Abort dialog

An operation is in progress. Please wait until the operation is finished. You may press the 'Cancel' button to abort the operation.

Add existing subclass

This dialog creates inheritance relationships between existing objects and the class currently in focus.

You may select one or more class names from the listbox.

New inheritances will be created automatically. You may use the controls below the edit field to define their inheritance mode. Mark the control 'set default' to store the settings as the new default mode for creating inheritances.

see also

Quickinfo

Add existing superclass

This dialog is used to add new inheritances to the currently focused class.

You may select one or more class names from the listbox.

New inheritances will be created automatically. You may use the controls below the edit field to define their inheritance mode. Mark the control 'set default' to store the settings as the new default mode for creating inheritances.

see also

Quickinfo

Add instance

This dialog is used to create one or more instances for the class currently in focus. Enter more than one name separated by commas.

see also

Quickinfo

Add new subclass

This dialog is used to create new classes that will be derived from the class currently in focus.

You may enter one or more new class names into the edit field (separated by commas).

As you create subclasses, new inheritances will be created automatically. You may use the controls below the edit field to define their inheritance mode. Check the box "set default" and the settings specified in the dialog will be used as the new default for future add operations.

see also

Quickinfo

Add new superclass

This dialog is used to create new classes that will be related to the class currently focused.

You may enter one or more new class names into the edit field (separated by commas).

As you create superclasses, new inheritances will be created automatically. You may use the controls below the edit field to define their inheritance mode. Check the box "set default" and the settings specified in the dialog will be used as the new default for future add operations.

see also

Quickinfo

Automatic save options

Automatic save means that OEW will remind you from time to time to save your work on disk. You may activate or deactivate this feature.

Define how often OEW will remind you to save. Intervals from 5 up to 60 minutes are allowed.

Check the 'Query' box, and OEW will ask you before it saves automatically. You may then confirm or cancel saving. In case you cancel, OEW will repeat the autosave request. Use the 'Repeat every' control to define how much time OEW will give you until it asks again.

see also

Quickinfo

Backup Files

It is recommended to work with backup files. In this case OEW will not overwrite your project's last version when saving, but renames it and then saves to a new file. Using this option you always have a certain number of last project versions as backups. This is useful if you make mistakes or if files get damaged.

When OEW renames a file, it keeps the first part of the filename, but changes the filename extension to a three digit number. The lowest non-existing filename is chosen. For example, if there are files with extensions from 000 to 137 and from 139 to 999, OEW will save the backup as number 138 and delete the file with extension 139. If you never interfere manually with this process you will have 999 backup files. In the described case, previous to saving, the oldest file had extension 139, becoming more recent up to extension 999, and 137 was the latest.

As OEW files may become quite big in large projects, the maximum number of backup files may be limited to a smaller value than 999. OEW will then ignore existing files with extensions larger than the defined limit.

Error

Error

You tried to assign a new default source module to the selected class(es).
You may not do so, as you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Message

OEW asks you to confirm your request to move a relationship.
Press 'Yes' to move the relationship, press 'No' to cancel.

Error

A 'part of' relationship defines that one object contains the other.
You tried to define that an object contains itself. This is not possible.

Error

An 'inheritance' relationship defines that one class is inherited from the other.
You tried to inherit an object from itself. This is not allowed.

Error

You tried to add an inheritance relationship that would cause a loop. This is not allowed.

Suppose you have three classes, Top, Middle and Bottom. Middle is derived from Top, Bottom is derived from Middle.

The error you see will be displayed when you try to derive Top from Bottom.

Error

You are not allowed to create an inheritance relationship between the classes you selected, because one already exists between them.

The existing inheritance may be in either direction, direct or indirect.

Error

You tried to connect two relationships, to build a relationship - inverse relationship pair.

You may not connect them, because one of the two relationships is of the type inheritance. These relationships may not have inverse relationships.

Error

You tried two connect two relationships. You may not do so, because one of the two relationships is already a relationship - inverse relationship pair.

Error

You tried to connect two relationships, but they can't be connected.

Two relationships that are to be connected must:
- exist between the same two classes
- have opposite directions.

Warning

For safety purposes, OEW asks you to confirm your rearrange request.
When you choose to proceed, the current diagram layout will be lost.
Press 'OK' to proceed and rearrange, press 'Cancel' to cancel.

Warning

For safety purposes, OEW asks you to confirm your rearrange request.
When you choose to proceed, the current diagram layout will be lost.
Press 'OK' to proceed and rearrange, press 'Cancel' to cancel.

Query

You have changed the default module for the selected classes.
OEW can move all members that already exist in the selected classes to the new default module.
Select 'Yes' to move. Select 'No', and no elements will be moved.

Error

Error

OEW couldn't write a backup copy of the objectbase you are saving. A file system problem occurred.

Please check your disk for sufficient space and your user account for sufficient access rights. Another possible reason for error: A file with the name that OEW used for the backup already exists, and you aren't allowed to overwrite the existing file.

Error

Error

Information

Error

You tried to delete one or more objects (classes or instances), but at least one of the objects is not accessible to you. As a result, your request cannot be executed, and OEW will not delete any objects.

Ask a supervisor to grant you access to the objects.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete, press 'No' to cancel.

Query

The OEW autosave feature is activated. OEW reminds you that the defined interval has expired and it is now time to save.

Please press 'OK' to save now. Press 'No' to cancel, this will give you three additional minutes without a save reminder.

Error

OEW tried to update the project file for the Borland C++ 3.1 compiler, but an error occurred in accessing the file.

Error

Error

You tried to move the selected slot(s) to a new module.
You may not do so, as you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Error

Query

Information

To use OEW's makefile generation functionality, you need to supply the filename that OEW uses for it. Enter it in the 'Global Filenames' dialog, accessible with the command 'Source Global Filenames'.

As there is currently no filename specified, OEW can't generate a makefile.

Information

An error occurred while writing the makefile.

Possible error causes:
- the path to the makefile filename doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Information

An error occurred while writing the linkfile (this is the file used by the makefile that contains linker commands).

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

You are evaluating a demonstration version of OEW. It has limits on the number of classes it can process. You have just reached one of those limits.

The retail version of OEW doesn't have this limit.

Information

You requested to generate changed modules to java sourcefiles. But this isn't necessary, because nothing has been changed since your last source code generation.

Query

You requested to close the last inheritance or relationship window of an objectbase. OEW shows this dialog to inform you that the data contained in the objectbase has not been saved.

Press 'Yes' to save your changes on disk. (If the file doesn't have a filename, a dialog will open where you have to enter it.)

Press 'No' and your changes will be lost.

'Cancel' doesn't close the window, your objectbase will stay open for further processing.

Error

During the parsing process (source code import), OEW tried to open the displayed file, but it was not able to access it.

Possible error causes:
- the file doesn't exist
- you don't have enough rights to access the file.

Information

You are evaluating a demonstration version of OEW. It has limits on the number of classes it can process. You have just reached one of those limits.

The retail version of OEW doesn't have this limit.

Error

You tried to assign a new module to the displayed class.
You may not do so, as you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Error

You are evaluating a demonstration version of OEW. It has limits on the number of classes it can process. You have just reached one of those limits.

The retail version of OEW doesn't have this limit.

Error

An error occurred while OEW tried to write the displayed filename.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

You tried to add a new global instance, but used a name that is already in use by another global instance.

Please try again using a unique name.

Error

You are evaluating a demonstration version of OEW. It has limits on the number of classes it can process. You have just reached one of those limits.

The retail version of OEW doesn't have this limit.

Error

You are evaluating a demonstration version of OEW. It has limits on the number of classes it can process. You have just reached one of those limits.

The retail version of OEW doesn't have this limit.

Information

Error

An error occurred while OEW tried to open the 'ObjectStore Source Generation schema' file.
Maybe you aren't allowed to access the file.

Error

OEW tried to open the 'ObjectStore Source Generation schema' file, but since there isn't a filename specified in the 'ObjectStore options' dialog, OEW can't open it.

Error

An error occurred while OEW tried to open the 'ObjectStore Header Generation schema' file.
Maybe you aren't allowed to access the file.

Error

OEW tried to open the 'ObjectStore Header Generation schema' file, but since there isn't a filename specified in the 'ObjectStore options' dialog, OEW can't open it.

Error

You tried to assign a new module to the displayed class.
You may not do so, as you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Error

An error occurred while OEW tried to open the 'ObjectStore schema file schema' file.
Maybe you aren't allowed to access the file.

Error

OEW tried to open the 'ObjectStore schema file schema' file, but since there isn't a filename specified in the 'ObjectStore options' dialog, OEW can't open it.

Error

An error occurred while OEW tried to open the 'Makefile schema' file.
Maybe you aren't allowed to access the file.

Error

OEW tried to open the 'Makefile schema' file, but since there isn't a filename specified in the 'ObjectStore options' dialog, OEW can't open it.

Error

You tried to add an inheritance relationship that would cause a loop. This is not allowed.

Suppose you have three classes, Top, Middle and Bottom. Middle is derived from Top, Bottom is derived from Middle.

The error you see will be displayed when you try to derive Top from Bottom.

Error

You tried to inherit a class from itself. This isn't allowed.

Error

You tried to add an inheritance relationship that would cause a loop. This is not allowed.

Suppose you have three classes, Top, Middle and Bottom. Middle is derived from Top, Bottom is derived from Middle.

The error you see will be displayed when you try to derive Top from Bottom.

Error

You tried to add an inheritance relationship that would cause a loop. This is not allowed.

Suppose you have three classes, Top, Middle and Bottom. Middle is derived from Top, Bottom is derived from Middle.

The error you see will be displayed when you try to derive Top from Bottom.

Warning

Error

You tried to create a new class, but another class with the name you entered already exists. Please enter a unique name.

Error

You checked the flag 'Template', but you didn't enter the template parameters. This is not allowed. Either enter the parameters of your choice, or uncheck the 'Template' flag.

Error

You tried to create a new class, but another class with the name you entered already exists. Please enter a unique name.

Error

You tried to create a new class, but another class with the name you entered already exists. Please enter a unique name.

Error

You tried to create a new class, but another class with the name you entered already exists. Please enter a unique name.

Error

You tried to create a new class, but another class with the name you entered already exists. Please enter a unique name.

Query

The active objectbase contains modules that are references to existing external Java files. OEW stored their file date and time. It just recognized that at lease one of those files differs from the stored information. The file has probably been changed manually using a text editor (or another application, or another OEW project).

Press 'Yes' and OEW will import the externally changed file(s). Press 'No' to not import.

Query

In an editor window, you requested to display the edit dialog for the object whose name is currently at the cursor position. But OEW couldn't find an object with that name in the objectbase.

OEW allows you to activate the Query window to allow you to start further searches. Press 'Yes' to open and activate it, or press 'No' to continue using the editor.

Error

You tried to open an objectbase file, but OEW was not able to access it.

Possible error causes:
- the file doesn't exist
- you don't have enough rights to access the file.

Error

You tried to open a file as an objectbase, but it isn't an objectbase.
If it is a text or Java file, try to open it using OEW's editor.

Error

You tried to open a file as an objectbase, but it isn't an objectbase.
If it is a text or Java file, try to open it using OEW's editor.

Error

You tried to open a file as an objectbase, but it isn't an objectbase.
If it is a text or Java file, try to open it using OEW's editor.

Error

You erased the contents of the 'Name' field that specifies the name of the class you are editing. This is not allowed.

You must enter a name in the 'Name' field.

Query

During your work with the current objectbase, some of the modules contained in it have become empty, that is, no elements are stored in them.

Such objects aren't necessary and can be deleted. Actually, it is recommended to delete them if you don't plan to add elements in the near future, because OEW will keep displaying this message each time it sees the empty module(s).

Press 'Yes' to delete the empty modules now, press 'No' to keep them and continue, or press 'Cancel' to keep them and interrupt the operation you requested that led to the displayed message.

Query

You requested to generate the files that are displayed in the message. OEW recognized that the file seems to have contents unknown to OEW, because it is newer or older than the last time OEW accessed the file.

OEW asks you to decide how you want to continue. Press 'Yes' to overwrite the file (and possibly lose changes made externally to this file), press 'No' to leave the file unchanged, and press 'Cancel' to cancel the file generation progress, that is, OEW will stop generating files (only applicable if you requested to generate more than one file, and there are still files to generate).

Error

An error occurred while OEW tried to write the displayed filename.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

An error occurred while OEW tried to write the displayed filename.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

An error occurred while OEW tried to write the displayed filename.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

An error occurred while OEW tried to write the displayed filename.

Possible error causes:
- the path to the file doesn't exist
- you do not have not enough rights to create a new or overwrite an existing file.

Information

Information

Error

You selected to redirect print output to a file, but didn't enter a filename.
Please type a filename into the field 'Output File Name' and try again.

Information

You requested to print to a printer, but there are no printers installed in your system.
Therefore, OEW is not able to print.

Error

You tried to rename the class that is displayed by changing the contents of the 'Name' field. Renaming is allowed, but another class with that name already exists. Please enter a unique name.

Information

You requested to display a preview of the output that will be generated. Unfortunately, the OEW preview logic requires that there is a printer installed in your system.

Since there is no printer installed in your system, OEW isn't able to display a preview.

Error

An error occurred while OEW tried to write the print output to the filename you entered.

Possible error causes:

- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

An error occurred while OEW tried to write the print output to the filename you entered.

Possible error causes:

- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

OEW tried to print your document on the selected printer, but an error occurred. OEW was not able to access the printer.

Error

Information

You requested to search for a string in the text, but you didn't enter a string.
Please enter the string or regular expression into the 'find' field.

Information

You selected to search for a regular expression, but the string you entered into the 'find' field isn't a regular expression as understood by the OEW editor.

Information

You requested to search for a string in the text, but you didn't enter a string.
Please enter the string or regular expression into the 'find' field.

Information

You selected to search for a regular expression, but the string you entered into the 'find' field isn't a regular expression as understood by the OEW editor.

Warning

You requested to search for a string (or regular expression) and to replace that string, but the 'replace with' field is empty.

OEW displays this warning, to ensure that you haven't forgot to enter the string

Press 'OK' to confirm your input. OEW will delete the occurences it finds. If you don't want it to delete, press 'Cancel'.

Query

You renamed the displayed class.

You may now make OEW automatically search the whole project for occurrences of the old name and replace them with the new name.

Press 'Yes' to replace, press 'No' if you don't want to replace.

Information

You requested to search only within the selected block of text.

In OEW, the cursor position defines where OEW starts its search. But the cursor isn't within the marked area. This is not allowed.

Information

You requested to search only within the selected block of text.

In OEW, the cursor position defines where OEW starts its search. But the cursor isn't within the marked area. This is not allowed.

Information

The text (or regular expression) that you are searching for couldn't be found within the region specified.

Please note that OEW started its search from the current cursor position. You may wish to move the cursor to the top of the text and try again.

Warning

The file that is currently being edited with the editor has been changed by another program, since OEW accessed it the last time.

Press 'OK', and OEW will overwrite the changes made in the background, press 'Cancel' to cancel saving.

Error

An error occurred while OEW tried to write the displayed filename.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

Error

Information

An internal error occurred while OEW tried to access a text resource that it assumed was contained in the file OEWRES1.DLL. Please check whether this file is the one that has been shipped with your active OEW program file (you could compare their dates).

If you can't correct this problem, please report this bug with the OEWRES1.DLL filename and the displayed ID.

Information

An internal error occurred while OEW tried to access a text resource that it assumed was contained in the file OEWRES.DLL. Please check whether this file is the one that has been shipped with your active OEW program file (you could compare their dates).

If you can't correct this problem, please report this bug with the OEWRES.DLL filename and the displayed ID.

Error

You requested to delete the currently selected slots, but currently there are no slots selected in the active slot list window.

Please select the slots you want to delete and try again.

Query

You have changed the module for the displayed classes.
OEW can move all members that already exist in the selected class to the new default module.
Select 'Yes' to move. Select 'No', and no elements will be moved.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'OK' to delete, or press 'Cancel'.

Error

You requested to edit slots, but there are no slots selected.
Please select one or more slots and try again.

Query

For safety purposes, OEW asks you to confirm your delete request. All the selected classes (and instances) will be deleted.

Press 'Yes' to delete, or press 'No' to cancel.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete the selected relationship(s), press 'No' to cancel.

Query

For safety purposes, OEW asks you to confirm your delete request. All the selected classes will be deleted.

Press 'Yes' to delete, or press 'No' to cancel.

Information

You tried to create a relationship between two classes, but at least one of them is inaccessible. The operation was cancelled.

Information

You tried to connect two relationships to a relationship - inverse relationship pair, but you are not allowed to access at least one of the involved classes.

The operation was cancelled.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete, press 'No' to cancel.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete, press 'No' to cancel.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete, press 'No' to cancel.

Error

OEW couldn't open the file you requested.

Possible error causes:
- The file doesn't exist any more.
- You haven't enough rights to access the file.

Error

You tried to move the displayed slot to a new module.
You may not do so, as you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Information

You tried to add an inheritance relationship that would cause a loop. This is not allowed.

Suppose you have three classes, Top, Middle and Bottom. Middle is derived from Top, Bottom is derived from Middle.

The error you see will be displayed when you try to derive Top from Bottom.

Query

You requested to paste the data currently contained in the clipboard.

The data may be inserted in two different ways, as a global object, or locally, as new parts of the class that is currently in focus. OEW asks you to choose between both possibilities.

Press 'Yes' to insert the clipboard data locally, press 'No' to insert the clipboard data globally.

Query

You requested to paste the data currently contained in the clipboard.

The data may be inserted in two different ways, as a global object, or locally, as new parts of the class that is currently in focus. OEW asks you to choose between both possibilities.

Press 'Yes' to insert the clipboard data locally, press 'No' to insert the clipboard data globally.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete, press 'No' to cancel.

Error

You didn't specify the 'Declaration' module for the class you are currently editing.
This is not allowed, please select a module from the list.

Error

You enabled the setting 'import module', but you didn't specify which module you want to be automatically included. This is not allowed.

Please select a module from the list on the right hand side of the 'import module' label, or disable the setting.

Error

The field 'File name' is empty. This is not allowed. Each module (except the default module) must have a filename associated with it.

Please enter a filename and try again.

Error

An error occurred while OEW tried to write the ObjectStore os_typespec source file.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

An error occurred while OEW tried to write the ObjectStore os_typespec header file.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

An error occurred while OEW tried to write the ObjectStore schema file.

Possible error causes:
- the path to the file doesn't exist
- you do not have enough rights to create a new or overwrite an existing file.

Error

You haven't entered a name in the 'Name' field.
A datatype without a name isn't allowed. Please enter a name.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete the instance, press 'No' to cancel.

Query

You requested to paste the data currently contained in the clipboard.

The data may be inserted in two different ways, as a global object, or locally, as new parts of the class that is currently in focus. OEW asks you to choose between the two possibilities.

Press 'Yes' to insert the clipboard data locally, press 'No' to insert the clipboard data globally.

Information

The value you entered in the field 'Repeat every' is less than the smallest allowed value. Please choose or enter an allowed value.

Information

The value you entered in the field 'Repeat every' is larger than the largest allowed value. Please choose or enter an allowed value.

Information

You tried to open an objectbase that is already open. You needn't open it again, use the open windows to work with it.

Token Too Large

ATTENTION!

Don't ignore this message or you will lose information.

OEW doesn't support method bodies longer than 30000 characters. Please avoid this limit by splitting it into two or more smaller ones.

OEW wasn't able to import everything correctly. Please use an editor to remedy the problem and afterwards import the file again.

Error

Information

The demo version can't print. However, you can view the printout on the screen. Therefore use the 'preview' button.

Error

You tried to rename the displayed datatype by changing the value in the 'Name' field.
Renaming is allowed, but another datatype with that name already exists. Please enter a unique name.

Error

You have entered an invalid value. Allowed values are 1 to 999.

Information

The name that you have entered could not be found in any view. Please check your input.

Error

You have chosen to print a page range, but didn't enter a range into the associated field.
Please correct the problem and try again.

Error

Your input for the field 'Number from' is not valid. The field must not be empty and must contain a number.

The documentation's page numbers will start with the number that is entered here.

The default value for this field is '1'.

Error

You didn't enter a name for the new layout.
A name is required to create a layout. Please enter a new name into the field and try again.

Information

The entry that you are trying to delete is one of the three predefined layouts 'Analysis', 'Design' oder 'Implementation'.

These layouts may not be deleted. You only can reset the changes that have been made to them to an inital state.

Question

You gave the command to delete a layout style. To make sure you really want to do this you are asked to confirm your delete request.

Error

You didn't enter a name for the new set of contents settings.
A name is required to create it. Please enter a new name into the field and try again.

Error

You tried to create a new layout using a name that is already used for another existing layout. This is not allowed.

Each layout must have a unique name.

Please enter another name and try again.

Error

You tried to move the datatype to a new module.
You may not do so, as you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Error

You tried to create a new set of settings using a name that is already used for another existing set. This is not allowed.

Each set of settings must have a unique name.

Please enter another name and try again.

Information

The entry that you are trying to delete is the predefined set 'Standard'.

This entry may not be deleted. You only can reset the changes that have been made to it to the initial state.

Question

You gave the command to delete a set of settings. To make sure you really want to do this you are asked to confirm your delete request.

Error

OEW has detected an error in the parentheses of the changed source code. You can ignore this message, but in that case You might encouter parsing problems.

Error

There is already a slot in that class with the same name. Please change the name of the slot.

Error

OEW couldn't start the external Editor. Please check the settings in the 'Editor options' dialog.

Error

OEW found an error during the generation of the SQL database scheme. Please refer to SQL support for more information.

Error

There already exists a SQL conversion for this OEW type. There can be only one conversion for each OEW Type.

Query

Choose 'Yes' if You want to delete the selected SQL conversions. Otherwise choose 'No'.

Query

You renamed the displayed datatype.

You may now make OEW automatically search the whole project for occurrences of the old name and replace them with the new name.

Press 'Yes' to replace, press 'No' if you don't want to replace.

Error

To delete datatypes, you must select them in the list.
Select one or more datatypes, then press the delete button.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete. Press 'No' to cancel.

Error

To edit a datatype, select one from the list first.
After you have selected one, press the 'edit' button again.

Error

To edit a container class, select one from the list first.
After you have selected one, press the 'Edit' button again.

Error

It is not possible to edit more than one container class at once.
Please select only one container class.

Information

Error

To delete a container class, select it from the list.
After you have selected it, press the 'Delete' button again.

Error

There is more than one container class selected in the list.
For safety purposes, it is not allowed to delete more than one container class at once.
Please select only one container class from the list and press the 'Delete' button again.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press the 'Yes' button to delete, press 'No' to cancel.

Error

Your input for the field 'page range' is not valid. Please check it for errors.

see also: Generate Documentation - Pages

Error

You changed the name of the cardinality.
Renaming is allowed, but another cardinality with that name already exists. Please enter a unique name.

Error

A 'part of' relationship defines that one object contains the other.
You tried to define that an object contains itself. This is not possible.

Error

An 'inheritance' relationship defines that one class is inherited from the other.
You tried to inherit an object from itself. This is not allowed.

Error

You tried to add an inheritance relationship that would cause a loop. This is not allowed.

Suppose you have three classes, Top, Middle and Bottom. Middle is derived from Top, Bottom is derived from Middle.

The error you see will be displayed when you try to derive Top from Bottom.

Error

You are not allowed to create an inheritance relationship between the classes you selected, because one already exists between them.

The existing inheritance may be in either direction, direct or indirect.

Error

You didn't enter a name for the relationship in the 'Name' field in the dialog's 'Relationship' section. This is not allowed. A relationship must have a name.

Error

No relationship type specified.

Information

Error

You are not allowed to create an inheritance relationship between the classes you selected, because one already exists between them.

The existing inheritance may be in either direction, direct or indirect.

Error

You are not allowed to create an inheritance relationship between the classes you selected, because one already exists between them.

The existing inheritance may be in either direction, direct or indirect.

Error

A class may not be inherited from its base class not more and not less than exactly once. Therefore, you have to specify 1:1 for the relationship cardinality.

Error

You selected a 'contains' relationship with an associated inverse relationship. But you selected a value other than '1' for the maximum cardinality of the inverse relationship.

For a 'contains' relationship, there is always only one containing object, therefore the maximum cardinality of the inverse relationship MUST be '1'.

Please select '1' in the field and try again.

Error

A 'part of' relationship defines that one object contains the other.
You tried to define that a object contains itself. This is not possible.

Error

You selected to create a 'contains' relationship between the classes.
This is not possible, because one of the classes is already inherited from the other (directly or indirectly).

Error

You are not allowed to create an inheritance relationship between the classes you selected, because one already exists between them.

The existing inheritance may be in either direction, direct or indirect.

Error

You are not allowed to create an inheritance relationship between the classes you selected, because one already exists between them.

The existing inheritance may be in either direction, direct or indirect.

Error

You pressed the assign buttons. This button is used to assign access rights to the selected group. But you didn't select a group.

Please select a group and try again.

Error

You pressed the 'Objects?' button. It is used to query and display the objects that the selected group may access. Therefore, it is necessary to have a group selected to use this button.

Please select a group and try again.

Information

Information

The value you entered in the field 'Save every' is less than the smallest allowed value. Please choose or enter an allowed value.

Information

The value you entered in the field 'Save every' is larger than the largest allowed value. Please choose or enter an allowed value.

Warning

As the text explains, you shouldn't disable the automatic save feature. Since no software is completely error free, it may appear that OEW stops execution due to an error condition without giving you the possibility to save your work.

To minimize possible data losses, OEW provides the autosave feature. Please don't disable this option.

Error

You tried to assign a new declaration module to the instance.
You may not do so, because you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Error

You tried to assign a new definition module to the instance.
You may not do so, because you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Error

You tried to rename the instance that is displayed by changing the contents of the 'Name' field. Renaming is allowed, but another instance with that name already exists. Please enter a unique name.

Query

You renamed the displayed instance.

You may now make OEW automatically search the whole project for occurrences of the old name and replace them with the new name.

Press 'Yes' to replace, press 'No' if you don't want to replace.

Error

You have entered a path that doesn't exist.
Please enter existing ones. Check for spelling.

Error

The entered pathname is not in a format allowed by your operating system.

Possible errors:
- invalid character(s)
- name(s) too long.

Error

The Login dialog is used to tell OEW who you are. Enter your name in the 'Name' field.

Information

Error

You entered a name in the 'Name' field that isn't known to OEW. Check for correct spelling.
If you don't have an OEW user account yet, ask your local system administrator to add one for you.

Error

The password you entered isn't the password defined for the user entered in the 'Name' field. Please try again.

Error

A system administrator has locked your OEW user account. You aren't allowed to access the OEW application.

Information

You tried to add a user, but a user with the name you specified already exists.

Please enter another user name. You can append a symbol or a number after the name to distinguish between the persons.

Information

There is already another user who has the requested initials assigned.
Please choose different initials for this user.

Information

Every user must be the member of a group.
Please select an existing group for this user.

Information

Another group with this name already exists. Please enter a unique name.

Information

The group level you entered isn't allowed.
Valid values are from -999 to 999.

Information

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete the user, press 'No' to cancel.

Information

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete the group, press 'No' to cancel.

Information

Error

You tried to delete a group that has users as members assigned to it. For safety purposes it is only allowed to delete groups that have no users assigned.

To delete this group, you have to remove all the users from this group. You may delete them or assign them to other groups.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete all the modules that are currently selected, press 'No' to cancel.

Error

You are evaluating a demonstration version of OEW. It has limits on the number of classes it can process. You have just reached one of those limits.

The retail version of OEW doesn't have this limit.

Information

You requested to generate the modules that are selected in the list.

But because all of the selected modules are of type 'library' and such modules are never written by OEW no files need to be generated.

Error

You tried to rename the displayed module by changing the contents of the 'Name' field.

Renaming is allowed, but another module with the name you entered already exists. Please enter a unique name.

Error

The 'File name' you entered already is in use by another module. Two modules cannot have the same file name.

Please enter a unique name.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete, press 'No' to cancel.

Information

You tried to delete an Instance module element, either the declaration or the definition.

OEW does not allow this. It is not possible to delete only one of them. Therefore, to delete all parts associated with an instance, you must use the delete function in the inheritance window.

When you see the instance in the inheritance window, select it and choose Edit Delete from the menu or use the instance's object menu.

Error

Error

You are currently editing an import statement.

Import statements define a class that you wish to be imported with an import in your Java sourcefile.

This dialog is used to specify which class you want to be included, but you didn't specify one. This is not allowed.

From the list, select the module of your choice.

Information

Error

You are currently editing an import statement.

Import statements define a class that you wish to be imported with an import in your class definition.

This dialog is used to specify which class you want to be imported, but you didn't specify one. This is not allowed.

From the list, select the module of your choice.

Query

For safety purposes, OEW asks you to confirm your delete request.
Press 'Yes' to delete, press 'No' to cancel.

Error

You tried to rename the displayed view by changing the contents of the 'Name' field.
Renaming is allowed, but another view with the new name already exists. Please enter a unique name.

Information

You didn't enter the line number where you want your cursor to be navigated to.
Please enter a number and try again.

Information

A line number is a number. A number consists only of digits, that is 0 1 2 3 4 5 6 7 8 or 9.
You entered characters other than digits. Please correct your input and try again.

Information

You requested to navigate the cursor in the current active editor to a line number that doesn't exist. Most likely you entered a number larger than the largest allowed value.

The information box shows you the range of allowed values. Please correct the number and try again.

Error

Information

The current text selection in the active editor is too large. OEW is not able to copy a block of this size to the clipboard.

Please try to copy a smaller block.

Information

The text in the clipboard is too large. OEW is not able to insert a block of this size.
Please try to insert a smaller text.

Information

Since you are not editing a file, but a text as part of an object, saving is not allowed here.

However, you may export the text. To do this, choose:

- Edit Select all
  Edit Save as.

Error

You tried to assign a new header module to the selected class(es).
You may not do so, as you aren't allowed to access the module you've chosen.
Please use a module you have access to.

Information

You requested to find the brace that belongs to the brace currently at the cursor position.
Such a brace doesn't exist.

Query

You requested to close an editor window. OEW shows this dialog to inform you that the text contained in the editor has not been saved.

Press 'Yes' to save your changes on disk. (If the file doesn't have a filename, a dialog will open where you have to enter it.)

Press 'No' and your changes will be lost.

'Cancel' doesn't close the editor window. It will stay open for further processing.

Message

You requested to delete a relationship. However, the relationship has an inverse relationship associated with it.

Press 'Yes' to delete both relationships, press 'No' and OEW will not delete the inverse relationship.

Error

You are evaluating a demonstration version of OEW. It has limits on the number of classes it can process. You have just reached one of those limits.

The retail version of OEW doesn't have this limit.

Information

Message

OEW asks you to confirm your request to move a relationship.
Press 'Yes' to move the relationship, press 'No' to cancel.

Error

A 'part of' relationship defines that one object contains the other.
You tried to define that an object contains itself. This is not possible.

Error

An 'inheritance' relationship defines that one class is inherited from the other.
You tried to inherit an object from itself. This is not allowed.

Error

You tried to add an inheritance relationship that would cause a loop. This is not allowed.

Suppose you have three classes, Top, Middle and Bottom. Middle is derived from Top, Bottom is derived from Middle.

The error you see will be displayed when you try to derive Top from Bottom.

Error

You are not allowed to create an inheritance relationship between the classes you selected, as one already exists between them.

The existing inheritance may be in either direction, direct or indirect.

Code generator options

Code generator is the part of OEW that writes C++ source code files from the information stored in your objectbase. It is activated whenever you tell OEW to generate or update source code.

Use this dialog to configure its options.

Replace Comment Variables

By default OEW replaces comment variables only when printing the documentation. If you require to replace in your source code, too, check this option. Please note that this process is not reversible. Once you reimport such sources you will no longer have variables but fixed information.

see also

- PathStyle
- Edit global filenames
- Quickinfo

Configure Documentation

The documentation you are going to create will be generated as a sequence of tables, with or without headings and some additional text between the tables. The contents of the tables and their appearance may be fully configured.

Being more detailed, you configure types of tables. The table with dataslots will appear once for each class, other tables, e.g. the one containing statistical information, will appear only once.

The dialog 'Configure Documentation' consists of several areas. At its top is a listbox (which may be opened) containing all the available tables, all the other areas show properties of the currently selected table and allow editing. From top to down these are:

- Table properties
- Table's cells properties
- Column layout graphical preview

If you want to copy settings from another layout use the button 'Copy' in the upper right corner.

Finally some tips on column widths.

Configure Documentation - Column layout graphical preview

The preview at the lower dialog's area doesn't show the table exactly as it will be printed, but is used to display the relations of the column sizes to each other. The table output will use the complete available space, but the division of the available space will be identical to what you see here.

You can change the widths directly in the preview using the mouse. Point with the mouse arrow on the vertical lines between the columns. You will notice that the mouse arrow changes its shape. Click and hold the left mouse button and move the mouse to the left or to the right to change the width. Release the mouse button at the position of choice.

Please notice: It's not possible to make columns smaller than a minimum, you can't delete them by dragging them to a size of zero.

Configure Documentation - Contents

Select all the components that you wish to be contained in the documentation.

Configure Documentation - Copy Button

This button allows to copy complete layout styles or only the layout of a single table. Use this button to open the Dialog 'Copy Layout'.

Configure Documentation - Copy Layout

You can copy the layout settings between different layout styles.

The target of this operation is always the style you are editing currently, you can't change the others.

You may choose the amount of data that should be copied. Select this using the setting for 'Target table(s)'. If you choose 'Currently selected table' only the currently shown table will be modifed by copying the layout from the other table. 'All Tables' will overwrite the complete layout you are editing and copies the settings for all tables.

Choose the source layout from the list.

Configure Documentation - Create new Layout

Before the new layout can be created, you need to provide some additional information.

It is required to enter a name for the new layout.

Furthermore you may create the layout based on another layout that already exists. All settings from the chosen layout will be copied. Please choose the base style from the list.

Configure Documentation - New Settings for "Contents"

You have to name the new set of settings that you are going to create before it can be created.

Configure Documentation - Page margins

Please enter the margin sizes you wish to use for the printout in millimeters (one inch equals 25.4 millimeters).

Configure Documentation - Table properties

The heading that will be printed in front of each table may be changed.
You may choose to print framses around the cells or not.
Choose the option 'On New Page' if a new page should be started each time this table appears.

Configure Documentation - Table's cells properties

For (nearly) every table it may be defined which columns are to be used. The two lists 'Available Columns' and 'Used Columns' show the current selection.

There are some tables where this is not true, and where you can't select the columns, but there are not many of them. For those tables only widths of columns may be configured. You can recognize such tables, as the lists 'Available Columns' and 'Used Columns' will be both empty.

If the lists contain entries, you may choose the columns. Use the buttons between the lists that contain arrows to move columns between the lists. Two buttons contain two arrows. Each of them moves all the columns to the shown direction. The other two buttons containing one arrow each are used to move only the current selection to the shown direction. The changes made are reflected in the graphical preview immediately.

To the right of the lists are another four buttons that are used to set positions and widths of columns. The buttons 'left' and 'right' mean 'move to'. They are only available while there is at least one column selected in the list 'Used Columns'. The buttons 'smaller' and 'bigger' are only available if there is exactly one column name selected. The field 'heading' is used to edit the heading of the selected column.

Configure Documentation - Tips on Column Widths

If You would like one or more columns to be of the same size, it's not required that you adjust this manually.

Just move all the columns that you want to have the same size to the list 'Available Columns' on the left. Then select all of them again (now in the list on the left) and move them to the right in one step (using the button showing one arrow to the right).

Note on the dialog's behaviour for column widths:

- If you add columns to the list of used columns, this won't change the size relations between the already used columns. All of them will be made smaller in the same ratio to make room for the new columns.
- To change a column size, but leave the size relations of the other columns unchanged, use the buttons 'smaller' and 'bigger'.
- If you want to adjust the widths manually, use dragging by mouse of the vertical lines in the graphical preview. This always changes the size of exactly two columns.

Object Engineering Workbench(TM) for Java(TM) - Help Contents

Create a relationship

Use this dialog to add a new relationship to the class you have selected. Its name is displayed at the dialog's top.

Below it, you see a list of all available classes to create a relationship to. Select the class of your choice. This is especially useful if the requested class isn't a member of the current view or is graphically not easily reachable in your display.

Press 'OK' and the new relationship will be created immediately and a dialog to edit the new relationship's properties will open.

see also

- Quickinfo

Data OEW operates on

OEW is a 'Java project editor'. It stores complete source code and logical information that belongs to Java projects and allows editing everything on an abstracter level.

There are two possible ways OEW stores data externally.
- in an OEW file. It is called Objectbase.
- in standard Java source files.

Due to the textual structure of source files, not all the data managed by OEW can be stored in them. Two examples are relationship names and views. Thus you should always store in objectbases. Use Java source files only for compiling and debugging purposes.

OEW allows you to have multiple projects opened simultaneously.

Database support

OEW offers support for SQL database management systems.

You may activate the database support either when creating a new objectbase, or later during your work using the 'Options' menu.

Some windows will contain additional elements to provide access to the database schema information, for example persistence or indexes.

SQL support

Default inheritance mode

When creating new inheritances with the mouse, OEW will not ask you for the mode to be used. It will use the inheritance mode configured in this dialog.

see also

- SQL support
- Quickinfo

Documentation

A superior software product requires superior documentation. This is true for object-oriented systems just as for conventionally engineered software. Good documentation enables a third party with no prior knowledge of the software to become acquainted with it. A new analyst might add new concepts to the model, a designer might add support for new technology to the system, or a programmer might fix a bug. You might desire to look the project over weeks or months after it has been completed.

Another important purpose of documentation is facilitating communication between different members of a project team. A print-out of intermediate results pinned to a wall illustrates a problem presented to an expert. Furthermore, such a print-out can give a detailed overview of large structures that simply do not fit onto a computer monitor.

Writing documentation requires a lot of effort. Besides, a psychological problem arises from having to describe a project that you thought was finally finished. Documenting is a monotonous task, unloved by creative developers. Frequently, documentation is not even compiled by the developers. This results in additional expenses generated while the person writing the documentation becomes familiar with the program.

Thus, many projects have incomplete or no documentation. There are two ways of solving this problem. A superior can exert pressure to make the developers create the documentation. Alternatively, documentation can become a by-product of development. OEW offers support for this possibility.

OEW offers two types of documentation.

- A variety of print-outs of the data stored in an OEW objectbase can be generated. These tables and diagrams excellently support communication in a team. In addition, they are very important in the final documentation. Combined with OEW's re-engineering features, they can also serve in documenting existing, non-OEW projects.
- On-line documentation is accessible from OEW's GUI. As you can look up any references in the comments directly, you can follow links much faster than on paper. This kind of navigation through a text resembles hypertext documents. Object-oriented software almost requires documentation that mirrors its structure. For this reason, the source code in an OEW objectbase can be labeled hypercode or object-oriented documentation.

Duplicate class names

During an operation that inserts classes into your objectbase, OEW found a name conflict. OEW asks you to decide how you would like it to resolve this and future name conflicts.

OEW displays the class name that is duplicate ('old class name') and suggests a 'new class name'. You may change the new class name manually.

Use the 'Yes' button, and OEW will replace the original class name with the new class name before it inserts it. If you changed the new class name manually and there is still a name conflict, the same dialog will be displayed again.

Use the 'No' button, and OEW will delete the existing class before it inserts the new one.

Use the 'Replace All' button, and OEW will behave like the 'Yes' button for the displayed name conflict, and further name conflicts will be corrected by automatically renaming the classes that are inserted.

'Replace none' will never rename classes that are inserted, but keep their names and delete the existing ones. This choice is the same as using the 'No' button, without further confirmation.

see also

Quickinfo

Edit access

Control users' write access rights to objects. It is not possible to define rights individually for each user, but only for groups of users.

Every object in the objectbase is assigned to a group. The 'group level' defines the access right. Only users that belongs to a group with the same or a higher level may change the object.

In the lower left corner, select the group you want to define access rights for.

In the upper left corner, select the type of object you want to define access rights for. As you select types, the list on its right hand side will change accordingly.

Press the 'Objects?' button to display the objects the group is allowed to access. They will become highlighted in the list. Change the selection using the left mouse button in conjunction with the 'CTRL' keyboard key. Save the newly defined rights using the 'Assign' button.

While having selected the type 'Classes' (in the upper left corner), you may use the button 'Sel. clas.'. This button will read the selection in your topmost inheritance or relationship window and highlight the classes in the right hand side listbox accordingly.

Proceed with caution, because you may forbid yourself to access objects if you make an error. In this case only an OEW administrator can undo your mistake.

see also

Quickinfo

Edit class

Use this dialog to edit the properties of a class.

see also

- Database support
- Quickinfo

Edit class information

This dialog shows user defined information about a class, who created and modified the class (last modification only) and which user group the class belongs to. The user group defines the access rights for the class.

see also

Quickinfo

Edit comment

This dialog is used to edit an object's comment (e.g. a method's implementation comment). The dialog's title bar informs you, which object this comment belongs to.

see also

Quickinfo

Edit container class

Use this dialog to edit the properties of a container class that is used in OEW to define cardinalities for relationships.

The 'maximum cardinality' name is the name OEW displays in other dialogs, where you may choose a cardinality to use. It must be unique. Usually one uses abbreviations like 'm' for 'many' or 'l' for 'list'.

In 'container class' specify the implementation class for the cardinality. This field is required and will be used when OEW generates source code. However, it is not required to be known in the objectbase.

To make OEW automatically generate an import command in modules where this container class is used as a cardinality for data members, check the appropriate checkbox and select the' needed module' (the module, where your container class is defined in). Usually this is a library module of a class library. Please note that OEW will not recognize usages within method bodies.

see also

Quickinfo

Edit container classes

This dialog displays a list of all container classes used as relationship cardinalities. You may add new, change or delete cardinalities. Please note that after deleting a cardinality you should edit all the slots using this cardinality. Otherwise OEW would generate this slots as Java arrays.

see also

Quickinfo

Edit data slot

A data slot is a class member attribute.

Use the 'min. card' and 'max. card' to specify minimum and maximum cardinality. Min. is used only for documentation purposes, and should be either 0 or 1.

Maximum cardinality can contain various values, as defined in dialog accessible through Datatype / Container classes:

'1': a standard slot will be generated, for example DayOfWeek a;

'm' or any other in OEW defined cardinality: A container class will be used to implement your data slot, e.g. 'OEWList<int> a' or 'LIST /*int*/ a'

'[]' will generate an array of unspecified size like int a[].

Any other value, e.g. '10' or 'MaxCount': a standard C++ array will be generated, i.e. 'int a[10]' or 'int a[MaxCount]'.

You may define the data slot as 'final'.

The 'initial value' is used to initialize the variable, e.g. 'int a = 0'.

When using OEW's database support, you see additional property flags.

see also

Quickinfo

Edit default comments

Various objects in OEW have comments assigned. Each comment is independent from other comments.

However, when a new object is created, its comment will be initialized with a predefined default comment. Use this dialog to specify default comments for the different comment types. By default all of them are empty.

Use the list on the left hand side to select the comment you want to specify. The edit field on the right hand side will immediately change to the selected comment. You may change this comment. Your changes will be stored when you select another comment type or press the 'OK' button. The 'Cancel' button cancels only editing of the currently edited comment.

see also

Quickinfo

Edit group

You may change the name of the group and its level. The name must be unique.
The group level defines the access rights for members belonging to this group.

see also

- User management
- Quickinfo

Edit groups

A list of the user groups that are know to OEW is displayed. You may add new, edit or delete groups.

see also

- User management
- Quickinfo

Edit inheritance

An inheritance between two classes has properties. If the 'implements' chackbox is checked, this inheritance will be generated as 'class a implements b'. Otherwise it will be generated as 'class a extends b'.

see also

- Quickinfo

Edit method

This dialog is used to edit a variety of different method types. The method's name that is currently being edited, is displayed in the dialog's title bar.

'Return type' is the type your method returns. Constructors and destructors do not have a return type. For methods that are supposed to return nothing please enter 'void' into this field.

Enter the parameters your method takes in standard Java notation, that is, separate the parameters with commas. Destructors (the 'finalize' method) do not have parameters.

If this method is able to return exceptions, mark throws and specify the possible exceptions.

Check the 'abstract' or 'final' boxes to give the edited method the property, respectively.

The 'body' editor field is the place to put the Java source code into.

The method body contains the method's algorithm in standard Java syntax.

see also

Quickinfo

Edit module

Use this dialog to change a module's properties and maintain its list of elements. A module is the counterpart to a C++ file.

Every module must have a unique 'name' and a unique 'filename'. The filename may contain a path. In case it is a relative one it refers to the project directory.

OEW doesn't generate files for modules that have the 'Library Module' flag turned on. It is assumed that the file doesn't belong to your current project, but to another one or is part of a class library and therefore may not be changed.

The dialogs lower area contains the module's elements. You can add new elements using the 'New' button. The new element is added before the first selected element, or at the end of the list if there is no selection.

The order of the elements may be changed using the 'Sort' button.

The list is capable of drag and drop operations. You may move elements within it (change the order) or between different modules.

see also

- PathStyle
- The Module Concept
- Quickinfo

Edit module element - comment

Specify the comment to be included in the module.

Check the 'generate single line comment' if you like. The comment will be generated into one or more lines, each beginning with the // comment marker. Otherwise the comment will be generated using /* and */ comment start and end markers.

see also

Quickinfo

Edit module element - import

Use this dialog to specify which file to include.

see also

- PathStyle
- Global filenames
- Quickinfo

Edit module element - meta command

A meta command is a place-holder that OEW will replace with the requested data at source generation time. Available meta commands are:

- 'Module comment'
  The comment assigned to the containing module.
- 'Includes'
  The include lines that OEW automatically knows to be required.
- 'Datatype include'
  A single include, including the global datatypes module.
- 'Forward declarations'
  Forward declarations for all classes (library and non-library) used in the objectbase.
- 'POET-Forward declarations' (only with activated poet support)
  Like 'Forward declaration' with the difference that for persistent classes the appropriate keyword is generated, too.

see also

Quickinfo

Edit module element - preprocessor statement

Preprocessor statements are those lines in your code beginning with an #.
Enter the complete statement into this dialog.
For #includes, please use #include module elements.
see also

Quickinfo

Edit module element - userdefined entry

You may use userdefined entry to enter code that is not handled otherwise by OEW. Two examples are global functions (including friend functions) or global variables. The entered text will be generated unchanged.

see also

Quickinfo

Edit modules

The list of modules that belong to your objectbase is displayed.
You may add new, edit, delete or generate modules.
see also:

- The Module Concept
- Quickinfo

Edit multiple classes

This dialogs allows you to quickly change properties of multiple classes.

At the dialog's left hand side you see a list of all classes. Select the classes you wish to change the options for. Use the left mouse button in conjunction with the 'CTRL' key on your keyboard to select more than one class. You may use the button 'Sel. class.' to make OEW select all the classes that are currently selected in your topmost inheritance or relationship window.

As you select classes, the controls on the dialog's right hand side will change. They show the settings that all selected classes have in common. If not all selected classes have identical settings, the property's associated control will be either empty or gray. After you marked the options press the 'Assign' button to store them.

see also

- Database support
- Quickinfo

Edit multiple slots

This dialog allows you to quickly change settings for more than one slot.

On the left hand side you see a list of slots that you have chosen to edit. Changes done within this dialog will be applied to all of them after you press the 'OK' button.

On the right hand side you see some controls that show the current settings of the slots on the left hand side, but only those that all slots have in common! Other controls, where the respective property is not the same for all slots, are empty or gray.

Use the controls to specify the new settings. If you need a new datatype, you may use the 'Types' button to open the datatype dialog and define it.

see also

- Quickinfo

Edit reference / member class

A reference / member class slot is a class member attribute and is used to implement relationships between classes.

Use 'value class' to scpecify the referenced / contained class.

Use the 'min. card' and 'max. card' to specify minimum and maximum cardinality. Min. is used only for documentation purposes, and should be either 0 or 1.

Maximum cardinality can contain various values, as defined in dialog accessible through Datatype / Container classes:

'1': a standard slot will be generated, for example String a;

'm' or any other in OEW defined cardinality: A container class will be used to implement your data slot, e.g. 'OEWList<String> a' or 'LIST /*String*/ a'

'[] ' generates an array with an undefined size, e.g. 'int a[];'.

Any other value, e.g. '10' or 'MaxCount': a standard Java array will be generated, i.e. 'String a[10]' or 'String a[MaxCount]'.

You may define the slot as 'final'.

The 'initial value' is used to initialize the variable, e.g. 'String a = """".

When using OEW's database support, you see additional property flags.

see also

Quickinfo

Edit relationship

This dialog is used to define a relationship's properties.

The classes that this relationship exists between are displayed in the upper left corner. The controls to the right allow you to change the relationship type (inverse relationships are always of type reference). In the lower right corner you may specify a comment for the relationship.

In the lower left corner, you may define the relationship and, as an option, its inverse relationship. Use the name fields to give them a name of your choice. Those names will be used to label the relationship lines in relationship windows. The name field for the inverse relationship defines whether there is an inverse relationship or not. Leave it empty and there will be no inverse relationship. It is not possible to specify an inverse relationship only.

You may tell OEW to automatically create and maintain a slot for a relationship by using the create slot checkbox. Access and edit the created slot using the edit slot button.

Use the min and max controls to define the cardinality of your relationship. The Maximum cardinality defines the appearance of the automatically maintained slot. A value of one will give you a simple slot, any other number creates an array. Choose a value from the list accessible via the arrow down button, and OEW creates code using a container class, as defined in OEW's container classes list. Minimum cardinality is for documentation purposes only.

Changing the relationships's associated slot may change the relationship type. For example, changing the type of the associated slot to 'integer' will delete the relationship.

The 'name' field defines whether an inverse relationship exists or not. Leave the field empty and there is none.

Inheritance relationships have neither a cardinality nor an inverse relationship.

see also

- Quickinfo

Edit slot

This dialog is used to edit the common properties of all kinds of slots.

The name has to be unique if it is not a method. In the dropdown box you can find names for the C++ operators and the names for constructor and destructor.

Depending on the type of the slots it has additional properties, to access them press the 'More' button. To create a constructor or destructor the slot type has to be 'method'.

With 'Access' you define a slot as 'public', 'protected', 'private' or '<default>'.

Methods and dataslots may be 'static', then they exist independently from the instances of their class.

In the dialog's lower area a specification comment may be added. Press 'default' to reset the field contents to the text that were defined in the 'Default comments dialog'.

see also

Quickinfo

Edit user

You should create a user account for every person that works with OEW. Use this dialog to set a user's account properties.

'Name' and 'Initials' must be unique, that is, no two users are allowed to have the same initials.

The 'group' controls the user's write access rights. A user may change an object if he belongs to a group with the same or a higher level than the object's assigned group level.

A 'supervisor' always has all rights, a locked user may not login to OEW, that is the account is disabled.

A user is a person that is allowed to use OEW. Use this dialog to manage an account for that person.

see also:

- User management
- Quickinfo

Edit users

This dialog displays the list of users known to OEW. You may add new, edit or delete user accounts.
see also:

- User management
- Quickinfo

Edit view

This dialog is used to edit the attributes of a view.

Specify a name and a comment.

Classes that are added (later) to your objectbase (manual or through import) can automatically be added to this view. Check the box 'automatically add new classes' to request this behavior. Otherwise, the set of classes for this view will not be expanded automatically.

The two lists at the dialog's bottom show which classes belong to the edited view (on the right) and all the other classes that don't (on the left).

Move classes from one list to another

- use drag and drop to move selected classes from one list to another.

or

- use the 'buttons with arrows'. For each direction there are two buttons, the arrow direction shows the direction items will be moved into.
  To move all classes from one list to the other, select the button with two arrows, respectively. To move only the classes selected, press the button with one arrow, respectively.

see also

The View Concept

Quickinfo

Edit views

This dialog displays a list of all available views.

You may add new, edit or delete views.

The list is capable of drag & drop operations. You may drag one view and drop it on 'inheritance windows', 'relationship windows' or 'list of classes windows'. The view will become selected as the current view in the drop window.

Use the 'search' button to find all views a specifc class belongs to.

see also

- The View Concept
- Quickinfo

Editor - Confirm replacement

The text (or regular expression) you are searching for was found. The occurrence is highlighted in the editor behind the dialog.

Use the 'Yes' button to replace the occurrence with the text you specified.The 'No' button skips this occurrence.

The 'Confirmation' setting controls how OEW will behave if the current search is continued. Check it, and OEW will keep asking. Uncheck it to make OEW replace automatically.

see also

- Quickinfo

Editor - Regular expressions

A regular expression is a string that contains wild-card characters. In the OEW editor you may use regular expressions to find text that matches some rules. Here is a list of rules the editor is capable of:

Use	to match
^word	word at the beginning of a line.
word$	word at the end of a line.
x[abc]y	xay, xby and xcy.
x[0-9]y	x0y, x1y, etc.
. (the period)	any char
xa*y	xy, xay, xaay, etc. The asterisk means "find the preceding char any number of times, even zero".
+	same as *, but at least one char. xy will not be found.
@	same as *, but maximum match within a single line.
#	same as @, but at least one char

If you need to use one of the special characters listed above without its special functionality, you may use the escape character \ (backslash) as a prefix for it. For example, \@ matches @.

Editor - block save as

Use this dialog to define the file, where you want the selected text to be stored.

Editor - find

This dialog is used to find a string or a regular expression within the edited text.
see also:

- Regular expressions
- Quickinfo

Editor - font

Specify the font to be used in editor windows. Please note that all editors in OEW use the same font. Changing the font in one editor will change it in all editor windows.

Only fixed width fonts are allowed for editors.

Editor - goto line

Use this dialog to jump directly to a line. Enter the line number you want to jump to and press 'OK'.

see also

- Quickinfo

Editor - open

Use this dialog to open an editor window. You have two choices:
- select an existing file and press 'OK'
- press the 'Create' button to create a new file.

Editor - open file to insert

Please select the file you want to be inserted into your edited text.

Editor - replace

This dialog is used to replace strings or regular expressions within the edited text. The 'replace with' string is not capable of regular expressions.

Use the 'OK' button to start the replace process. Use the 'replace all' button, and the search will be continued automatically after each occurrence until the end of the text is reached.

see also:

- Regular expressions
- Quickinfo

Editor - save / save as

Use this dialog to define where you want the file to be stored.

Editor Options

If the functionality of OEW's built in editor isn't sufficient for you, you may configure an external editor. Everywhere a button 'Editor' is shown beside an editor-field, you may use your editor of choice.

Use this dialog to specify whether you want to use the internal or an external editor, and enter the command required to start it.

Every time you press a button with the label 'Editor', OEW will create a temporary file, will fill it with the current editfield contents and start the editor with the filename as parameter. But OEW doesn't watch whether the editor is closed or the file is modified. You must manually tell OEW to read in the modified file contents by pressing the button (meanwhile showing the label 'get!') again .

Editor window

An editor window is used to display and let the user change textual information, for example source code or comments. Use the keyboard keys to navigate through the text and type characters, as you would do in any text processing software.

Different occurrences of editor windows

Keyboard commands

Change the text display

Editor statusbar

Comment variables

Editor window - Change the text display

Syntax highlighting

The OEW editor is C++ syntax sensitive. Special elements of the C++ language are visualized with different colors. As some editors are used to edit sources, syntax highlighting is enabled for them. Other editors that are used to comments have syntax highlighting disabled by default. For your convenience, you may toggle this mode manually using the editor's option menu.

Font

You may change the font used in editor windows. Use the font command in the editor's option menu. Please note that all editors in OEW use the same font. Changing it in one window will change all other editor windows as well.
Only fixed width fonts are allowed for editors.

Editor window - Comment variables

Comment variables are symbols that are contained in comments. They will be replaced with the actual information at print time.

For example, there are symbols for 'name of current user', 'creation date of a class', etc.

All comment variables have in common that they start with @@. You don't need to remember all of them, OEW helps you with remembering them.

As OEW knows whether an editor is used to edit a comment or not, OEW appends the available comment variables to the editor's menu in cases when it is.

When activating a comment editor's popup menu, you will see a submenu entry 'variable' at its bottom. Open it, and you will see a list of available variables for this comment. Select the variable, and it will be inserted into the editor at the current cursor position.

To replace the comment variables in the generated code you have to set the appropriate options in the 'Code Generator Options' dialog.

Editor window - Different occurrences

There are three possible kinds of editor windows used in OEW.

- Windows as part of the workbench's workspace.
  They are of the same kind as inheritance windows or slot windows, for example. You may resize them and change their position within the workspace.
  They are controlled with the workbench's menu at its top, and are listed in the workbench's window menu.
- As elements of dialogs. It is not possible to resize or move the position of these editors.
  They are not controlled by the workbench's menu, but have their own popup menu. To access it, click the right mouse button once while the mouse cursor is positioned over the editor.
  As these editor elements are sometimes very small, it is often necessary to enlarge them. Most editor elements have a button labeled 'Editor' placed near them. Press the button and an enlarged editor window will open.
- Enlarged editor windows are handled the same way as editor elements, but they are larger and are sizable and movable. An enlarged editor is associated with exactly one editor element in a dialog. By default, the enlarged editor will have the same size and position as the dialog containing its associated editor element.
  After you are finished editing, close the window.
  The enlarged editor is modeless, so you are allowed to switch to other windows or dialogs while the editor stays open.

Editor window - Editor statusbar

Each editor has its own statusbar. A horizontal scrollbar is integrated into it at its right hand side. At its left hand side you see four fields containing information on the current editor status. There contents, from left to right, are:

- Cursor position in the format Line:Column
- Insert or Overstrike. New characters are either inserted at the cursor position or replace the character at the cursor position.
- The 'modified' status. While this field is empty, either you haven't changed the field, or you have saved it. An asterisk (*) in it shows a 'modified' or 'unsaved' state.
- First character of two character keyboard combination.

Editor window - Keyboard commands

The editor understands most keyboard commands of the classic Wordstar keyboard command set. Here is a list of keys understood:

Key	Function
F11	Open the dialog for the symbol at cursor position
Arrow keys	Cursor movement
Ins	Toggles overwrite mode
Del	Edit/Delete
Backspace	Deletes character left of cursor
Home	Beginning of line
End	End of line
PgUp	One screen up
PgDn	One screen down
Ctrl-Home	Beginning of text
Ctrl-End	End of text
Ctrl-Right	Word right
Ctrl-Left	Word left
Shift-Del	Edit/Cut
Shift-Ins	Edit/Paste
Ctrl-Del	Edit/Delete
Ctrl-Ins	Edit/Copy
Ctrl-PgUp	Top of screen
Ctrl-PgDn	Bottom of screen
Alt-PgDn	Scrolls one page up
Alt-PgUp	Scrolls one page down
Alt-Left	Scrolls screen 20 chars to the right
Alt-Right	Scrolls screen 20 chars to the left
Alt-Down	Scrolls screen one page down
Alt-Up	Scrolls screen one page up
Strg-Alt-PgUp	Scrolls to start of text
Strg-Alt-PgDn	Scrolls to end of text
Ctrl-W	Scrolls screen one line down
Ctrl-Z	Scrolls screen one line up
Ctrl-S	One char left
Ctrl-D	One char right
Ctrl-A	Word left
Ctrl-F	Word right
Ctrl-E	One line up
Ctrl-X	One line down
Ctrl-R	Page up
Ctrl-C	Page down

Ctrl-G	Deletes to end of line
Ctrl-H	Deletes char left of cursor
Ctrl-Y	Deletes cursor line
Ctrl-T	Deletes word under cursor
Ctrl-N	Enters new line behind cursor
Ctrl-V	Toggles overwrite mode
Ctrl-Q-D	Move to end of line
Ctrl-Q-E	First line of screen
Ctrl-Q-X	Last line of screen
Ctrl-Q-R	Start of text
Ctrl-Q-C	End of text
Ctrl-K-D	Activates menu
Ctrl-K-L	Selects cursor line
Ctrl-K-Y	Edit/Delete
Ctrl-K-V	Moves selection to cursor position
Ctrl-K-T	Selects word under cursor
Ctrl-K-B	Set start of block
Ctrl-K-K	Set end of block
Ctrl-L	Search/next
Ctrl-Q-A	Search/replace
Ctrl-Q-[	Search/matching parenthesis
Ctrl-Q-B	Move to start of selection
Ctrl-Q-K	Move to end of selection
Ctrl-K-L	Load block from file
Ctrl-K-S	Edit/Save selection
Ctrl-K-H	Toggles display of selection
Ctrl-K-0..9	Sets bookmark 0..9 to cursor position
Ctrl-Q-0..9	Moves cursor to bookmark 0..9
Ctrl-K-U	Indent selected block one level
Ctrl-K-I	Unindent selected block one level
Tab	Indent selected block one level
Shift-Tab	Unindent selected block one level
Shift-Alt-Left	Indent selected block one level
Shift-Alt-Right-	Unindent selected block one level

Execute

This dialog allows you to start another program from inside OEW, for example your project or a debugger. Enter the command you wish to be executed. The output of the started Program is shown in the docking output window.

Please note that during execution of an external Process it is not possible to execute a command from OEW's menu.

see also

- Quickinfo

Explorer

The explorer gives you a different view to your objectbase. The window is separated into three areas:
- The context tree on the left side. In this tree you can select the object you want to edit.
- The editor window with the associated 'apply' and 'reset' buttons.
- The tab bar at the lower end of the window. It shows all opened contexts.

Context tree

With a click on the right mouse button you open a context menu for the clicked element.

Editor window

In this part the source code of the element selected in the context tree is displayed and can be modified. Changes will be accepted when you click the 'apply' button or when you select a different object in the context tree.

Source code will be displayed only for objects which have an implementation and an associated module. For example, if you select a dataslot there will be source code only if the slot is a static dataslot. Otherwise you have to select the class to edit the slot declaration.

If an object has no module assigned, the will be no source code, too. In this case, a message in the status line appears. To edit such an object in the explorer you have to assign a module to it. You can get to the appropriate dialog very quickty by using the context menus in the context tree.

To reduce the probability of errors, the changed source code will be checkt for misplaced parentheses before the changes are applied. In case of an error OEW will display a message.

Tab bar

The tab bar shows all opened contexts. You can jump to a context by clicking the tab or by pressing the 'Alt' key together with the associated number.

Contexts

There are three kinds of contexts: LRU, 'normal' context and query context. You can open many contexts at the same time. The advatage of this is that you can edit multiple objects at the same time without having to apply the change every time you want to see another object.

LRU

The 'LRU' ('last recently used') contains an entry for each newly opened context. This enables You to re-open a closed context (e.g. for a specific class).

Query context

Selecting the 'query' item in the context ment opens a query dialog. The result of these queries are displayed in query contexts.

Explorer popup menu

Depending on the type of context and clicked object the menu contains some of the following items:

Expand

Shows all children of the clicked object

Collapse

Hides all children of the clicked object

Collapse parent

Shows all objects on the same level as the clicked object

Context new

Opens a new context for the whole object base.

Context new from here

Opens a new context with the clicked object as root. Using that feature You can open contexts for special classes or slots if you want to access these objects often.

Context close

Closes the current context.

Query

Opens the query dialog

Slot filter

Here you can select a filter for the displayed slots.

sieh auch

Explorer

Explorer query

In this dialog You can specify what and where you want to search and where the result should be displayed.

If you start the query from a query context you can scpecify whether the new result should overwrite the old result in that query context or if a new query context is to be opened.

sieh auch

Explorer

Find and replace

Use this dialog to globally find text contained anywhere within your objectbase. Additionally you may replace the found occurrences with a new string.

Use the controls in the dialog's lower left corner to specify how the find and replace process will behave:
- if you don't want OEW to replace, check the 'find only' control.
- to make OEW ask you whether you want to replace an occurrence or not, check 'query replace'.
- check 'match case' for a case sensitive search
- check 'find whole words' to find no substrings, but whole words only

Use the controls in the dialog's lower right corner to specify where to search. You may select to search in whole modules, or in classes and their elements only. The displayed list will be filled with all elements of the selected type. Manually select the objects you want to search in, or select all of them using the 'All' button.

After you have set the options, use the 'Start' button to start the search.

see also

Quickinfo

Find class in views

Enter the name of a class you are searching for. After you press OK, OEW will look in all views for the entered name, and select all the views where it is contained.

If no views will be selected, no view contains a class with the entered name.

Found

This dialog is shown during the search and replace process. This dialog displays the context of the object displayed in the lower area.The found occurrence is marked and may be edited.

Press 'next' to continue searching without replacing.

Press 'replace' to replace and continue. You may mark a different area to replace first.

Press 'replace all' to stop OEW from showing this dialog and to replace all occurrences.

You may cancel at any time. In case you have selected 'find only', after you cancel a dialog will open that allows to edit the last found context.

see also

Quickinfo

General concepts

Generate Documentation

This dialog is used to control the documentation's appearance.

The options in this dialog are:

- Output Mode
- Output Target
- Classes (extent of documentation)
- Pages (ranges and labels)

Depending on the Output Mode, not all options are available.

The button 'Print' starts the generation.

'Preview' will show a preview of the printout on the screen. Please note that preview is only available for output Mode 'Using Tables' and only if there is at least one printer installed in your system.

see also

- Quickinfo

Generate Documentation - Classes

This setting controls which part of the classes, that are currently contained in the objectbase, will be contained in the output.

The choice 'All' is always available. 'In View' is only available if there is currently a view selected in the active window, and 'Selected' is only available while there are classes selected in the graph.

Note: The slots' order within the classes depends on the selected 'Slot Filter'.

Generate Documentation - HTML Files

OEW doesn't create the Hypertext system as a single file, but creates one file for each page. The start page will have the filename that you choose. All the other files simply get running numerical filenames.

To view the files, an appropriate application is required. Such applications are used to view the Internet's World Wide Web pages. Examples are Netscape Navigator and Mosaic.

Note: Netscape Navigator and Mosaic are protected trademarks of other companies than Innovative Software.

Generate Documentation - Hypertext

Hypertext is the right choice for the Output Mode, if you want to use the documentation as a quick reference. Two advantages are a clear structure and quick navigation to the sought information.

Currently, OEW supports Hypertext generation as MS Windows Helpfiles and as HTML (Hypertext Markup Language), the Internet's Hypertext language.

- MS Windows Help files
- HTML files

You may add instructions to your comments to structure them or to add additional jumps. To learn how this works read Hypertext-Comments.

Note: MS Windows is a trademark of Microsoft Corporation.

Generate Documentation - MS Windows Helpfiles

OEW is not able to create help files directly, because they must exist in a special compiled format. Therefore Microsoft's help compiler must be used. It isn't contained in the OEW package, but usually shiped with every computer language product that creates MS Windows applications. If you aren't sure whether it is available on your computer, look for files name HC, HCP or HC31.EXE on your harddisk.

What OEW creates isn't a help file, but a help project that must be compiled with the mentioned compiler. The project consists of three files with different file extensions: HPJ (for Help Project File), RTF (contains the textual information that will be displayed in the help file) and BAT (a batch command file that compiles the project).

Try to start the batch file created by OEW (extension BAT), best from an MS-DOS session, and watch the screen output. In most cases this will work, i.e. the help compiler will be found and the project compiled. If not, you may have to edit the batch file and arrange availability of the help compiler.

Generate Documentation - Manual Style

This creates a template for a manual. The template may be futher processed with any word processing software that is able to read RTF files.

The output is provided with appropriate heading styles. Assuming it is supported by your word processing software, these styles will enable it to automatically generate the table of contents for the manual.

Generate Documentation - Output Mode

The following output modes are available:

- Using Tables
- Manual Style
- Hypertext

Generate Documentation - Output Target

Depending on the output mode the output must be sent to a file or may also be sent to a printer. The dialog that is always shown gives only those options that are possible.

Select a target for the output if there is more than one available (i.e. printer or file). Note: If you change other settings in the dialog, this may disallow the currently selected output target and will change the target to an allowed one.

For each possible output target there are perhaps further options.

Output Target Printer

There are three text labels shown, each of them having a button on its right showing the text 'Select'. The label shows the setting setting and using the button it is possible to change it. The three options are the printer itself, the font that is to be used and the page margins that should be used on the paper.

Output Target File

Enter the filename for the output file into the field, with or without path, or use the button 'Browse'.

If you don't enter a file extension, OEW will choose one for you (according to the file type that will be created), and perhaps changes it while there are other options changed in the dialog.

In some cases you are also allowed to control the file format.

Generate Documentation - Pages

This area is availably only when output goes to a printer.

Do you want the documentation that you are generating to be part of another document? In this case you perhaps wish that all pages are numbered continously, including the pages generated by OEW. To adjust the page numbers for this purpose, enter the required start page number in the field 'Number from'.

Furthermore it is possible to print only selected pages. This is useful if you have a large printout and some pages didn't work, maybe due to printer problems. Therefore do not select 'All' but 'Range' instead, and enter the required pages or page ranges into the field.

Separate the different page numbers or ranges using a semicolon (;). A range is entered for example like this: 5-19

Generate Documentation - Table Contents

This controls what information will be contained in the documentation, e.g. whether statistics or module sources will be contained.

Near the label 'Contents' there is the button 'Edit' und below a list (that may be opened) with sets of 'Contents' settings.

The list always contains a set named 'Standard', a predefined set that uses the complete information. This can be schanged and it is further possible to define addtional sets.

Using the button 'Edit' you can manage and edit the contents. Select the button (point and click using the mouse, or press the spacebar while the button is selected) to open a menu. In it you will find the commands 'New', 'Edit' and 'Delete'.

'Edit' jumps directly to the dialog 'Documentation Contents', where the currently selected set may be edited.

Using 'New' a new set of contents will be created that you first have to name. Afterwards you may define the contents.

The menu entry 'Delete' deletes the currently selected set of contents. An exception is the set 'Standard' that can't be deleted, but may also be reset to the initial state.

Generate Documentation - Table Layout

The layout option allowsyou to fully configure the properties of all tables contained in the documentation, e.g. which columns are to be used and their widths. It is possible to manage multiple complete sets of those settings, i.e. multiple documentation layouts.

The required controls are located in the dialog's left hand area, below the label 'Using Tables'. The button 'edit' is lcated near the label 'layout' and below a list with names of layout styles.

This list always contains at least the entries 'Analysis', 'Design' and 'Implementation'. These are predefined layout styles for the three typical phases of software development. The layout styles may be modified and it is possible to define additional ones.

Using the button 'Edit' you can manage and edit the layouts. Push this button (click and hold with the mouse, or press the spacebar while the button is seelcted) to open a menu. It contains the entries 'New', 'Edit' and 'Delete'.

The command 'Edit' directly opens the dialog 'Configure Documentation' where the layout itself may be modified.

Using the menu entry 'New' you create a new layout that you first have to name. Furthermore a new layout isn't empty. It copies the settings of an already existing layout of choice. Afterwards you may modify the layout.

The command 'Delete' deletes the layout that is currently selected in the list, unless the selected entry is one of the three predefined styles. Those styles won't be deleted, but reset to their initial state.

Generate Documentation - Using Tables

This means an output mode that shows nearly all the documentation information in a clear table appearance. The shape and contents are, within certain limits, fully configurable.

- Table's shape (layout)
- Table's contents

see also: Documentation output to a RTF-File.

Global filenames

Use this dialog to specify the default path and extension of new modules used in your current objectbase.

'Path' : The paths where OEW will store new created modules by default. They are relative to the path where your objectbase is stored.

'Extension' is the default filename extensions that OEW uses when it automatically creates new modules.

see also

Quickinfo

Glossary

- A -

- abstract datatype
- ancestor class
- attribute

- B -

- base class

- C -

- C++
- C++ reference
- C++ sourcefiles
- CASE
- calling parameter
- cardinality
- child class
- class hierarchy
- class
- constructor
- container class

- D -

- datatype
- data member
- derived class
- descendant class
- destructor
- drag and drop

- E -

- encapsulation
- enumeration

- F -

- focus (OEW term)

- G -

- global

- I -

- inheritance
- inheritance diagram
- initial value
- instance

- L -

- W -

Glossary - C++

The language OEW is based on C++ is a hybrid programming language.

C++ combines the concepts of encapsulation, inheritance, and polymorphism and abstract datatypes with functional programming. It is an object-oriented programming language.

Things possess attributes and exhibit behavior. C++ extends the notion of a structure to contain functions as well as data members, and so it can model the way an object's complete identity is expressed through a single language construct, the class.

C++ is the object-oriented successor to C, developed at AT&T Bell Labs. It provides support for the three key defining attributes of an object-oriented language: data abstraction, inheritance and polymorphism.

OEW/C++ supports C++ for implementing object-oriented applications.

Glossary - C++ reference

On the C++ level, there are two types of references allowed. OEW defines the term 'C++ reference' to distinguish between both types.

- standard reference, using a pointer, such as:

```
int *a;
```

- C++ reference, such as:

```
int &a;
```

Glossary - C++ source files

OEW reads and writes the standard source code files. These are files containing all the declarations and definitions programmed in C++.

In OEW terms, these source code files are called Modules.

There are two basic types of modules:

- header files
  These files typically have extensions of ".h" or ".hpp" or ".hxx"
- source files
  These files typically have extensions of ".cpp" or ".cxx". The extension ".c" is commonly reserved for source code files containing only statements in the C programming language (C is the ancestor of C++, a subset).

You read (import) these files through 'importing source code'.

OEW writes these files through 'generating or updating source code'.

Glossary - CASE

Computer Aided Software Engineering

Glossary - abstract datatype

An abstract datatype or class contains at least one pure virtual function (for which no code is implemented in the class definition). Objects of an abstract class cannot be instantiated. The slots of an abstract class can be inherited by other classes. This capability is essential to polymorphism, because in the derived classes the code for the virtual function can be supplied in any way that is required by the object calling the function.

An abstract class must have at least one child class to supply the missing function definition, otherwise a compiler error results. Concrete classes can have abstract classes derived from them, but these in turn must have other concrete subclasses.

Glossary - ancestor class

Synonym for base class or superclass. A type from which an object type inherits. The type named in an object type definition statement is called the immediate ancestor.

Glossary - attribute

The attributes of an object describe its state. Attributes can be recognised in the descriptions of real world objects. Data members store the attributes of an object. Member functions define an object's behavior. In OEW/C++, data members and member functions are generally referred to as slots. This is intended to express that slot contents may be data or methods.

Glossary - base class

A base class is the class from which a derived class inherits its attributes and behavior. Its slots (data members and member functions) are inherited by the derived class.

Glossary - calling parameter

Parameters in function declarations describe the structure of messages being passed to and from objects.

Glossary - cardinality

Cardinality defines the extent of a relationship, that is how often a value may be contained within a slot or a relationship. There is a difference between minimum and maximum cardinality. Min. defines how often the value is contained at least and can be 0 or 1. Max. defines the highest allowed number of objects that may be contained and may be 1 or 'many'. Max. cardinalities other an 1 are defined often using container classes.

Glossary - child class

Synonym for derived class.

Glossary - class

A class is the definition of the structure and characteristics of a number of similar objects. The idea of a class is closely connected with the process of classification of individuals of the same type. A class can inherit characteristics from another class in a class hierarchy. Classes can be superclasses or subclasses depending on from where you view the inheritance hierarchy.

A class is the definition of an object type from which an instance is created. A class describes both the instance variables (or data members) and methods (or member functions) for manipulating an object.

Glossary - class hierarchy

All classes within an object oriented system are arranged in an inheritance structure which is arranged in a hierarchy. The sum of all classes and their related inheritances is therefore called a class hierarchy. In the graphical representation this hierarchy appears as a tree (with single inheritances only) or as a network (using multiple inheritance).

Glossary - constructor

A constructor is a member function that has the same name as its class. There can be more than one constructor in a class. Constructors are called to create an instance of their class.

Glossary - container class

A container class defines objects as collections of other objects. Most third generation language data structures are implemented as container classes in an object-oriented environment. All types of list, tree, stack and queue objects appear in container classes, since the elements of each of these data structures are also objects in their own right.

In OEW a container class may be assigned a cardinality that can be used as maximum cardinality for a slot or a relationship. OEW then generated source code that uses this container class.

Those classes are required to be either typeless (storing a void pointer) or a template class with exactly one parameter that specifies the type of the managed object.

Glossary - data member

A data member is a variable that is declared within the scope of a class definition. By default, all members of a class are private to that class, but they may be explicitly declared to be public and thus directly accessible to other parts of the program.

Glossary - datatype

OEW/C++ allows user-defined datatypes. These may be either defined globally or locally to a class.

OEW/C++ differs between two kinds of datatypes, the enumeration and the typedef, both of which can be assigned to a library.

Code for datatypes in a library is not generated.

Glossary - derived class

A class which inherits some or all of its attributes (slots) from another class is derived from that class.

Glossary - descendant class

All classes following from a superclass are descendant classes. There can be more than one step in the class hierarchy between a superclass and a descendant class.

Glossary - destructor

A destructor is the member function of a class that has the same name as the class preceded with a tilde (~) sign. If a class has a destructor function, it is always called when an instance of its class is deleted.

Glossary - drag and drop

Drag and drop means that you may move or copy items using your mouse. Many windows in OEW support drag and drop.

To drag and drop: Move your mouse cursor over the object, press and hold the left mouse button, 'drag' the object, move the mouse cursor over the destination of choice and 'drop' the object, releasing the left mouse button.

While 'dragging', the mouse cursor changes its shape. (If the mouse cursor doesn't change its shape, the object doesn't support drag and drop).

While you move a dragged object over the screen, the shape of the mouse cursor may change again, depending on the window currently under the cursor. While your mouse is over a window that can't be used as a destination for the object you are dragging, its shape will change to a stop sign. A picture of a paper depicts a 'move' operation, a picture of a paper containing a plus '+' sign depicts a 'copy' operation.

Glossary - encapsulation

Encapsulation places all the data and functionality of a datatype in a controlled and abstract structure. Classes are designed to give the programmer the freedom to encapsulate data members or instance variables and member functions or methods within a single structure.

Glossary - enumeration

An enumeration type is a datatype. It declares a set of constants.
Enumerations allow slots to take one of a series of predetermined constants.

Glossary - focus (OEW term)

When the cursor moves over a class or instance in the main window, the class or instance comes into focus. The focus is shown by a shadow around two edges (usually the right side and the bottom) of the class icon or the instance icon that is currently in focus.

Glossary - global

An object is 'global' if it is not locally defined within a class.

Glossary - inheritance

Inheritance in object-orientated systems is the process by which the attributes (data members), and behavior (methods or function members) of one or more classes are available for new classes to inherit. Characteristics which are common to several classes can then be defined in one class in the class hierarchy and inherited by its subclasses.

Glossary - inheritance diagram

The inheritance diagram is the graphic display of the class hierarchy in the main window of OEW/C++. It is used to modify the objectbase and the inheritance hierarchy of its classes and instances.

Inheritance diagram is an OEW term.

Glossary - initial value

The initial value is a default value set for a data member or instance variable when an instance of a class is created.

Glossary - instance

Instances are individual, discrete occurrences of objects in a class. Both the behavior and the structure of an instance are defined in the class definition, however, no memory is allocated to an object until it is declared (as an instance of its class). In the classes of the class hierarchy, the structure and the behavior of a number of individual instances is defined.

Instances are created by the compiler at program run time. Exceptions are static instances.

Static instances are defined in the program. In these static instances, the parameters of the constructor that instantiates an instance of the class of which the constructor is a function are defined.

Instances can be persistent.

In the OEW/C++ drawing of the class hierarchy, instances are displayed on the diagram in red, and are connected to their class with a broken line.

Glossary - library class

A library class is a class that isn't defined or programmed in your current objectbase, but only (re)used. As a result, there is no need to generate source code for those classes, as they are usually already compiled and linked to your application through static or dynamic libraries.

OEW has the ability to mark classes with a library flag. OEW will not generate source code for those classes.

In the graphic display of the class hierarchy, the library classes are shown with a capital 'L' on the right hand side of the box depicting the class.

Glossary - lifecycle

Projects are organized into phases that concentrate on different levels of development. The phases of development in a project are analysis, design, implementation and maintenance. Together they are parts of the development lifecycle.

Because a common way of thinking, referred to as 'object-oriented', is used in each phase in the modeling and implementation of interacting software objects in an application, seamless movement between the phases of the lifecycle is possible.

For the management and documentation of a project, OEW/C++ provides the database of who created objects and when, and who last changed an object and when.

Glossary - local

An Object that is member of a class is called a 'local' object in OEW.
For example 'local datatype' or 'local class'.
Local is a synonym for 'nested'.

Glossary - main program

C++ uses a similar convention to C, with execution beginning at function main(); it also uses the same argc and argv[] conventions as C to pass arguments to the program at runtime.

To define a global function in OEW create a userdefined entry in a module.

Glossary - member

Member is a term for either:
1. attributes and functions that are members of a class
2. objects that are instances or members of a class

Classes have both data members and function members. OEW/C++ uses the term slot to refer to both data and function members.

Members of a derived class are also members of the base class, but not vice versa.

Data members and member functions are only accessible within the context of the class with which they are associated. The compiler wants to know the instance or instances of a class you want to access or alter, and can determine whether such access is allowed. The class declaration encapsulates members, hiding information from unauthorized change. By default, all members of a class are private to that class, but they may be explicitly declared to be public and thus directly accessible to other parts of the program.

Glossary - member function

A slot (member) of a class of type method is also called a member function.

By default, all members of a class are private to that class, but they may be explicitly declared to be public. This makes them directly accessible to the member functions of other objects that wish to pass messages to objects in that class. In this way the interface between an object and other objects with which it interacts is defined with a set of public member functions.

Glossary - message passing

In object-oriented systems methods or member functions in classes are called by passing messages between objects. One object sends a message to another object asking it to perform some work. The receiving object has methods that react to the message and returns some information.

Object-oriented systems are often described as systems where processing happens exclusively through the passing of messages. Message passing is a similar concept to parameter passing in functional programming.

Message is the term used for a method when it is applied to an object. Methods define the messages which can be sent to a class of objects. In most ways, calling a message is similar to a standard function or procedure call.

Glossary - method

Methods are procedures or functions that are declared and defined in a class statement. They are used to pass messages between objects.

Method is a synonym for member function.

Glossary - module

A C++ project is divided into modules, the standard C++ text files. OEW maintains a list of all the modules that belong to a project. When importing C++ files or creating new classes, new modules are created automatically.

Each module contains a list of elements, e.g. class definitions, methods, include statements. Each list is updated automatically by OEW as you change, delete or create new objects. You are allowed to change the order of elements in a module manually. It is also possible to move elements from one module to another.

Module properties

Module elements

Types of module elements

Special modules:

Default module

> The default module specifies the elements that will automatically be entered into new modules as they are created. Changes to the default module are not reflected in existing modules. It is not possible to delete this module.

Global types

> All global datatypes will automatically be placed into this module.

Glossary - nested

An Object that is member of a class is called a nested object. 'Nested' is a synonym for 'local'. For example 'nested datatype' or 'nested class'.

Glossary - object

Objects are instances of a class. The class contains the definition of the object. An object is a member of its class. In object-oriented systems, an object contains the data that describes it and the functions that it performs. A model of objects describes their structure and behavior.

For each object, the class defines the attributes (data members) of the object, and the methods (member functions) that determine its behavior, as well as information about its connections to other objects. When an object is declared, memory is allocated and its constructor function is called to initialize its data members.

Glossary - object menu

An Object menu is a popup menu that belongs to a special object. Not every window or object displayed has an object menu associated with it.

To see if it does, just move the mouse pointer over the object and click once with the right mouse button. In general, you access the object menu in the same way.

In windows or lists that have a focus you may access a popup menu with the spacebar.

Glossary - object-oriented analysis

In analysis it is necessary to find objects that exist in the application, and identify what they do, how they do it and how they interact with other objects. In OEW/C++ classes are defined to represent these objects.

Glossary - object-oriented design

In OEW/C++ design is the further specification of slots (attributes, methods and/or datatypes). This is done in the class dialog box for the class being designed. The design steps add details that are important for the implementation of the application using the object-oriented features of C++.

Glossary - object-oriented programming

To design and implement objects in software, it is helpful to use a programming language that supports the object-oriented concepts of encapsulation, inheritance and polymorphism.

C++ is an object-oriented language.

Glossary - object-oriented systems

The data structures and functions that describe an object and how it behaves are described in the class declaration. Functions and data structures are inherited by subclasses. Individual objects are specific instances of a class, created from the 'framework' defined by the class and its ancestors in the class hierarchy.

Object-oriented software emulates real world objects by creating objects in software, using the three concepts of encapsulation, inheritance and polymorphism. These concepts are supported by object-oriented programming languages, data bases and computer-aided software engineering tools.

OEW/C++ is an object-oriented software engineering tool that supports the phases of analysis, design, implementation and maintenance in the development lifecycle.

C++ supports object-oriented programming language concepts.

Glossary - objectbase (OEW term)

OEW stores all project data in a single file called the Objectbase File. Files of this type have the suffix of ".oew". (or '.nnn', where each n is a digit. OEW creates a backup file each time you save, incrementing 'nnn' by one.)

The objectbase stores information about classes, their attributes and methods, and the relationships between them. Information about instances and datatypes is also stored. Comments and descriptions documenting these objects are also stored in the objectbase, along with information about who has created and changed them.

Glossary - parent class

The class which is an immediate superclass of another class which inherits all of the parent's slots.

Glossary - persistence

Persistent objects maintain their current state when a program is not running. Persistence means that the objects are filed automatically with their structure and their current state when the program is ended or aborted.

Glossary - phases of modeling

Software is created in phases. Each phase specifies in more detail what the application is supposed to do and how it will do it. Traditionally analysis is followed by design and then programming. In an object-oriented approach it is possible to move freely between the phases of development.

Glossary - pointer

Pointers are variables containing addresses of other variables, including objects and structures. Pointers can be used to build dynamic data structures with which one can create arrays of objects of indefinite length. If pointers reference other objects and if the objects are supposed to be persistent, the problem of integrity arises.

Glossary - polymorphism

Polymorphism is the ability of a member function to modify its implementation based on the specific class of the object it is called by. When a class is created, it can replace the implementation of member functions it inherits from its base class or classes. The C++ concept 'virtual function' to implement this behavior.

Glossary - re-use

Re-use is one of the goals of software development. When software can be used like ready made building blocks, the development of software takes on the qualities of engineering and building processes rather than the craft process in which every development is started from scratch each time.

Object-oriented programming languages such as C++ that support encapsulation, inheritance and polymorphism make it possible to construct software that has a high degree of re-usability.

As a software engineering tool for C++, OEW/C++ provides direct support to the developer working with an object-oriented approach to software development.

Glossary - reference

The semantic relationships between objects are modeled with references. A reference is a relation between an attribute in one class and instances of another class. Instances of the class being referenced are the only values allowed.

Glossary - root class

A root class is always at the top of a 'tree' in the class hierarchy. A root class has no superclasses. In one objectbase, more than one root class can be created.

Glossary - slot (OEW term)

Slot is a general term for the attributes, methods and datatypes declared and defined in classes.

Slots can be displayed in the class icons in the main window, or viewed in the list box in the class dialog box.

Slot is an OEW term.

Glossary - state

Each instance of a class has a current state. The state of an instance is defined by the contents of its data members. The state of an instance can only be changed by accessing the instance through the public member functions of its class. If the instance is persistent, its state can be saved.

Glossary - static

A declaration modifier that, when applied to global variables and functions, means that the variables are not accessible outside of the file in which they are declared; when applied to local variables, it means that they continue to exist even after the procedure in which they are declared has exited; and, when applied to class declaration fields, it indicates that the fields are shared by all instances of the class.

Glossary - subclass

Subclasses are classes that are immediately below their respective superclasses in the inheritance hierarchy. Subclasses inherit the attributes and behavior of their superclasses. A subclass is also a child class and a derived class of its superclasses.

Glossary - superclass

Superclasses are classes that are immediately above their respective subclasses in the inheritance hierarchy. Superclasses pass their attributes and behavior on to their subclasses. A superclass is also a parent class and an immediate ancestor class of its subclasses.

Glossary - template

Code for some classes is the same regardless of the type of objects in the class - the only thing that changes is the type of the objects in the class (for example, a List class). Such a class can be declared a template class. Put another way: when you are able to write each method for a class without knowing anything about the type of objects in the class, then the behavior of objects in the class does not depend on their type, and such classes are ideal candidates for being declared as template classes.

Templates are the C++ mechanism for parameterizing a class or function on the basis of type. Another term for template classes is parameterized classes.

A template is written type-independent but is type-safe in use. For example: if you create a List class with objects of a type that is not yet known when the template is used for objects of a specific "known" type, then it is type-safe.

Templates can be used to specify functions as well as classes. They can take more than one parameter, and they can take non-type parameters.

Template is a reserved word.

Glossary - title bar

A title bar is the topmost bar of a window. In OEW, it displays the window's type and the containing objectbase's name.

Windows, where a view is activated, additionally contain the view's name in brackets.

Glossary - typedef

A typedef name is a name given to a type via a type-name definition introduced by the keyword typedef. A type-name definition allows a name to be given to a type that would otherwise be represented by a combination of symbols.

Typedefs declare aliases that are more meaningful names for function and variable declarations.

Glossary - view (OEW term)

To manage a large number of classes, it is convenient to divide the objectbase into smaller subsets of an objectbase. Those subsets are called views. A view stores the set of classes it contains.

You may select views in inheritance and in relationship windows, limiting the displayed classes to those contained in the selected view. A view automatically stores two layouts (a submodel), one for an inheritance window, one for a relationship window.

Each window can display only one view at a time, but you may have opened multiple inheritance windows (or relationship windows) for the same objectbase, each displaying a different view.

The display speed in inheritance and relationship windows depends on the number of visible classes. Therefore, it is recommended to use views, escpecially with large objectbases.

Glossary - virtual function

Virtual functions are functions which are declared in a superclass, inherited by and redefined in a number of subclasses. They are used to implement polymorphic behavior, where objects from different classes respond to the same message in different ways.

When a function is declared as virtual then its definition, that is, its behavior, depends on the class of the object pointed to or referenced in the call to the virtual function. This makes it possible for objects to respond to messages depending on the class of the object pointed to or referred to, as this determines which function definition will be used. This means objects from different classes can respond to the same message in different ways. This is what is referred to as polymorphism (from Greek, meaning: many shapes) where the same function behaves in a variety of ways.

A virtual function is called by a pointer or reference. It is a requirement that a virtual function's declaration in derived classes must precisely match its declaration in a base class.

For non-virtual functions, the type of the pointer or reference rather than the class of the target object determines which definition is used.

Glossary - workbench

In Computer terms, a workbench is a kind of software that you use to 'work'. By using it, you can produce something. The 'bench' is the application's main window that contains the objects you are working on.

Glossary - workspace

The workspace is the area between OEW's menu and its lower border, except for the toolbar and statusbar (if active and visible).

How to ...

How to ... - Frequently asked questions and their answers

How do I make a class?
- click on Edit / New class or
  press "+" button on the toolbar or
  press <crtl> + N on the keyboard.

How do I move a class?
- hold down the <Ctrl>+ right mouse button and drag.

How do I make a member i.e. a slot?
- doubleclick on a class
- click on the white slot list with the right mouse button
- from the menu select "New Slot"

How do I make a member function (method)?
- double-click on a class
- click on the white slot list with the right mouse button
- from the menu select "New Slot"
- choose "Method" as the slot-type
- press "More" to enter the design and programming dialog

How do I define an inheritance relationship?
- drag a line with the left mouse button pressed from one class to another.
- release the button.

How can I remove an inheritance relationship?
- drag a line with the left mouse button pressed from the superclass to the subclass,
- release the button.

How do I make other kinds of relationships?
- press the relationship window button of the toolbar to call up the Object Relational Modular.
- The first thing you want to do is call the View/Reformat/Class Hierarchy Menu to make the graph a bit prettier.
- drag a line (left mouse button) to connect two classes and an 'Edit Relation' dialog pops up.
- choose the type of the relation (in both directions) and press OK.

How to get submodels?
- you can collect parts of the whole hierarchy in so-called Views.
- a view is a collection of classes and their relationships. Each class can appear in multiple views.
- to create a view click on the magnifying class icon on the toolbar and define views as described in the online help.

How to ... - Importing source code that uses class libraries

When importing source code that uses classes from class libraries, it is recommended not to import the library's files, because this will import too much unneeded data and will hinder OEW from operating quickly.

Import your project's source files only. When OEW asks for the include path, do not supply the directory name where your library's header files are located. For every class that is used in OEW, but whose definition can't be found, OEW automatically adds an empty library class to your objectbase.

If you insist on importing a whole class library, it is recommended to import those files in an extra step, using the 'import as library' option.

How to ... - Using classes from libraries

To use a library class in an objectbase, the library class doesn't have to contain its definitions, it may be empty. OEW uses only the name of a library class and the library module to generate correct source code. As OEW is not a compiler, it does not check statements within method source code for correctness, so it is ok for a library class to be empty. You should assign these classes the flag library class.

We do not recommend importing whole class libraries, even when working on projects that use classes contained in them. Just add new empty classes to your objectbase as you need them, and give them the flag library.

Hypertext Comments

In addition to the hyperlinks that are automatically created by OEW, it is possible to add userdefined jumps to your comments. This is achieved by entering control sequences into your comments that OEW will recognize and replace during generation.

Hypertext control sequences are of the form:

%type of hyperlink%jump target%text to be shown%

There are different types of hyperlinks. We distinguish between 'internal' and 'external' jumps.

Internal jumps

External jumps

Hypertext Comments - External jumps

If you want to supplement the generated Hypertext documentation you perhaps also want to define jumps between the generated text and your manually created text.

To achieve this you may enter control sequences into comments that OEW will replace during generation.

Depending on the type of hypertext, the system must be given different control sequences. MS Windows Help files require 'context strings' as jump targets, on the other hand HTML requires an 'URL'. This is the reason why there are two different control sequences for:

For MS Windows Help files:

```
%w%Context String (jump target)%Text to be shown%
```

For HTML:

```
%h%URL (jump target)%Text to be shown%
```

You must make sure that the jump targets are available. For MS Windows help files this means that you must add your own RTF files to the help project. To do this, edit the HPJ file.

Hypertext Comments - Internal jumps

Internal jumps are jumps to other pages within the set of pages generated by OEW. Maybe you wish that the comment for class 'a' refers to class 'b'. Therefore, add the following statement to the comment of class 'a':

```
%i%b%Reference to b%
```

The output will show the text 'Reference to b'. By clicking that text the system will jump to the page containing class 'b'.

This means that the hyperlink type for internal jumps is 'i'. It works for all kinds of hypertext created by OEW.

Import from objectbase

Use this dialog to specify which parts of the selected objectbase you want to import. The name of the objectbase you selected is displayed at the dialog's top.

You may select 'import as library' to give all imported classes the library flag.

In the list on the dialog's left hand side all classes, datatypes and container classes are displayed. Select the objects you want to import manually, or use the 'All' or 'None' buttons to select objects in the listbox.

Often the objects that you want to import, themselves use other objects that may be contained within the objectbase, too. Press the 'Used Objects' button, and OEW will examine each object you selected for those usages. The used objects will be added to the selection in the listbox.

As OEW doesn't allow different objects to have identical names. OEW prevents name conflicts by automatically renaming imported objects. Symbol names in the list box that are followed by an arrow and another symbol name show you the new name OEW would use.

After you're finished selecting objects, use the 'Import' button to start the import process.

see also

Quickinfo

Import from objectbase - file open

Select the objectbase you want to import from.

Important terms

Data OEW operates on
Objectbase
Modules
Views
User management
Database support

Inheritance window

An inheritance window is used to display and edit the inheritance structure between classes, and to access the displayed objects.

Window contents

Mouse and keyboard commands

Manage what and how it is displayed

Inheritance window - Add new class(es)

Double-click with the left mouse button in the window's background. A dialog will appear where you may specify the class(es) to add.

Inheritance window - Create, edit, remove inheritances

To inherit one class from another, drag a line from the baseclass to the new subclass. You do this by clicking the left mouse button over the base class, holding it pressed, moving the mouse to the new subclass (the area will scroll when you reach the window's border) and releasing the button over it. The properties for an inheritance defined this way is controlled in the 'default inheritance mode' dialog.

To change an inheritance double click the middle of the inheritance line. A dialog showing its properties will be displayed.

You may remove an inheritance by drawing the same line again. The inheritance line will be removed.

Inheritance window - Drag and drop

Copy / move objects

Drag and drop objects between different windows. Drop them into a rectangle to make them local, drop on the background to make them global.
Standard operation is move. Press and hold CTRL to copy.

Add / Remove classes to / from current view

While there is a view active, you may drag and drop class names from a 'list of classes window' or an 'Edit view dialog' to change the view contents.

Import source code / objectbase

Drag and drop C++ source files or objectbase files from the File Manager to import them.

Inheritance window - Edit slots

To access a class' slots window, either double-click on its rectangle or bring the class into focus and press the 'Enter' key on your keyboard.

Alternatively you can display the slots in the inheritance window directly. Therefore select 'Show slots' in the class' object menu.

Inheritance window - Focus

At any time, one class within the window is focused. Commands that operate on a single class will use the focused class. The focused class has a larger black border than other classes. The focus changes automatically as the mouse cursor moves over the window. The keyboard arrow keys move the focus one class into the respective direction.

Inheritance window - Manage what and how it is displayed

Multiple inheritance tree display
Show inheritance handles
Sort alphabetically
Slot filter
Font
Views
Slot display
Subclass' display

Inheritance window - Mouse and keyboard operations

Scrolling
Zoom
Focus
Add new class(es)
Select class(es)
Edit slots
Create, edit, remove inheritances
Rearrange the class order
Object menus
Drag and drop

Inheritance window - Object menus

Objects within inheritance windows that have object menus assigned:

- Class rectangles
- Inheritance handles
- The background

You may open the class and instance object menus with the spacebar key on your keyboard while the object is focused.

Inheritance window - Rearrange the class order

As new classes are added to your objectbase, they will be appended at the bottom of your inheritance window, with two exceptions:

- Sub- or superclasses of other classes. They will be placed around the related class.
- You have the option 'Sort alphabetically' in the view menu active.

Otherwise you are allowed to move classes or subtrees up and down within the same inheritance level.

Press and hold the 'CTRL' key on your keyboard, press and hold the right mouse button while the mouse cursor is over the class you want to move. Move the mouse up or down and watch the gray line, indicating the new position. Release the mouse button when it is at the position of your choice.

Inheritance window - Scrolling

Since the inheritance tree contained in a relationship window may be larger than the window itself, you are required to scroll the visible area to the region you are interested in. You may do so using the scrollbars at the window's right and bottom.

For a quicker positioning you can use the 'overview window'. To navigate quickly to a special class use the 'list of classes' window.

Inheritance window - Select class(es)

Some menu commands operate on the set of selected classes in the active window.

To select a class, click it once with the left mouse button. The class will be displayed like a pressed button.

To add further classes to the selection, press and hold the 'CTRL' key on your keyboard before clicking with the mouse.

Holding the 'Shift' key while clicking selects the class and all its subclasses.

To deselect all classes click in the background.

The 'Edit' menu contains a command 'Select All' to select all classes.

The set of selected classes in the inheritance window and in its associated 'list of classes' are identical.

Inheritance window - Slot display

For each class you may decide whether OEW displays its slots within the inheritance window or not. To enable slot display for the selected class(es), from the view menu choose

Show slots

To disable slot display for the selected class(es), from the view menu choose

Hide slots

Please note: Slot display is further controlled by the Slot filter.

Inheritance window - Subclass display

For each individual class in an inheritance window, you may toggle the display of its subclasses on or off.

To disable the display of subclasses for the selected class(es), from the view menu choose

Hide subclasses

To enable the display of subclasses for the selected class(es), from the view menu choose

Show subclasses

To enable the display of all subclasses, from the view menu choose

Show all subclasses

To distinguish whether a class has hidden subclasses, OEW displays three short lines at the right hand side of those rectangles.

Inheritance window - Views

What are views?
Edit view
Select view
Turn off view

Inheritance window - Window contents

Rectangles

Each rectangle represents either a class.

Rectangles may contain more than one line of text. The first always contains the class' name. Each other line shows a slot's declaration.

Base level classes are displayed at the window's left hand side. The more they are displayed at the right, the more base classes they have.

Lines

Inheritances are displayed as solid lines. Instances are connected to their type class by a dotted line. Lines may have a smaller rectangle displayed at their center, called inheritance handles. The line's style tells the inheritance mode:

solid: extends

dotted: implements

Gadgets within rectangles and their meaning

Red vertical line - an abstract class

P - a persistent class

L - a library class

Inheritance window - Zoom

At the lower edge of the inheritance window You will find two scrollbars. Use the left scrollbar to scroll the contents of the inheritance window horizontally. The right scrollbar can be used to determine the zoom factor. The zoomfactor can be set between 5% (left side) and 100% (right side) in steps of 1%.

Please note that not all fonts can be resized to arbitrary sizes. If you want to zoom a window, please select scalable fonts like truetype fonts.

Innovative Software GmbH
Kaiserstr. 65
60329 Frankfurt/M
Germany

Phone: +49 (0)69 236929
Fax: +49 (0)69 236930
BBS: +49 (0)69 236934
E-Mail: 100272.515@compuserve.com or oew@isg.de
WWW: http://www.isg.de

Inside OEW

OEW is easy to use although it is complex. Powerful features, e.g. the parser and the code generator, interact in an intuitive way. This ease of use has a slight drawback: there are no special menu commands to access some of the most powerful applications of OEW. This effectively obscures that OEW can accomplish these applications at all. The ability to handle multi-user projects, for example, is one of these features. The user administrator built into OEW is an indication that it must exist, yet how is it accessed?

The following text will cover some capabilities and applications of OEW not obvious at first glance:

- how to split a project among multiple developers
- how to assign access rights in a multi-developer project
- container classes and the corresponding code
- relationships and the corresponding code
- automatic conversion to members and references
- specifics of OEW's Symbolic Parser(TM)

As these are fairly advanced topics this text requires some experience with OEW. As an example, it does not explain which buttons have to be pressed to import a file from a library.

List of classes window

This window displays a list of classes, sorted alphabetically. By default, it lists only the classes that are currently displayed in its associated inheritance or relationship window.

It helps you to

- find classes quickly in its associated inheritance or relationship window.
  Click once on the class name in the list of classes, and its associated window will scroll the visible area to make the clicked class visible.
  Double-click a class to open its slots window.
- manage contents of views using the object menu or drag and drop.

The class' selection is identical with its associated (inheritance or relationship) window.

see also

The window it belongs to

List of classes window - The window it belongs to

Every 'list of classes' window is associated with exactly one inheritance or relationship window. As it is possible to open a 'list of classes' window only from these two window types, there is no need to tell OEW which window the list of classes belongs to. It is always the window that was active when a 'show list of classes' command was given.

Login

You must identify yourself before you are granted access to OEW.
Enter your name or select it from the list and enter your password.
You may close the OEW application using the 'Quit' button.

see also
Quickinfo

Long operation status

An operation that may need some time is in progress. Please wait until the operation is finished.

see also

- Quickinfo

Make options

Use this dialog to set the name of the java compiler you are using.

see also

Quickinfo

Module elements

Each module contains a list of its elements that are class definitions, methods or comments, for example. OEW preserves the element order of modules. The user is allowed to change this order manually.

Each element contained in an objectbase that has a source code equivalent is automatically assigned to exactly one module. It is not possible for elements to be contained in more than one module, or even in no module.

see also

Types of Module elements

Module move in

While editing a module, you told OEW to 'move in' module elements into it from another module. At the dialog's bottom the target module is displayed.

Use the controls 'from module' and 'elements' to specify the module elements. As you select a 'from module' (i.e. the source module), the elements list will be filled. Select the module elements of your choice.

You may choose to move the elements, meaning delete them from the source module, or to copy them only, leaving the source module unchanged.

Please note that declarations and implementations can be contained in only one module, and therefore may be moved but not copied.

see also

Quickinfo

Module move out

While editing a module, you selected one or more elements and told OEW to move them into another module. Use this dialog to specify where the elements should be moved to.

The source module is displayed, and you may select the target module from the list below.

see also

Quickinfo

Module properties

Each module has a module name and an actual file name. Each OEW generated module has a module comment assigned.

You can mark modules as library. Library modules are never generated during the source code generation.

see also

Quickinfo

New class

This dialog is used to add one or more new root class(es) to your objectbase.
Enter the names of the new classes. Use commas to separate the names.
For example, entering

```
Car,Wheel,Engine
```

will create three classes.
see also

Quickinfo

New objectbase

You want to create a new objectbase. Use this dialog to set its initial properties.

see also

- Objectbase templates
- Database support
- Inheritance window
- Relationship window
- Objectbase information
- Global filenames
- Default inheritance mode
- Default module
- Make file options
- Quickinfo

OO Analysis

OO Analysis - Analyzing Relationships

Lines in OEW's Inheritance windows represent a generalization. They correspond to the is_a relationships of an Entity-Relationship-Model (ERM) graph. However, an inheritance line also means slots (i.e. attributes and methods) are inherited from the superclass. In this way, a single slot definition can be used in many classes.

Member classes, representing attribute values of a complex type, implement the ERM's has_a relationships, also called whole_part relationships. The object model makes referential integrity a minor issue. Some class libraries or modern object-oriented databases automatically ensure referential integrity, using the has_a relationship. This is a great advantage in building models that rely heavily on this relationship, e.g. in part list administration. OEW allows you to graphically model data structures for the object-oriented databases ObjectStore and POET, automatically generating the database information required.

In contrast, the traditional way of building a complex, structured data model begins with the creation of a data model free of redundancies. Afterwards, this semantic data model has to be normalized in a complicated process to create a database structure. At run-time, the complex structure is re-integrated by a join. This results in slower programs in addition to the mental overhead we just described. Even worse, update and delete operations repeatedly cause problems in traditional database systems.

How are relationships other than is_a and has_a modeled in object-orientation? Flight Events, for example, calls for a "transports" relationship between the plane classes and their cargo. Considering inheritance, these relationships must be inherited, too. Only two transports relationships are needed, one between CargoFlightEvent and Cargo, the other between PassengerFlightEvent and Passenger. ComboFlightEvent automatically inherits both of these relationships so no additional ones are required here.

To create these relationships, an ERM with the features of object-orientation is needed. OEW's Object Relationship Modeler window enables you to graphically edit object relations. An Object Relationship Diagram only shows data relationships, not message passing (function calls).

OO Analysis - How do you analyze a project

How do you analyze a project? Start with an identification of terms pertaining to the problem. These terms must be precisely identified to ensure effective communication between experts in the problem domain and analysts. Each of the terms is entered as an OEW class or slot and described in the comment fields. The program's implementation should be disregarded at this time - a thorough understanding of the problem and a definition of terms understood by experts and developers is the aim. The result, a first list of terms concerning the project, will be thoroughly modified and revised during further stages of analysis and design.

Our example, the Flights project, serves to manage flight events of different types. Passengers and/or cargo items can be transported by a plane. The purpose of the application is to calculate the revenue from each flight event. From the users point of view, the program should be easy to use, furnishing a graphical user interface.

First of all, the important terms from this example project must be found.

- Flight event
- Freight
- Cargo
- Passenger
- User
- Revenue
- UserInterface

This is quite an arbitrary selection of terms, but it already shows the fundamentals of OOA. Objects are modeled, not the functions working with these objects. In comparison to SA, starting with a functional decomposition, this is a totally different approach.

The next step uses generalization to structure the model. The analyst tries to find terms belonging to the same superset. In a typical bottom-up approach this superset is inserted in OEW as a superclass. He decides that Freight is a superclass for both Cargo and Passenger.

Now the analyst determines whether classes should be specialized in a top-down approach. Discussions with the experts may also yield additional terms needed in the model. He finds subclasses of FlightEvent such as CargoFlightEvent, PassengerFlightEvent. ComboFlightEvent is a subclass of both CargoFlightEvent and PassengerFlightEvent, using multiple inheritance.

It is obvious that OOA combines top-down and bottom-up approaches. In fact, the two approaches are so thoroughly intermingled that these extremes should be discarded, labeling OOA an inside-out approach instead. A rapid change of contents is very typical for the beginning of an OOA.

OO Analysis - Non-automatic slots

Now new slots must be added to complete the analysis of structure and behavior of the classes. There is a rule-of-thumb for adding slots. Take a class at the very right of the hierarchy and think of a slot for this class. In Flights, you might think of a slot Name for PASSENGER. Before adding the slot to the class you have to check if the slot also characterizes the class's superclass. If so, you repeat this until you find the leftmost class that needs this slot. After entering the slot, you must determine if the slot is needed by all classes derived from this class. If this is not the case, the model must be revised. For a valid model each slot must be assigned to all classes of a distinct subhierarchy. Each slot should be defined only once and passed to other classes by inheritance (multiple inheritance included) or by the has_a relationship. Only a bad analysis has redundant slot definitions.

Follow this guideline for each slot you insert into the model. This might result in drastic changes. Both the analysts and the experts must understand these changes. After adding all important slots, remove classes without slots from the model.

When creating a class, an analyst associates certain properties with it. This results in another style of adding slots, starting on the left hand side of the class hierarchy. Again, the model is verified using the inheritance mechanism: each slot should be defined only once, and be valid in the entire subtree of its class.

OO Analysis - The Flights Example

OO Analysis - Using classes ... or slots?

Our modeling of the revenue gained from a flight event has changed. When starting OOA we made it a class, because it is one of the key terms of the problem. Now we have re-modeled it as a slot, Revenue, of the class FlightEvent. The question when to use a class or a slot instead is a very essential one in object-orientation and any other data structure modeling technique.

The answer to this question shows one of the great advantages of object-orientation: information hiding. Languages like C++ allow information hiding. An access function can be called to retrieve data from an object, which obscures the way the data is internally accessed. From outside a class, there is no difference between data embedded directly in an object, data that is obtained internally by a pointer operation, or data stored in an objectbase. The internal functioning of the access function does not matter, it is simply called.

For the Flights project, the Revenue slot might contain the data or permit functional access. In any case, to query the revenue from outside the class, an access function is used. This function might return the value of a slot or calculate the revenue from a complex REVENUE object. Viewed from outside the class, there is no difference. (However, a revenue has no complex structure, it is simply a numeric value. This indicates there is no need for a REVENUE class).

This approach can be used to implement an OOA, as it permits complex attributes. However, discussion on programming languages shows that you cannot model data structures without considering their use. If slots are nested too deeply, a standardized access by an SQL-like language might not be possible. Too many access functions have to be newly programmed, considerably increasing programming time. In addition, program maintenance is more difficult if each access function performs a complex operation. Classes that are too deeply modeled are also hard to re-use. Note: a simply-structured class is easy to re-use. Consider Stroustrup's stream classes, for example. He did not use a single class FILE_INPUT_OUTPUT_STREAM but a set of classes. In this way, the model can be wholly or partially re-used, offering maximum flexibility.

Our guideline on modeling classes or slots: use a class for a term with a complex structure. There must be various access methods for such a structure. A slot with multiple values but a single access method should be modeled using the many cardinality, resulting in a substructure of the slot. This is still valid if the order of the slot's values is important, e.g. when modeling an EMPLOYEE. This class has a slot, first names, with a set of values with a fixed order.

A deep structural nesting, using classes instead of slots with multiple values, makes re-use of the classes in another application difficult. The same reasoning limits the number of slots of a class. The better a single class is suited to fulfill the needs of a specific application, the harder it can be re-used. Use hierarchies of small classes to ensure efficient re-use.

Object-oriented data models need not be normalized. A good OOA makes this step of conventional data modeling obsolete. The data model is closer to reality, e.g. the employee Cooper with his three names John, Louis and Martin. This is an advantage of object-orientation over other data models.

OO Analysis - summary

The guidelines given above are a first definition of OOA. They can serve to evaluate an existing model to decide whether it can be implemented or if modification is required. During OOA, the following tasks have to be completed:

- Creating classes from the key terms of the problem
- First classification by creating super- and subclasses
- Analyzing relationships, automatic slot generation
- Inside-out modeling of other slots, assigning these to the correct hierarchy level
- Assigning methods to classes
- Completeness check
- Constant feedback from the experts

OEW supports class creation, storing all structural elements in a central objectbase. Diagram print-outs facilitate communication of analysts and experts. Your OOA can be completely and concisely commented on-line and on paper. To achieve this aim, each class and each slot - especially methods - must receive a comprehensive comment.

OO Analysis defined

Booch's definition of object-oriented analysis (OOA) is similar to the analysis definition of SA, but based on object-oriented terms. OOA describes the requirements for a new system from the perspective of classes and objects, using the vocabulary of the problem domain.

This sentence seems to imply that there is a single method to derive a valid object-oriented model. If so, software production would degenerate to simply applying this method to the problem. Up to now, nobody has found this method. Booch states that only guidelines on how to design software can be given. Our evaluation of analysis techniques is even more critical: currently, we only have rules of thumb that decisively identify a bad analysis result. Nevertheless, this is a great asset to evaluating an analysis from an architectural point of view.

Naturally, this statement cannot be proven, but we can cite an analogy in computer science: structured programming. The software crisis led to intensive research on source-based design maxims. Dijkstra, among others, could prove that any problem can be solved on a von Neumann architecture using three elementary control structures: sequence, loop and selection. This was the foundation for modern programming. Techniques like Nassi-Shneidermann diagrams, action diagrams (mini-specs) and pseudo-code were quick to follow, only allowing structured programming. These structured techniques verify the architecture and design of a program. They do not bar logically faulty elements from the structure charts. There are only rules for the syntactic checking of the charts, but no semantic checking is provided. In analysis we are looking for the semantics, the "what" of a problem domain. That means, simply applying a "method" in a project can not guarantee the required solution in the problem's domain. We believe that this was one of the reasons why Booch and Rumbaugh called their books "Object-oriented design" and "Object-oriented Modeling and Design" in the first place. Other authors used the term "Object-oriented Analysis" in their books' titles but the actual content primarily discusses design (syntactic) issues rather than analysis (semantic) issues.

OOA is the analyst's task. It is the key to success of a software project. Usually the difficulty of analysis grows with the number of analysts - each analyst has his/her own ideas on how to solve the problem! OEW can help in clarifying concepts and ideas with its consistent and unequivocal documentation.

OO Design

OO Design - Accepting analysis

The first step in design is especially important when analysts and designers are different persons. The designer must thoroughly understand the results of OOA to accept them. While studying the documents of analysis, some terms or structures may seem illogical or impossible to implement. In this case, analysts and experts must be notified. Analysis of the project must be revised. The result of the first step of design is acceptance of the results of OOA by the designer. The acceptance of analysis is a time well-suited to transfer the responsibility for the project from the analyst to the designer.

OO Design - Technically completing the model

In the second step, the model must be completed from the technical point of view. Classes or methods may need additional design features to implement their functionality. Furthermore, this includes implementing the cardinalities with container classes. As you will remember, we assigned the cardinality many to the slots pPassenger and pCargo when we were analyzing the relationships of the classes. The designer implements this cardinality with a list class. He creates a common superclass, OBJECT, for all classes that can be held in lists, and derives a class OBJECTLIST from the class LIST.

Afterwards, a cardinality l is assigned to the container class ObjectList, and used to replace the cardinality many. This procedure might seem a little complicated, but is quite useful for larger applications. In such an application, various criteria apply, e.g. memory usage or run-time efficiency. The cardinality many must be separated into various cardinalities, each corresponding to a container class best suited to meet one of these requirements.

During this step, the designer adds interface classes and methods to use databases, communicate with other computers, etc. He manipulates the model in the same way as an analyst, adding classes, creating relationships and describing methods. One of his most important jobs is to check if existing solutions can be re-used in the project. Many of the functions listed above are available in commercial class libraries. The designer must integrate these classes into the model.

OO Design - The user interface

In the next step of design, the user interface of the application is defined. Screens are designed together with experts of the problem domain. If desired, this can take place concurrently with other design activities. OEW is not a GUI prototyper, but many such products are available on the market. In Flights, we used Borland's and Microsoft's MFC class libraries (please refer to the appropriate example files). But you can use the class library of your choice together with OEW. Most of these libraries would work with the Flights example, though you would have to adapt the derived classes and the method bodies. Many GUI interface tools directly generate C++ source code. This code can be imported into the OEW objectbase. Now the designer can create methods to correctly interface with the user interface components.

Based on the user interface the designer now sketches the program sequence. This mainly consists of comments to guide the programmer. For an event-driven system, such as Flights, the designer enters a list of key events into the objectbase information.

To efficiently prepare object-oriented programming, the designer must document his design. OEW supports this with a multitude of comment fields and output options. There is no fixed order for the activities listed above. Concurrent activities and steps backward cannot be avoided in an inside-out project. However, this is not classic prototyping. Experience with interpreter systems has shown that a project should be engineered, not incrementally developed. Especially in large projects, quality of work can only be ensured in this way.

OO Design - Tuning the model

The third step of design prepares all components of the model for programming, assigning suitable datatypes to all slots. In the Flights example, the designer noticed that using the type integer for the revenue from a flight event assumes a single currency. By defining and assigning the datatype Currency, he prepares the program for future change to an international version.

The designer decides which classes will be implemented as structs or unions. Slot access (public, private or protected) is specified, as well as inheritance type. The parameters of each method, special method attributes (virtual, static, ...) and constructors and destructors are defined. Again, the designer must take care to create a model with parts that can be re-used.

The Flights example shows two typical design problems in a complex class hierarchy with multiple inheritance. To avoid multiple definition of FlightEvents's slots in ComboFlightEvent, inheritance must be virtual.

The access function for FlightEvent's slot Revenue must refresh the value of Revenue from the plane's current configuration. Obviously, FlightEvent must define a method that is overridden in each of its subclasses. A realistic model must keep the different flight events in a single list, so the access function must be virtual.

The GetRevenue access function completely hides the slot Revenue of PLANE and its subclasses. If this function calculates the revenue value, the slot is no longer needed and can be removed from the model. Only a design where each unit of freight automatically updates its plane's revenue requires the Revenue slot. The designer must determine which of these models will be implemented.

OO Design - summary

During OOD, the following tasks have to be completed:

- Accepting object-oriented analysis
- Expanding the model to suit technical requirements
- Checking existing projects and class libraries if re-use is possible
- Preparation of OOA classes and slots for programming
- User interface definition
- Sketching program sequence
- Checking integrity and completeness
- Especially during interface design, analysts and experts should be consulted to ensure that design meets their requirements.

OEW supports the design of classes and slots, storing all structural elements in a central objectbase. OEW visualizes the design. Diagram print-outs facilitate communication of designers, analysts and experts. Your OOD can be completely and fully commented on-line and on paper. In addition to commenting classes and slots, the program sequence or the key events should be commented. In OEW, the objectbase and module comment fields are especially suited for this purpose.

OO Design defined

Object-oriented design (OOD) is the technical specification of the project. The results of OOA are used to create a model that can be implemented. This terminology agrees with the traditional technique of Structured Design as described by Constantine and Yourdon, for example.

This agreement does not result in a consistent definition of OOD in literature. The limits of OOA and OOD vary from author to author, if any are given. The contents of OOD differ as well, and all terms of OOA (inheritance, message passing, etc.) are used to describe it. If you are familiar with traditional software engineering, this diversity may be confusing and even result in rejection of object-orientation.

Object-orientation is quite a young technique of software engineering. Very few large projects have been completed. This is the cause of many of these problems. As object-orientation becomes more widespread, clearer definitions can be expected. Another explanation is the origin of object-orientation. First experiences and results were gained on single-user, stand-alone interpreter systems, e.g. using SMALLTALK. This caused OOA, OOD and object-oriented programming (OOP) to intermingle in a procedure called prototyping. This is a viable approach for experimental application design. But we believe that building successive prototypes until the final solution is attained is not applicable in professional software engineering, especially in larger projects.

In OOD, the designer translates the problem description from OOA into a model that can be implemented on a computer. An OOA shows the users' view of the problem. All the terms that are modeled are relevant and comprehensive for an expert of the problem domain. What about technical specifications? For example: which ways will be used to store the data required? Which screen layouts will be used? Which classes, variables, methods must be added to implement the analysis? Which procedures need optimization? Questions like these define the tasks of a designer. OEW helps you to transfer smoothly from OOA to OOD.

OO Programming

OO Programming defined
Accepting design
Implementing basic methods
A draft main program
Completing the program

OO Programming - A draft main program

The next step implements a draft program testing the methods under "working" conditions. OEW supports this white-box testing, integrating make-tools and compiler IDEs with OEW development.

OO Programming - Accepting design

Once again, the first task of design is acceptance of the prior stage: of OOD. The programmer must become familiar with the model. If he detects problems or faults, he must notify the designer, maybe even the analyst. This step of OOP results in consent to the results of OOD. Now responsibility for the project should be transferred to the programmer.

OO Programming - Completing the program

Now a working program is created, featuring full functionality. This is quite similar to traditional programming. The program structure directly results from the class hierarchy, however, and the ideas it represents. Development is local, i.e. the programmer does not work on the program as a whole, but on methods that are components of classes. As a result, the superstructure of a program is much more visible, and OEW enhances it even better. The final program must now be tested and presented to experts in the problem domain. Only they can determine if the program is useful.

During OOP, the following tasks have to be completed:

- Accepting the results of OOD
- Implementing basic methods
- Testing these methods using a draft main program
- Completing implementation of all classes and methods
- Integration and testing
- Checking integrity and completeness
- Analysts and experts should be consulted to ensure that the program meets their requirements.

OEW supports programming by storing structural data and method bodies in its central objectbase. OEW generates the complete, commented source code for your application from this information on modules, classes and methods. Finally, you can have OEW generate the executable program.

Errors found during testing can be fixed in OEW. It is also possible to fix them directly in the source files, e.g. in a compiler's IDE. Nevertheless, your OEW project won't lose correspondance to your sources. When the program is running or changes in OEW are required, you can refresh the objectbase using import from source. By keeping the objectbase up-to-date you prepare software maintenance as well as changes and improvements of its functionality.

OO Programming - Implementing basic methods

It is common use to start programming with the methods added during design as these usually serve basic functions. The functionality of the program relies on these methods. In addition, the programmer must be familiar with the classes and methods that will be re-used, as their functions are similar. This procedure - starting with the basic functionality of the program - is more so a bottom-up approach.

OO Programming defined

In object-oriented programming (OOP), the design is translated into executable code. Compilers, debuggers and other powerful "lower-CASE" tools support the programmer during this stage of software engineering. We perceive OOP as structured programming with objects. In the same way as structured programming sets rules to avoid "garbled code," we wish to give guidelines for object-oriented programming in the following paragraphs.

Object-orientation originates from the research of programming languages. As a result, a large part of object-oriented techniques and standards concern OOP. As OOP is a topic older than OOA and OOD, all the terms of object-orientation are used to describe OOP.

The aim of OOA and OOD is an optimized preparation for OOP. Thus, our minimum requirements for a programming language are classes, inheritance, message passing (method calls), access control, and dynamic linkage. A programming language with only some (ADA), very few (C) or even none of these properties (COBOL) makes implementation of an OOA difficult if not impossible. We are aware that this strict statement is contradictory to many authors. Coad and Yourdon, for example, discuss implementation of OOA in COBOL. In our opinion, constant support by a method and the asserted tool means using the results of OOA and OOD in object-oriented programming. An object-oriented analysis cannot be "compatible" with a programming language that is not object-oriented. The structure of data and functionality simply doesn't match. Discarding their design to start from scratch does not tend to make developers happy. Besides, how do you maintain a program that only vaguely resembles its analysis? Although it might be possible in principle to implement an OOD in a conventional programming language, the expenses are too high. More so, COBOL or FORTRAN on a mainframe is the very opposite of the principles discussed in this text.

OEW currently uses the C++ language for OOP. This does not prevent using OEW for OOA of other object-oriented languages as Objective-C, extensions to Pascal, SMALLTALK or CLOS.

As the foremost aim of software engineering, the programmer creates an executable program during OOP. The engineering of industrial products is automated at least partially by machines. In the same way, OEW is meant to automate a part of OOP. OEW offers source code management, automated code generation, updating of debugged sources and support for the other tools of programming, e.g. compilers and debuggers.

Object-Orientation

Object-Orientation is currently the catch-word in computer science and business. However, the terms in object-orientation are not clearly defined. As a consequence, nearly everything is object-oriented, object-based, object-structured, etc.

Recent discussions on CASE and computer languages have added to this variety of terms. Additionally, there were many attempts to apply the results of object-oriented research in database technology. The main terms of this sector are structurally object-oriented, behaviorally object-oriented and fully object-oriented. It is interesting to see that object-orientation creates a link between programming languages databases and CASE.

The ancestor of object-oriented programming languages is SIMULA 67, a language conceived for simulation software. The first complete, stand-alone object-oriented development systems were built in the early '70s in XEROX's Palo Alto Research Center. The main aim of XEROX's first research on SMALLTALK systems was to improve the communication between human beings and computers in the Dynabook project.

Artificial intelligence research - especially expert systems - also had a strong influence on object-orientation. The developers of the time were guided by concepts like those of Marvin Minsky, described first in his "frame"-paper and summarized later in his "The Society of Mind". Minsky and other authors explained the basic mechanisms of human problem solving using frames, a concept well-suited for computer implementation. This explains the similarities between classes in systems like SMALLTALK and LOOPS, and units in KEE.

Object-orientation in structure and behavior

Meanwhile, several systems using object-oriented methods have been created. These systems, using stand-alone environments and unique interpreter languages, were meant to rebuild real objects and their relationships as closely as possible. Such an object has the following properties:

a unique identifier, e.g. the name of the object

a list of attributes

possible values for these attributes

information on relationships to other objects

Modeling objects implies modeling their structure as well as their behavior, resulting in structurally object-oriented and behaviorally object-oriented systems. Systems encompassing both features are fully object-oriented. These definitions originate from database technology.

Structural object-orientation means class-instance hierarchies of objects. Such a hierarchy is created either bottom-up or top-down. The bottom-up method aggregates object attributes, the top-down approach calls for a step-by-step specialization of classes until instances - mirroring reality - are attained.

As an example for object-oriented data structuring, let us create a top-down specialization of a root class, PLANE. A PLANE has several attributes, e.g. a plane_ID. The subclasses of PLANE, PASSENGER_PLANE and CARGO_PLANE, have additional attributes. An important characteristic of an object-oriented model is that a derived object inherits the attributes of its superclass. In our example,

both subclasses of PLANE inherit the plane_id.

Inheritance offers various advantages in the design of data structures. Foremost, an attribute only needs to be defined once - in the superclass - and is passed to its subclasses. In this way, it is easier to conceptualize and mentally encompass the data model. No redundant redefinition is needed, as a subclass inherits all the attributes of its superclass. Furthermore, structural errors in the design can be avoided. Another advantage of inheritance: you can solve the general problem first, creating subclasses later to cover exceptional cases. Again, this feature of object-oriented analysis mirrors human behavior. Finally, a hierarchical object model is modular, resulting in more flexible software as its components can be easily replaced.

In addition to inheritance, object-orientation does not require attributes to be atoms. The value of an attribute may be a list or a complex object. For example, the attribute Cockpit_Crew of a PLANE must allow for a single pilot as well as a list of persons for larger planes.

Object-oriented behavior calls for functional properties of a class. Data elements and operations on data are encapsulated in a class together. These operations are also used for message passing, allowing objects to communicate. Each object can process a fixed set of messages, its protocol. To access an object's internal data structure, other objects must use the message passing mechanism. There are two modes of message passing:

The caller sends a message, and waits for it to be completely processed.

The caller resumes its current operations immediately after sending the message. This asynchronous mode of message passing is used in implementing concurrent processes.

Fully object-orientated systems are structurally object-oriented and have object-oriented behavior. Functional and data elements are handled the same way. They are all inherited by derived objects. Full object-orientation combines data and process modeling.

C++ doesn't appear to support message passing, so why explain this concept? If you study the fundamentals of message passing you will find no conceptual difference between a synchronous message exchange and a call of a function of the object that receives the message. Most interpreter systems internally translate message passing into function calls, anyway. Thus, C++ does in fact support message passing.

A concise definition of terms

The terms used in object-orientation are definitely not standardized yet. Attributes, for example, are called members in C++ but slots in KEE. If attributes are functional, they are labeled methods in SMALLTALK and KEE, member functions in C++ and routines in Eiffel. These are just two items of a long list that do not facilitate the understanding of object-orientation at all. For this manual, we use an efficient subset of all terms:

Object

A real or artificial, individual unit. Persons, ideas, or items are real objects, container objects like hash tables are artificial. "Object" is quite a general term. Non-object-oriented fields use this term as well. Therefore, we do not think it useful for precise definitions in a model intended for computer implementation. We will use this term in a "colloquial" sense when the distinction between natural, artificial and computer objects is of no consequence.

Class

A class represents a set of individual instances with similar structure and behavior. A class might be a superclass and/or a subclass of other classes.

Instance

A class defines structure and behavior of an arbitrary number of instances. In compiler languages like C++, these instances are typically created at run-time.

Attribute

An object has several properties, its attributes. This is a term from the traditional Entity Relationship Modeling (ERM) technique. We will not use the term attribute to describe the properties of a class because it is widely used in non-object-oriented fields.

Slot

ERM objects - or entities - only have atomic attributes with no internal structure. The "attributes" of a class may have a complex structure. To set them apart from ERM attributes, we call them slots. A slot may be data, function or even a datatype.

Method

We also wish to set the function slots of a class apart from the functions of traditional programming languages like C or LISP. Therefore, we call these functions "local to a class" methods.

Objectbase file open

This dialog is used to open an objectbase file. They usually have the ".oew" extension. Select the file you wish to open and press the OK button.

To open objectbase files that have other extensions (for examples, the backup files automatically generated by OEW), select all files from the list of file types.

Please note: The file "users.oew" that is stored in OEW's home directory does not contain an objectbase, but OEW's user list.

Objectbase information

This dialog displays information about the active objectbase and permits you to change it.

You may specify a name for the objectbase that is independent from and may be longer than the objectbase filename. It will appear in the title line of all OEW print-outs.

You may enter a description and a comment.

see also

Quickinfo

Objectbase save as

Specify the filename and directory where you want the objectbase to be saved.

siehe auch

objectbase templates

Objectbase templates

All objectbases saved in a directory called 'template' in the OEW directory (where OEW.EXE is located in) can be used as objectbase templates when You create a new objectbase. If you want to create a new objectbase without using a template select 'none'.

If you save an objectbase into the 'template' direcotry, OEW saves the name of the objectbase (not the file name!) in a file called 'contents.ini' in the template directory. The name of the objectbase can be changed in the 'Edit objectbase information' dialog.

Overview window

An overview window displays the whole diagram contained in an inheritance or relationship window. The display will be zoomed to the needed level, either minimized or maximized to make the overview diagram use exactly the overview window size.

A dark gray rectangle within the overview window shows which part of the diagram is currently displayed in the associated inheritance or relationship window.

The window, the overview belongs to

Mouse operations in the overview window

Overview window - Mouse operations in the overview window

You may control the visible area that is currently displayed in the associated relationship or inheritance window.

With your left mouse button, click once on any point within the overview window. The dark rectangle will change its upper left corner to the position you have clicked on. The visible area in the associated window will change immediately.

Overview window - The window the overview belongs to

Every overview window is associated with exactly one inheritance or relationship window. As it is possible to open an overview window only from the both mentioned window types, there is no need to tell OEW which window the overview belongs to. It is always the window that was active, when a 'show overview' command was given.

Parser status

OEW is currently importing source code files. It reads, parses and converts them to its internal format. This may be a long duration process.

You see three lists of files. On the right are the files OEW is finished with. In the middle you see what is currently in progress. While reading a special file, OEW might encounter imports in it, depending on the import options you have chosen, OEW may have decided to parse another file first. That's the reason why you sometimes see more than one file currently in progress. OEW is finished with the files displayed on the right.

The line at the bottom displays the currently read file and the currently read line number. OEW updates this display twice a second.

see also

Quickinfo

PathStyle

In the file 'oewsys.ini' in the OEW direcotry You can define an internal variable which is used upon code generation and file name creation.

[CodeGen]
PathStyle=n

The variable n can have the following values:

0=Version dependant

Depending on the platfor version of OEW the following value is used:

Windows 16 bit: MS/DOS

Windows 32s, Windows NT, Windows95: Windows NT

OS/2: OS/2

Unix: UNIX

1=MS DOS

File names will be shortened to 8+3 characters and converted into lower case.

A backslash is used to separate directory names.

Text files will be written using CR/LF.

2=OS/2

File names will not be shortened but converted into lower case.

A backslash is used to separate directory names.

Text files will be written using CR/LF.

3=Windows NT

File names will not be shortened but converted into lower case.

A backslash is used to separate directory names.

Text files will be written using CR/LF.

5 or 6 = UNIX

File names will not be shortened but converted into lower case.

A slash is used to separate directory names.

Text files will be written using LF.

Print

Use this dialog to specify the way you want OEW to print. Press the 'Preview' button to get an impression of how the printed papers will look like, or use the 'Print' button to print immediately.

Printing to a physical printer is always allowed. Printing to RTF file or plain text file is allowed if your print-out does not contain graphical information. Use the controls in the upper left corner to specify your requested print-out target.

By using the controls at its right hand side, you may either select a physical printer, or specify the output file name.

Use the control in the lower left corner to choose a font for your print-out.

- this font will only be used if you print to a physical printer.
- due to some complex internal calculations, the font measure you choose using the font select button may differ slightly from the font measure displayed here.

In the lower right corner area, you see a descriptive name for the printout you're going to make, and an 'Option' button to its right. If this button is available, use it to access the special options available.

Please note: OEW's preview functionality requires a printer driver installed in your system. If no printer is installed, it will not function.

see also

Quickinfo

Print Graphic

This dialog is used to control and start printing of OEW graphics.

In the upper area are the options for the output device. There are three text labels, each having a button to its right showing the text 'Select'. The label shows the current setting, the buttons are used to change it. The settings are the output device (the printer), the font that is to be used and the page margins that will be used on the paper.

In the lower area you configure the pages (ranges and labels).

Use the button 'Print' to start printing.

'Preview' will give you a preview of the printout on the screen. Please note that preview is only possible if there is at least one printer installed in your system.

see also

Quickinfo

Print Graphic - Pages

Do you want the printout that you are generating to be part of another document? In this case you perhaps wish that all pages are numbered continously, including the pages generated by OEW. To adjust the page numbers for this purpose, enter the required start page number in the field 'Number from'.

Furthermore it is possible to print only selected pages. This is useful if you have a large printout and some pages didn't work, maybe due to printer problems. Therefore do not select 'All' but 'Range' instead, and enter the required pages or page ranges into the field.

Separate the different page numbers or ranges using a semicolon (;). A range is entered for example like this: 5-19

Print documentation options

Use this dialog to specify
- the parts you want your printed documentation to contain
- the logical style (analysis, design, implementation)
- the physical style (tables with frames or not)
- the number OEW should use to label the first page number
- the set of classes you want the documentation to be printed for

see also

Quickinfo

Print preview

This dialog gives you a preview for the document you're going to create, one page at a time.

If you are printing to a physical printer, "what you see is what you get". If your print target is a file, the preview only displays the textual information and layout that will be contained in your file.

Use the buttons '+' and '-' in the upper right corner to zoom in or out the displayed page. Current zoom factor is displayed above them. Zoom factor 0 means OEW displays the full page.

Below those buttons, there are four buttons to navigate through the document's pages. Current displayed page number is displayed above them.

Use the 'Print' button in the lower right corner to start print. You may decide not to print, use the 'Close' button.

Please note: OEW tries to simulate the print-out as well as possible. It may fail to display a good preview if the font you've chosen is not scaleable.

see also

Quickinfo

Print status

OEW is currently sending a document to your printer (or file). Please wait until this operation is finished.
You may cancel the operation using the 'Cancel' button.
see also
Quickinfo

Print to file - File save as

You told OEW to print into a file. Use this dialog to browse through available directories and enter or select a filename you wish to be used.

Printer selection

Use this dialog to set the printer that you wish to be used by OEW.

Query window

You can use a query window to access objectbase elements by their name.

Enter the name of the object that you are looking for (OEW accepts the asterisk (*) wild-card at the end of the name), and click on the 'Display result' button.

The list in the window's lower left corner displays the search results. Every element whose name matches your search criteria is displayed here.

To see the source code for an element, click once on its name in the list. It will fill the text window in the lower right corner.

To edit an element, double-click on the name in the list.

Quick Start Tutorial - Step 1 of 25

While trying this tutorial, you are required to switch between this help window and the OEW main window. You may do so by using the left 'Alt' on your keyboard (it is on the left hand side of your spacebar), holding it pressed and hitting the 'Tab' key once (it is the key that is labeled with two arrows, one to left, one to right), and release the 'Alt' key.

Each time you will read an instruction, please use this keyboard combination to switch to your OEW window, and after you have tried the instructions, go back to this help window using the same keyboard command. Please try now to switch and come back.

proceed

Quick Start Tutorial - Step 10 of 25

You just added an inheritance from class Plane to class CargoPlane that means that CargoPlane is inherited from Plane. OEW changes the contents of the 'inheritance window' accordingly: OEW displays the PassengerPlane on the right hand side of Plane. This visualizes that PassengerPlane is an inherited class of Plane. Also, a line is drawn from Plane to PassengerPlane.

The inheritance you have created has options. In the middle of the line, there is a small rectangle. Try to double-click on it with your left mouse button. A dialog opens that shows the current settings. We don't want to change any options now, please use the 'Cancel' button to close the dialog.

proceed

previous step

Quick Start Tutorial - Step 11 of 25

Now it is time to fill our classes with elements. Elements of classes are called 'slots' in OEW. OEW is able to display slots within the inheritance window. By default, this functionality is disabled for each class. We want to enable it for the 'Plane' class. Open the object menu of Plane and select 'Show slots'. A line will be added within the rectangle. It separates the name of the class from the list of the slots contained in the class. You won't see any slots, because there are currently no slots contained in the class Plane.

proceed

previous step

Quick Start Tutorial - Step 12 of 25

Again, open the object menu of class 'Plane', select 'New Slot'. A dialog appears where you tell OEW what kind of slot you request. Actually, a new slot was already created in the moment you selected 'New slot'. OEW used defaults, and it displays the properties of the newly created slot. We want to change only the name of the slot. It should be highlighted and be 'unnamed'. Simply overtype it with 'Freight'. Press 'OK' to confirm your input.

proceed

previous step

Quick Start Tutorial - Step 13 of 25

Back in the inheritance window, the Plane rectangle now contains an additional line 'Protected integer Freight'. You may reaccess the just displayed dialog by double-clicking this line. If you choose to try this out now, the dialog will be displayed. Use the cancel button in it to close it.

proceed

previous step

Quick Start Tutorial - Step 14 of 25

Suppose that you now want to test what you have created. From the file menu select 'Save as'. (It is recommended that you create a new directory for the objectbase first using your file manager). The directory that you choose to save your objectbase file in will automatically be the place for the files created in our next step.

proceed

previous step

Quick Start Tutorial - Step 15 of 25

From the 'Source' menu select 'Generate all'. This will give you a dialog that displays a list of Java files that will be generated. These files will contain the objects you have been creating in the standard syntax of the C++ programming language. Press 'OK' to confirm. After a short moment, OEW will be ready. From the 'File' menu, select 'Save'. This will ensure that OEW will remember that it has written the Java files.

Please note: While you were working with OEW, it automatically decided that you will require files when you decide to test your project. It automatically created 'modules' (Java files stored internally in OEW) and placed every object you created in one of them. It provided the modules with default names that it derived from the class names. Actually, OEW creates new modules every time you manually add new classes to your project.

proceed

previous step

Quick Start Tutorial - Step 16 of 25

Let's take a look at the files we've just created. From the 'File' menu select 'Editor' to access OEW's built in Java file editor. A dialog opens that allows you to select the file you wish to edit. The dialog's directory list should already show the directory where you saved your objectbase, and in the list on the left hand side the newly created files should be displayed. Select the file 'plane.java' and press the 'OK' button. OEW opens the file and displays it. Don't be worried about the different colors. The OEW editor understands some Java, and shows you the different object types using different colors.

proceed

previous step

Quick Start Tutorial - Step 17 of 25

Now you will get to know OEW's ability to re-engineer text files that are created or modified with a file editor, either with the one you just opened, or even with any other text editor. We want to demonstrate this now. Please move your cursor to the line where the data element 'int Freight' ist declared. Please change the word Freight to something different, for example to 'FREIGHT' in all upper case. After you are finished, choose 'File Save' and then 'File Close'.

proceed

previous step

Quick Start Tutorial - Step 18 of 25

We are back in the inheritance window, and OEW still displays the word 'Freight' in the old way. This is not an error, as OEW doesn't change the contents of an objectbase without being explicitly instructed to do. We want to do this now. From the 'Source' menu, select 'Import changed modules'. OEW will recognize that only the file plane.hpp is 'new', and add only this file to the list. Simply press the 'Start' button to continue.

After a very short moment, OEW will be finished reading the file and updates the display. The word 'Freight' should have changed now to the text that you typed in the text file.

proceed

previous step

Quick Start Tutorial - Step 19 of 25

The second most important window in OEW is the relationship window. It is not as powerful as the inheritance window, e.g. it does not provide the possibility to show slots in the diagram, but you may position your classes freely on the screen, and additional lines are displayed, showing the other types of relationships that may exist between classes.

From the 'Window' menu, select 'New relationship window'. To improve the window layout in your workspace, select 'Arrange horizontal' from the 'Window' menu.

proceed

previous step

Quick Start Tutorial - Step 2 of 25

We want to have enough workspace in OEW, so please maximize the OEW window. Do this by using the arrow showing upwards in the upper right corner of the OEW window.

proceed

previous step

Quick Start Tutorial - Step 20 of 25

Just as you may draw lines in the inheritance window, you may do so in the relationship window. But by default, the created relationship isn't of type 'inheritance', but of type 'references'. Now try this: In the relationship window, draw a line from Plane to PassengerPlane. A dialog opens where you may change the properties of the relationship just created.

On the upper right hand side, you see three controls that are inside a box labeled 'Type'. Please click with your mouse on the control labeled 'references' (This makes the 'OK' button 'clickable'). Take a look at the middle left hand side, there is a box labeled 'Relationship'. Inside you see a small box labeled 'Create slot'. It is checked, that is, you see a small cross inside it.

proceed

previous step

Quick Start Tutorial - Step 21 of 25

Now press the 'OK' button. The dialog is closed, and both windows, the relationship and the inheritance window, are updated. In the relationship window, a new line was added that visualizes the Relationship you just created.

Unfortunately, it is likely that you can't read the text that is displayed in the middle of the window. The reason for this is that two texts are displayed at the same position. But you can rearrange the classes. Move the rectangle in the middle slightly to the right. Try this: Press and hold the 'Ctrl' key on your keyboard (the key in the lower left corner), move your mouse cursor over the middle rectangle, click and hold your right mouse button once, now move the mouse to the right. The rectangle will follow your mouse cursor. Release the mouse button, and the lines will be redrawn. You should now be able to read all the texts.

The contents of the inheritance were updated, too, as mentioned above. The Plane rectangle now contains an additional slot: the data member that is the reference to the class PassengerPlane.

proceed

previous step

Quick Start Tutorial - Step 22 of 25

Let's try if the opposite way of adding relationships works, too. In the inheritance window or the relationship window (it doesn't matter which you choose) move the mouse cursor over the class PassengerPlane, and double-click with the left mouse button.

A new window type will open, the 'Slots Window'. Similar to the command 'Show Slots' that we used earlier, this window gives you a list of all the slots contained in a class. It is the white area in the window that is currently empty, as no slots are in the class PassengerPlane. (The main reason for having the functionality of showing slots twice: with the slots window, it is easier to maintain large lists of slots.)

proceed

previous step

Quick Start Tutorial - Step 23 of 25

The white area (the slot list) has an object menu. Open it and select 'New slot'. Enter the word 'TestSlot' into the field 'Slotname', press the 'Tab' key (the key showing two arrows, one to the left, one to the right) once, and enter the word 'CargoPlane' into the field 'Type'. Press the 'OK' button when you are finished.

proceed

previous step

Quick Start Tutorial - Step 24 of 25

You are back in the 'Slots window'. Use the 'Close' button to close it. As you can see, the relationship window saw the new created relationship and displays an additional arrow for it. You may edit a relationship, for example we choose the relationship line labeled 'inheritance'. Try to move your mouse cursor exactly over the arrow in the middle of the line, and open its object menu. As you can see, you are allowed to edit or delete a relationship.

proceed

previous step

Quick Start Tutorial - Step 25 of 25

You are finished learning the most important parts of OEW and the standard OEW development cycle. You may now try to browse through the other menu commands, open the assigned dialogs, read their help texts. Open the objectbases contained in your examples directory and see what's possible.

Context sensitive help is available everywhere. Press the F1 keyboard key to access it. Where available, you may use the 'Help' button, too.

previous step

Quick Start Tutorial - Step 3 of 25

The kind of file, that is managed by OEW and accessed via the file menu are objectbases. These are binary files. To start your work, select 'File New'. A dialog will open where you may set initial properties of your new objectbase. For our tutorial, the default settings are fine. Just press the 'OK' button.

A window called inheritance window will open. It is one of the most important windows displaying information. Initially, it will be empty. We want to enlarge it: From the 'Window' menu select 'Tile'.

proceed

previous step

Quick Start Tutorial - Step 4 of 25

During our tutorial, we don't want to be disturbed by 'save now' reminders, because we don't want to save our test data. It is not necessary to understand this now. Please select 'Options Warnings' from the menu to disable warnings, after that select 'Options Automatic Save', click on 'Deactivated' and press 'OK'. (Please do this only for the current tutorial, and not for every project you are working on, unless you know the meaning of these settings.).

proceed

previous step

Quick Start Tutorial - Step 5 of 25

Now try this: Move your mouse cursor inside the 'inheritance window' and press the right mouse button once. A menu opens. Select 'New Class'.

A dialog appears, where you may add new classes to your project. Please enter:

```
Plane, CargoPlane, PassengerPlane
```

and press the 'Add' button.

Now you will see three rectangles within the inheritance window. Each represents one class. By default, the inheritance window displays all the classes contained in your project.

proceed

previous step

Quick Start Tutorial - Step 6 of 25

proceed
previous step

Quick Start Tutorial - Step 7 of 25

You are now back in the 'inheritance window'. As you can see, there are still only three rectangles. Datatypes are not displayed in the inheritance window.

Now we want to change the properties of classes.

proceed

previous step

Quick Start Tutorial - Step 8 of 25

Move your mouse over the rectangle labeled 'Plane' and press the right mouse button once. An object menu opens. Select the command 'Edit class', it is the second command from the top. This dialog allows you to change the properties of the class. We don't want to do this now, close the dialog using the 'Cancel' button.

proceed

previous step

Quick Start Tutorial - Step 9 of 25

Classes may be in relation to each other. One possible relation is the C++ feature 'inheritance'. We now want the class 'CargoPlane' to be inherited from the class 'Plane'. You can use the 'inheritance window' that is displayed to add this inheritance. Please read the following paragraph first, store it in your mind, and try it out then in one step:

Move your mouse cursor over the rectangle labeled 'Plane', click your left mouse button down and leave it clicked! Don't release the button. Move your mouse cursor over the rectangle labeled 'CargoPlane'. (A line will be drawn while you move the mouse cursor). Now release the mouse button.

proceed

previous step

Quick reference window

Use this window to get information on any class in your objectbase, displayed within a single window.

Click a class in the list at the window's left hand side. It will fill the upper right-hand corner with its slots.

As you click on classes or slots, the text window in the window's lower right corner will become filled with information on the item you clicked.

Quick Start Tutorial

This text gives a quick introductory to the basic features of OEW. As OEW is a complex application with a variety of different features, we recommend to work through this tutorial, to ensure that you have seen the most important parts of OEW at least once.

However, you should know what kind of software OEW is (you may read What is OEW?). In addition, it is required that you have some knowledge of the C++ programming language.

proceed

Quickinfo

In OEW there is a special feature available that explains nearly everything it displays.

To enable it, select 'QuickInfo' from the Help Menu, and move the mouse cursor to the element of interest (without clicking buttons). OEW will display a small bubble that contains some descriptive information.

There is help information on the menu commands, too. Press and hold the mouse button while the mouse cursor is over the menu bar, and move from item to item to see it.

RTF file

OEW creates a lot of 'text styles' which are applied to the various parts of the generated RTF file. For example, if you would like to change the style of all the cells containing the class name within all class tables, simply use your word processing software, and change the style named 'ClassTabDCol1'. In addition, all the print styles are part of a hierarchy. There are some basic styles, which are the bases of more special styles. For the base hierarchy, from which the specialized styles are derived, see the following diagram:

- none
  - BlankLine
  - PageHeader
  - PageFooter
  - Text
  - TableTitle
  - TableStyle
    - HeaderTableStyle
      - AuthorH
      - NameH
      - CommentH
      - DefinitionH
      - TypeH
      - CardinalityH
      - FlagH
    - DataTableStyle
      - AuthorD
      - NameD
      - CommentD
      - DefinitionD
      - TypeD
      - CardinalityD
      - FlagD

All specialized styles that are part of one table type start with a common prefix. These prefixesfor the various tables are:

Prefix	of the styles in the table containing...
InfoTab	objectbase information
StatTab	statistical information
ClassTab	list of classes
RelTab	list of relationships
SupSubTab	list of super- and subclasses
LocTypeTab	list of local datatype slots
DataSlotsTab	list of data (attribute) slots
MethodsTab	list of method slots
InstTab	list of instances
GlobTypeTab	list of global typedef datatypes
GlobEnumTab	list of global enum datatypes
ViewsTab	list of views
ContTab	list of container classes

ModTab	list of modules
ModElTab	list of module elements
ModSrcTab	module source
GroupTab	list of user groups
UserTab	list of users

Most of those tables are preceded by a title line. This title line has a style named 'table prefix' + 'Title', for example the title's style of the objectbase information table is 'InfoTabTitle'. All title styles are derived from 'TableTitle'.

Within the table, there are different styles for header and data rows, and different styles for each column. Style names that have 'H' after the table name prefix are header row styles.

The ones with 'D' are data row styles. These have the styles 'HeaderTableStyle' and 'DataTableStyle' as (indirect) bases. Each Column of a Table has the additional suffix 'ColX' where X is a number. For example: The cells that contain the total counts in the statistical information table, do have the style 'StatTabDCol4'. This logic is consistent for all styles of all the table cells. To find out which number to add for a particular cell, please make a test print-out and look at the desired cell. Please note: The column numbers do not necessarily have to be consecutive. Not all the columns are printed in all the print modes. But it is guaranteed that column numbers will be identical in all different print modes.

There is another derivation: Depending on the type of information contained in a particular cell, its style is derived from one of the intermediate bases named 'AuthorX', 'NameX', 'CommentX', 'DefinitionX','"TypeX', 'CardinalityX' and 'FlagX', where 'X' is either 'D' for data rows or 'H' for the header row. For example, 'ClassTabDCol1' contains a name (the class name) and therefore it is derived from 'NameD'. The header of this column has the style 'ClassTabHCol1', and it is derived from 'NameH'.

With word processing software that supports writing macros, you're enabled to access all the information you need using an automated process. Locate the text passages by searching for the style names.

References

A list of references can never be complete. For this reason, the following can only introduce the topic of object-orientation.

Minsky's original concept of modeling knowledge in frames is among the bases of object-orientation. He later compiled his ideas in his very comprehensive

Minsky, M. The Society Of Mind. New York: Touchstone, 1985

There are hundreds of essays and books on software production. The following books not only describe its basic structure but also delineate the principles of traditional software engineering,

Boehm, B. W. Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981

Constantine, L., Yourdon, E. Structured Design. Englewood Cliffs, NJ: Prentice-Hall, 1979

DeMarco, T. Structured Analysis and System Specification. Englewood Cliffs, NJ: Prentice-Hall, 1979

Ward, P., Mellor, S. Structured Development for Realtime Systems: Introduction and Tools. Englewood Cliffs, NJ: Prentice-Hall, 1985

Yourdon, E. Modern Structured Analysis. Englewood Cliffs, NJ: Prentice-Hall, 1989

Object-orientation is also a widely discussed discipline. Goldberg's books discuss SMALLTALK. Object-oriented development in Eiffel cannot be separated from the name Meyer. Coad and Yourdon have a very pragmatic approach to analysis and design of object-oriented systems. Booch not only proposes a unique design technique but also gives a multitude of examples in the C++ programming language. His book also contains a very detailed list of references. Rumbaugh describes a very popular way of modeling and design. Last but not least, Martin and Odell propose the notation that we have chosen.

Booch, G. Object-Oriented Analysis and Design with Applications. Redwood: Benjamin/Cummings, 1994

Coad, P., Yourdon, E. Object-Oriented Analysis, 2nd edition. Englewood Cliffs, NJ: Prentice-Hall, 1991

Goldberg, A., Kay, A. et. al. SMALLTALK 72, Manual. Palo Alto: XEROX PARC, 1976

Martin, J., Odell, J. J. Object-Oriented Analysis & Design. Englewood Cliffs, NJ: Prentice-Hall, 1992

Meyer, B. Object-Oriented Software Construction. London: Prentice-Hall, 1988

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: Object-Oriented Modelling and Design. NJ: Prentice-Hall, 1991

Finally, Gupta and Horowitz as well as Kim and Lochowsky treat object-orientation from a database view.

Gupta, R., Horowitz, E. (Ed.). Object-Oriented Databases with Applications to CASE, Networks, And VLSI CAD. Englewood Cliffs, NJ: Prentice-Hall, 1991

Kim, W., Lochovsky, K. Object-Oriented Concepts, Databases, and Applications. Reading, MA: Addison-Wesley, 1989

Relationship window

A relationship window is used to display and edit the relationship structure between classes contained in your objectbase, and to access the displayed objects.

Window contents

Manage what and how it is displayed

Mouse and keyboard operations

Relationship window - Add new class(es)

Double-click with the left mouse button in the window's background. A dialog will appear where you may specify the class(es) to add.

Relationship window - Change size of class rectangles

All class rectangles in the relationship window have the same size, all the time. At the time you open a new relationship winodw OEW determines the size from the longest class name. To avoid difficult to read diagrams, especially if large class names are used, you may change the size of the rectangles.

To change this size, move your mouse cursor to the lower right corner of any class rectangle. The mouse cursor changes its shape to a cross. Click and hold the left mouse button and move the cross to the new size. Release the mouse button, and the display will become redrawn. Names will be shortened to fit the display area.

Relationship window - Drag and drop

Copy / move objects

Drag and drop objects between different windows. Drop them into a rectangle to make them local, drop on the background to make them global.
Standard operation is move. Press and hold CTRL to copy.

Add / Remove classes to / from current view

While there is a view active, you may drag and drop class names from a 'list of classes window' or an 'Edit view dialog' to change the view contents.

Import source code / objectbase

Drag and drop C++ source files or objectbase files from the File Manager to import them.

Relationship window - Edit slots

To access a class' slots window, double-click on the class or press enter while your class is focussed.

Relationship window - Focus

At any time, one class within the window is focused. Commands that operate on a single class will use the focused class. The focused class has a larger black border than other classes. The focus changes automatically as the mouse cursor moves over the window.

Relationship window - Grid

You can set up a grid for the relationship window. All further positioning will then be adjusted according to the grid.

Relationship window - Manage what and how it is displayed

Font for classes
Font for relationships
Change size of class rectangles
Reformat like class display
Show derivations / contains / inheritance relationships
Show relationship labels
Views
Grid

Relationship window - Mouse and keyboard operations

Scrolling
Zoom
Focus
Edit slots
Add class(es)
Select class(es)
Object menus
Rearrange the class order
Relationships
Drag and drop

Relationship window - Object menus

Objects within relationship windows that have object menus assigned:

- class rectangles
- relationship line's arrows
- the background

Relationship window - Rearrange the classes, relationships and labels

To move an object manually to another position:

- press and hold the 'CTRL' key on your keyboard
- while the mouse cursor is over the object you wish to move, press and hold the right mouse button
- move the object to the position of your choice
- release the mouse button and the keyboard key

To move more than one object, you may select any number of objects and move them the same way.

Relationship window - Relationships

Create a relationship

Draw a line from one class rectangle to another: Move the mouse cursor to one class, press and hold the left button and move the mouse to another class, release the button.

or

Open a class' object menu and select 'Add new relationship'.

Now a dialog will open where you can edit the new relationship.

Edit a relationship

Double click on the line's arrow.

Select a relationship

With your left mouse button, click once on the arrow of the relationship line in its middle. OEW draws small rectangles at the relationship's end to visualize the selection.

Move a relationship

While a relationship is selected, click and hold with the right mouse button on the rectangle at its line end, move the rectangle to the new class and release the button.

Delete a relationship

While a relationship is selected, press the 'DEL' key on your keyboard.

Relationship window - Select class(es)

Some menu commands operate on the set of selected classes in the active window.

To select a class, click it once with the left mouse button. The class will be displayed like a pressed button.

To add further classes to the selection, press and hold the 'CTRL' key on your keyboard before clicking with the mouse.

Holding the 'Shift' key while clicking selects the class and all its subclasses.

To deselect all classes click in the background.

The 'Edit' menu contains a command 'Select All' to select all classes.

The set of selected classes in the inheritance window and in its associated 'list of classes' are identical.

You can draw a rectangle around multiple classes to select all classes contained within the rectangle.

Relationship window - Views

Relationship window - Window contents

Rectangles

Each rectangle represents a class.

Lines

Each line represents one relationship.

Texts

Each text represents either a relationship label or a global label.

Line colors

Green lines are 'inheritance' relationships. They correspond to the lines in the inheritance window.
Blue lines are 'contains' relationships. Other common names for this kind of relationship are 'whole-part', 'member class', 'includes', 'using' or 'has_a'.
Black lines are 'references' relationships.

Arrow in the middle of a line

Each line has an arrow, indicating the relationship direction.
Inheritance: A => B means B is inherited from A
Contains: A => B means A contains B
References: A => B means A references B

Line labels

Each line may have a label, depending on the option 'Show relationship labels' in the view menu. The label is the relationship name.

Line gadgets at the line's end

They visualize the relationship's cardinality.
Inheritances always have cardinality one to one, so their lines are not marked in a special way.
Standard cardinality for 'contains' and 'references' relationships is also one to one. A short line orthogonal to the relationship line is the standard display.
An additional '0' means minimum cardinality is zero. A crow's foot, three short lines from the orthogonal line to the class rectangle, means many.

Lines labeled twice

Visualizes a relationship with an associated inverse relationship.

Relationship window - Zoom

At the lower edge of the relationship window You will find two scrollbars. Use the left scrollbar to scroll the contents of the relationship window horizontally. The right scrollbar can be used to determine the zoom factor. The zoomfactor can be set between 5% (left side) and 100% (right side) in steps of 1%.

Please note that not all fonts can be resized to arbitrary sizes. If you want to zoom a window, please select scalable fonts like truetype fonts.

Relationship window - scrolling

Since the relationship diagram contained in a relationship window may be larger than the window itself, you are required to scroll the visible area to the region you are interested in. You may do so using the scrollbars at the window's right and bottom.

For quicker positioning you can use the 'overview window'. To navigate quickly to a special class use the 'list of classes' window.

Relationship window - Edit label

In this dialog You can change the text and appearence of the label.
Each label belongs to it's view and is not displayed in other views.

Reverse Engineering

Software engineering doesn't terminate when the program is finished. On the contrary, during the life cycle of complex software many modifications take place, leaving very few of its properties unchanged. Internal data and process structure are subjected to these changes in the same way as the user interface. This applies for mainframes, workstations and PCs, for individualized and commercial software.

There are a variety of reasons accounting for changes in software:

- Completing software in a given time causes many user requirements to be neglected. Later in its life cycle, these concepts (among others) are used to improve on the finished software.

- There is progress in technology. Hardware and software systems have to adapt. They must support new interfaces and devices. A database system might replace a filing system, or a new operating system might be installed. An extreme example that is currently in fashion is downsizing. Many firms replace their mainframes with workstation or PC networks to reduce expenses, even though new software might cause a short-term cost increase.

- New applications expand the requirements existing software must meet. Cost-accounting software, for example, might have to be changed to allow access by executive information systems (EIS).

These are only some of the things that result in software changes. Nowadays, software maintenance accounts for the largest part of software cost. We do not know if this will change for object-oriented systems. Consequently, speeding up program development is only one purpose of object-orientation. It is even more important to reduce maintenance costs.

Many authors on object-orientation use the term re-use when describing this requirement. What is it that should be re-used? Examples used to show the advantages of object-orientation in this area tend to be quite trivial, e.g. base classes for windowing systems. Other authors propose the re-use of the classes of a project, e.g. a class INVOICEFORM that can be used in many applications.

We believe that elaborating on such examples is missing the point. It is not very complicated to program windowing systems even in C or COBOL. The appropriate tool libraries are available on the market. They are linked to an application using the #include or Copy commands, respectively. Source generators are also used to manage windows and other "objects" in languages that are not object-oriented. Re-using existing software by simply creating a software clone is not the problem.

What software engineers desire is to be able to re-use the ideas and concepts inherent in a program. The cost distribution described above is the best example for this: understanding an application so as to be able to change it is the problem. Thus re-use means "similar to...". The revision of a project should abide by the same rules of engineering as its creation. As a consequence, we call this process Re-engineering.

The Re-engineering cycle:

1 Old application source code
2 Source code analysis
3 Model of the old application
4 Model of the new application

5 Source code generation
6 New application source code

(steps 1-3 are called Reverse Engineering, steps 4-6 are called Forward Engineering)

The first requirement for re-use of a project's structures and concepts is understanding them. A developer can use the commented source code and the documents from analysis and design to achieve this aim. If the program was built with OEW, re-engineering can use the .oew objectbase, accessing and modifying analysis and design information. In this case, re-engineering consists of the same steps as building a new project.

If only sources of a project are available, you can either modify them directly or use OEW to create an objectbase from the sources. This approach gives you visual access to the underlying concepts and structures. The reverse engineering features of OEW can be used on code from external sources as well, so you can easily visualize class libraries. You can work with any of these sources as with any other objectbase, using the tools OEW offers to support OOA, OOD and OOP. This means you have the tools you need for future maintenance of your applications now.

SQL - Edit conversion

In this dialog you define a SQL conversion. Please enter the OEW (Java) datatytpe and the according SQL type.

Example OEW type: String, SQL type: VARCHAR

SQL Options

In this dialog You can specify the name of the file into which OEW will generate the SQL database scheme. Also You can select which Syntax is used in that file.

Pressing the 'types' button opens the 'SQL typeconversions' dialog where You can edit the default conversions.

SQL support

Basics

If SQL support is selected, OEW generates a 'CREATE TABLE' statement for each class that has the 'create table' attribute. OEW uses only slots which are not marked as 'transient'.

For a complete description on how OEW generates the SQL database scheme please refer to the OEW manual.

Code generation

The generation of the Java source files is not affected by the SQL support. OEW will only generate an additional file containing the statements for the table creation. The name of this file and the used language can be specified in the 'SQL options' dialog.

The wollowing dialogs can be used to set the desired SQL flags:

SQL Options

Edit class

Edit multiple classes

Edit datatslot

Edit reference / member class slot

Edit inheritance

Default inheritance mode

SQL typeconversions

In this dialog you edit the default conversions OEW uses to convert Java datatypes into SQL datatypes. OEW uses these conversions only if there is no SQL datatype specified for a slot.

Use 'new' to create a new conversion.
Use 'edit' to edit the selected conversion.
Use 'delete' to delete selected conversions.

Select definition

You requested to jump to the symbol that is at cursor position in your current active editor.

The symbol was ambiguous, there is more than one symbol with that name. The displayed list contains all of them. Please select the symbol of your choice.

see also

Quickinfo

Select files to import

You requested to import source code files. Use this dialog to specify the files you wish to import. The contents of modules existing in the objectbase that have identical filenames will be replaced.

OEW also supports importing of comiled class files. For these files, only the declaration of the class is imported. The code is not decompiled.

On the left hand side, you see the files you've already chosen. The list is initially empty. Use drag and drop from the File Manager, or the buttons on the right hand side to add files to the list.

To add new files to your objectbase, use the 'Add' button. To add those files that already belong to your project, but have changed externally since you imported them the last time, use the 'New Mod.' button. To add all files that belong to your project use the 'Modules' button.

Use the 'Add URL' button to add an URL. This enables you to parse .class files directly from the internet.

The controls at the dialog's left hand side and the 'include path' field at its top specify which files OEW will read. 'Import selected files' only is the default choice. You may specify 'execute import', and while importing, OEW will examine the source code for imported classes and import them, too.

OEW will find these files only if they are in your project's directory, or if you entered their directories in the 'include path' field (separate the paths with semicolons). OEW imports only files it can find; it won't generate any error messages for files it can't find.

see also

Quickinfo

Select files to import - file open

Select a file you want to import. Press 'OK' to add the file to the list of files to import. This dialog will be re-opened automatically. Press 'Cancel' after you are finished adding files.

Unfortunately we were not able to provide a multi-selection list box. You may select only one file at a time to add to the list. However, we provided the possibilty to add all the files with a specific file extension. To add all files with a specific extension, select one file with the extension of your choice, check the 'All' box and press 'OK'.

Select view

You requested to select and activate a view in your current active inheritance or relationship window.

Use the list at the left, containing all available views, to select the view of your choice. If you need to define a new view, use the 'Edit' button.

You may activate a view as exclusive or non-exclusive. Exclusive tells OEW to display only classes contained in the view. Views used in the non-exclusive mode display all classes, but classes not contained in the view are displayed differently (they don't appear in three-dimensional form, but are flat).

see also

- The View Concept
- Quickinfo

Separation Instructions in Comments

When generating the documentation, OEW replaces in the output modes 'Manual Style' and 'Hypertext' separation instructions contained in comments.

Separation instructions are used to structure comments into subareas. The subareas will be separated by larger titles.

Example: If a comment contains the following text:

```
a
$New area$
b
```

it will appear in the generated documentation similar to the following:

a

New area

b

Therefore a separation instruction looks like:

```
$Title$
```

It is necessary that the $-chars are at the beginning and the end of a line. If there are other chars in front or behind them, the instruction won't be recognized.

Slot filter

You may choose to display only special kinds of slots in list of slots. In the dialog's 'Filter' section, check all kinds you want to see.

Specify a sort order for list of slots in the lower left corner, and a display mode for slots in the lower right. The display mode tells OEW what strings it should use to describe slots in lists.

Analysis does not display method parameters. All non-numeric cardinalities are displayed as many. Design displays the return type and the parameters. Java displays the slots in Java style.

see also

- Quickinfo

Slots window

This dialog is used to provide access to a class' slots, they are displayed in the list at the dialog's lower area. Double-click a slot to edit it.

The display depends on the slot filter.

For your information, at the dialog's top the class declaration is displayed. You may edit the properties using the 'More' button. Edit the class information using the 'Info' button.

Manage what and how it is displayed

Mouse operations

Slots window - Manage what and how it is displayed

Show implementation

Own slots

Slot filter

Slots window - Mouse operations

Double-click a slot to edit it.

There are drag and drog operations possible:
- Drag any number of selected slots
- Drop objects on the list of slots

You may drag and drop within the same (sort the list) or between different windows (move or copy objects).

Please note: To see the manually set order of a slot list, make sure that the Slot filter is set to 'unordered'.

Slots window - Own slots

Toggle the list contents using the 'Own slots' setting. Turn it on to display only the class' own slots, turn it off to display all the slots, including those derived from base classes.

Slots window - Show implementation

The 'Show implementation' setting controls the action that takes place when you double click a method. If it's checked, you'll jump directly to the 'implementation dialog'. Otherwise - and for non-method slots - it will open the dialog with the common slot properties.

Sort items

This dialog is used to edit the order of items within a list.

The list at the left contains all the items. Select items using the left mouse button. The buttons in the upper right corner act on the currently selected items.

The 'Sort' button sorts the items alphabetically. Exception: Slots are sorted by the current slot-filter.

The 'Group' button groups the selected items together.

You may use drag and drop to move the selected classes to a new position.

see also

- Quickinfo

Statistical information

This dialog displays statistical information for the set of classes you requested.

The data is organized in a table with four columns. The first column contains a description for the numeric data that is displayed in the other columns.

The Workbench

After you have started the OEW application (and optionally logged on), you see its main window, that is called the workbench.

It has same some components:

- Menu
- Toolbar
- Tab bar
- Statusbar
- Workspace

The kind of product that is processed by OEW, is the objectbase. The workbench allows it to have multiple objectbases opened simultaneously.

You may drag and drop files from File Manager to open an objectbase.

The waterfall model of software engineering

Most enterprises produce software to support other activities such as the production of goods and services. Just like a machine, software is a tool, an asset. This puts software production on par with the production of other assets. The planning and construction of machines is engineers' business, so why shouldn't software be engineered, too?

Among the first to investigate the problems of contemporary software production, Boehm concluded that software production must abide by the rules of engineering. He explained that the failure of many software projects of the early '70s was due to the lack of proper tools and systematic engineering. He proposed different stages of software engineering, each with a well-defined result. Development must not proceed to a new stage unless the previous stage is completed. Errors in earlier stages may become obvious at any stage, thus the back-pointing arrows. Boehm's model has been criticized, modified and expanded by many others. Based on his basic ideas we propose the following model for engineering object-oriented systems:

First of all, a business process analysis establishes the relevant business events. This gives an overview of current and planned business processes and the basic concepts and algorithms to solve the problems. We call this design-in-the-large. Now the different projects can be identified and have a priority assigned.

In order of priority, a concept of implementation is drafted for each project. This concept is presented to the experts in this problem domain and implemented with their consent. OEW's business is object-oriented analysis, design and programming. Finally, a test run under working conditions must prove that the implementation has been completed successfully.

Usually, new requirements will result when the program is in use. The time for re-engineering has come. Very thorough changes may require a new design of the whole project, smaller modifications affect the existing waterfall.

A completely new design and similar projects are sped up greatly if an old project can be partly or completely re-used. In such a case, completely implemented ideas, designs or modules can be integrated into the new project. One of the great strengths of object-orientation is that these transitions are executed very smoothly.

These files will be generated

OEW is going to generate (write) one or more source code file(s). A list of those files is displayed.
Confirm the operation using the 'OK' button.

If You don't want one or more of the listed files to be generated, select these files and push the 'remove' button.

see also

Quickinfo

Additional information:

OEW will overwrite each file listed. Changes made to them (using a file editor) will be lost. However, OEW keeps track of files (and their dates) it processed (generated or imported) as a part of your active objectbase. So it is able to recognize changes made to those files by another program. If this should be the case, OEW will give you a warning and the possibility to cancel before it overwrites.

Types of module elements

- Slot
- Comment
- Class Definition
- OEW meta command
    module comment
    autogenerated impors
  User defined
    Any code you desire, e.g. comments, import and package statements.

User management

To allow multiple users editing a single project, you may give each user an account with individual access rights. This way you control who accesses what.

Access to an object is based on the user level of its creator. Each user belongs to exactly one group, and the group defines the user level for its members. Access is granted to users with the same or a higher user level (than the creator). Objects that the current user is not allowed to modify are displayed in gray. However, you may use classes you aren't allowed to modify, for example you may create a subclass for such classes. Please note that supervisor users may access any object, regardless of their user level. Locked users are not allowed to log in.

If you want to test the access mechanism, then follow this example.

- Create three user groups (for example, Developers, Contractors and Guests). Set the levels for the groups to 900, 600 and 500 respectively.
- Add three users, one in each group. Bill belongs to Developers, David to Contractors, Justin to Guests.
- Logon as David and create the classes Wheel and Engine.
- Logon as Justin and create the classes Body and Door.
- Logon as Bill and create the classes Vehicle, Car and Truck, where both Car and Truck are derived from Vehicle.
- Now again log on as each user to see which objects you are allowed to access.

View menu - Edit view

To define views using a dialog, from the view menu choose

Edit views

See also:

The view concept

Edit views dialog

View menu - Font

Control the font that is used to display classes and slot labels in the inheritance window.

View menu - Font for classes

Choose the font used to display class names in class rectangles in the relationship window..

View menu - Font for relationships

Choose the font used for relationship labels in the relationship window.

View menu - Multiple inheritance tree display

OEW allows two ways of displaying a multiple inheritance structure.

- All classes are displayed only once, all of them labeled black (the rectangles). As a result, more than one inheritance line may connect to a single rectangle. This may be hard to read.
- Classes having multiple base classes are displayed as often as they have base classes. To mark these rectangles, they are labeled with class names in blue.

To toggle the two display modes in the class hierarchy window from the view menu choose:

Multiple inheritance tree display

View menu - Reformat like class display

Reformats your current relationship window display. The current layout will be lost. The resulting layout will be like the class layout in your inheritance window.

View menu - Select view

To activate an already defined view from the view menu choose

Select view

see also:

The View concept

View menu - Show derivations / contains / inheritance relationships

Enable or disable the display of lines for each of three relationship types. This can make a diagram more readable.

View menu - Show inheritance handles

This define whether small rectangles in the midlle of inheritance lines in the inheritance window are displayed or not.

However, even if the rectangles are hidden you may access the inheritances.

View menu - Show relationship labels

Enable or disable the display of labels at relationship lines. The labels are the relationship names.

View menu - Slot filter

Change the slot filter that controls which type of slots are displayed and in which order they are displayed in the Slots window and in the inheritance window.

View menu - Sort alphabetically

You may control the order an inheritance window uses to display the classes. To toggle between the display modes, from the view menu choose

Sort alphabetically

When activated, the root classes and subclasses are sorted alphabetically from top to bottom.

Deactivate this option to go back to your manually (or automatically) set class order.

Please note: As derived classes have to stay in the surrounding of their base classes, ordering does only take effect within the same inheritance level of one tree.

View menu - Turn off view

To deactivate the active view and display all classes contained in the objectbase, from the view menu choose:

Turn off view

see also:

The View Concept

What is OEW?

Innovative Software's The Object Engineering Workbench(TM) for Java(TM) (OEW) is a graphical design tool for Java applications. OEW is a road-tested working solution providing Object-Oriented Analysis, Design, Implementation and Documentation for Java based applications.

Automated Code Generation and full Reverse Engineering are key functional components of the OEW development environment. OEW completely supports the Java 1.0 standard. It has a built-in code manager. There is no longer a gap between model and code, they are simply two separate windowed views. The Symbolic Parser technology used in OEW allows it to handle even incomplete projects or syntactically incorrect code. OEW can import and export entire programs. In addition, OEW can generate SQL database schemes.

OEW is controlled by a powerful and intuitive "drag&drop" user interface for several graphical environments and platforms, including a context-sensitive on-line help.

For more information on OEW such as product integration notes, product documentation, white papers, reference lists, etc. please contact Innovative Software or your local distributor.

What is the support OEW offers the software engineer?

The title question can be rephrased as "What does a software engineer require?", immediately adding "Does OEW do this?" This chapter tries to answer both questions.

The first step of program development is to find a way of processing a specific task set by the design-in-the-large. This might be easy, e.g. in book-keeping, where many tomes and laws describe exactly how to proceed. Other problems require more attention. It might not even be feasible to apply solutions from theory as they are too difficult or too costly to implement. Many of these problems, e.g. the packing of containers with cargo, are solved using practical heuristics.

During analysis, smaller, independent sections of the problem are isolated. These divisions of the problem will be implemented separately. An analysis is not a description of the actual situation. It must describe "what" the computer is supposed to do, using a language understandable for both the analyst and an expert in the problem domain.

After analysis has defined the purpose of the application, the design clarifies "how" the computer should do its work. This is the technical specification of the program, preparing the programming.

While the above definitions are widely accepted, there is a controversial discussion on how to apply these concepts. Should DeMarco's Structured Analysis be applied, or Ross's SADT? Maybe Jackson's approach is the solution? These questions cannot be decisively answered, let alone in a single manual. However, it is our opinion that object-orientation is very well-suited for attaining the aims of software engineering. Meyer and Booch list various reasons backing this opinion. In addition, the inventors of SA and SD seem to be of the same opinion - at least, they are among the strongest supporters of object-orientation now.

OEW offers support for many tasks that analysts and designers have to perform. It allows graphical modelling of the object modell. All OEW objects, like classes, slots, datatypes, method bodies, etc. can be created, specified, maintained and documented through its user interface. The main goal is to increase the productivity during the development cycle. OEW can not automatically find "solutions" in a given problem domain, e.g. the right algorithms in an optimization problem like the Travelling-Salesman-Problem, but it can assist the team in developing a Java based program more quickly once the right solution strategies have been found.

Who uses OEW?

OEW accompanies its users along all phases of the software life cycle. This implies that OEW's users belong to the following groups:

- Project Managers
- Analysts
- Designers
- Programmers

Larger projects can only be completed successfully by a team. To coordinate such a team, the Project Manager requires clear and precise documentation, especially when existing code is re-used. OEW offers consistent project documentation and features especially useful for the project manager. For example, it is possible to restrict specific tasks - as the programming of a certain group of classes - to a single group of developers using OEW's access control facilities.

A Systems Analyst analyses technical or business processes. He creates an abstract model that can be partially or fully automated. This requires analyzing both the static and dynamic structures of the problem. Modeling takes place completely in OEW's class hierarchy graphs and object relationship models.

A Designer takes the abstract model from analysis and adds those classes and attributes needed to implement the model on a computer. His detailed specification of data structure and processes is required for effective programming. OEW supports the designer in many ways. The parser, for example, facilitates the use of class libraries and the re-use of existing code.

A Programmer receives both the analyst's and the designer's results. He proceeds by creating executable source code. In addition to its code generation facility, OEW offers support for compiler integration: the special features of many compilers are supported, program building can be controlled from OEW's user interface, etc.

You might also have other purposes for using OEW, for instance:

- Software Engineering Departments have to set - and control - specifications concerning the use of methods and tools. They are especially interested in well-commented results. OEW furnishes such results with its on-line documentation and print-outs.

- Trainers and teachers can use OEW to visualize object-orientation. Coaching in object-orientation is much more efficient using visual examples and the right tool than in a purely theoretical or code-based way. OEW helps you train your employees faster and more thoroughly.

Workbench dialogs

A dialog is a window that is fixed in size and displayed in front of all the other windows. OEW uses a large number of dialogs, but they are not listed here separately. Instead, access each one's help text while it is shown using the 'Help' button or the F1 key on your keyboard.

Many dialogs in OEW are modeless. That means, you may leave a dialog open and switch to another window or dialog.

Please note that the menu you can see at the workbench's top doesn't give commands to the active dialog, but to the topmost window.

Workbench menu

The workbench has a menu below its title bar. The displayed menu depends on the active window that is displayed on the workbench's workspace.

If there is a dialog active, the menu still belongs to the most recently activated window.

Take a look at the window's and dialog's title bars (at their top). The color tells you which one is active.

Workbench statusbar

The statusbar is the gray line displayed at the window's bottom. It is possible that it is deactivated so that you can't see it. Check the command 'Statusbar' in the options menu to make it visible.

On its left hand side you see some status information which contains 'OK' most of the time. While highlighting commands in the menu, or holding toolbar buttons pressed, you see here a small help text describing the object you are selecting.

On the right hand side, you see your computer's current system date and time.

Workbench tab bar

At the lower end of the workbench window above the statusbar the tab bar is located. In the tab bar there is a tab for each window opened, You can activate a window by clicking on it's tab or by pressing 'Ctrl' and the number associated with that window's tab.

Workbench toolbar

The toolbar is the button line displayed in the upper area of the window directly below the menu. It is possible that it is deactivated so that you can't see it. Check the command 'Toolbar' in the options menu to make it visible.

Toolbar buttons serve as a shortcut for often-used menu commands. Click once on a button to start its associated action. The picture in the button gives you an impression of the button's meaning.

The buttons contained in the toolbar depend on the current active window.

Workbench Windows

While working with OEW, you will use a lot of different windows and dialogs, each with a different appearance and behavior. The most important windows are

- Inheritance window
- Relationship window

These are the most important windows because you can't have an opened objectbase without having at least one of those windows open.

Other important window types are

- Overview window
- List of classes window
- Slots window
- Editor window
- Quick Reference window
- Query window
- Explorer

Automatic conversion to members and references

When a class type is entered into the objectbase OEW checks whether this type has already been used as a slot type before. If so, it converts the plain slot type to the appropriate member class or reference type. As an example, the slottype c is changed to reference to c as soon as the class c is entered into the objectbase.

Deleting a class deletes all slots of the member class or reference types referring to it.

container classes and the corresponding code

OEW offers a facility to effectively manage container classes in a Java project. When generating code for a data slot with multiple values, Suppose the class Vehicle has defined a relationship to the class Tire:

Generating code yields the class declaration

```
class Vehicle
{
        List /*<Tire>*/ Tires;
};
```

OEW has generated a comment specifying which type of object the container holds. Otherwise, this information would be lost when the file is parsed again, destroying the relationship between VEHICLE and TIRE.

Parsing containers

When parsing OEW can only translate container classes to those cardinalities already defined in the objectbase. Therefore, all the cardinalities used in the project have to be defined before parsing starts.

how to assign access rights in a multi-developer project

OEW's user administration is a valuable assistant in the management of multi-developer projects as it offers the following items:

There can be any number of user groups.

Each group has an access level from -999 to +999.

Each user is a member of a group. He can access the objects of the objectbase at the access level of his group.

Each object can be accessed for reading by any developer.

A user can only modify an object if it was created by his group or by a group of lower level.

Supervisors can modify all objects.

The presumed development team has two developers, Mr Flynn and Mr Tracy, as well as a coordinator, Mrs Hepburn. Mrs Hepburn has decided to strictly limit the access of each developer to the other's class to read-only. She has assigned the classes to the developers as follows:

Developer	Classes
Flynn	SuperClass
Tracy	SubClass

Mrs Hepburn asks a supervisor to create three user groups for this project: Project_Coordinator, Project_Developer1 and Project_Developer2. The groups have the following attributes and members:

Group name	Members	Level
Project_Coordinator	Hepburn	10
Project_Developer1	Flynn	0
Project_Developer2	Tracy	0

This level assignment assures that

The developers can only read, but not modify each others results.

The coordinator can read and modify both results.

In the example, the analysis was completed by a single developer, the classes and modules have to be specifically aassigned to their developers using Edit/Edit access.

Classes that a user isn't allowed to modify are displayed slightly different than standard classes, they are displayed in light gray.

Do not forget to assign the classes and the modules to their users! A user cannot modify a class if he cannot modify the module it is located in.

how to split a project among multiple developers

There are two reasons why it can be desirable to split an OEW project into multiple objectbases:

The simplest reason is that the objectbase has too many classes to manage efficiently in a single .oew file. Even though OEW's view concept largely offsets this situation, some functionalities of OEW slow down, e.g. saving the objectbase. Splitting up the objectbase is a remedy for this kind of problem.

Another even more important reason for dividing a project among various objectbases is having many people work on the same project. OEW does not support access to an .oew file by more than a single user. However, if the project is devided into multiple objectbases, it can be used by a team of developers, each accessing a single objectbase.

A large project consists of different components, each of which corresponds to a distinct set of classes. A small project will have some classes providing basic functions, some which govern the essentials of the application, and the classes of the user interface. Larger projects can be further divided.

Multiple developers working on a project should also have distinct responsibilities. Each developer should be responsible for a well-defined set of classes. This structure can be mirrored in the design of the objectbases the project is comprised of. If user management is used in addition, each developer has "his" set of classes in "his" objectbase. He can access all the other objectbases to see which classes and slots are defined there, but cannot modify them. His part of the application interfaces with the other objectbases through the use of library classes.

For example, consider a project with two classes only, SuperClass ---- SubClass.

As this project consists of two classes, it can be divided among two objectbases. The first objectbase declares the class SuperClass and the corresponding module, superclass.java. As a superclass is independent of its descendants no further modifications are required.

The second division of the project declares the class SubClass. In its declaration the SuperClass is needed. As it is declared in the other project, it has to be a library class here.

Generating code now produces the same results as when there was only one objectbase for the project.

relationships and the corresponding code

Most relationships are visible as inheritance lines or slots in the class hierarchy. When such a relationship is deleted, the effects are immediately visible:

- An inheritance line vanishes.
- The slots corresponding to a relationship are deleted from their classes.

This obvious fact has a more subtle counterpart: Whenever a class is deleted from the objectbase, all its relationships lose their meaning. Consequently, OEW deletes all the slots modeling relationships to and from this class. It can sometimes be quite astonishing to see slots disappear from a class when another class is deleted!

specifics of OEW's Symbolic Parser(TM)

OEW serves as a system to efficiently manage Java projects in an object-oriented manner. This aim differs significantly from a compiler's, where executable code has to be generated. Consequently, the Symbolic Parser does not parse source code as a compiler does. This fact can be used to work even more efficiently with OEW if you are aware of its advantages and drawbacks.

Missing header information

First of all, the code OEW reads does not have to be as complete as the code a compiler needs to build an application. For example, OEW does not have to read the classes imported by the code parsed.

However, OEW cannot produce information out of thin air. In the case of automatically added library classes , there is no way to find out which file contains its declaration. Therefore, OEW assigns the class to the file where it was found (used)..

Parsing Libraries vs. manually entering library classes

As stated in the previous paragraph, parsing with execute imports can create many undesired objects,resulting in chaos in a formerly orderly objectbase. The safest route is not to import libraries to set the filenames of library modules at all.

If you heed this advice there is only one time when you import a class library: when you wish to examine it. Usually this means an import of the complete library.

When such an import is complete, save the objectbase of a library you want to use! It is a priceless repository for looking up the filenames of the modules it contains.

Unfinished projects

OEW's parser does not require a syntactically correct project to work. This fact mekes it possible to import a partly-finished project, e.g. when a project is updated in OEW to continue working after correcting some mistakes in a compiler IDE. As with library classes, the rules OEW uses to place the code parsed are based on reasonable guidelines.

This tolerance for errors is limited, however, when there the errors occur in the declarations describing the class hierarchy. OEW has to display each detail of these declarations, so an error here becomes immediately apparent in the display. This is especially true if a { or } is missing. The example for this leads to a totally garbled class hierarchy:

Admittedly, there is a way of realizing there was a mistake: counting braces. However, it would not really help as this error cannot be corrected automatically. OEW tries to parse smoothly, ignoring the error.

Using the include path during import

The import source dialog's include path entry field is deceptive. Copying the compiler's CLASSPATH setting usually results in long imports with many classes. Since this is not always desired it is important to know how to properly use OEW's include path.

Our guidelines are simple:

Save before import.

Use import selected file(s) only as often as possible, e.g. when updating debugged projects. The new mod. button shows exactly which files to import after a change.

Clear the include path whenever using 'execute imports' or import all. These options may come in handy if you have changed some header files.

There are two important applications where you do not leave the Include Path blank:

¨ When parsing a class library, use execute imports or import all. Make sure only the directories of this specific class library are in the Include Path.

¨ When parsing a complete project, make sure only the directories of the project are in the Include Path.

The statements above are applications of the basic rules:

Make sure only the directories you need are included.

Make sure these directories contain no files that you do not want.

In case of doubt, create a temporary directory where you move all the files you are not sure about.