ToolManager

An extension for the Amiga Workbench

Version 2.1
16 May 1993

1 Important notes

Welcome to the wonderful world of ToolManager 2.1 :-)

- ToolManager and its concepts have drastically changed (see Appendix B [History], page 30) since the release 1.5.
- Starting with the ToolManager 2.0 release, this program has a *GiftWare* option. If you like the program and use it very often, you should consider to send a little donation to honor the work that the author has put into this program. I suggest a donation of US $10-$20 or 10-20 DM. Please don't send cheques or money orders from outside Europe, because most often cashing those items costs more than what they amount to.

  If you don't send the donation or can't afford it, you needn't feel bad about it. Please send me a note saying that you are using ToolManager anyway (I like to get fan mail :-). See Chapter 2 [Authors address], page 2.
- Users of ToolManager 1.X/2.0 can start with the quick installation chapter (see Chapter 3 [Quick installation], page 3). Some features haven't changed and the rest is fairly easy to find out by trial & error. For a detailed description of the new concept & features browse the reference part of this document (see Chapter 8 [Objects], page 12).

  You **must** remove any running ToolManager 1.X/2.0 or the new version won't work. The new version cannot read the old 1.X configuration file format (Sorry).
- First-time users should read the entire document to understand the concept and purpose of the program. Start with Chapter 4 [Introduction], page 4.
- ToolManager 2.1 uses some features of AmigaOS Release V38 (and higher) and it supports the new AmigaOS networking features, which will (hopefully) be available soon to all Amiga users. If you are still using Release 2.0 (referred to as V37 in this document), you need not worry since ToolManager doesn't rely on these features. All extended features are marked in this documentation.

2 Where to send bug reports, comments & donations

The author can be reached at the following addresses:

Postal address:

Stefan Becker
Holsteinstrasse 9
5100 Aachen
GERMANY

Please use the following address after the 1-July-93:

Stefan Becker
Holsteinstrasse 9
52068 Aachen
GERMANY

InterNet Electronic Mail:

stefanb@pool.informatik.rwth-aachen.de

3 How to install ToolManager 2.1 the fast way

The basic ToolManager 2.1 installation consists of the following four parts:

'`Libs/toolmanager.library`' ⇒ '`LIBS:`'

This is the main program of ToolManager. It handles all programs, menus, icons and docks (see Chapter 10 [Library], page 23).

'`Prefs/ToolManager*`' ⇒ '`SYS:Prefs`'

This is the editor for the configuration (see Chapter 9 [Preferences], page 19).

'`WBStartup/ToolManager*`' ⇒ '`SYS:WBStartup`'

With this utility you can start and stop ToolManager. If it resides in the WBStartup drawer, ToolManager gets always loaded when your machine boots up.

'`L/WBStart-Handler`' ⇒ '`L:`'

This program starts programs by the Workbench startup method. It is a seperate process, so that you can quit ToolManager even if you have still programs running that were started by it with the WB method.

After copying these files, you should quit any older version of ToolManager running on your machine and double-click the ToolManager icon in the '`WBStartup`' drawer. Now you can start the preferences editor and play around (Use the "Test" button instead of the "Use" button while testing). You should be able to figure out most features with trial & error, for further information look into the ToolManager object descriptions (see Chapter 8 [Objects], page 12).

The distribution includes an example configuration file called '`TM_Demo.prefs`'. You can load it into the preferences editor with the `Open` menu item.

4 What is ToolManager?

ToolManager is a flexible program to manage the tools in your working environment. It can start Workbench and CLI programs, ARexx scripts and generate HotKey events. It even can issue commands to a ToolManager running on a remote machine. The user interface consists of menus, icons or dock windows. If you like a noisy computer, you can associate a sound to each of these items. See Section 8.3 [Sound], page 14.

ToolManager can add items to the Workbench `Tools` menu. If you select such a menu item, the program associated with it will be started. Every selected icon on the Workbench will be used as an argument for the program. This feature is only available when the Workbench is running. See Section 8.4 [Menu], page 14.

ToolManager can add icons to the Workbench window. When you double-click such an icon, the program associated with it will be started. If you drop some icons on this icon, the program will be started with these icons as arguments. This feature is only available when the Workbench is running. See Section 8.5 [Icon], page 15.

ToolManager can create a dock window from a collection of programs. This window can be opened on every public screen. Each program is represented by an image or a button gadget. To start a program you simply click on the image or the button gadget. If the dock window has been opened on the Workbench screen and the Workbench is running, you can also drop some icons on the image or the button gadget to start the program with arguments. See Section 8.6 [Dock], page 15.

Additionally you can assign a Hot Key to each program. If you press this Hot Key, the program will be started. Note that *no* arguments can be passed to the program if you use this startup method. See Section 8.1 [Exec], page 12.

5 The concepts behind ToolManager

ToolManager 2.1 uses a new object-oriented approach to provide a flexible and extendable system. This approach made it possible to enhance several ToolManager features of the 1.X versions, e.g. you can now have multiple docks.

An object is a collection of data which describes its features. Each object has a name and a type. You can create as many objects of each type as you want, but the name of each object has to be unique, because it is used as a reference to this object.

Currently there are seven different types of objects: Exec, Image, Sound, Menu, Icon, Dock and Access. The first three of them are basic objects; that means they don't reference other objects. They provide data or services for the complex objects.

The last four object types are complex objects; that means they reference simple objects and rely on them to get access to data or services. The reference is done by name, and if no simple object with this name exists, the complex object will ignore it. Note that this may reduce the functionality of the complex object, e.g. an Icon object *needs* the data from an Image object, so if this object doesn't exist it won't create an icon.

For a detailed description of all object parameters see Chapter 8 [Objects], page 12.

6 A guided tour through ToolManager

So you haven't understood a word until now? Confused by objects, programs and links? Don't despair, help is on the way.

I will now guide you through a step-by-step example on how to configure ToolManager. All you need is to install ToolManager and to run the preferences editor. After each step, use the "Test" button in the main window to test the configuration.

As an example we use the text display program More in the drawer '`SYS:Utilities`'. First we must tell ToolManager which program we want to use. Information about programs is stored in Exec objects. Just select "Exec" as object type in the main window of the preferences editor and press the "New" button.

After pressing the button you will see the "Edit Exec Object" window. Now open the Utilities drawer in your Workbench partition, move the More icon out of the drawer and drop it on the edit window. As you can see, the editor has now set the name of the object and the command to the program name, and the current directory to System:Utilities. Press the "OK" button to use the settings.

You can't do much with the Exec object alone, so as next step we want to add this program to the "Tools" menu of the Workbench. Select "Menu" as object type and press the "New" button. Now you will see the "Edit Menu Object" window. Change the name of the object to "Display Text".

ToolManager has to know which program it should start when the menu item is selected, so we link an Exec object to the menu object. Press the "Exec Object" button and select the object "More" from the requester. Now press "OK" button and the "Test" button in the main window. You can now see an entry in the "Tools" menu. Select a text file from the Workbench and choose the new menu entry. The program "More" should start and display the text. This is easy, isn't it?

Now we can go a step further and create an icon object on the Workbench. For an icon we need some image data, which is stored in an image object. Select "Image" as object type and press the "New" button. The "Edit Image Object" window will open. Change the name to "Image for More" and drop the More icon from the Utilities drawer on the window. Press "OK" to use the settings.

In the next step we will create the icon object. Select “Icon” as object type and press the “New” button. Change the name of the object to “Show Text”. Press the “Exec object” button and select the object “More” from the requester. Press the “Image object” button and select the object “Image for More” from the requester. Set the X position to 100 and the Y position to 50. Press the “OK” button and the “Test” button. After a short delay an icon will appear on the Workbench, on which you can drop the icons of your text files to display them.

I’m sure you now have an idea how to use ToolManager objects and in which way you have to link them together to build your environment. Now you can figure out the rest of the features by trying them out one by one. You may also look at the demo configuration in the file ‘`TM_Demo.prefs`’.

7 Description of all files in the distribution

The complete ToolManager 2.1 distribution consists of several directories which are explained below. Note that the distribution is split up into three parts, so you may not have all directories which are mentioned below.

7.1 The Docs directory

This directory contains the documentation for ToolManager. The documentation is available in four different formats and several languages. Additionally there is a file in AutoDoc format describing the ToolManager shared library interface.

Prefix '`TM_<language>`'

This file contains the documentation for the specified language. Currently available languages are: Deutsch, English, Français, Svenska.

Postfix '`.doc`'

This file contains the documentation as plain ASCII text.

Postfix '`.dvi`'

This file contains the documentation in TEXs DVI format. To get a printed manual, run this file through a TEX printer driver.

Postfix '`.guide`'

This file contains the documentation in AmigaGuide format. Although it is only plain ASCII with some commands, you need AmigaGuide to exploit the hypertext links in it.

Postfix '`.tex`'

This file contains the documentation in Texinfo format, as specified by the Free Software Foundation (FSF). Together with the '`texinfo.tex`' macro package, you can use TEX and '`texindex`' to create a file in DVI format (see above).

'`toolmanager.doc`'

This file contains the ToolManager shared library interface description in AutoDoc format.

7.2 The Goodies directory

This directory contains additional program packages which are useful for ToolManagers operation.

'`GetPubName.lha`'

This little program prints the name of the frontmost public screen either to stdout or into an environment variable. It was written by Michael "Mick" Hohmann.

'`upd1_20.lha`'

The program `upd` was written by Jonas Petersson. It is a small program which opens an ARexx port and waits for commands. Via ARexx you can order `upd` to play sampled files. ToolManager uses this feature to implement its Sound objects. See Section 8.3 [Sound], page 14.

7.3 The Graphics directory

This directory contains a rich collection of images from which you can choose your favourite ones. Just load them as Image objects into ToolManager (see Section 8.2 [Image], page 13).

The files were contributed by various people (see Appendix C [Credits], page 33). Each of them got a seperate sub-directory in the distribution. As the files were created by different authors, they come from different environments (palette, depth, resolution, size) and have different design styles. So not all images may look good on your machine.

To differentiate the image formats that are supported by ToolManager, each file has a postfix which describes the file format:

'`.anmb`' This is an IFF ANIM file created by a paint/animation program. It can contain several pictures. Although ToolManager can load complete ANIM files, you must use something like DPaints "AnimBrush" feature to cut out the interesting part of the animation.

'`.brush`' This is an IFF ILBM file created by a paint program. It contains only one image.

'`.info`' This is a normal Amiga Icon created with IconEdit (or something similiar). It can contain two images.

7.4 The L directory

This directory contains only one file, namely '`WBStart-Handler`'. You *must* copy this file to the '`L:`' directory, or otherwise ToolManager won't be able to start any Exec objects by the WB startup method (see Section 8.1 [Exec], page 12).

The complete package WBStart 1.2 may be found on Fish Disk #757.

7.5 The Libs directory

This directory contains only one file, '`toolmanager.library`'. This is the main program for ToolManager and must be copied to the '`LIBS:`' directory.

7.6 The Locale directory

This directory contains all files for ToolManagers Locale support. As locale.library is new with V38, you need not copy these files if you are using V37. If you are using V38, choose the files for your language and copy them to the appropriate places.

'`Catalogs/<language>/toolmanager.catalog`'

This is a translation file for the specified language. Copy the file for your language to the directory '`LOCALE:Catalogs/<language>`'.

'`Languages/<language>.language`'

Some languages are not supported by the standard V38 Locale distribution. So some of the translators have supplied a '`.language`' file, so that ToolManager can use their translation files. Copy the file for your language to the directory '`LOCALE:Languages`'. Additional available languages are: Finnish (suomi), Eefeler Platt (eifel).

7.7 The Prefs directory

The ToolManager preferences editor and its icon reside in this directory. Copy both files to the directory '`SYS:Prefs`'. For further information on the editor see Chapter 9 [Preferences], page 19.

7.8 The Programmers directory

This directory contains all files which are needed by the various computer languages and their compilers to use the ToolManager shared library interface. Look into the sub-directory '`examples`' for some examples on how to use this interface. For a complete interface description read the file '`Docs/toolmanager.doc`'.

Currently supported languages/compilers are: AmigaOberon, DICE C, M2Amiga Modula-2, MANX Aztec C and SAS C.

7.9 The Scripts directory

This directory contains a collection of ARexx or Shell scripts which can be used in ToolManagers Exec objects. Note that they may be specific to the authors environment or shell, so you may have to modify them.

7.10 The Source directory

This directory contains the complete source code to ToolManager 2.1 and its utilities. Each program has its own sub-directory. The author provides the source code as an example for OS 2.x/3.0 programming.

The '`locale`' sub-directory is of interest for translators. If your language is not supported in this release and you want to do the translation, look at the file '`empty.ct`'. Just fill in the empty lines and send the file to me. Maybe it will be included in the next release.

7.11 The WBStartup directory

Only one program resides in this directory: `ToolManager`. This utility starts and stops ToolManager 2.1. Most of the time this utility will reside in the '`SYS:WBStartup`' directory, but it can be used from the Shell too.

8 ToolManager objects reference

This chapter describes the ToolManager objects in detail. Each object has a type and a name. The name is used to reference the object. There are six different types of objects:

8.1 Exec objects

Exec objects describe programs or actions which are started by ToolManager. Three different types of programs are supported: CLI, Workbench and ARexx. Three different types of actions are supported: Dock, Hot Key, Network. Each Exec object has the following parameters. The defaults are set in parantheses:

`Arguments` (Yes)
: This switch controls the handing over of arguments to the program. If a program doesn't support arguments or doesn't need them, you can switch off the argument passing.

`Command`
: The file name of the program or action to start. This name may be relative to the current directory. If the type is Dock, the command describes the name of the dock object, which should be opened/closed. For the type Hot Key this string must be a Commodities Input Description String (see Chapter 11 [Hot Keys], page 24). A remote command (type Network) is described as `object@machine`, which tells the ToolManager running on `machine` to activate the Exec object named `object`.

`Current Directory` ('`SYS:`')
: The name of the current directory for the program. Note: ARexx programs ignore this parameter.

`Delay` (0)
: After activation of an Exec object, ToolManager waits `Delay` seconds before it starts the program. If this value is negative, the program will be started every `Delay` seconds. To stop an Exec object which is waiting for execution, just activate it again. Note: If `Delay` is set, the program will be started without arguments.

`Exec Type` (CLI)
: This specifies the type of the program or action. It can be one of: CLI, WB, ARexx, Dock, Hot Key or Network.

`Hot Key`
: You can set a Hot Key for each Exec object. If this Hot Key event is generated, the program will be started. Note: The program will be started with no arguments.

`Output File` ('`NIL:`')
: This is the file name of the output file. This is only useful for CLI programs.

`Path` (path from ToolManager process)

This string sets the command search path for the program. You can specify several directories by seperating the names with a ";". This is only useful for CLI programs.

`Priority` (0)

This sets the priority of the new process which runs the program.

`Public Screen` (default public screen)

You can set the name of the public screen which should be moved to front before the program is started. This only works in conjunction with the `To Front` parameter.

`Stack` (4096)

This sets the stack size of the new process which runs the program.

`To Front` (No)

If you set this parameter the public screen specified by `Public Screen` is moved to front before the program is started.

8.2 Image objects

Image objects specify the image data which is used by ToolManager for icons or docks. This object type has only one parameter:

`File Name` This specifies the name of the file from which ToolManager should read the image data. ToolManager tries to detect the type of image data automatically:

1. It tries to load it as IFF data. Currently ToolManager can read ILBM (one image) or ANIM (two or more images) files.
2. It tries to read in an icon file. An icon can have one or two images.

Animations are currently only supported by Dock objects. Icon objects only retrieve the first and the second image from the animation to build a two image icon. If you want to make an animation for ToolManager, you should follow these design rules:

Image 1 This should be an image which represents the inactive state.

Image 2 This should be an image which represents the selected state. Normally this is an inverted copy of the first image.

Image 3 to N-1

These are the images for the animation. Each image will be shown for 1/3 of a second.

Image N The last picture of the animation will be shown one second. After this the first picture will be shown again.

8.3 Sound objects

A Sound object can be used to make ToolManager noisy. ToolManager itself has no ability builtin to play sound data, it uses ARexx to activate an external sound player daemon. This object type has two parameters:

`Command` This sets the ARexx command which ToolManager sends to activate the external sound player. For `upd` this could be something like `file samples:boing` which instructs `upd` to play the IFF sample '`samples:boing`'. See Section 7.2 [Goodies], page 9.

`ARexx Port`

This specifies the ARexx port where ToolManager should send `command` to. The default is `PLAY` which is the port for the program `upd`.

8.4 Menu objects

Menu objects control the entries in the Workbench Tools menu. The object name is used as the menu text. To activate such an object, just select the menu entry. Menu objects only work when the Workbench is running.

This object type has two parameters:

`Exec Object`

This is the name of an Exec object which should be activated when the menu entry is selected. Every icon which is selected at this time will be used as an argument for the program.

`Sound Object`

This is the name of a Sound object which should be activated when the menu entry is selected.

Note to ToolManager 1.X users: To simulate the old tool type "Dummy" just create a Menu object and specify *no* Exec and Sound object.

8.5 Icon objects

Icon objects describe application icons in the Workbench window. Such an object can be activated by double-clicking the icon or by dropping some icons on the application icon. Icon objects only work when the Workbench is running.

The parameters for this object type are as follows:

`Exec Object`

This is the name of an Exec object which should be activated when the icon is selected. Every icon which is dropped on the application icon will be used as an argument for the program.

`Image Object`

This is the name of an Image object. The image data of this object is used to build the application icon.

`Left Edge` (default: 0)

This sets the left edge for the application icon.

`Show Name` (default: Yes)

If this parameter is set, the object name will be used as the name for the application icon.

`Sound Object`

This is the name of a Sound object which should be activated when the icon is selected.

`Top Edge` (default: 0)

This sets the top edge for the application icon.

Note: The Workbench is *very* picky about the position of icons. If you specify coordinates which the Workbench doesn't like, it will ignore them and place the icon somewhere else.

8.6 Dock objects

Dock objects describe windows. These windows combine several tools which are represented by images or gadgets. To start such a tool just click on its image or gadget. Of course you can drop some icons on the image or gadget to supply arguments for the tool.

Each dock object has several parameters. The defaults are set in parentheses:

`Activated` (Yes)

A dock window can be active (open) or not (closed).

`Backdrop` (No)

This tells the dock window to go immediately to the back after opening.

`Centered` (No)

If this parameter is set, the window will always be centered to the current mouse position when it opens.

`Columns` (1)

This parameter sets the the number of tool columns. Tools are always sorted row-wise, starting at the leftmost column and filling up to the rightmost column.

`Font` (Screen font)

If you have a dock window with the parameter `Text` set, you can choose the font for the button gadgets with this parameter.

`Frontmost` (No)

If you set this parameter, the dock window will always open on the frontmost public screen.

`Hot Key` You can set a Hot Key for each Dock object. If this Hot Key event is generated, the activation status of the dock window will be toggled; that is it will be closed or opened.

`Left Edge` (0)

This parameter sets the left edge of the dock window. If the parameter `Centered` is set, this parameter will be ignored.

`Menu` (No) You can add a small menu to each dock window. This menu has two items:

`Close Dock` Close dock window.

`Quit TM` Quit ToolManager

`Pattern` (No)

The dock window automatically adjusts its size to the largest image. Each dock entry has the same size, and smaller images are centered, so they have a blank border around them. If you don't like this blank border, set this parameter and the border will be filled with a pattern.

`PopUp` (No)

When this parameter is set the dock window will be closed automatically after selecting one dock entry. This is especially useful in conjunction with the parameters `Centered`, `Frontmost` and a Hot Key of the class `rawmouse` (see Chapter 11 [Hot Keys], page 24).

`Public Screen` (Default public screen)

Specifies the public screen on which the dock window should open. If the dock window was opened via Hot Key, the public screen will be moved to front after the window has been opened. This parameter will be ignored if the parameter `Frontmost` is set.

`Sticky` (No)

Normally a dock window stores its last position when you close it and pops up at the same position when you re-open it. If you want the dock window to open always at the same position, you must set this parameter.

`Text` (No) You can choose between images and button gadgets in dock windows with this parameter. Button dock windows are especially useful when used in conjunction with the parameter `PopUp`.

`Title` This specifies the dock window title. If you supply a title, the dock window will be a normal OS 2.0 window with dragbar, close gadget, depth gadget and a border. If you *don't* supply a title, you will get a dock window with only a dragbar and *no* border.

`Top Edge` (0)

This parameter sets the top edge of the dock window. If the parameter `Centered` is set, this parameter will be ignored.

`Vertical` (No)

If the dock window has the new window design (that is: only a dragbar and no border), this parameter sets the orientation of the dragbar. This parameter is ignored if you supplied a window title with the parameter `Title`.

8.7 Access objects

Access objects control the access rights for network requests. Per default *every* request is denied, so a remote ToolManager can't harm the operation of your machine by activating some of your Exec objects. With Access objects you can allow specific machines to activate some of your Exec objects.

The name of an Access object has a special meaning. It is matched with the name of the remote machine from which a network request was sent. ToolManager uses the following three step matching scheme:

1. Match with the complete host name
2. Match with the realm name
3. Look for the Access object named `anyone`

If a corresponding object is found, then this object is used for the access rights of the remote machine. The object named `anyone` is used for any network request, for which a corresponding Access object can't be found.

The Access object type has only one parameter:

`Exec Object`

This parameter can be used several times and specifies which Exec objects can be activated from the remote machine. If you don't specify *any* object name, then the remote machine can activate *all* Exec objects on your machine.

9 The ToolManager preferences editor

With the preferences editor you can manage the global configuration of ToolManager. This configuration gets automatically loaded when you start ToolManager. To start the editor just double click its icon. You will then see the main window.

Most of the gadgets in the editor windows have keyboard shortcuts. They are marked with an underscore (_). Note that if a string gadget is active, you must first press the return key before you can use the keyboard shortcuts.

9.1 Main window gadgets

The main window has several groups of gadgets:

Object type

With this cycle gadget you can choose the type of objects that you want to create or edit.

Object list

This gadget shows the list of all objects of the current type. You can select an object by clicking on its name. If you double-click one item, an edit window will open.

Move object

When an object is selected, you can move it around in the list with these gadgets. If you click on the `Sort` gadget, the items in the list will be sorted alphabetically.

Manipulate object

These gadgets manipulate objects. The `New` gadget creates a new object of the current type which is selected automatically. When you click on the `Edit` gadget, an edit window for the selected object will open. With the `Copy` gadget you can make a copy of the selected object. The `Remove` gadget deletes the selected object.

Configuration

You have several choices to save the configuration. With the `Save` gadget you can save the configuration permanently into the file '`ENVARC:ToolManager.prefs`'. For a temporary change use the `Use` gadget, which will save the configuration into the file '`ENV:ToolManager.prefs`'. This file will not survive a machine reset. To test the new configuration without leaving the editor, use the `Test` gadget. The `Cancel` gadget will quit the editor without saving.

9.2 Main window menus

The main window has several menu items:

`Project` With the menu items `Open` and `Save As` you can load and save the configuration. The `About` item opens an information requester. Selecting the `Quit` item will leave the editor without saving.

`Edit` With these menu items you can restore older configurations. The `Last Saved` item loads the last saved configuration from the file '`ENVARC:ToolManager.prefs`'. With the item `Restore` you can load the configuration that was active before you started the editor from the file '`ENV:ToolManager.prefs`'.

`Settings` You can choose with the `Create Icons` item wether the menu item `Save As` should create an icon or not.

9.3 Create objects window

If you drop an icon on the main window, the "Create objects" window will open. Here you can choose what objects should be created from this icon. This can be used to add a program to your configuration very easily and fast.

You can just create an Exec or Image object from the icon, if you select one of the first two choices. But you can also create a complete Menu and/or Icon object if you select one of the last three choices.

9.4 Edit windows

Each object type has a different edit window to set the object parameters. For a detailed list of all object parameters see Chapter 8 [Objects], page 12.

Every edit window has a string gadget for the object name. This name is important, because it is used to reference this object. Note that there is currently no builtin cross-reference. So if you change the name of an object which is already referenced by another object, this reference will *not* be updated. You have to update this reference by hand.

The button gadgets in the edit windows open different types of requesters. You can choose an item by clicking on it and pressing the `OK` gadget, or you simply double-click it. To leave a requester

without changes, use the `Cancel` gadget. If you wish to clear a field which can only be choosen by a requester, open the requester and press the `OK` gadget *without* selecting an item.

The edit windows for the object types Exec and Image have an additional feature. You can simply drop an icon on them to set the parameters from this icon.

9.5 Tooltypes

When you start the preferences editor from the Workbench you can set several tooltypes in the program icon or configuration file icons to control it.

`USE` If you set this tooltype in an icon for a preferences file, the editor will install this file as current configuration file.

`SAVE` If you set this tooltype in an icon for a preferences file, the editor will install this file as current and as permanent configuration file.

`PUBSCREEN`
This tooltype tells the editor to open its windows on a specific public screen. If you don't supply this tooltype, the default public screen will be used.

`CREATEICONS`
When this tooltype is set to `YES`, the editor will create an icon for every preferences file that is created with the `Save As` menu item.

`DEFAULTFONT`
The editor normally uses the public screen font to draw its gadgets. If you set this tooltype to `YES`, the editor will use the system default font instead.

`XPOS` This specifies the initial X position of the editor main window.

`YPOS` This specifies the initial Y position of the editor main window.

`MINLISTCOLUMNS`
This specifies the minimum number of columns in the list gadgets.

`MINLISTROWS`
This specifies the minimum number of rows in the list gadgets.

9.6 CLI Arguments

When the preferences editor is started from the shell, it uses the following command line template:

```
FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K,DEFAULTFONT/S
```

`FROM` This parameter specifies the name of the preferences file which the editor should load.

`USE` If you use this parameter, the editor will install the file specified as the `FROM` parameter as current configuration file.

`SAVE` If you use this parameter, the editor will install the file specified as the `FROM` parameter as current and as permanent configuration file.

`PUBSCREEN`

This parameter tells the editor to open its windows on a specific public screen. If you don't supply this tooltype the default public screen will be used.

`DEFAULTFONT`

The editor normally uses the public screen font to draw its gadgets. If you use this parameter the editor will use the system default font instead.

10 The ToolManager shared library interface

The ToolManager handler is embedded into a Amiga shared library. This library offers several functions to create and manipulate ToolManager objects, so that you can use them in your programs.

There are currently six functions available:

`AllocTMHandle()`

In order to create ToolManager objects you must first allocate a TMHandle. This handle stores all information about your objects and is used to reference them. Note that the information stored in this handle is *only* accessable by the program which creates it.

`FreeTMHandle()`

This function frees a TMHandle and all ToolManager objects associated with it. Each `AllocTMHandle()` must be matched with a `FreeTMHandle()`!

`CreateTMObjectTags()`
`CreateTMObjectTagList()`

This function creates a ToolManager object. You must supply a name, the object type and various tags for the object parameters. The name of the object is important, as it is used to reference the object.

`ChangeTMObjectTags()`
`ChangeTMObjectTagList()`

You can modify the parameters of a ToolManager object with this function. The object state will be updated to reflect the new parameters. Note: Currently Image objects can't be modified.

`DeleteTMObject()`

With this function you can delete a ToolManager object. If the object is linked to other objects, these objects will be notified to update their state.

`QuitToolManager()`

This function tells the ToolManager handler to quit as soon as possible.

The complete library interface description is available in AutoDoc format (see Section 7.1 [Docs], page 8).

11 How to define a Hot Key

This chapter describes how to define a Hot Key as an Input Description String, which is then parsed by Commodities. Each time a Hot Key is activated Commodities generates an event which is used by ToolManager to activate Exec objects or to toggle Dock objects. A description string has the following syntax:

```
[<class>] {[-][<qualifier>]} [-][upstroke] [<key code>]
```

All keywords are case insensitive.

`class` describes the InputEvent class. This parameter is optional and if it is missing the default `rawkey` is used. See Section 11.1 [InputEvent classes], page 24.

Qualifiers are "signals" that must be set or cleared by the time the Hot Key is activated; otherwise no event will be generated. For each qualifier that must be set you supply its keyword. All other qualifiers are expected to be cleared by default. If you want to ignore a qualifier, just set a `-` before its keyword. See Section 11.2 [Qualifiers], page 25.

Normally a Hot Key event is generated when a key is pressed. If the event should be generated when the key is released, supply the keyword `upstroke`. When both press and release of the key should generate an event, use `-upstroke`.

The key code is depending on the InputEvent class. See Section 11.3 [Key codes], page 26.

Note: Choose your hot keys *carefully*, because Commodities has a high priority in the InputEvent handler chain (i.e. will override existing definitions).

11.1 InputEvent classes

Commodities supports most of the InputEvent classes that are generated by the input.device. This section describes those classes that are most useful for ToolManager Hot Keys.

`rawkey` This is the default class and covers all keyboard events. For example `rawkey a` or `a` creates an event every time when the key "a" is pressed. You must specify a key code for this class. See Section 11.3.1 [rawkey key codes], page 26.

`rawmouse` This class describes all mouse button events. You must specify a key code for this class. See Section 11.3.2 [rawmouse key codes], page 27.

`diskinserted`

Events of this class are generated when a disk is inserted in a drive. This class has no key codes.

`diskremoved`

Events of this class are generated when a disk is removed from a drive. This class has no key codes.

11.2 Qualifiers

Some keyword synonyms were added to Commodities V38. These are marked with an `*`.

`lshift`, `left_shift` *

Left shift key.

`rshift`, `right_shift` *

Right shift key.

`shift` Either shift key.

`capslock`, `caps_lock` *

Caps lock key.

`caps` Either shift key or caps lock key.

`control`, `ctrl` *

Control key.

`lalt`, `left_alt` *

Left alt key.

`ralt`, `right_alt` *

Right alt key.

`alt` Either alt key.

`lcommand`, `lamiga` *, `left_amiga` *, `left_command` *

Left Amiga/Command key.

`rcommand`, `ramiga` *, `right_amiga` *, `right_command` *

Right Amiga/Command key.

`numericpad`, `numpad` *, `num_pad` *, `numeric_pad` *

This keyword *must* be used for any key on the numeric pad.

`leftbutton`, `lbutton` *, `left_button` *

Left mouse button. See note below.

`midbutton`, `mbutton` *, `middlebutton` *, `middle_button` *

Middle mouse button. See note below.

`rbutton`, `rightbutton` *, `right_button` *

Right mouse button. See note below.

`repeat` This qualifier is set when the keyboard repeat is active. Only useful for InputEvent class `rawkey`.

Note: Commodities V37 has a bug which prevents the use of `leftbutton`, `midbutton` and `rbutton` as qualifiers. This bug is fixed in V38.

11.3 Key codes

Each InputEvent class has its own key codes:

11.3.1 Key codes for InputEvent class `rawkey`

Some keywords and synonyms were added to Commodities V38. These are marked with an *.

`a-z`, `0-9`, ...

ASCII characters.

`f1`, `f2`, ..., `f10`, `f11` *, `f12` *

Function keys.

`up`, `cursor_up` *, `down`, `cursor_down` *
`left`, `cursor_left` *, `right`, `cursor_right` *

Cursor keys.

`esc`, `escape` *, `backspace`, `del`, `help`
`tab`, `comma`, `return`, `space`, `spacebar` *

Special keys.

`enter`, `insert` *, `delete` *
`page_up` *, `page_down` *, `home` *, `end` *

Numeric Pad keys. Each of these key codes *must* be used with the `numericpad` qualifier keyword!

11.3.2 Key codes for InputEvent class `rawmouse`

These keywords were added to Commodities V38. They are not available in V37.

`mouse_leftpress`
: Press left mouse button.

`mouse_middlepress`
: Press middle mouse button.

`mouse_rightpress`
: Press right mouse button.

Note: To use one of these key codes, you must also set the corresponding qualifier keyword, e.g.

```
rawmouse leftbutton mouse_leftpress
```

11.4 Examples for Hot Keys

`ralt t` Hold right Alt key and press "t"

`ralt lalt t`
: Hold left *and* right Alt key and press "t"

`alt t` Hold either Alt key and press "t"

`rcommand f2`
: Hold right Amiga key and press the second function key

`numericpad enter`
: Press the Enter key on the numeric pad

`rawmouse midbutton leftbutton mouse_leftpress`
: Hold middle mouse button and press the the left mouse button

`diskinserted`
: Insert a disk in any drive.

Appendix A Most asked questions about ToolManager

Here are the answers to the most asked questions about ToolManager:

- Why can't ToolManager create multiple "Tools" menus or sub-menus?

  Multiple menus or sub-menus are currently not supported by the system software. To create them, you have to *hack* them into the system software, which can result in an unstable system. I don't want to produce unstable software, so I won't implement such a thing in ToolManager.
- WB programs won't start, but all other exec types work fine.

  ToolManager relies on the program `L:WBStart-Handler` to start WB programs. There are two reasons, why ToolManager can't execute this program:

  The file '`L:WBStart-Handler`' doesn't exists. Please copy it from the distribution archive.

  The execute flag (e) isn't set on this file. Use the following command to set this flag: `protect L:WBStart-Handler +e`
- How can I create a horizontal dock window?

  Just set the number of columns to the number of entries in the dock object.
- How can I create an output window for CLI programs?

  Output windows can be created by using the `CON:` device. Use the following file name to create an auto-open window with a close gadget which doesn't close after the program has quit:

  ```
  CON:10/10/640/100/Output-Window/AUTO/CLOSE/WAIT
  ```

  The `CON:` device has many options, please consult your AmigaDOS manual for further information.
- How can I put the arguments in the middle of a CLI/Arexx command line?

  Normally all arguments are appended to the command line. To insert the arguments anywhere in the command line, ToolManager uses the same `[]` syntax, which is used by the AmigaShell command `alias`. So for example

  ```
  Dir [] all
  ```

  will insert all arguments before the keyword `all`.
- How can I clear a link from a complex object to a simple object?

  After pressing the "xxx Object" button just press the "OK" button *without* selecting an object. This means that you choosed no object, and therefore the link will be cleared.
- How can I create sub-docks?

  You must use Exec objects of the type Dock. Put such objects in the entries of your main dock and they will open/close the other docks.
- ToolManager is dead after starting a Network command.

  There is currently a problem with the network software, which doesn't timeout local requests. So if your machine is called `Host1` and you have an Exec object of the type Network with the

command `Object@Host1`, ToolManager will run into a dead-lock when you activate it. Please use only names of remote machines!

Appendix B The History of ToolManager

2.1, Release date 16.05.1993

- New Exec object types: Dock, Hot Key, Network
- New Dock object flags: Backdrop, Sticky
- New object type: Access
- Network support
- Editor main window is now an AppWindow
- Gadget keyboard shortcuts in the preferences editor
- New tooltypes for the preferences editor
- Several bug fixes
- Enhanced documentation

2.0, Release date 26.09.1992, Fish Disk #752

- Complete new concept (object oriented)
- (Almost) Complete rewrite
- ToolManager is now split up into two parts
- Main handler is now embedded into a shared library
- Configuration is now handled by a Preferences program
- Configuration file format has changed again :-) It is an IFF File now and resides in ENV:
- Multiple Docks and multi-column Docks
- Docks with new window design
- Dock automatically detects largest image size
- Sound support
- Direct ARexx support for Exec objects
- ToolManager can be used without the Workbench. If the Workbench isn't running, it won't use any App* features.
- Locale support
- Path from Workbench will be used for CLI tools
- Seperate Handler Task for starting WB processes

1.5, Release date 10.10.1991, Fish Disk #551

- Status Window: New/Open/Append/Save As menu items for config file
- Edit Window: File requesters for file string gadgets
- Added a Dock Window (a la NeXT)

- Added a DeleteTool
- A list of all active HotKeys can be shown
- Tools can be moved around in the list
- Icon positioning in the edit window added
- Name of the program icon can be set
- CLI tools can have an output file and a path list
- Uses UserShell for CLI tools
- Maximum command line length for CLI tools is now 4096 Bytes
- AppIcons without a name are supported now
- Workbench screen will be moved to front if you pop up the Status window
- Workbench screen can be moved to front before starting a tool via HotKey
- TM will wait up to 20 seconds for the workbench.library
- Added a DELAY switch which causes TM to wait <num> seconds before adding any App* stuff
- renamed some tooltypes/parameters
- some visual cues added
- some internal changes

1.4, Release date 09.07.1991, Fish Disk #527

- Keyboard short cuts for tools
- AppIcons for tools
- Menu item can be switched off
- Configuration file format completely changed (hopefully the last time)
- CLI commandline parsing is now done by ReadArgs()
- Status & edit window updated to new features
- Safety check before program shutdown added
- Menu item “Open TM Window” only appears if the program icon is disabled
- WB startup method changed. Now supports project icons
- several internal changes

1.3, Release date 13.03.1991, Fish Disk #476

- Now supports different configuration files
- Format of the configuration file slightly changed
- Tool definitions can be changed at runtime
- Now supports CLI & Workbench startup method
- Selected icons are passed as parameters to the tools

- Now uses the startup icon as program icon if started from Workbench
- The position of the icon can now be supplied in the configuration file
- The program icon can now be disabled
- New menu entry “Show TM Window”
- Every new started ToolManager passes its startup parameters to the already running ToolManager process

1.2, Release date 12.01.1991, Fish Disk #442

- Status window changed to a no-GZZ & simple refresh type (this should save some bytes)
- Status window remembers its last position
- New status window gadget “Save Configuration”: saves the actual tool list in the configuration file
- Small bugs removed in the ListView gadget handling
- Name of the icon hard-wired to “ToolManager”

1.1, Release date 01.01.1991

- Icons can be dropped on the status window
- Status window contains a list of all tool names
- Tools can be removed from the list

1.0, Release date 04.11.1990

- Initial release

Appendix C The author would like to thank...

ToolManager has gone through many major evolutionary phases since its first implementation in mid-1990. This development would have been impossible if I hadn't received the enormous feedback from various ToolManager users. Many ideas & features resulted from this source...

Therefore I would like to thank:

For Alpha/Beta testing, ideas & bug reports:

> Amiga section of our local computer club (Computerclub an der RWTH Aachen), Olaf 'Olsen' Barthel, Georg Hessmann (Gucky), Markus Illenseer (ill), Klaus Melchior, Rickard Olsson (Richie), Matthias Scheler (Tron), Ralph Schmidt (laire), Roger Westerlund (Budda), Juergen Weinelt, Brian Wright (SteveVai), Petra Zeidler (stargazer) and many others...

Matthew Dillon

> Without your **excellent** C development system DICE and various other tools, ToolManager wouldn't exist!

For their excellent graphics work:

> Andreas Harrenberg, Georg Hessmann, Michael "Mick" Hohmann, Markus Illenseer, Oliver Koenen, Klaus Melchior, Rickard Olsson, Jan Peter, Matthias Scheler, Brian Wright

For the translations:

> Tomi Blinnikka (suomi), Jorn Halonen (norsk), Dr. Peter Kittel (deutsch), Jasper Kehlet (dansk), Klaus Melchior (eifel), Rickard Olsson (svenska), Rullier Pascal (français), Marc Schaefer (français), Tor Rune Skoglund (norsk), Reinhard Spisser (italiano), Andrea Suatoni (italiano)

All gals & guys at West Chester:

> For developing the Amiga and its superb operating system.

All users who sent me money:

> I didn't ask for it in the 1.X releases, but it's nice to see when someone appreciates my work.

All users who sent me a note:

> I really enjoyed reading your letters!

and all I forgot to mention...

Index

M

O

P

Q

R

S

T

U

V

W

Table of Contents