



Version 4.0
System Administrator's Guide

MetaCase Document No. SAG-4.0

Copyright © 2004 by MetaCase Oy. All rights reserved

First Printing, 2nd Edition, May 2004.

MetaCase
Ylistönmäentie 31
FIN-40500 Jyväskylä
Finland

Tel: +358 14 4451 400

Fax: +358 14 4451 405

E-mail: info@metacase.com

WWW: www.metacase.com

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including but not limited to photocopying, without express written permission from MetaCase.

You may order additional copies of this manual by contacting MetaCase or your sales representative.

The following trademarks are referred to in this manual:

CORBA and XMI are registered trademarks and UML and Unified Modeling Language are trademarks of the Object Management Group.

HP and HP-UX are trademarks of Hewlett-Packard Corporation.

Linux is a registered trademark of Linus Torvalds.

MetaEdit+ is a registered trademark of MetaCase.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

Motif is a trademark of the Open Software Foundation.

Pentium is a trademark of Intel Corporation.

Solaris and Sun SPARC are registered trademarks and Java is a trademark of Sun Microsystems.

UNIX is a registered trademark of X/OPEN.

Product support

Available to sites with a current maintenance agreement or 90 day limited warranty. If you purchased MetaEdit+ through a distributor, you must contact that distributor for assistance.

For product support please use the following options:

- The FAQ pages at www.metacase.com provide answers to the most common questions, and the support pages provide downloadable patches that upgrade your MetaEdit+ environment with new features and corrections.
- E-mail: metaedit.support@metacase.com,
- Fax: +358 14 4451 405, or
- Telephone: +358 14 4451 401 (6.00–14.00 GMT/UTC)

When you contact our support team please have ready,

- Your license number,
- Meplus*.err file(s) from your MetaEdit+ directory,
- MetaEdit+ version number and patches loaded (use **Help | About**),
- Hardware platform (memory size and disk space),
- Operating system and its version,
- Network information, if applicable,
- Printer information, for printing problems,
- Steps that led to the problem, and
- Can the problem be reproduced on more than one machine?

Preface

Thank you for purchasing MetaEdit+[®], a new generation metaCASE tool. MetaCase sincerely hopes that this tool will offer you the functionality you need for your work.

PURPOSE AND ORGANIZATION OF MANUAL

This system administrator's guide provides a thorough introduction to the tasks of the system administrator in MetaEdit+, as well as a reference manual covering the complete set of concepts, features, and commands relating to MetaEdit+ system administration.

This manual has three chapters and an index:

- **Introduction** to the basic tasks and concepts of MetaEdit+ system administration (Chapter 1).
- **MetaEdit+ client administration** (Chapter 2), describes the functionality available to the system administrator in a MetaEdit+ client, single-user or multi-user version.
- **MetaEdit+ server administration** (Chapter 3) discusses the system administration operations related to the MetaEdit+ server that is required for multi-user version.
- **Index**, containing an alphabetical reference to all MetaEdit+ system administrator functions and commands.

Together with this system administrator's guide two additional manuals from MetaCase Consulting may be needed: 'MetaEdit+ User's Guide' and 'MetaEdit+ Method Workbench User's Guide'. The former manual introduces you to the standard CASE features of MetaEdit+. The latter manual introduces you to the metaCASE features of MetaEdit+, and describes how you can customise and develop your own methods within MetaEdit+.

AUDIENCE

This user's guide is intended as a reference for MetaEdit+ system administrators. For installation you should have an additional set of instructions. It is assumed that the system administrator is familiar with the concepts and basic usage described in the other MetaEdit+ manuals.

Please note that in this manual it is assumed that you have a working knowledge of the operating system you use. Whilst it is not necessary to be a system administrator for your operating system to be a MetaEdit+ system administrator, such knowledge would be useful. For an introduction and guidance about your platform and operating system, please see the manuals that came with your system.

USAGE OF THE MANUAL

MetaEdit+ is a dynamic product under continual improvement, and there may occasionally be some differences between what is described in the printed manual and what is found in the current version. The HTML-based manual, however, is updated for any new changes or improvements as it comes together with the software installation package and patch files. Therefore, you are strongly advised to use the HTML-based version of the manual.

Conventions

Throughout the manual, you will find special notes and comments that point out important features and characteristics of the MetaEdit+ environment. These notes are printed in *italics* and are marked by an arrow (→) in the left margin. The steps required for performing MetaEdit+'s various functions are indented and numbered: 1), 2), 3) etc.

List dialogs

MetaEdit+ makes extensive use of list dialogs for selecting among elements. To quickly select a known element in the list, simply type the first few letters of that element's name when the dialog opens. This moves the cursor to the first element whose name begins with those letters. Pressing enter will choose the framed element, closing the dialog. Pressing space selects the framed element, and resets the typed buffer, so you can start typing a different name. You can also double click an element to choose it and close the dialog.

Some dialogs allow multiple selections: use shift-click or shift-space to select a contiguous section of the list, and control-click or control-space to select individual elements. Again, a double click first performs the selection operation (modified by shift or control keys), and then closes the dialog.

The Windows user interface standard prevents resizing of modal dialogs, which can make life difficult if not everything is visible in the default size. To help in such situations, MetaEdit+ includes a triangular resize corner at the bottom right of most dialogs. By clicking and dragging the resize corner, you can resize the dialog window to be larger.

Contents

1	Introduction	1-1
1.1	Responsibilities	1-1
1.2	Repository Concepts	1-1
1.3	Repository Files	1-1
2	MetaEdit+ Client Administration.....	2-1
2.1	Repository Administration.....	2-1
2.1.1	Databases Roots File.....	2-1
2.1.2	Database Roots Browser	2-2
2.1.3	Creating a New Repository	2-3
2.1.4	Converting Repositories.....	2-5
2.1.5	Backups.....	2-6
2.1.6	Reconstructor	2-6
2.2	MetaEdit+ System Administration	2-7
2.2.1	User management.....	2-8
2.2.2	Repository Options	2-10
2.2.3	Working with projects.....	2-11
2.3	Importing and type deletion	2-12
2.3.1	Importing methods and models.....	2-12
2.3.2	Removing types	2-15
3	MetaEdit+ Server Administration.....	3-1
3.1	Repository Files on Unix	3-1
3.2	Starting the Server Launcher.....	3-4
3.3	Creating a New Multi-user Repository	3-4
3.4	Starting a server for a repository	3-4
3.5	Client Login	3-5
3.6	Server Browser	3-6
3.7	Suspending a server	3-7
4	Index.....	4-1

1 Introduction

In this introductory section we will familiarize ourselves with MetaEdit+ system administrator's responsibilities and some of the basic concepts of the ArtBase repository system employed by MetaEdit+ and its management. The rest of this manual is divided into two sections that describe the client and server system administration tasks respectively.

1.1 RESPONSIBILITIES

The responsibilities of the system administrator in MetaEdit+ are two-fold, reflecting the distinction between the client and server side administrative tasks. Regarding the MetaEdit+ client and repository, the system administrator is responsible for such tasks as creating new users, importing models and methods, and probably also for making backups of the repository at appropriate intervals. For the MetaEdit+ server, the administrator's tasks include setting up and starting up the server for a multi-user repository, and making sure all clients can connect to the repository. Also, to be able to troubleshoot problems arising from the technical environment, the administrator should be familiar with the operating system on which the clients and server will run.

1.2 REPOSITORY CONCEPTS

A *repository* or database is the largest unit of data in MetaEdit+: there may be several repositories, but no data in one repository can be used or referenced directly from a second repository. A repository is composed of *areas*, which correspond to the MetaEdit+ user-visible concept of *project*.

In a multi-user environment, each repository requires its own *MetaEdit+ server* program to be running for it. In general, each site will only be using one repository at once: each MetaEdit+ server program running requires a server license from MetaCase Consulting. In a single-user environment there is no server: the MetaEdit+ client accesses the database files directly.

The server for MetaEdit+ is a separate program with its own user interface. A server must be running for a repository whenever a client tries to login to that repository. The server thus normally runs on a machine that is routinely kept on all the time. As the server program user interface requires a user to be logged in to the operating system, this machine is normally a workgroup server, rather than a departmental or larger server.

1.3 REPOSITORY FILES

The repository files are generally arranged in a single directory hierarchy. The highest-level directory contains one file `artbase.roo` that contains the names and paths of all the

repositories, and a subdirectory containing each of these repositories. Initially there is probably only one repository, but later you may make backup or checkpoint repositories, copies of your main repository made at a certain time.

In a repository directory there are two files, **manager.ab** and **trid**. The **manager.ab** file contains the names and paths of all the areas in that repository, the names and encoded passwords of all the users in that repository and any disk name mappings needed to access the repository. The repository directory also contains several subdirectories:

areas	Contains a directory for each physical area in the repository
users	Contains a directory for each user in this repository
backup	Contains a copy of the repository from the last successfully committed transaction
comm	Only created when a multi-user repository is used. Contains information about multi-user access using file communication.
counters	Only created when a multi-user repository is used. Normally empty.

2 MetaEdit+ Client Administration

This section describes the extra facilities available in MetaEdit+ that belong to the system administrator. These facilities include:

- Repository administration (Section 2.1), for administration of whole repositories,
- System administration (Section 2.2), for administration within a single repository, and
- Importing methods and models from other repositories (Section 2.2.3).

2.1 REPOSITORY ADMINISTRATION

In this section we examine the system administration actions that operate at the level of whole repositories: creating a new repository, backing up and restoring repositories, and reconstructing damaged repositories. All these functions can be accessed from the MetaEdit+ Startup Launcher as shown in Figure 1.

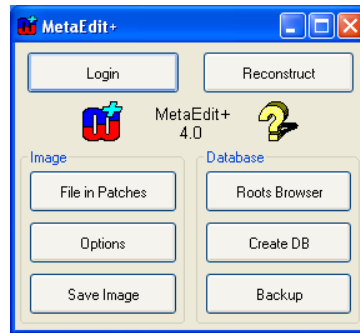


Figure 1. Startup Launcher.

2.1.1 Databases Roots File

The Databases Roots file, `artbase.roo`, consists of a list of the available MetaEdit+ repositories and their paths. A repository must be listed here for you to be able to log in to it. Initially there is only one repository, 'demo', supplied with MetaEdit+. Note that you may have other backup or checkpoint repositories to maintain a history of the evolution of your repository, or even different repositories. Initially it is best not to have several different repositories in use, as it is not possible to share data in a real-time manner between repositories: an important feature of MetaEdit+. Instead, create several projects within the same repository.

The path for each repository should point to the directory where that repository's info file, `manager.ab`, is. With the single user version, the paths to the repositories should generally be relative paths from the MetaEdit+ startup directory. In the multi-user version, the paths to the repositories are generally fully qualified, e.g. UNC paths or mounted drives or directories.

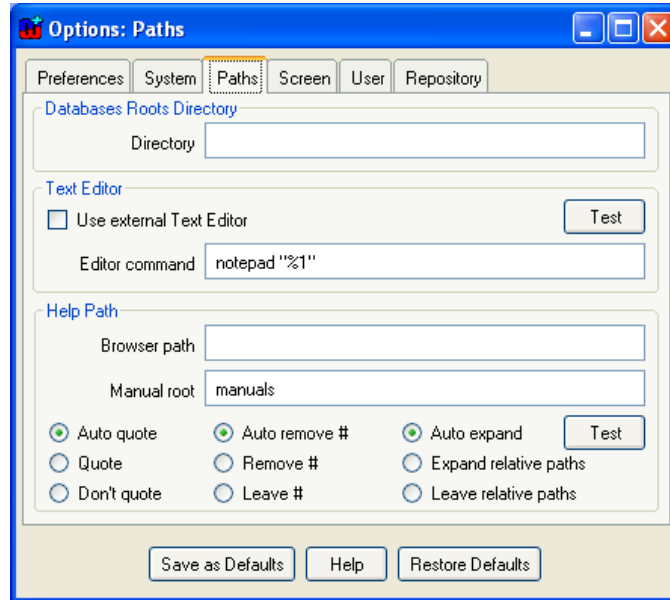


Figure 2. Paths page in Options Tool.

To set the path where the client can find the `artbase.roo` file, open Options Tool by pressing **Options** button in the Startup Launcher and go to the **Paths** page (as shown in Figure 2). On top of the page, enter the valid path into the text field called **Databases Roots Directory**.

2.1.2 Database Roots Browser

To view and edit the Databases Roots file, `artbase.roo`, use the Database Roots Browser (shown in Figure 3). The Roots Browser can be opened either from the MetaEdit+ Startup Launcher (press **Roots Browser** button) or from the **Maintenance** menu in the MetaEdit+ server (see Section 3.4).

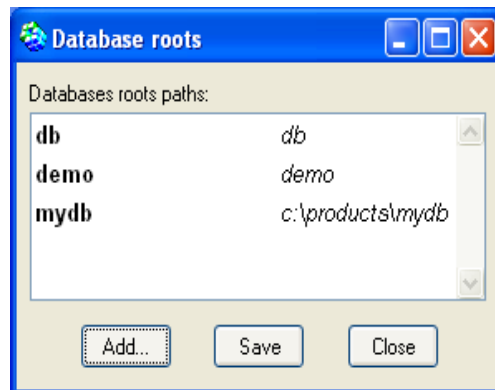


Figure 3. Roots Browser.

In the Roots Browser, the following operations are available:

rename opens a dialog where you can change the name of the selected database.

change path opens a dialog where you can change the path of the selected database: note that this does not move the database.

You can also double-click a database in the list to edit both the name and path of the selected database.

edit manager.ab opens a text editor window on the `manager.ab` info file of the selected database. This file lists all the users, areas and network disks translations, plus the name of this database, its path, and any server running on it. Each file name is preceded by a letter, indicating which platform's conventions the name obeys, e.g. **F** for FAT file names, **N** for NTFS file names, or **U** for Unix file names. If you use only relative paths, with 8 or fewer characters, lower case, and no / or \ path separators, this letter will not need changing. When you save the file (from the pop-up menu), you will be prompted for the password of `sysadmin`, the first user, according to the encoded password in your *edited* copy of the file. If the password is not valid, a dialog warns that the file was not saved.

add DBS or the **Add...** button prompts you for the name and path of a database to add to the list.

remove DBS removes the selected database from the list: note this does not affect that repository itself.

save accepts the changes you have made, and saves them to the `artbase.roo` file. Note that this is necessary after any changes, except for editing a database's `manager.ab`.

cancel rejects the changes you have made, and reloads the contents of the list from the `artbase.roo` file.

2.1.3 Creating a New Repository

To create a new repository, you should start MetaEdit+, but do not log in yet. First determine which path you want to create the repository into: if the directory does not yet exist, it will be created, provided that its parent directory already exists.

→ *In the multi-user version of MetaEdit+, new repositories are created from the server: see Section 3.3. After that basic creation, you can proceed by pressing **Login** in the Startup Launcher, and choosing the repository you created in the server, marked New Database in the dialog list of repositories. A dialog will open and you can continue from step 3) below.*

- 1) Press the small 'down arrow' button to show the full Startup Launcher, then press **Create DB**.
- 2) A dialog will open (Figure 4) prompting for the basic information of the new repository, with fields for the repository name and path, and for the system administrator user name and password. Fill in each field, moving to the next with the tab key or mouse. The name of the repository is that which will be shown to the user: there are no restrictions on the format of the name. The path should in general be a relative path, and you should make sure that the path you enter is legal in all client platforms that will access the repository.

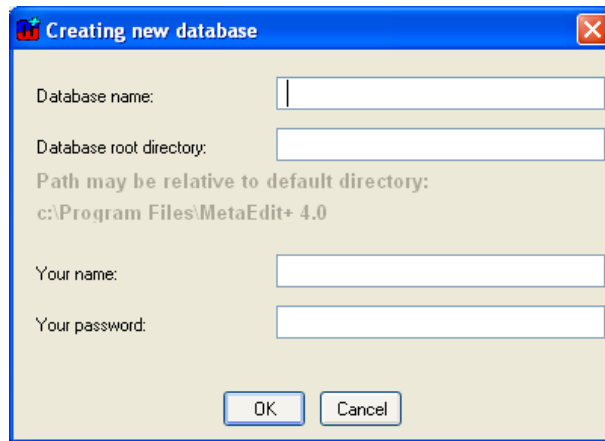


Figure 4. Creating new database.

- 3) Fill in the fields for the system administrator name and password. In general, the name for the system administrator is 'sysadmin', rather than any particular user's name.
- 4) When you have filled in all the fields, press **OK**, and you will be prompted to enter the password again for confirmation.
- 5) The new repository will now be created and added to the Database Roots file, `artbase.roo`. You are now logged in to the repository as the system administrator, and the basic information you have entered is committed.
- 6) Login proceeds as normal, opening the main MetaEdit+ Launcher, and asking for which projects to open and choose as default project. As no projects exist yet, just press **Cancel** in those dialogs.

The repository is now created. However, it is normal to continue and define at least one extra user and one project, and set metamodeling options, before logging out. You should be careful working with MetaEdit+ in a repository with no projects open, especially if no projects exist. In particular, you should avoid trying to open editors on models or metamodels. It is thus safest to create a project now.

- 1) In the Main Launcher, choose **Repository | Options** and select the **User** page. Choose **New User...** (see Section 2.2.1) from the pop-up menu to create a new user.
- 2) Now move to the **Repository** page, press **Metamodellers** (see Section 2.2.2) and choose those users who will have permission to metamodel.
- 3) Set the metamodeling security level from the pull-down list (see Section 2.2.2).
- 4) Now would be a good time to create some projects: at least one should in general be created. In the Main Launcher, select **Repository | New Project...** A dialog will appear informing that the current transaction will be committed during the creation of a new project. Proceed by answering **Yes**. You will now be prompted for the name of the new project.
- 5) Your transaction will be committed, and the project will now be visible in the list of open projects.

After performing the above operations you can commit and log out, or continue by making metamodels: until metamodels exist, it is of course not possible to create any models. You can also import a metamodel patch that was previously exported with the Type Manager in another repository. If for instance you wanted to make a new repository containing just the

SA/SD and UML types from the demo repository, you could select their top level graph types in the Type Manager and export them. You could then create a ‘types’ project in your new repository and import the patch to there.

If you wanted to maintain the project structure (shared types in mcc, a project each for SA/SD and UML), you would use **Select All Types in Project...**, and choose and export mcc, SA/SD and UML in turn. You should then create corresponding project names in the new repository and import these patches in the same order. Finally, since mcc also contains some types that are only used by metamodels other than SA/SD and UML, you could commit and then use a Type Manager in the new repository to delete these unnecessary types.

2.1.4 Converting Repositories

Sometimes it is necessary to move repositories between different operating system platforms or make conversions between single or multi-user repositories. MetaEdit+ handles most aspects of these kinds of situations automatically, but there are still a few things for the system administrator to know regarding these issues.

→ *Remember to make sure that all clients are closed and users logged out, suspend the server and backup the database before carrying out any operation explained below.*

Moving Repositories Between Platforms

To move a repository to another platform:

- 1) Copy the repository directory and files within to the target platform. If necessary, check and set the access rights for the repository files to match the target platform after copying.
- 2) Check and change the path names in `artbase.roo` and `manager.ab` to follow the target platform’s conventions (see Section 2.1.1).

Converting a Single-User to a Multi-User Repository

In principle, no conversion is required to use a repository created with a single-user version of MetaEdit+ via a server as a multi-user repository. However, make sure to carry out the platform conversion operations, if the repository was moved to a different platform.

Converting a Multi-User to a Single-User Repository

To convert a multi-user to a single-user repository:

- 3) Start the server and log in as `sysadmin` with a multi-user client, but do not open any projects.
- 4) Choose **Repository | Options** to open the Options Tool, and go to the **Repository** page.
- 5) Press the **Convert to Single** button and answer **Yes** to the dialog that appears. This will consolidate all the different users’ committed transactions together into a form that the single user version can read: without this the single user version will see an ‘old’ version of the data in the repository.
- 6) Close MetaEdit+, committing changes if asked, and suspend the server.

You can now make a copy of the database files, and that copy of the database can now be used with the MetaEdit+ single-user version. You can also restart the server on the multi-user repository.

2.1.5 Backups

It is a good idea to take backup copies of your database at regular intervals. Make sure no users are logged in when you take the backup. Similarly, in the multi-user version, the server should not be running when the backup is taken. If it is, when you restore the backup and attempt to start up a server on the database, you will be prompted to reconstruct all users who were logged in, and the changes from their open transactions will be lost.

Backups can be taken either manually via your operating system (which will normally be faster), or using the **Backup** button in the MetaEdit+ Startup Launcher. If you make the backups manually, you should include all files in the database directory (**manager.ab** and **trid**), plus all directories there (**areas**, **users** and **backup**), their subdirectories and files. Make sure that any empty directories are included and restored as empty directories.

Backups with the **Backup** button simply take a copy of all the files in the database directory and its subdirectory. For this reason you should be careful not to use root directories — or any other directory that contains files — as database directories, or as the directory into which the backup copy is made. The **Backup** command also adds an entry into **artbase.roo** for the backed-up copy of the database: it prompts for a name for the backup.

2.1.6 Reconstructor

Most problem conditions that can arise are handled automatically by MetaEdit+ and ArtBase. For example, if a user uses emergency exit, his user details in the repository are left in an inconsistent state; however, the next time he logs in, his user is automatically reconstructed.

Hardware or software failures can however cause the database to end up in an inconsistent state. The ArtBase Reconstructor, available from the **Reconstruct** button in the Startup Launcher, can help in these situations. To apply the Reconstructor you must first select a repository and open it with the user name and password. All users can reconstruct their own user account as described in the 'MetaEdit+ User's Guide', but other reconstruction operations require system administrator rights.

The Reconstructor shows a list of all users and a list of all areas. The radio buttons allow you to select an action to apply to the selected users or areas. First choose an action, then select the items to apply that action to, and finally press the **Apply** button.

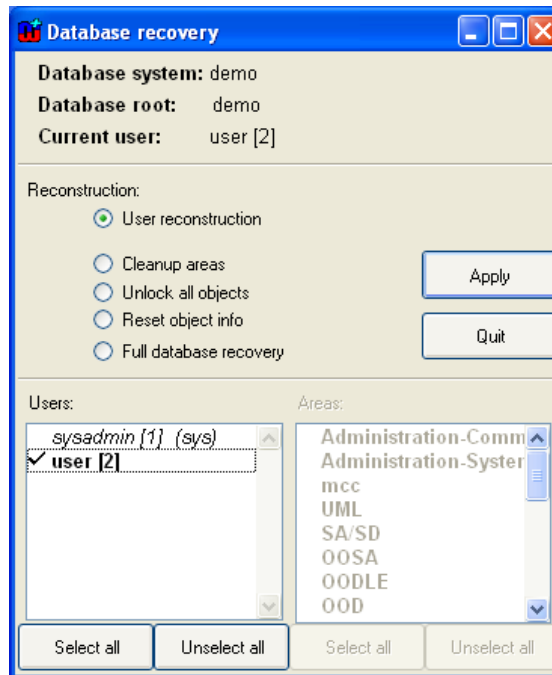


Figure 5. Repository reconstruction.

User reconstruction will rebuild the user information for the selected users, so that they can log in again.

Cleanup areas will delete any temporary files that may have been left in the selected areas' directories.

Unlock all objects will remove all locks from all objects held by the selected users in the selected areas, useful if a user has crashed in the database, and his locks have not been released.

Reset object info will delete any dynamic files (*.cif) that may have been left in the selected areas' directories.

Full database recovery will delete the current database files, replacing them with those from the automatically created backup directory. As this destroys data, you should take a backup of the whole database first. Note that full database recovery is different from restoring the whole database from a backup copy that you have created: full database recovery copies files from the database's own backup directory, i.e. the state after the last successful commit, whereas restoring from your own backup replaces both the current files and the backup files from your copy of the whole repository, at the state when you made that backup.

2.2 METAEDIT+ SYSTEM ADMINISTRATION

In this section we examine the administration tools for use within a single repository. To use the system administrator tools, you must start MetaEdit+ and login as `sysadmin` to a database. If you will be working solely with database data such as users and projects, there is no need to open any projects. If you will be deleting metamodel information with the Type Manager tool, you should open all projects.

2.2.1 User management

All user management tasks can be carried out from the Options Tool's **User** page. This page shows a list of all users for the current repository (Figure 6).

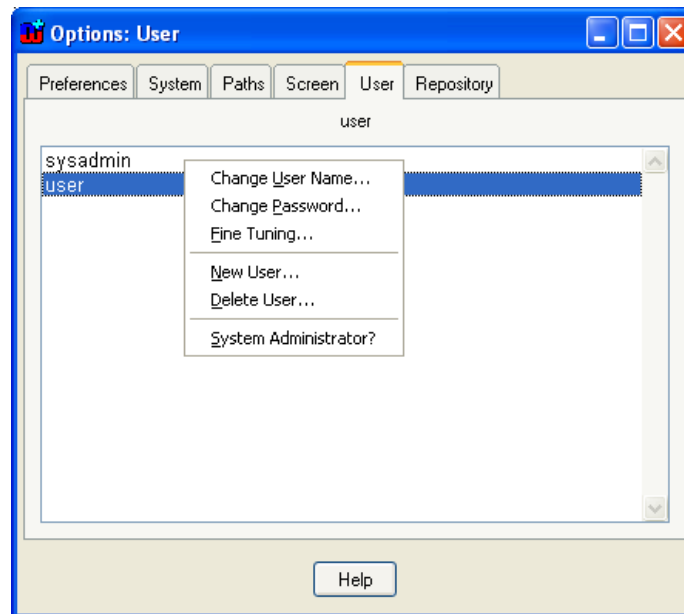


Figure 6. User page in Options Tool.

Each user is allowed to change his own user name, password and fine tuning settings. A system administrator can also create or delete users and grant or deny system administrator privileges for other users.

Change User Name

To change a user's name, select the user in the list and choose **Change User Name...** from its pop-up menu. Enter the new name into the dialog that opens.

Change Password

To change a user's password, select the user in the list and choose **Change Password...** from its pop-up menu. Enter the new password when prompted and re-enter it for confirmation in the next dialog that opens.

Fine Tuning

As computer power and in particular memory size constantly increases, we recommend checking the Fine Tuning settings for your users, and increasing the values for MaxPersistentObjectSize, MaxImageSize, MinObjectAmount, MinPersistentObjectsSize..

The ArtBase repository system used by MetaEdit+ maintains a cache of all objects read from the database, to speed up access times by reading objects from memory rather than across the network or from disk. There are various settings that affect the size and behaviour of this cache that can be accessed by Fine Tuning dialog. To open this dialog, choose the user whose settings you want to view or change from the list and select **Fine Tuning...** from the pop-up menu.

The default ArtBase settings (shown on the right) are the defaults for all users in each new repository. For a large repository, these default values may be somewhat low. The values of

greatest interest are those beginning with **Max** or **Min**: these specify the bounds above which objects will be flushed from the cache, and below which objects will not be flushed from the cache.

The current size of the image, and amount and size of persistent objects in the cache, can be seen by selecting **Repository | Statistics...** in the main Launcher. By checking those values for a 'typical' session, and knowing the amount of physical and swap memory on a user's platform, you can estimate efficient values for the settings.

In general, if you are running reports on a database that is much larger than your available physical memory, and the operating system seems to be using disk swap space all the time, you should allow the ArtBase cache to be flushed earlier, by decreasing the **Min** and/or **Max** values. Similarly, if the ArtBase cache seems to be flushing often (visible in the Repository Transcript in the Options Tool, and also as a pause of several seconds while no actions are possible), and you have sufficient physical memory, you should try to prevent the ArtBase cache flushing so soon, by increasing the **Min** and **Max** values.

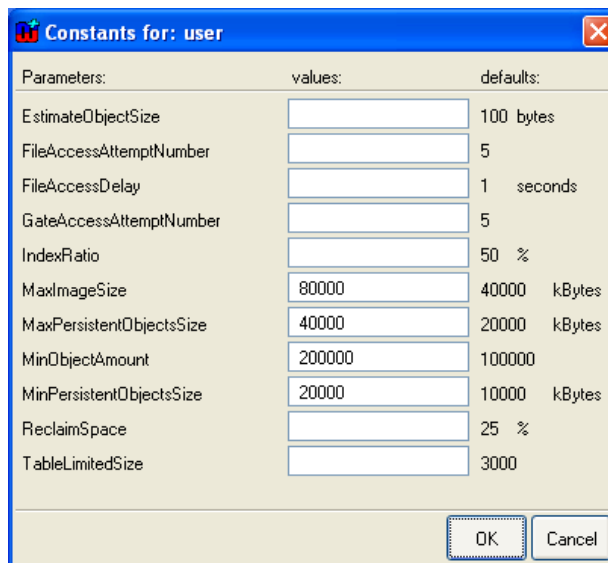


Figure 7. Fine tuning cache and memory.

New User

To add a new user, select **New User...** from the list's pop-up menu. Enter the name and password for the new user in the dialog that opens. When prompted, re-enter the password to confirm it.

Delete User

To delete a user, select the user in the list and choose **Delete User...** from the pop-up menu. Answer **Yes** to the dialog that prompts to confirm the delete operation.

System Administrator

To grant or deny system administrator privileges for a user, select the user in the list and toggle the **System Administrator** menu item in the pop-up menu.

→ *Note that these system administrator rights are no way related to those of your operating system: they exist purely within a MetaEdit+ repository.*

2.2.2 Repository Options

In the main Launcher select **Repository | Options** and choose the **Repository** page of the tabbed window that opens. Most actions here are only available to the system administrator, but normal users can check the completeness of method specifications, and view settings concerning the repository.

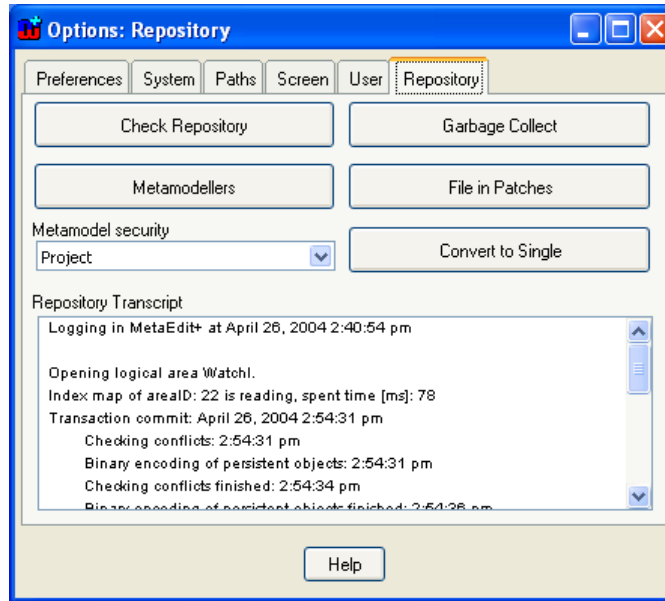


Figure 8. Repository Options page.

Check Repository

Check Repository reports on types which do not have complete specifications, e.g. do not have an identifying property or symbol. The incomplete types are reported in a list dialog, from which they can be selected for editing.

Metamodelers

The system administrator specifies which users have the right to metamodel (**Metamodelers**). This includes not only using the features of the Method Workbench version of MetaEdit+ to create and modify types, but also the metamodeling features of the standard version of MetaEdit+: changing symbols, property dialogs, and saving reports.

→ *It is possible to write and run reports without saving them in the repository: this does not require metamodeling rights.*

Enforcement of metamodeling security level and metamodeler rights is by means of locks. The lock is only taken at the time the changes are saved: users can open metamodeling tools without acquiring any locks or having metamodeling rights, but they will be unable to save any changes they make. In the standard (non-Method Workbench) version of MetaEdit+, the metamodeling tools exist, but all save options are greyed out, i.e. unavailable.

Metamodel security

MetaEdit+ allows you to modify methods, even while you or other users are using them, constantly making sure that the repository is kept consistent. However, it can be rather unsettling to users to find the methods they are using changing under their feet! To control

this situation, the system administrator can define the **metamodel security** level: are other users allowed to be logged in while someone is metamodeling. Currently there are three settings, **exclusive**: the metamodeler must be the only user in the database, **single**: there can be only one metamodeler in the whole database at a time, but any number of simultaneous non-metamodeling users, and **project**: only one metamodeler can access a specific project, but any number of non-metamodelers may work in the project, and other metamodelers may work in other projects. The initial default is **exclusive**.

Garbage Collect

This option will condense the database by removing all old versions of objects, and perform a global database garbage collect, removing all objects from the database that are no longer used, e.g. all deleted objects. The net result is a smaller, faster database. Because of the amount of data that must be read and maintained in memory, this is both a time consuming and memory intensive operation. We advise taking a backup of your repository before performing garbage collect.

File in Patches

There are two types of patches in MetaEdit+: image patches update the MetaEdit+ program itself, and repository patches import methods and models and thus update the Object Repository. All users can file in image patches, but only a system administrator can file in repository patches.

The process of filing in image patches is described in the MetaEdit+ User's Guide. The use of repository patches is described in detail in Section 2.3.1.

Convert to Single

To convert a multi-user repository for single-user use, press the **Convert to Single** button and answer **Yes** to the confirmation dialog. For more information about converting repositories, see Section 2.1.4.

Repository Transcript

The **Repository Transcript** text box shows the log of the repository operations for the current session. Please note that the transcript is not updated in real time, so to view possible recent events, activate some other page in the Options Tool and then go back to the **Repository** page.

2.2.3 Working with projects

Projects are the way the MetaEdit+ repository arranges type and instance data internally. Each piece of data is stored in exactly one project in the repository, but can be referred to by elements stored in other projects. When such a reference is followed and the project of the referred element has not been opened explicitly, this project will be opened seamlessly in the background.

There is no separate tool for managing projects as these tasks have been incorporated in the normal Project list found in the Graph Browser, Type Browser, etc. Use any of these as your starting point for managing projects. Creating, opening and closing projects are tasks allowed for all users. Renaming projects, however, requires system administration privileges.

Creating projects

To create a new project, select either **Repository | New Project...** from the main Launcher's menu bar or **New...** from the pop-up menu for the current browser's Projects list. MetaEdit+

will first ask you to accept the committing of the current transaction (answer **Yes**) and then prompts for a name for the new project. Once you have entered the name, MetaEdit+ will commit the changes to make the project creation permanent and shows a dialog reporting that a new project has been created and opened.

The subdirectories and files for the new project will have names based on just the alphanumeric characters in the project name.

Opening projects

To open one or more projects, select either **Repository | Open Project...** from the main Launcher's menu bar or **Open...** from the pop-up menu for the current browser's Projects list. You will be presented with a list of possible projects to open. Choose the desired projects from this list and press **OK**.

Please note that sometimes projects can be opened in the background, although they have not yet been explicitly opened and hence do not appear in the list of open projects in browsers. Such projects are marked with an asterisk '*' in the list of possible projects to open.

Closing projects

To close projects, select **Close...** from the pop-up menu for the current browser's Projects list. You will be prompted with a list of open projects. Choose those projects you want to close from this list, and press **OK**.

Renaming projects

Unlike other project related tasks, certain restrictions apply for renaming projects:

- Renaming requires system administration privileges.
- Renaming is an exclusive operation, i.e. the system administrator must be the sole user logged into the repository when this task is carried out.
- Only projects that have not been open during the current transaction can be renamed.

To rename a project:

- 1) Make sure that other users are not logged in to the repository and that you are logged in as a system administrator.
- 2) If the projects to be renamed are open, close them and start a new transaction either by committing or abandoning the current one.
- 3) Select **Rename...** from the pop-up menu for the current browser's Projects list.
- 4) A list of possible projects opens. Choose the project you want to rename from this list.
- 5) Enter the new name for the project when prompted. After this, MetaEdit+ will show a dialog reporting that the renaming was successful.

2.3 IMPORTING AND TYPE DELETION

2.3.1 Importing methods and models

In MetaEdit+ the system administrator is responsible for importing methods and models exported from other repositories into his repository. This section describes the importing functions and gives guidelines for applying it in typical scenarios.

Method patches and model patches

There are two types of repository patches that the system administrator can file into the repository: method patches and model patches. Method patches include only types, and are made with the Type Manager (see 'MetaEdit+ Method Workbench User's Guide'). Model patches include instances (design data) together with their types (method). This patch is made with the Graph Manager (see 'MetaEdit+ User's Guide'). Both types of patch are imported in the same way.

Repeated export and import

Importing will affect large amounts of data, and must perform complicated mappings from the structure and contents of one repository to another. It is thus worth spending time making sure you understand a little of what happens during import, especially if you want to repeatedly export and import models from one repository to another, called the source and target repositories here.

The target repository remembers from which source repositories imported instances have come, so if the same graphs are exported again later, the existing imported instances in the target repository are updated, rather than creating new 'duplicate' instances. Previously imported instances will thus be updated in the correct project automatically, if the source repository name given when exporting the previously imported patch was the same as for this patch. Exporters should thus take care to always use the same, unique name for the source repository when exporting.

Recognising previously imported types to be updated is based on unique internal names, stored with each type, and is thus not dependent on the name given for the source repository when exporting. However, if there are instances of port types within type definitions, these instances are handled in the same way as other instance data and the behavior is therefore affected by the source repository name. When importing, you are expected to first open all the areas where types to be updated exist. If a type is imported that already exists in another, unopened project, MetaEdit+ will warn you and ask for permission to open that project.

Whilst updating types works between any repositories, updating instances works only in one direction: from the source (exporting) repository to the target (importing) repository or repositories. It is not possible to update the exported models in the source repository by importing the same models back from the target repository.

Similarly, if A exports to B, and B exports to C, it is not possible to update models in C by directly exporting from A. Instead, B must import the models from A, and then re-export them for C to import. In many situations, it would be better for A to export one patch at the start, which both B and C could import. Later, A could export the newer versions of the same models, and again both B and C could import that same patch.

Instructions for importing

The normal state for importing is to have just logged in: if you are already logged in, and have a sufficiently recent backup of the repository, you can continue from step 3. Make sure however that no other users are logged in, and no editor or browser windows are open during import.

- 1) Before importing either kind of patch you should take a backup of your target repository.
- 2) Start MetaEdit+ and log in to the target repository as system administrator. You must be the only logged in user.

- 3) Open all projects containing types (at least those types that might possibly also be in the patch to be imported), and also those projects where you want new types and/or instances to be created.
- 4) Select a default project: you will be prompted during import to choose a default project for new types, and a possibly different default project for new instances.
- 5) Select **Repository | Options** in the Main Launcher and press the **Repository** page's **File in Patches** button. A file dialog opens, and you can select the patch file to be filed in.

After the import, briefly check the imported models and types before committing. Check especially that new types and instances have been created in the projects where you expected them, and that previously imported instances have been updated, rather than new 'duplicate' instances created (which would indicate that the name given for the source database when exporting this patch was not the same as for the previous patch).

→ *Note that all your designs (instances) — also those that were not imported — are changed to reflect changes in the method (types) in the same way as if the changes had been made with the metamodeling tools.*

Import-Export Scenarios

This section describes some scenarios of how the import/export functionality might be used:

○ Simple model exchange

If one user simply wishes to give a model to another user, he can export it to a file and the other user can import it, even if the second user has totally different methods in his repository. In that case the second user should be aware that the types of the imported methods will be created in the current default project in his repository. He may wish to create a new project to hold the first user's models and methods.

○ One source, many targets

A common situation is that one developer or organisation is working on a set of core models, and other developers or organisations are each working on their own sets of models, which reuse and refer to these core models. The core developer or organisation can export the current state of their repository (the source repository), or just the relevant parts of it, to a file. This file can then be imported into an empty repository, and the resulting repository (the target repository) can be given to each other developer or organisation. (If the others have already started work, the file is simply imported into their existing repositories).

Work continues in both source and target repositories. Whilst the core developer or organisation need not see the changes made by the other developers or organisations, the latter must keep their repository up to date with the changes made to the core models. Thus the core developer or organisation periodically exports the changed or new core models to another file, and this is then given to each other developer or organisation, who import it into their repository. When exporting, the same name should be given each time for the core (source) repository, so that imports into target repositories update the core models there rather than creating new ones.

○ Versioning

If a model created in a repository is exported and then imported to the same repository, it creates a full copy of that model and all its components, but uses the existing types. This is useful for making versions: the current state is exported and considered as frozen as a version in its project, a new project is created and the file imported with the new project

as default. This creates a new working copy in the new project, and development can continue with that. This process can be repeated as often as necessary, always exporting the current working version and thus freezing it, importing it into a new project and continuing work in that new project. As the same data is never exported twice, it is in principle safe to give the same source repository name each time when exporting; in practice, it may be wiser to give a different name, avoiding problems if you export from a frozen project by mistake.

→ *There is absolutely no connection between the data in the frozen version and the new working version: changes in the new version will not affect the frozen version. Users should no longer change the frozen version; their work from now on should happen in the new project.*

Hints for importing and exporting

Exporting uses large amounts of memory. If you see a lot of cache reclaiming, try setting higher values in Fine Tuning (see 2.2.1).

You can export and import from the same source repository to the same target repository many times; subsequent imports update existing imported objects, providing the database name specified when saving the model patch was the same each time.

You cannot import changes back from the target repository to the source repository (this just creates new copies the first time, then updates them on subsequent occasions).

There is no warning if existing types are to be replaced by ‘smaller’ or ‘older’ versions, causing existing property data from your own models to be lost.

There is possibly a danger of Graph Manager updates never ending if cache reclaim happens often — be patient though, as it often takes a while, especially if there are many models.

Graphs not selected on the left in the Graph Manager, but reachable from those selected (e.g. explosion, decomposition, reused elements), are exported as empty graphs, i.e. with just their properties, but no contents. The same is true for graph types in the Type Manager.

2.3.2 Removing types

The Type Manager is a tool for exporting method specifications, and deleting no longer needed parts of methods. Here we focus on deleting types, which requires system administrator rights: exporting methods is available to all users and is covered in the Method Workbench User’s Guide.

Deletion of methods or parts of methods — types — that you do not need reduces the size of your database and thus improves speed, in particular the time taken to open a project. This permanent removal is in addition to the removal of types that can be done from the Types Browser: removing a type in that way only removes it from most system dialogs, and it remains in the database, and can be salvaged at any time. Type deletion with Type Manager, on the other hand, is a once and for all deletion operation, and as such **we advise taking a backup of your repository before performing type deletion.**

The Type Manager can be opened from the main Launcher by choosing **Metamodel | Type Manager**. The Type Manager shows all the methods and their component types in currently open areas. You can see the methods on the left, and expand the display with the check box underneath the list to show all the component techniques (Graph types) of those methods. The Type Manager works on the principle that the user can be sure what methods he needs, but

may be mistaken in thinking that a given type is *not* needed: that type may in fact be used in some complex way from another type that he does want.

In the list on the left in the Type Manager you choose all the methods and techniques that you want (initially all are chosen), and press the ‘→’ button. MetaEdit+ calculates the dependencies between types: this involves loading all those types, so will take a while the first time. The list on the right then shows all the types in the open areas, with a check mark if they are required by the methods you have selected on the left. Types in the list on the right that have no check mark will be deleted when you press the **Delete Unselected Types** button.

Before deleting types, make sure that the graph types unchecked in the left list are also unchecked in the right list. If a graph type that you thought would be deleted is still checked in the right list, you can choose **Show Type Users...** from its pop-up menu in the right list, and will see which types you have chosen to keep still refer to it. Most often, they will refer to it by explosion or decomposition links, and you may want to remove these links from the chosen types, to allow this unwanted graph type to be deleted.

You can use the Type Manager freely to look at types and their dependencies, without deleting anything. When you want to delete, you should make sure that you have all projects open, and you must be the only logged in user.

All instance graphs whose types are to be deleted will be removed, and explode or decompose links to those graphs will be removed. Under normal circumstances, this will ensure that there are no instances left in the repository of types that are to be deleted. You are however responsible for making sure that no other instances of types to be deleted exist. This problem occurs most often with object types previously in a graph type, but now removed: as no graph type refers to that object type, it would be deleted, but it may still have instances in graphs made when that object type was legal.

The types will then be deleted in two phases, with a transaction commit at the end of each phase. You will then be prompted to exit and restart MetaEdit+, to ensure that no references to the deleted types are present in your image, and to perform a database garbage collect, which will reclaim the space occupied by the deleted types.

3 MetaEdit+ Server Administration

The server files, normally including the repository, must be installed on a machine that all client machines will be able to access. A list of currently supported server/client combinations is available from MetaCase Consulting, but at the time of writing a server may run under Unix and Windows NT/2000/XP. All client machines must have read and write access for the repository files and directories. Similarly, all file names must be appropriate for all client and server platforms to be used.

All the files necessary for running the server image (`mep4serv.im` image, and `visual` on Unix platforms or `visual.exe` on Windows platforms) should be placed in the repository root directory. Only the system administrator need have access to these.

3.1 REPOSITORY FILES ON UNIX

On Unix, the file system group of all client users must be the same, and be the group owner of the database files and directories. This is because shared access to files in the MetaEdit+ repository is by group permissions: the only way to allow several users to access a file, but not all users.

The simplest way to accomplish this is to assign all users of multi-user MetaEdit+ to the same group, as defined by their primary group in `/etc/passwd`. In this case the access rights for directories and files should be as follows:

- directories: `rwrxwxr-x`
- files: `rw-rw----`

Please note that it is not sufficient enough to simply create a new group (e.g. 'metagr') in `/etc/group` and add MetaEdit+ users to that, because although users could read files in the MetaEdit+ repository on the basis of their being in the metagr group, and the repository files being owned by that group, when a user writes a file (create or change) the file takes on the group ownership of the user's primary group. Thus the group ownership of the files in the repository would change as they were used, meaning other MetaEdit+ users (not part of that user's primary group) would no longer be able to access those files.

This situation can however be annoying: MetaEdit+ users probably did not all have the same primary group in `/etc/passwd` prior to MetaEdit+ being used, and forcing changes to their primary groups may produce problems elsewhere. To overcome this problem, you might want to explore the `setgid` bit feature that allows file system to use directory's own group instead of user's primary group in write operations. Instructions for this are in the next section.

Using Setgid and /etc/group

To find out whether your Unix OS supports the `setgid` bit feature, you must have a user who belongs to two groups, e.g. 'someuser' in groups 'main' (defined in `/etc/passwd`) and 'other' (defined in `/etc/group`). Make two directories owned by him and his 'other' group, and make

one setgid. Create a new file in each directory, and see whether the one in the setgid directory is owned by the 'other' group. E.g.:

```
su
cd ~someuser
mkdir setgid nosetgid
chmod g+s setgid
chown someuser.other setgid nosetgid
su someuser
cat >setgid/tmp
typesometext
^C
cat >nosetgid/tmp
typesometext
^C
ls -lg setgid/tmp nosetgid/tmp
exit
rm -R setgid nosetgid
exit
```

If the 'ls -lg' step gives group ownership for `setgid/tmp` as 'other' (and `nosetgid/tmp` as 'main') then the setgid bit will work on your system as described above. If you have users on non-Unix platforms you should also check that this behavior is the same from their OS through their network software.

When a directory has setgid on, files written by a user in that directory will have the same group owner as the directory, rather than the primary group of the user. Thus we can create a new secondary group (e.g. 'metagr') in `/etc/group`, add all MetaEdit+ users to it, set the group owner of the MetaEdit+ repository directories and files to 'metagr', and set the setgid bit for all repository directories and subdirectories (including the root directory where `artbase.roo` is).

E.g. if `artbase.roo` is in `/home/meshare`, add the following line to `/etc/group`, replacing the number with a new unique group number, and the users `mepuser1` etc. with the names of the MetaEdit+ users (of course, since you are making changes to the system and your MetaEdit+ repository, you should be careful, log what you're doing and have backups).

```
metagr:*:12345:mepuser1,mepuser2,mepuser3
```

Then become root, go to the directory where `artbase.roo` is, set the ownership, permissions and setgid bit for that directory, `artbase.roo`, and recursively for all database subdirectories and files there. Change the names of the user and database directories (i.e. those containing `manager.ab`) appropriately. The example below is for `csh` in SunOS 4.1.3 - you may need to make changes for your shell and OS, e.g. for `sh` use `for dbdir in db1 db2 db3 do` `done` etc.

```
su
cd /home/meshare
chown mepuser1.metagr . artbase.roo
chmod 770 .
chmod 660 artbase.roo
foreach dir (db1 db2 db3)
  chown -R mepuser1.metagr $dir
  chmod -R 660 $dir
  find $dir -type d -exec chmod 770 {} \; -exec chmod g+s {} \;
end
```

The net result should be something like this:

```
$ >ls -ldg . artbase.roo db1 db1/areas db1/areas/mcc db1/manager.ab
drwxrws--- 2 mepuser1 metagrp 512 Oct 17 23:14 .
-rw-rw---- 1 mepuser1 metagrp 2342 Oct 17 23:14 artbase.roo
drwxrws--- 2 mepuser1 metagrp 512 Oct 17 23:14 db1
drwxrws--- 2 mepuser1 metagrp 512 Oct 17 23:14 db1/areas
drwxrws--- 2 mepuser1 metagrp 512 Oct 17 23:14 db1/areas/mcc
-rw-rw---- 2 mepuser1 metagrp 2412 Oct 17 23:14 db1/manager.ab
```

You should of course test these changes by using MetaEdit+: make sure all users can work with the repository, and that when they commit their changes, the directories and files for changed projects/areas do not lose their permissions, group owner or setgid bit (the user owner will change to be that of the user who last committed a change in that area, but that is normal). In particular, you should make sure if you have users on non-Unix platforms that the changes made from their OS through their network software do not upset these settings.

Using MetaEdit+ with Samba

The Samba suite is a set of protocols and programs that enables sharing of files and printers between Unix and Windows in a network environment. In some cases it may be required that a MetaEdit+ repository will be located on such a shared resource. The typical scenario is that the MetaEdit+ repository is installed on a Unix server and its home directory is then shared via Samba for Windows-based workstations to use. In this section we explain what kind of Samba configuration is needed for this kind of installation. However, we do not cover the basics of installing the Samba suite itself here, so you are expected to have some previous knowledge about Samba.

First you need to have the MetaEdit+ repository installed in your server. This may be accomplished by following the standard MetaEdit+ installation procedure or by manually copying the necessary files into a suitable directory. Make sure that the file access rights are set correctly as explained above.

The next thing to do is to set up the directory sharing for the MetaEdit+ repository. Let us assume that in our Unix server we have a general MetaEdit+ home directory `/home/meshare` containing one or more repositories we would like to have a remote access to. The solution here is to share `/home/meshare` via Samba. To do this, we need to edit the Samba configuration file `/etc/samba/smb.conf` (the file path may vary on different platforms) and add the following definition to the 'Share Definitions' section in `smb.conf`:

```
[meshare]
    comment = meshare
    path = /home/meshare/
    public = yes
    writable = yes
    inherit permissions = yes
    security mask = 0
    map archive = no
```

By this definition we now share the `/home/meshare` directory under the alias `meshare` that allows the network users to access the files within `/home/meshare` from their workstations. We want this directory to be publicly available with write access and this is why we have set both `public = yes` and `writable = yes`. It is, however, important to ensure that server-side file access permissions remain correct if our workstation platform is different from the server platform. Basically, what we want to do is to retain all the access rights that we set originally according to the guidelines given in the previous section. In our example the following three lines in the sharing definition take care of this:

```
inherit permissions = yes
security mask = 0
map archive = no
```

The three lines ensure that: the permissions for a new file are set according to the parent directory's permissions; the permissions of a file can not be changed by client software; and the DOS archive bit is not mapped to the Unix executable bit (as Samba would otherwise do by default). These settings may vary depending on your setup. For further information on how to modify the share options, please consult the Samba documentation.

3.2 STARTING THE SERVER LAUNCHER

To start the Server Launcher:

- On Windows platforms, run `visual.exe` with `mep4serv.im` as an argument.
- On Unix platforms, run the `mep4serv` executable with `mep4serv.im` as an argument.

3.3 CREATING A NEW MULTI-USER REPOSITORY

The procedure for creating a new repository differs slightly in the single and multi-user versions: in the multi-user version the creation is started from the server, whereas the single-user version it is started from the client. Here we describe the operations in the server. After this you should continue as described under client administration in Section 2.1.3.

In the Server Launcher, press **Server Start Up**, and choose **New database** from the pop-up menu that opens. You will be prompted for the new repository's name and root directory. The name of the repository is that which will be shown to the user: there are no restrictions on the format of the name. The path should in general be a relative path, and you should make sure that the path you enter is legal in all client platforms that will access the repository. Next you will be asked for the socket number, and should accept the default given (initially 6666) unless you know this is already in use. The server is now started on the new repository, and you can proceed as described in Section 2.1.3.

3.4 STARTING A SERVER FOR A REPOSITORY

When starting the server for a repository for the first time, select **Maintenance | Set Databases Roots Filename**, clear the directory field and accept `artbase.roo` as the filename. Next select **Maintenance | Browse Databases Roots**, and a Databases Roots Browser will open (see Section 2.1.1). There should be one repository shown with its path. Select this and from the pop-up menu change the path of the repository to one which will be accessible by all clients, normally an absolute network path such as a UNC or mapped drive or directory. Choose **save** from the Roots Browser pop-up menu, and close the roots browser.

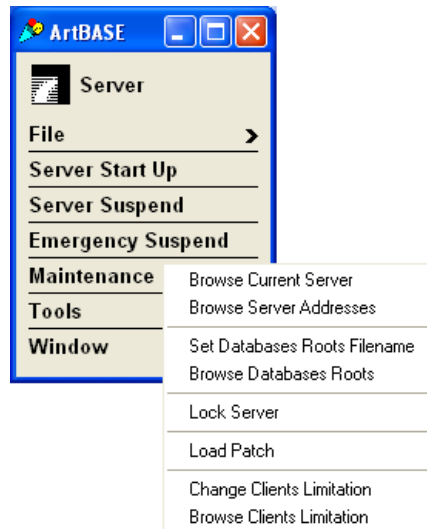


Figure 9. Server maintenance menu.

Now choose **Server Start Up**, and select the name of the repository from the pop-up list that appears. You will be prompted for a socket number, and should accept the default 6666 unless you know this is already in use. You will be prompted for your login name (sysadmin in the default 'demo' repository) and your password (initially also sysadmin). These can be changed later.

At this stage the server has been started on the repository, and nobody is logged in: the password request was to verify your authorization to start the server on that repository.

3.5 CLIENT LOGIN

You should now start a MetaEdit+ client, either on the same machine or another machine. Login as system administrator to the database for which you started the server, but do not open any projects. You can also perform the other system administration activities that are common to both single and multi-user versions of MetaEdit+, as described in Chapter 2. These include creating new users, assigning a metamodeling security level, defining which users can metamodel, and filing in database patches. Finally, logout by closing the MetaEdit+ Main Launcher, and commit your changes. The database is now ready for other users to log in to.

The client can employ two methods to communicate with the server. It first tries to set up a socket connection and if this fails it will establish a file-based method of communication. The most common cause of a client failing to open a socket connection, and thus using the slower file communication, is a problem in DNS. MetaEdit+ tests whether socket communication is possible by trying to obtain an IP number for the name 'localhost', which should always give the IP of the client computer. If the DNS server cannot provide an answer, the test fails. It is, however, also possible that current operating system uses another mechanism to implement the name lookup, e.g. WINS – although this may be slower, as lookup probably first tries true DNS and then falls back to WINS. Also, when a server is running on a database, the last line of `manager.ab` includes the server name in the format the server application receives from then operating system. All clients must be able to resolve this name to the IP of the server.

As a last resort, one workaround for DNS problem is to add an entry for 'localhost' to your client's local 'hosts' file (or similar), assuming of course that your network software uses such a file. The IP number should in general be 127.0.0.1 and not the actual IP of your client computer. The line to add will often look like this (the separator character is a tab):

```
127.0.0.1    localhost
```

There are also certain other conditions that may hinder the connection performance. If there is communication over a non-LAN network, the ping time may become a bottleneck: each object read requires a message from client to server and a reply. When logging in and opening a project thousands of objects are read, each taking roughly double the ping time. Thus in non-local networks the ping time, not the bandwidth, is normally the limiting factor. Over a LAN, the limiting factor is normally the CPU speed of the client as it reconstructs the database objects from the data received over the network.

Another issue to check if poor performance is encountered is the behavior of your anti-virus software. A bug in certain versions of Norton AntiVirus in the SmartScan feature is known to cause a delay of 1 second for each access to any file over a network connection.

3.6 SERVER BROWSER

Choosing **Browse current server** from the **Maintenance** menu of the server launcher opens a browser on the current state of the server, showing logged in users.

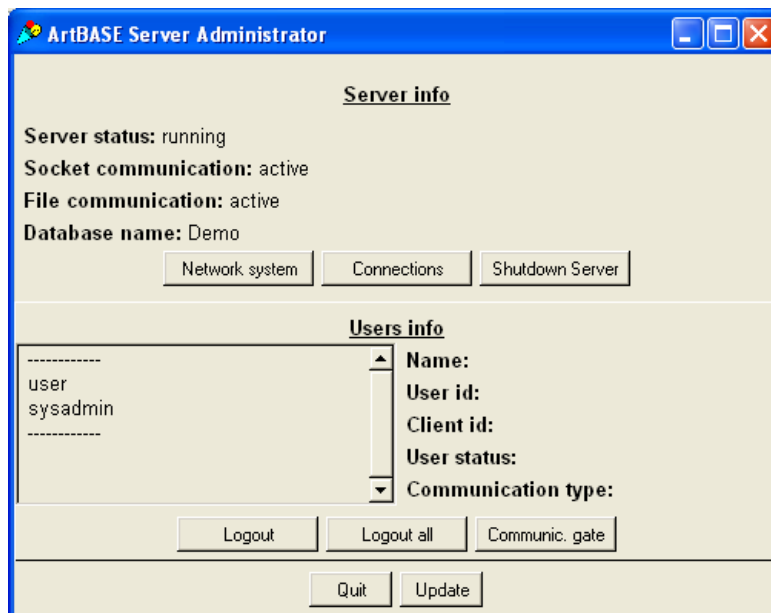


Figure 10. Server Browser.

By selecting a user you can see the current state of that user on the right. Pressing **Logout** will logout the selected user. **Communication Gate** shows the details of the communication protocol being used between the selected client and the server: this can be either socket or file communication — socket communication is generally noticeably faster.

Pressing **Connections** will open a window with two lists, the first showing users connected to the repository via the socket gate protocol, the second those via the file gate protocol: users in

long transactions are not shown, as they have no current connection to the repository. The popup menu in each list allows you to browse the selected user's connection, or disconnect it.

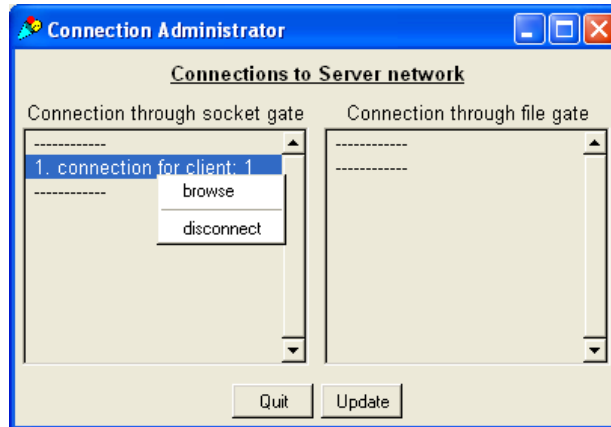


Figure 11. Connection Administrator.

Browsing a connection opens a Communication Gate Administrator for that connection, where you can see the details of the connection, including the network address of the client. Pressing **Release** will release that communication gate, useful if a gate remains open when a user is in fact no longer logged in, e.g. if his client has crashed.

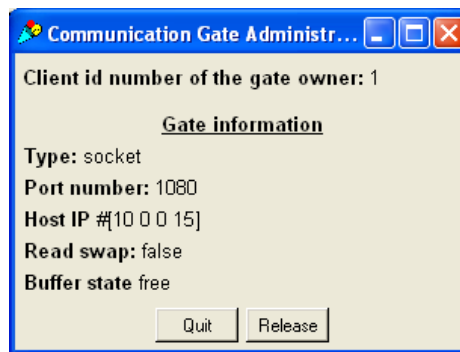


Figure 12. Communication Gate Administrator.

3.7 SUSPENDING A SERVER

Each server can only run one repository at a time: to run another repository, you need to start another server, or to suspend the current server first. Similarly, the server can not remain running if the server program is closed (via **File | Exit**) or killed. The server requires its windowing session to be and remain alive.

If you do not wish to remain logged in e.g. overnight, you must suspend the server by choosing **Server Suspend** from the Server launcher, and then close the server program by **File | Exit**. The server should then be restarted the next time you log in.

4 Index

—A—

access rights
 Unix file permissions, 3-1
 area, 1-1
 physical, 1-2
 artbase.roo, 1-1, 2-1, 2-2

—B—

backup, 1-2, 2-6, 2-7

—C—

Check repository, 2-10
 checkpoint. *See* backup
 Cleanup areas, 2-7
 client, 3-1, 3-5
 Communication Gate, 3-6
 Communication Gate Administrator, 3-7
 Connection Administrator, 3-7
 crash, 2-6

—D—

Databases Roots Browser. *See* Roots Browser
 Databases Roots Filename. *See* artbase.roo

—F—

file communication, 3-6

—G—

garbage collect, 2-11
 group
 Unix, 3-1

—I—

importing, 2-12

—M—

manager.ab, 1-2, 2-1, 2-6
 editing, 2-3
 Metamodel security, 2-11
 Metamodellers, 2-4
 Metamodelling rights, 2-10
 metamodelling security level, 2-4

Methods
 remove, 2-15

—O—

Options Tool, 2-10

—P—

patch
 image, 2-11
 repository, 2-11, 2-13
 path
 absolute/relative, 2-3
 translation, 2-3, 3-1
 permissions. *See* access rights, Unix file
 permissions
 Product support, v
 project, 1-1, 2-11
 closing, 2-12
 creating, 2-4, 2-11
 opening, 2-12
 renaming, 2-12

—R—

remove
 methods, 2-15
 types, 2-15
 repository, 1-1
 converting from multi to single-user, 2-5, 2-11
 converting from single to multi-user, 2-5
 creating, 2-3
 creating multi-user, 3-4
 files, 1-1
 moving to another platform, 2-5
 patch, 2-13
 reconstructing, 2-6
 transcript, 2-11
 Repository page. *See* Options Tool
 Reset object info, 2-7
 Roots Browser, 2-1, 2-2, 3-4

—S—

Samba, 3-3
 server, 1-1, 3-1
 exiting, 3-7
 launching, 3-4
 starting for repository, 3-4

Index

suspending, 3-7
Server
 maintenance menu, 3-5
Server Administrator, 3-6
Server Browser, 3-6
socket communication, 3-5, 3-6
sysadmin, 2-7, 3-1, 3-5

—T—

Type Manager, 2-15
Types
 remove, 2-15

—U—

unlock objects, 2-7
user, 1-2
 change name, 2-8
 change password, 2-8
 creating new, 2-9
 fine tuning settings, 2-8
 granting administrator privileges, 2-9
 removing, 2-9
User reconstruct, 2-7

—V—

Versioning, 2-14