory Version of ***objectiF***. Here, we'd like to illustrate how you use this online document. This document was created with Adobe Acrobat 2.0 and is contained in a PDF file (Portable Document Format). You will need the Acrobat Reader to be able to read the PDF file. The Reader is supplied together with Introductory Version of ***objectiF***. We will explain here the essential Reader functions. Detailed instructions can be obtained with the F1 function key or over the ***Help*** menu. The corresponding online handbook (help_r.pdf) is opened in either case.

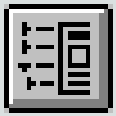
Default Settings That Make Reading Easier:

The increased use of 256 color displays induced us to setup our documentation in accordance with to this standard. Of course our documentation can still be read well with a 16 color display. If it is possible, however, we recommend a configuration with 256 colors.
The following settings should be made after opening a document to achieve optimal reading results:
The magnification level of the document should be adapted to the size of your screen: Click this button to zoom the document to fill the width of the screen.
With the ***Zoom*** function you can change the magnification level of the document's screen representation. Comfortable reading results can be achieved with the ***Fit Visible*** command.
Have an outline of the document displayed; click this button in the Acrobat toolbar. The display window will be divided vertically, and the bookmarks defined for the document are displayed with a page icon in an overview area on the left. To move to one of the displayed topics, just click it or the corresponding page icon.



Fill the width of the screen

Zoom function



Show bookmarks

This icon to the left of a bookmark means that subordinate bookmarks have be defined. Click the icon to show the subordinate bookmarks.
Click this icon to hide the subordinate bookmarks again.

Show subordinate bookmarks

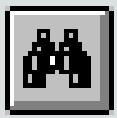
Hide subordinate bookmarks

Hypertext Links

You can jump to any topic of interest from any item in the table of contents with a click of the mouse.
We have also defined hypertext links for individual terms, which we set off in color and italic style. Just move the mouse pointer to such a term; the mouse pointer becomes a pointing hand. With a mouse click you "open" that page in the document linked to the colored term, where the term could appear again – possibly in another context – or where it is illustrated in detail.

Text Search

If you are looking for a specific term, you can look it up in the alphabetically ordered index of the corresponding document. You can move quickly to the given page by moving the scroll bar to the indicated page with the mouse: The beginning of the document is at the top and the end is at the bottom. Stop when the number of the desired page is displayed.
Or select the "Find" function from the Acrobat Reader toolbar. That is another way to find any term quickly. To find the next instance of the term, press the F3 function key.



Search

Printing Documents

To print a document, select first the ***Print Setup*** command from the ***File*** menu, and specify the appropriate settings in the dialog box that appears. Afterwards, select the ***Print*** command in the ***File*** menu.

The Introductory Version

A very warm welcome to ***objectiF***. Have you started down the object-oriented path to software development? Then why don't you take a look at ***objectiF***, our object-oriented development environment? Because we conceived ***objectiF*** for professional object-oriented application development.

Discover *objectiF* ...

... and convince yourself how easy and effective this tool really is. You will find the demo version of ***objectiF*** on the enclosed CD ROM. You can explore each "corner" – from system analysis to the generation of source code.

If you would like more information on *objectiF* beforehand, ...

... then in the following pages you will find a product description as well as abundant information that we hope will inspire you to start on your own tour of ***objectiF***.

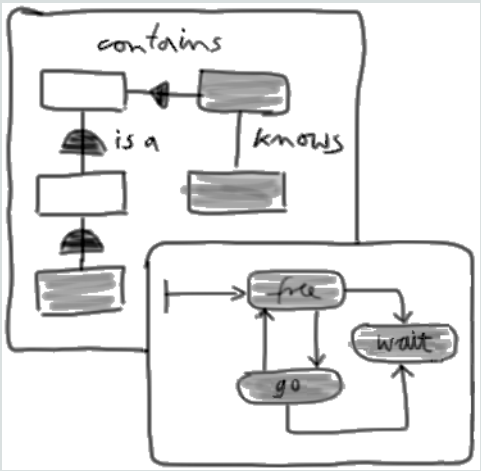
What Is *objectiF*?

objectiF is an integrated software development environment that offers complete support through all phases, from object-oriented analysis to implementation in C++.
objectiF was conceived for teamwork in LANs. It has its own integrated ODBMS – we speak of an object base – which guarantees safe multi-user operation in networks.
Moreover, with its import/export functions for the exchange of data between different object bases ***objectiF*** supports the logically consistent integration of decentrally developed results.
Teamwork in LANs, decentralized project teams, divisional work loads distributed over stand-alone PCs – ***objectiF*** is easily adapted to the different forms of project organization.

Is *objectiF* Right for You? Yes, ...

... if you develop systems with a design span that is no longer manageable by 2-3 man teams,
... if you work on projects at the beginning of which the predominant tasks have little to do with object-oriented programming, but all the more with the precise specification of user requirements,
... if you depend on smooth communication between several development teams, perhaps even geographically removed,
... if you are serious about reusing frameworks, purchased or developed in-house, class libraries, or component software.
In project situations of this type, model-based procedures – supported by graphic modelling tools – have proven their worth. And this is just what ***objectiF*** offers.

On the side: There is a direct correlation between precise designs and the use of diagrams. In his often quoted book The Timeless Way of Building, *the architect Christopher Alexander states: "My experience has shown that many people find it hard to make their design ideas precise. They are willing to express their ideas in loose, general, terms, but are unwilling to express them with the precision needed to make them into patterns. ...*



If you can't draw a diagram of it, it isn't a pattern. If you think you have a pattern, you must be able to draw a diagram of it. This is a crude, but vital rule."

What You Need for *objectiF*

A platform usually present anyway. Because the lower the investment needs for hardware and system software, the more likely a tool will achieve widespread acceptance.
For ***objectiF*** you will only need

- Windows PCs, stand-alone or in a network, 80486, or better, with at least 66 MHz and 16 MB RAM.

With respect to software, ***objectiF*** requires

- Microsoft Windows 3.1x or Windows 95,
- Microsoft Word 6.0a or later or any other word processor that supports OLE 2.0.

To apply ***objectiF*** in a multi-user environment, we recommend

- Novell Netware 3.11, or later, or
- Microsoft Windows NT Advanced Server.

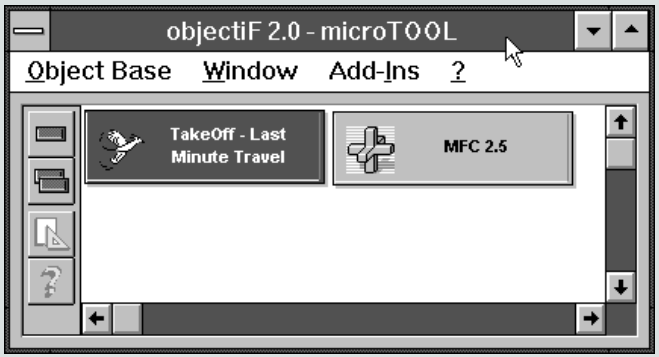
To be able to offer optimal object-oriented programming support, ***objectiF*** is specialized in C++.

And What *objectiF* Can Do for You

You will be actively supported in your project work with

- graphic tools for specifying classes with their structural relationships and message connections in ***class models***,
- graphic tools for modelling ***object life cycles***,
- ***code-generating components*** that generate, in C++, header files and large portions of the implementation files and that maintain, at all times, consistent code and graphic models,
- a ***method editor*** of a very special type that actively supports you, wherever the tool cannot perform the programming completely on its own, by utilizing its knowledge of syntax and the given context to supply you with the appropriate programming suggestions,
- a ***publishing component*** for developing project and product documentation that satisfies the elevated layout and presentation demands placed on requirement documents, handbooks, etc., and

*If you come along with us – here in the margins – we will show you some of **objectiF**'s striking functions. For our example we have taken a system for last minute travel reservations, implemented with **objectiF** in C++ utilizing the Microsoft Foundation Class Library (MFC).*
*Is **objectiF** installed? Then please start it by double-clicking the program icon in the **objectiF** program group. A window will open containing two rectangles, both of which represent model units that we refer to as* subjects.



*The gray rectangle represents the Microsoft Foundation Class Library, which we imported to enable its use for specifying and programming our example system with **objectiF**. The green subject icon refers to the classes of the example system developed with **objectiF**.*

- a reverse-engineering component enabling ***objectiF*** to incorporate classes, class libraries, and components that were not created with ***objectiF*** .

As *add-ins*, ***objectiF*** additionally offers you

- a graphic editor for modelling ***interaction diagrams***, with which the flow of messages between instances of selected classes can be specified,
- numerous ***evaluation*** and test functions for documentation purposes, and
- object-oriented ***software metrics*** to support quality control throughout the life of a project.

If however, for your specific project situation, you should find something missing from this complex store of functions, ...

... then the solution is called "Customizing"

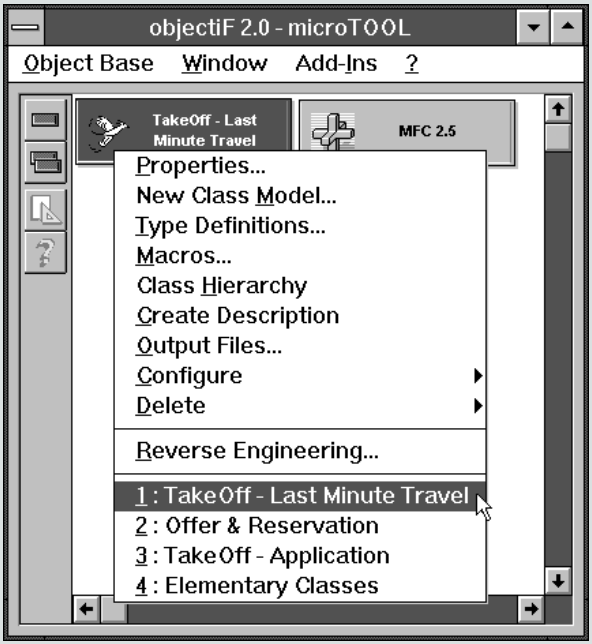
Customizing denotes the process of extending ***objectiF*** functionality with user-specified tool functions. In the sense of OLE Automation, ***objectiF*** is component software. That means a number of its internal classes are defined as exposed classes and are accessible from "outside" – from foreign applications. This represents the base technology allowing you to extend ***objectiF***'s functionality with add-ins. The add-ins can be developed with ***objectiF*** in C++, or when minimal specification efforts are required, in Visual Basic, without the assistance of this tool.
The group of ***objectiF*** 's exposed classes, and thus its extendability, covers at present the following application areas:

- publishing,
- evaluating and testing for quality control, and
- generating classes for persistent storage in object-oriented or relational data bases.

A Whole Tool Kit in One,...

... the ***objectiF*** *Add-In Manager* (incidentally, an add-in itself) makes sure of that. With its help, you will be able to integrate your new tool functions into the ***objectiF*** menus.

For now, let's not worry about the Main Menu or the toolbar, but instead, let's explore together what's behind the subject icons. If you click one with the right mouse button, a context menu appears with a list of all of the commands it knows. Try it out on the Take Off – Last Minute Travel *subject.*



In the lower part of the menu you will find a list of the class models *contained in the subject. One of them has the same name as the subject:* Take Off – Last Minute Travel. *Please select it with the mouse. Incidentally, the elements within the windows, the windows' background, and even highlighted code, all react to a click of the right mouse button in the same way, by opening a context menu.*

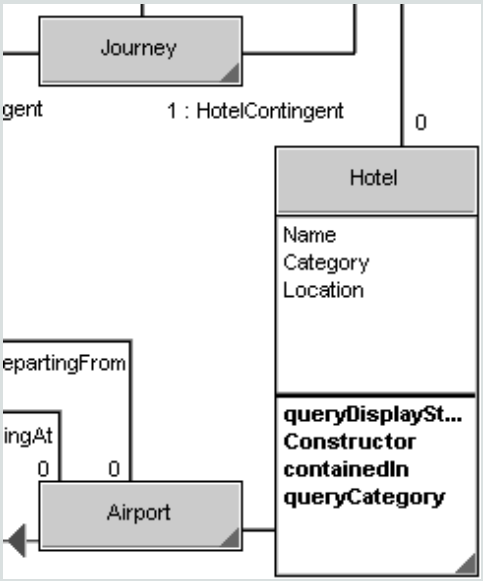
How You Work with *objectiF*

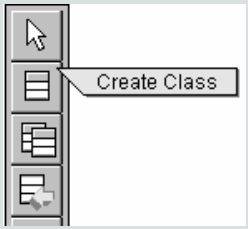
Object-oriented application development starts with analysis. Only if the selected analysis methods actively apply the concepts of the target language – abstraction, encapsulation, inheritance – we can speak of a process that consistently applies object-oriented methods.
Such processes enable the productive and automated utilization of analysis knowledge during implementation. And that's just what ***objectiF*** does for you.
We would like to illustrate the effectiveness of the object-oriented methods with ***objectiF*** . Follow us on an excursion from analysis, over software design, all the way to implementation in C++.

OOA and OOD – Methods in Practice

objectiF is based on the object-oriented analysis and design methods, OOA and OOD, from Coad/Yourdon.
OOA and OOD are simple and well-founded in practice. They encompass a very compact, easily understood graphic notation for classes and their interrelations, which has equally proven its worth during both analysis and design.
The clear notation of OOA/OOD and its simple application with ***objectiF*** enable you in short time to develop, discuss, and evaluate alternative models for problem-domain and technical solutions. Documentation suitable for reviews is a prerequisite for effective quality control during analysis and design – as well as a good reason for applying OOA and OOD with ***objectiF***.
Coad and Yourdon conceived OOA/OOD for a wide range of applications. Regardless of whether you are developing technical or commercial applications, are realizing large projects or small tasks, with this method base, ***objectiF*** is just the right tool for you.

This is how class models appear: The rectangles represent classes. The ***Open*** *option in a class's context menu opens the list of its attributes and methods. Try it out on the* Hotel *class. Double-clicking one of the attributes or methods presents a definition box. Take a closer look at one.*





Would you like to insert a class of your own? Let us suggest the class Room. *Proceed by clicking the button in the toolbar for adding classes:*

Afterwards, you can click any free place next to Hotel *and enter the class name.*

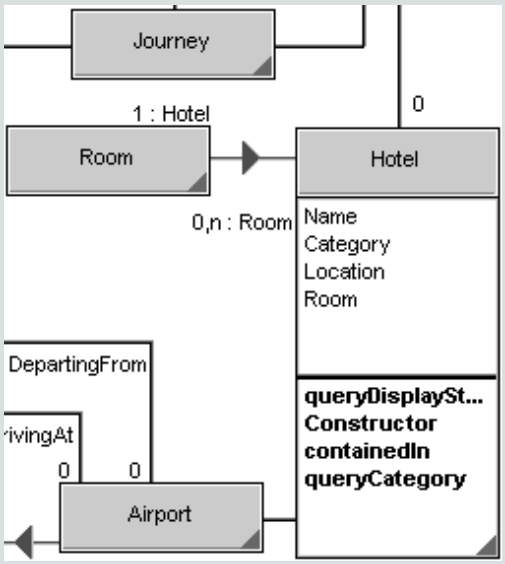
From OOA to OOD with the Class Model

Characteristic of object-oriented processes is the seamless transition from analysis to design. The two procedures differ primarily in the objects being modelled – not the way in which the results are represented.
OOA with ***objectiF*** means modelling the problem-domain classes of a system.
OOD means working out a technical solution for the requirements of the problem domain. This step can often be reduced to developing the base-technology classes from which – expressed simply – the problem-domain classes modelled during OOA inherit their technical behavior.
The ***class model*** is the common modelling tool of OOA and OOD. It displays at a glance all of the central aspects of an object-oriented design:

- the *classes*,
- their static properties, the *attributes*,
- their behavior, the *methods*,
- *inheritance structures,*
- *message connections*, and
- *structural relationships* at the instance level.

With ***objectiF*** the elements of a class model are precisely specified through the definition of their properties. In this manner, you decide in the class model, among other things, whether a class is to be *abstract*, a method *virtual*, methods and attributes *protected*, *public*, or *private* – specifications that ***objectiF*** uses automatically during implementation. The details of relationships can be extended as well. For example, structural relationships can be described by formulating their semantics and specifying cardinality and conditionality.
While you are still modelling a class model from the problem-domain perspective, ***objectiF*** is already "thinking" about system realization. For every structural relationship that you enter into the model, it automatically generates, for example, those attributes that will be reciprocally assigned references to the instances linked by the relationship.

Let's take a closer look at the class model now: The green lines in the class model represent structure relations at the instance level, the semantics of which can differ from case to case. The red lines with a triangle have the fixed meaning of "consists of" or "contains". We call them aggregation structures.
A hotel contains rooms, doesn't it? Then select the button with the red triangle in the toolbar. Now, click first Hotel *and afterwards* Room *– you just modelled an aggregation structure. At the same time,* ***objectiF*** *added a* Room *attribute to* Hotel*, which instances of* Hotel *will use to manage their rooms.*



The new class, Room*, still needs appropriate methods. You can add them with the help of the class context menu or ...*

By the way, if you cancel such a modelling step, ***objectiF*** deletes the added attributes as well, ensuring in this way consistent solutions.

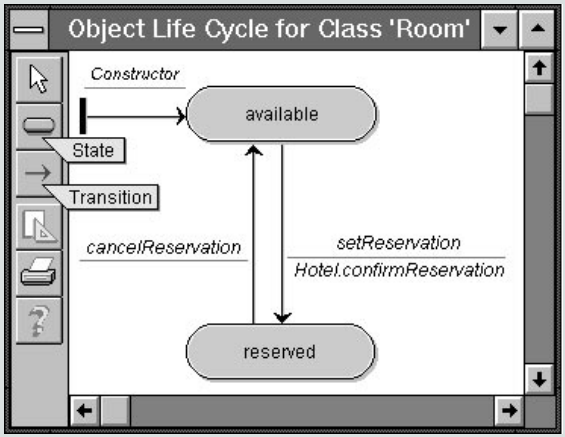
Object Life Cycles – an Effective Instrument for Modelling Behavior

What happens when ...? This question is important not only for real-time systems; today more and more commercial systems incorporate time-critical, event-dependent components, too. Let's formulate the question in an "object-oriented" manner: When, under what conditions, do the instances of a class display their behavior? ***objectiF*** offers a special instrument for modelling this aspect: the ***Object Life Cycle***.
Object life cycles are described with the help of *state transition diagrams*. An object life cycle represents the connections between

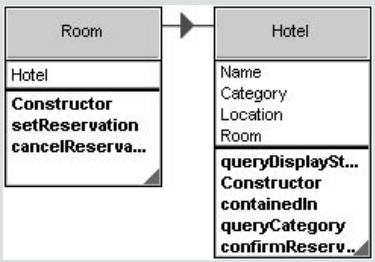
- the *states* that an instance can go through during its existence,
- the *events*, in the form of messages, that will affect it,
- the *actions*, in the form of function calls, invoked by the state transition and
- the *following states* reached.

The creation of an object life cycle is optional with ***objectiF***. However, for classes whose instances move through complicated life cycles or are responsible for controlling tasks, you should not do without this powerful modelling tool.
If during your work on an object life cycle you extend the store of behavior displayed by a class, the new methods are automatically inserted into the class model. ***objectiF*** takes on the responsibility of safeguarding consistent object state diagrams and class models. As end-user, you can concentrate entirely on the essence of the modelling tasks.

... specify them with the help of an object life cycle for the Room *class: Select the* ***Create Object Life Cycle*** *option in* Room*'s context menu. An empty diagram is presented. Here's how to add* states*: Select the button with the oval icon, click a free place in the window, and enter a name for the state. Here's how to add* transitions*: Select the button with the horizontal arrow icon, and click the positions for arrowtail and arrowhead. A dialogbox opens. You can choose or enter the name of an event to label the transition. In the transition's context menu you'll find the* ***Actions***



option for the entry under the separating line of the label. Events and actions are expressed by methods. Room *doesn't have any yet; why don't you add some – and then take a look at the class model:*



REUSE – a Capital Concern for *objectiF*

The term *object-oriented* is often automatically associated to *reuse*. If, however, we understand reuse to be more than the application of the inheritance concept, it is in no way a natural consequence of object-oriented methods.
Three things are necessary to achieve a high level of reuse:

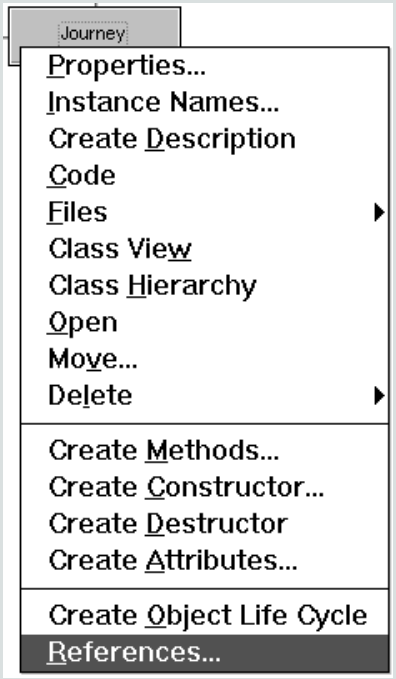
- explicit plans for the reuse of the projected results,
- comprehensible documentation, because you only *will* reuse what you understand, and
- easily accessible elements, because you only *can* reuse what you are capable of finding.

With its *subject concept*,
objectiF offers very effective support.

With Subjects You Have Class Complexity Under Control

Class models are grouped together with ***objectiF*** to form larger model units that we refer to as *subjects*. Experience has shown that a subject will typically contain some 20-30 classes that pertain to the same problem-domain or technical topic and partake in an extensive exchange of messages. Each class of a subject can be prepared for reuse in other subjects by declaring it as *public*, or its usage outside can be excluded with a *private* declaration. The public classes represent the *subject interface*. One glance is enough to immediately assess the store of available classes.
objectiF's subject concept enables the practical reuse of classes – especially beyond the bounds of a project.

Have you been waiting for the term "inheritance?" We would like to make that up to you right now with the aid of the Journey *class.*
Select the ***References...*** *option from the class's context menu.*
objectiF *reacts to this by displaying a list of all the documents containing the* Journey *class. In our case, the list is quite small, but it contains the class model* Offer & Reservation. *Please open it.*



You'll find Journey *again in* Offer & Reservation ...

How to Realize Software Architecture Concepts

Subjects display another positive trait with respect to OOD: versatility in realizing concepts for software architecture. Here are three examples of the modelling freedom that ***objectiF*** opens up for you with its subject concept:

Frameworks

A large potential in reuse can be maintained by developing frameworks; frameworks offer problem solutions at a generalized level. They contain abstract classes, connected to one another by relationships, which are then offered to the application developer for their concrete realization.
A framework is modelled with ***objectiF*** in the form of a subject. The capability of enabling specialization for selected classes with the *public* property represents a powerful instrument for modelling the frameworks as well as for using them during application development.

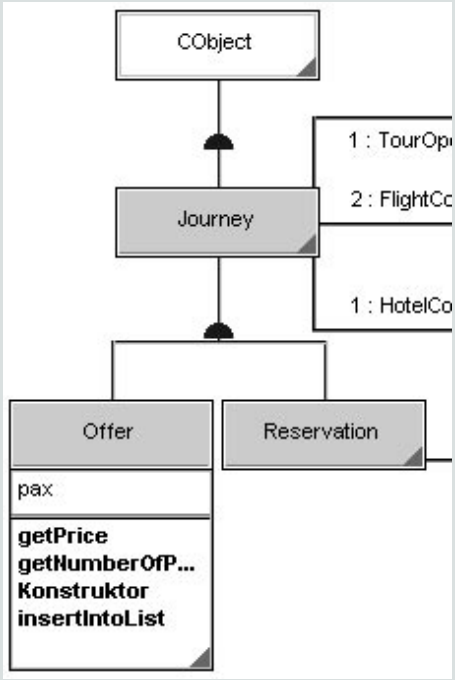
External Class Libraries

Here is another example of the efficient way ***objectiF*** uses subjects, just part of the standard package: linkage to an external class library, namely, the Microsoft Foundation Class Library (MFC). The MFC classes are summarized in their own subject and can be specialized in user-specific subjects or used as elementary classes in attribute definitions.

Components in the Sense of OLE Automation

For the development of component software with the help of OLE Automation, a subject takes on the meaning of a component. All of the classes of a component that are accessible to other components through messages are specially marked with ***objectiF*** as *exposed classes* of the subject. Exposed classes of other applications supporting OLE Automation – e.g., Microsoft

... as a specialization of the CObject *class. The* CObject *class has a white background because it has been* imported *from a foreign subject. It comes from the Microsoft Foundation Class Library, that is, from an external class library, which we mapped to an* ***objectiF*** *subject. Imported classes can be used, but they may not be modified.*



Offer *and* Reservation *are specializations of the* Journey *class. You can see what's "special" about them when the class icons are opened. Would you like to view the inherited attributes and methods as well? Then select the* ***Full Class View...*** *option from the class's context menu. A window with two lists opens. The context menu for the list titled "Methods" contains the* ***Show Inherited Methods*** *option.*

Excel – can be represented with ***objectiF*** as subjects as well and used exactly as if they were developed with ***objectiF*** themselves.

objectiF – the OOP Specialist for C++

Object-oriented programming (OOP) in C++ means coding a class declaration for every designed class and a function definition for every specified method. The precise class specification in the class models pays for itself here – in the form of better quality and increased productivity; you can leave a large part of the work up to ***objectiF*** now.
Let's start with the class declaration: ***objectiF*** is not only capable of representing a class model graphically but also from the perspective of C++. It can display every specified class, together with the attributes, methods, and method parameters that were defined for it, in the syntax of a class declaration with a member list. You will only need to extend the C++-specific declarations of the member functions, which during OOA/OOD have not been of interest yet.

objectiF Stands for Implementations True to Specification

Graphic specification and class declaration are simply two sides of the same thing being modelled with ***objectiF***. For example, if you later modify, extend, or delete a method or attribute in the class model, you will immediately find the corresponding modification has been made in the class declaration – and vice versa.

objectiF Becomes Your Personal Assistant, ...

... when you are writing the code for methods, i.e., function definitions. The C++ declaration at the beginning of function definitions is generated by ***objectiF*** . You will still have to write the code for the function body – but even here, ***objectiF*** is an active partner, providing you with its knowledge of the correct

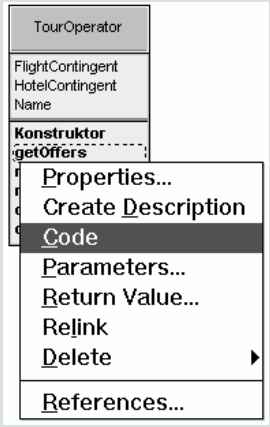
Please return now to the Take Off – Last Minute Travel *class model, where you added your own* Room *class. Please select this class now because we would like to use it to illustrate that for* ***objectiF*** *, the code and the model are simply different means of representing the same content.*

Select the ***Create Code*** *option from the* Room *class's context menu.* ***objectiF*** *displays the class declaration for* Room*. Arrange the windows such that you can view both the class declaration and the open class icon for* Room*.*
Use the ***Create Attribute...*** *option to extend* Room *with a new attribute. Give it the name* Bath*; specify it as* public *and its type as BOOL. The corresponding modifications are automatically performed in the class declaration.*

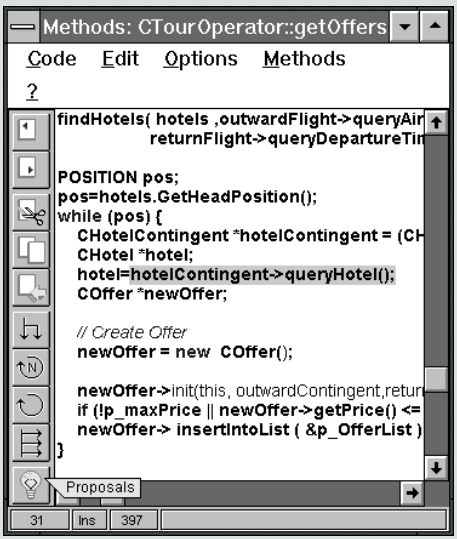
syntax and given specification. In this manner, a new, highly productive kind of programming arises.
This is how it works: ***objectiF*** knows the basic syntax structures of a function body. A function body, expressed in simple terms, is nothing more that a procedural sequence of often no more than 15-20 messages. Principally, a message begins with a variable for the receiver object. A method name, that has to be compatible with the variable, follows as well as arguments that the method expects as regards both type and meaning. When you formulate a message to an object and need a variable for this, then only those attri-butes, parameters, and local variables that have already been assigned an object are of interest. It's exactly this set that ***objectiF*** offers you to select from in this context. ***objectiF*** combines its syntax and context knowledge for this. After you complete your selection, ***objectiF*** offers you appropriate messages, but of course, only those that are understood by the receiver object and can be provided with arguments in the current context.
You will have just completed your selection of a variable when a list of appropriate arguments is offered to you on the screen. To save you some typing, ***objectiF*** enters the best alternative – from its viewpoint – directly into the code. A suggestion list displays all the alternatives. You can confirm the suggestion or select an alternative.

Kind of Fuzzy – the Method Editor

objectiF uses stringent criteria – e.g., type checks and tests of specification knowledge – in combination with Fuzzy Logic to determine which suggestion is the best. For example, a variable is always a good choice from ***objectiF*** 's perspective when an object was assigned to it shortly before, as the return value of a message – the closer the better.
Incidentally, ***objectiF*** is very forgiving. It offers you very efficient help even then when you disagree with its suggestions or even switch them off altogether because ...



Let's have ***objectiF*** *code a "piece" of a method – for example, from* getOffers*, a method of* TourOperator*. First click* getOffers *with the right mouse button to open the corresponding context menu, and then select the* ***Code*** *option. The code for* getOffers *is already complete; let's delete a line of code, or a part of it – e.g., the one highlighted: Now, with the bottom toolbar button, ask* ***objectiF*** *for suggestions to complete this line of code.*

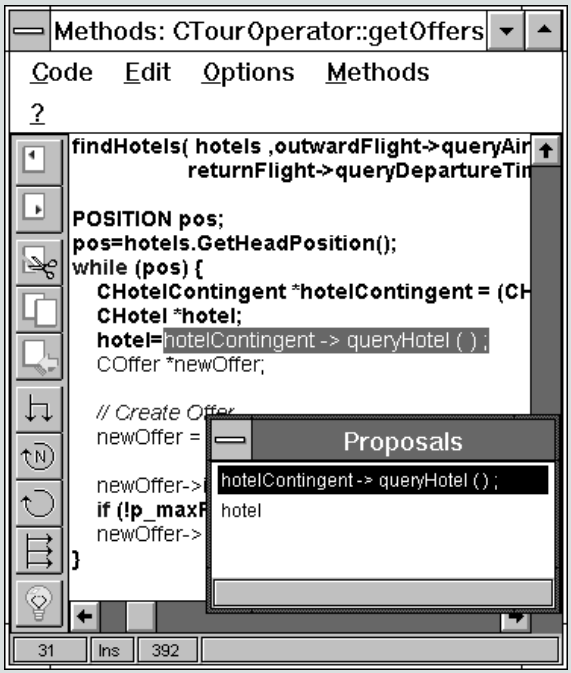


... Code Is More Than Just a Text String for objectiF

Even as you are inputting, ***objectiF*** decomposes, with the help of an incremental parser, every statement into its syntactic elements and checks if they are correct with respect to the syntax of C++. All of the syntactically correct elements are stored as objects in the object base and are put into relationship with the results already found there. This aspect of the ***objectiF*** architecture has one very beneficial effect on your work: Any violation of C++ syntax or disagreement with the connections specified in the class model is immediately visible on the screen, as you type, from the color and type used to represent the input characters. All of the input that ***objectiF*** cannot make sense of, either syntactically or semantically, is represented in black. If the input code turns green, then you can rest assured that it is syntactically correct and logically consistent with the OOA/OOD results found in the object base.

Objects understand messages. For the elements of correctly input C++ statements, which ***objectiF*** manages as objects, there is no exception to this. A pleasant consequence for you: For every highlighted, syntactically correct element in the code you can open a context menu providing you with helpful functions for programming. For example, in the context menu for the name of a method found in a message you will find the ***Code*** option, with which you can directly branch to the code for the corresponding method. Also offered is the ***Parameter...*** option, which opens a list containing all the parameters defined for the method. Moreover, the context menu of a method name enables you to edit the ***Properties*** of the method – in the simplest case its name. Incidentally, the modifications are effective not only locally. They are performed consistently *throughout* the code, as well as in the graphic models, where the method appears.

objectiF *offers you two suggestions, each of which results in a semantically and syntactically correct statement. The original statement is, of course, offered as well. Double-click this entry – the line of code is right back the way it was.*



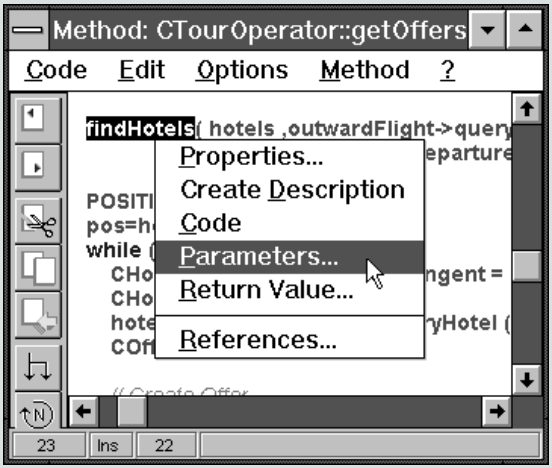
A Maintenance Plus

Readable, semantically expressive C++ code is the guaranteed result of programming with ***objectiF***. The technique supported by ***objectiF*** for assigning names makes a substantial contribution to this: During OOA/OOD, different names can be assigned, from the problem-domain perspective, to instances of the same class. Each name reflects a role from the problem domain. During the naming process, ***objectiF*** displays all of the names already used for instances of the given class. Thus you can always rest assured that you assign the same name to instances fulfilling the same role. The semantically meaningful instance names selected during OOA/OOD are used by ***objectiF*** in the function definitions as variable names, sometimes somewhat modified to satisfy C++ syntax. Using these names for formal parameters, local variables, and function arguments makes every single statement of the function definition easily understood, without requiring you to search through the code to find the intended meaning of variables. Thus the code becomes a readable representation of the problem-domain specification.

How the Code Gets to the Compiler

It is but a small step from the classes coded with ***objectiF*** to compilable source code. Your role in the process is simply to assign output files to the classes of the subject. ***objectiF*** does the rest. It generates compilable *.h* files and *.cpp* files together with include and forward declarations.
If you find errors testing and debugging outside ***objectiF***, you can correct the code directly, that is without returning to ***objectiF***. ***objectiF*** reads the corrected code and relinks it with its graphic specification.

Before you leave the Method Editor you should test the effectiveness of the context menus for code. Here's our tip: Select the ***Parameter...*** *option from the context menu of* findHotels *– a few lines above the statement just coded.*



For a class developed with ***objectiF****, the path to the compiler starts at its context menu ...*
There you will find ***Assign****, a sub menu option of* ***Files****; it has to be invoked first to specify the output files for the code. Afterwards, the .h and .cpp files can be generated with* ***Files/Generate****.*
The C++ code for a class in ***objectiF*** *should always reflect the most current stage of development. If while testing, you modify the code outside of* ***objectiF*** *, you should make use of the* ***Files/Relink*** *option to link the modified code back to the class in* ***objectiF*** *.*

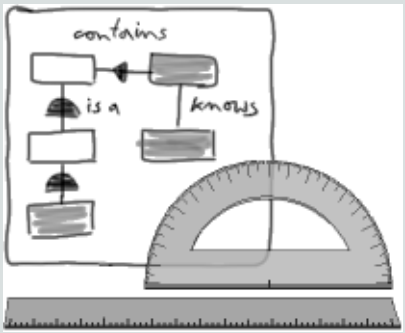
With *objectiF* Software Quality Is Measurable from Project Beginning

Alternative design solutions, their discussion-ready representation, and the realization of architecture concepts are, as illustrated, easily accomplished with ***objectiF***'s model-based method procedures. But which of the developed alternatives is the best? Is the quality of the classes, frameworks, or components high enough to really justify their reuse? ***objectiF*** can help you answer these questions, which concern the quality of the design and product, with its *object-oriented metrics* and object base *evaluations*.

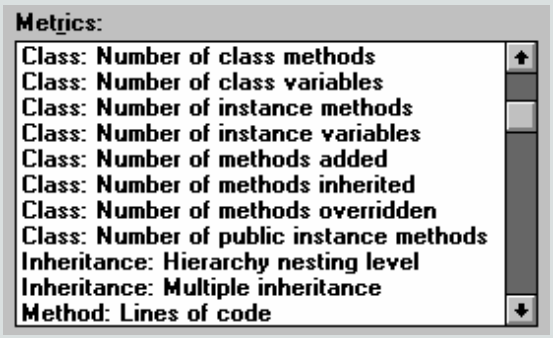
Numerous quality characteristics, like class and method size, inheritance hierarchy depth, and the degree of class specialization can be measured already at the design level and evaluated with comparative values. ***objectiF*** offers you a number of such procedures for measuring and evaluating under the term *metrics*. What classes does the subject interface consist of? Where does a class appear as type in a parameter declaration? For what classes output files have been generated already? Information of this type, covering the developmental stage of the results found in the object base, is produced with the help of *evaluations*. Under this term, ***objectiF*** offers you content data, proof of usage, tests of completeness, and code listings.

Metrics and evaluations are implemented in Visual Basic as ***objectiF*** add-ins, and their source code is available to you, too. That means you can modify, adapt, or extend them to fit your personal requirements.

If project-specific metrics and evaluations should become necessary as your work proceeds, then we recommend the following: Develop your own add-ins, and turn ***objectiF*** into your own personal assistant for quality control and documentation.



Measure? What for? In Decline & Fall of the American Programmer, *Edward Yourdon answered: "The baseball player needs to know some things about his own process; for example, 82 percent of the time he swings at a curve ball, he misses, while 73 percent of the times he goes after a fast ball, he gets on base. And software engineers need information about* their *process in order to learn how to change, how to improve."*

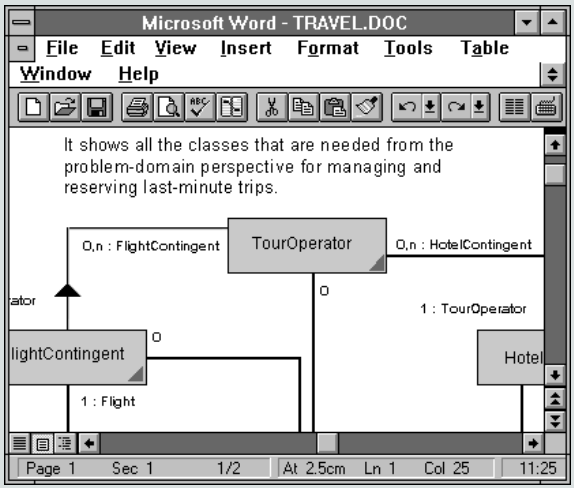


Then let's do some measuring: To output your results, you will need MS Word 6.0a, or later. If you have MS Excell 5.0, or later, installed, you will also be able to produce graphic representations of your results. You'll find ***Evaluations*** *offered in the* ***Add-Ins*** *menu from the Main Menu.*

Target-Group Documentation – a Clear-Cut Case for *objectiF*

Let's stay on the topic of documentation: If you want to output your results on paper, ***objectiF*** has the fitting support for you in every stage of your project.
For example, if during OOA and OOD you need to document subjects, classes, methods, attributes, etc., you can use the professional word processor of your choice, e.g., Microsoft Word for Windows. The only prerequisite is the capability of the word processor to function as OLE server.
Or would you like to summarize your results in an extensive text document, for example, a requirements document? This task is accomplished by the combined efforts of ***objectiF*** and a word processor supporting OLE 2.0 – but now the roles are switched: ***objectiF*** appears in the role of OLE server, while the word processor functions as OLE client.
In this way, even elevated layout standards can be reached, and the documentation is automatically guaranteed to be consistent with the development results.

*Thank you for following us this far. It has not been possible for us to show you any where near all of the **objectiF** functions, for example, the documentation capabilities, ranging from text descriptions of individual diagram elements to the complete documentation of a finished product. Why don't you try to create a description now?*



*Or you could develop an **Interaction Diagram**...*

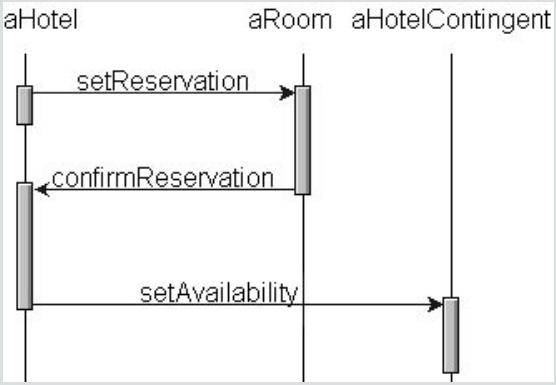
microTOOL –
Your Partner on the Way from OOA to C++

Our short presentation of ***objectiF*** ends here. We are sure, though, working with ***objectiF – the Introductory Version***, you will find many more interesting and useful functions not touched upon in this brochure.
Convince yourself – even for your day-to-day project work, ***objectiF*** lives up to all the expectations raised by the demo version.
Then, let us help you develop your enterprise-specific path from OOA to OOP with ***objectiF***.

microTOOL GmbH
Voltastrasse 5
D-13355 Berlin
Phone (+49 30) 467 086-0
Fax (+49 30) 464 47 14
CompuServe: 100272,1713

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: mtool@ikp.atm.com.pl

Interaction Diagrams *are used to describe the flow of messages between the instances of different classes. The following is an example of such a diagram type, offered in the* ***Add-Ins*** *menu:*



Have fun testing!