What's New In Max/MSP 4.5.5

This short document highlights some of the new features in Max/MSP 4.5.5. We'll start by discussing additions to the patcher editing environment that permit customization to make your work with the software more productive. In addition, we've made the clipboard much more useful, and made small changes to the way you edit text objects in the patcher.

Object improvements continue to focus on the new objects added for version 4.5, highlighted by the new **mxj~** object, multi-threading support in **js** and **jsui**, and continued enhancements to the **pattr** family of objects.

The documentation for version 4.5.5 has been thoroughly revised to take into account the new features mentioned here. However, this document provides useful introductions to the changes in one place.

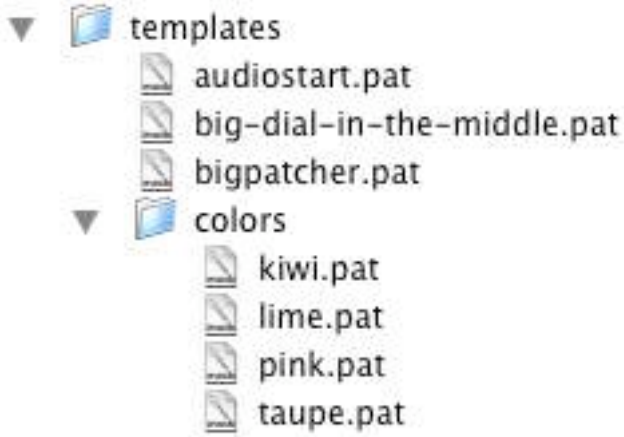
New Editing Features: Templates, Shortcuts, Clippings, and Prototypes

The new editing features share a general motivation: the ability to identify and exploit frequently used pieces of your work (or parts of other people's work) to avoid tedium and repetition in patching and configuring.

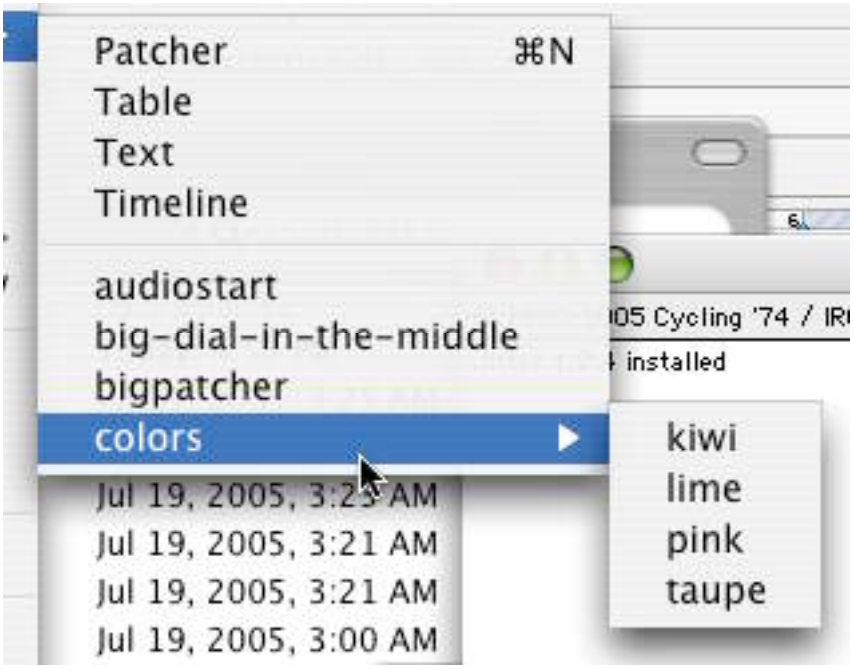
Templates

Templates are probably the easiest to understand of the new editing features. The patches folder inside the Max application folder now contains a folder called templates. In this folder you can place what some applications call "stationery." In Max's case, these are patcher files you wish to use as a starting point for further work.

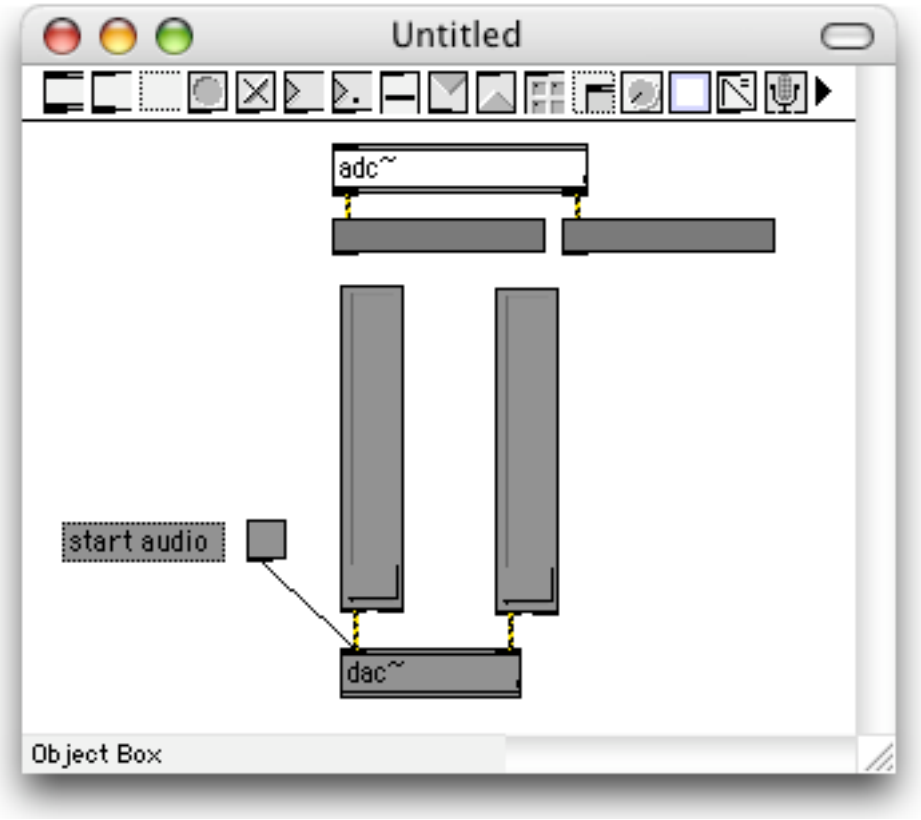
Here are the files you might see in a typical templates folder:



The New submenu of the File menu in Max reflects this organization as shown below. Templates appear in the New menu below Patcher, Table, Text and Timeline.



When you choose one of these items in the New menu, the patcher file is opened, but the window is untitled, unlocked, and not marked as modified. Here is the audiostart patcher opened as a Template (we've made it smaller to fit on the page):



The idea of the audiostart template file is that pretty much every audio patch you make will have a **dac~** object, some **gain~** sliders, a **toggle** to start audio etc. Why put them in your patch manually each time?

Other templates might save particular sizes and shapes of patcher windows or include a particular color background (that's what the user did above with the colors folder). We're sure you'll think of other Templates that will be useful starting points. For example, if you write plug-ins, you could create a template with the **plugin~, plugout~,** and **plugconfig** objects as well as the testing mechanisms you commonly use.

If you want to modify a template file, choose Open from the File menu and navigate to the templates folder inside the patches folder inside the application folder. Only when opened via the New menu do Template files behave differently.

Shortcuts

Max power-user Jasch created an object called _ (underscore) that did exactly what the **prepend** object did with "set" as an argument because he was tired of typing "prepend set" all the time. Inspired by his efforts to reduce RSI among Max users, we have added the ability to set up text macros for things you type into object boxes. As you'd expect, version 4.5.5 comes with Jasch's underscore shortcut. To try it out, create a new object box, then type the underscore character followed by the Escape key (ESC). The underscore will be replaced by "prepend set" and the insertion point will be after the word set (in the unlikely event you want to type something else).

Another example: let's say you do a lot of work with multichannel audio I/O. Now you can type `d6 <ESC>` instead of `dac~ 1 2 3 4 5 6`.

To add your own shortcuts, you place a message to the max object in a text file in the init folder as follows:

```
max shortcut <shortcut-text> <replacement-text>;
```

The shortcut text must be a single symbol, which means if you want to include a space in the shortcut, you will need to put all of the text in double quotes. The replacement text does *not* need quotes around spaces. For example, to replace underscore with prepend set, you would add the following message to a text file in the init folder:

```
max shortcut _ prepend set;
```

Examine the files *max-shortcuts.txt* and *audio-shortcuts.txt* in the init folder for examples and inspiration.

Clippings (Paste From…)

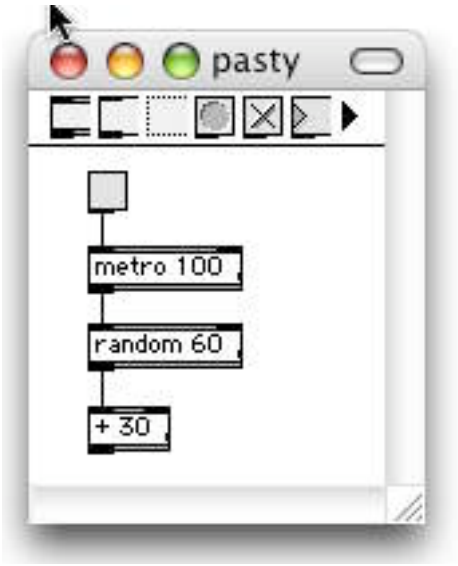
Many people end up patching the same little things every time they use Max. It doesn't help that much to use a subpatcher or abstraction for these "bits of code" because you often want them in the same patcher that you're working on. Abstractions have the disadvantage that they can't be modified easily. And both subpatchers and abstractions put the commonly used group of objects in a different window where it's hard to get to them.

If you find yourself repeating the same patch over and over again, you might the new clippings folder, located in the patches folder inside the Max application folder. The contents of the clippings folder is added to submenus of a new **Paste From…** item in the patcher contextual menu. Perhaps you've never even used the patcher contextual menu,

but we think you should consider checking it out, because **Paste From…** could save you a lot of time. **Paste From…** pastes the *contents* of a patcher file right into the patcher you're working on, with the top-left corner of the patcher window located at the current cursor position (i.e., where you clicked to obtain the contextual menu).

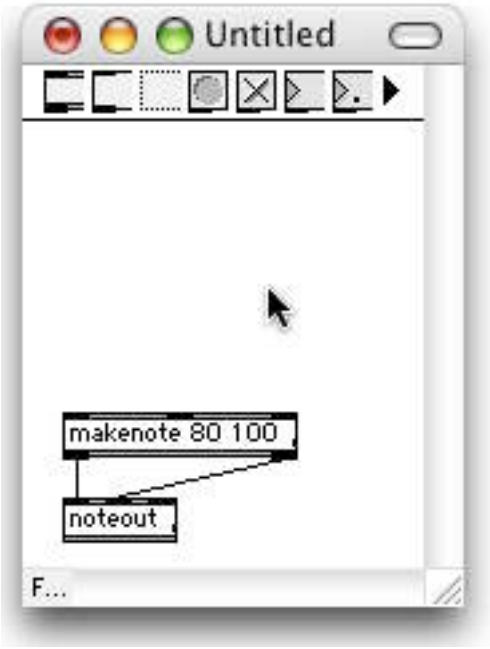
To obtain the patcher contextual menu, control- (Mac) or right- (Windows) click in a *blank space* in an unlocked patcher (i.e., not on an object or patch cord). **Paste From…** is the last item and its submenu will list all of the patchers in the clippings folder.

Choose one of the items in the submenu. Its contents will be pasted at the location you clicked to get the submenu. For example, consider the following patcher window:

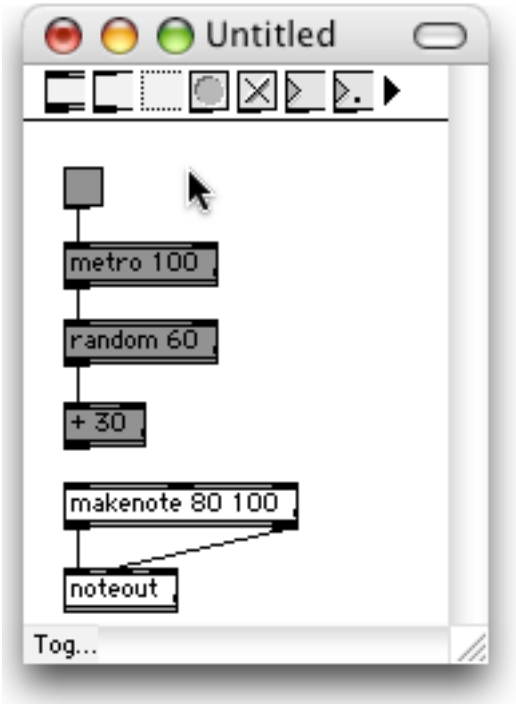


We saved the above window as a file called **pasty** in the clippings folder inside the patches folder in the Max/MSP application folder.

Now we can use **Paste From…** to put this into another patcher, which could really use some random notes.



Control- or right-click to obtain the Patcher Contextual Menu. Then choose pasty from the **Paste From…** submenu. The objects in pasty appear where you clicked.



The **Paste From…** menu contents from the clippings folder contains some very basic ideas to get you started. You can use the **Other…** item to open any patcher and paste its contents into the patcher you're working on.

Prototypes

Prototypes transform individual user interface objects with commonly used combinations of settings.

Some of the newer Max user interface objects have a fair amount of tweaky configurations you can set in an Inspector window. In particular, you can make beautiful sliders and dials with objects such as **pictslider** and **pictctrl**, but once you've made them, they're probably sitting in a patcher somewhere. You have to remember where the patcher is, copy the object out of the file, and then paste it into the patcher you're editing. Or, often as not, recreate the object from scratch.

Prototypes turn retrieving pre-configured user interface objects into a one-step process. Prototypes contain all of the settings for an object that you would otherwise set one at a time in the object's Inspector window.

Prototypes can be applied to all user interface objects except object boxes.

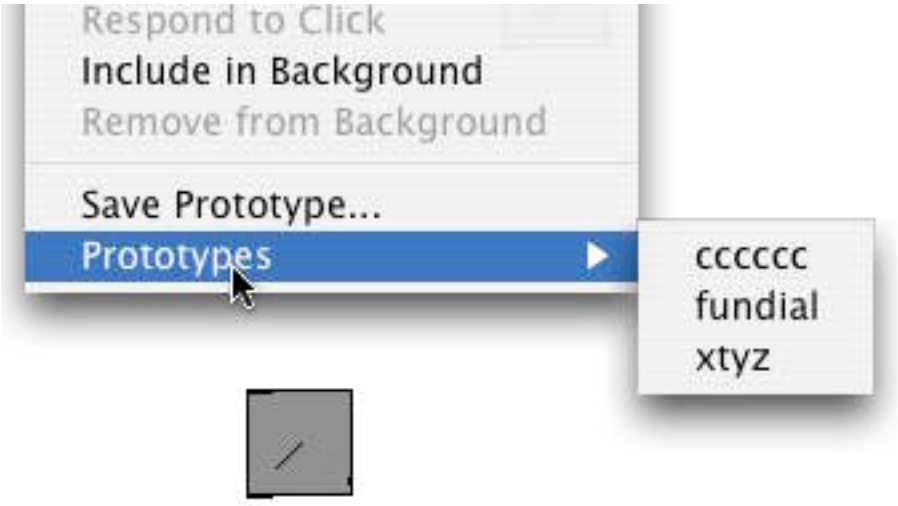
When you move the mouse over a user interface object's icon in the patcher window palette or scroll through the menu of icons to the right of the palette, you'll see that some objects have a number of prototypes listed in parentheses after the object description in the assistance area of the patcher window. For example, the text below will tell you that the **pictctrl** object has one prototype:

Picture Control (1 Prototype)

When you create a new **pictctrl** object, you'll get a default object. It has a generic size, no associated picture, and no behavioral settings. It's pretty useless. But it can instantly be made useful by selecting one of the object's prototypes. Here's how you do it:

When creating a new object, position the cursor where you want the object to go, then click. But instead of releasing the button as you normally would do, hold it down for a second or so. You'll see a menu listing all of the available prototypes. Choose one and the prototype will be applied instantly to the object.

- At any time after creating an object, select the object and choose an item from the Prototypes submenu of the Object menu. The prototype you choose will be applied to the selected object.

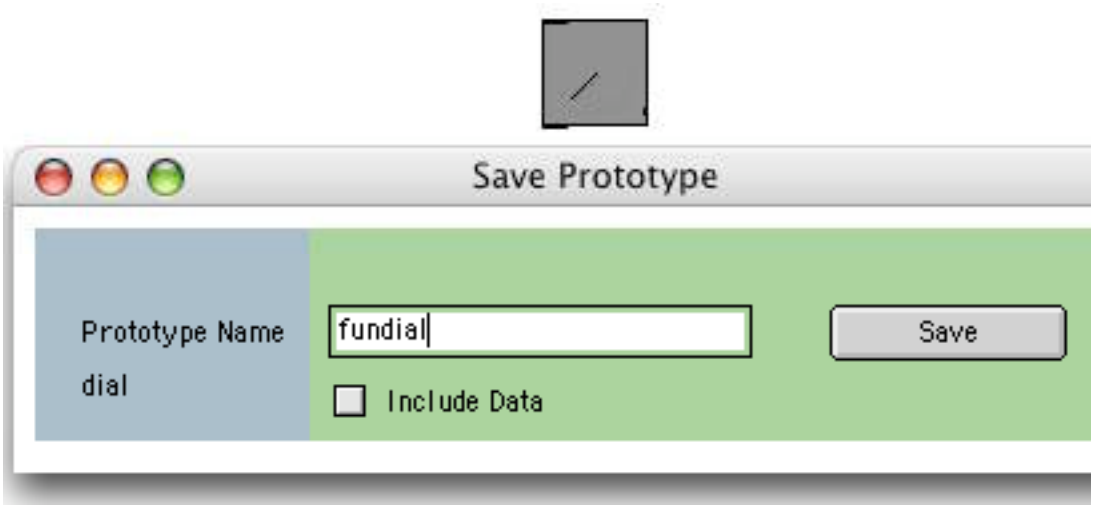


- Use the object contextual menu to obtain the Prototypes submenu when control- or right-clicking on an object in a patcher. The prototype you choose is applied to the object on which you clicked.

Applying a prototype is undo-able, but you can only apply a prototype to one object at a time.

Saving Prototypes

Once you have a collection of object settings that you like, you can save it as a prototype to use later. Select the object you want to save and choose **Save Prototype…** from the Object menu.



Name the prototype in the window that appears, then click the Save button. Your prototype is saved in a subfolder of the object-prototypes folder.

If you use an object with a prototype in a patcher you save, you don't need to worry about keeping the prototype around for the next time you open the patcher. The prototype feature is really an editing tool, it merely *replaces* the object you have with a new object that is created according to the instructions in the prototype file. In other words, unlike an "abstraction" a prototype is not a reference to an object. If you save over an existing prototype, all of the objects that were created with that prototype will be unaffected.

Prototypes and Object Data

A prototype can contain preset data from an object—check *Include Data* before saving the prototype. The data in the existing object is always replaced, either by the preset data in the prototype, or by the default data. In some cases, the "data" of an object is not necessarily its preset data. For example, the text of menu items for the **umenu** and **ubumenu** objects is saved with an object in a patcher, not in a preset (the current menu item selected is saved in a preset).

An object's connections, patcher scripting name (if any), and imageburger are preserved when a prototype is applied.

Prototypes for the bpatcher Object

One of the most powerful uses of the Prototype feature is its ability to create a collection of commonly used patcher elements using **bpatcher** objects. The prototype will save the current settings of the bpatcher (for example, the visible area of its client patcher). This could be useful if you are trying to create a catalog of visual "components" that you want to patch together.

Confused About All the New Editing Features?

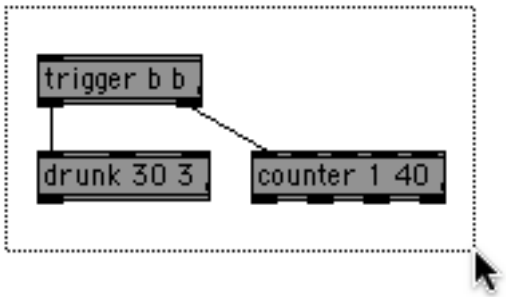
This handy comparison chart might be helpful.

Feature	Location	Applies To…	How to Use	Useful For…
Templates	:patches:templates	Entire patcher windows	Choose from File->New menu	Creating common starting points
Shortcuts	Init folder in Cycling '74 folder	Text in object boxes	Type shortcut then press the Esc key	Reducing excessive typing of object names and arguments
Clippings	:patches:clippings	Inserts a patcher's contents in another patcher	In an unlocked patcher, control- or Right-click on blank space to get a menu, then choose an item in the Paste From… submenu	Adding commonly used groups of objects to patches
Prototypes	:patches:object-prototypes	Individual user interface objects	Select an object, then choose Prototypes from the Object menu	Creating pre-configured user interface elements

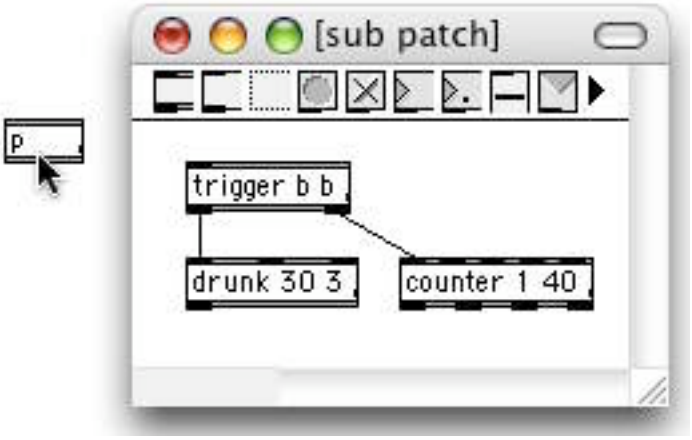
Encapsulation and De-Encapsulation

Have you ever wanted to clean up a patch you are making by putting a group of objects in a subpatcher? Now you can do it in two steps.

- Simply choose the objects you wish to place in the subpatcher.



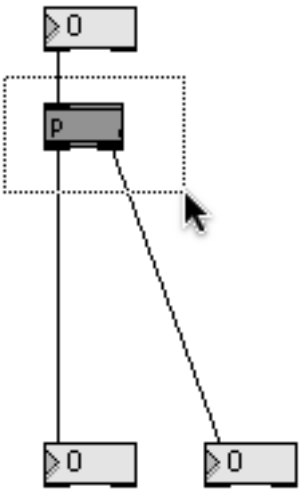
- Then, choose **Encapsulate** from the Edit menu.



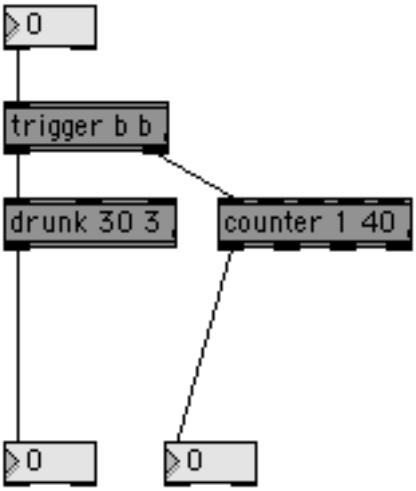
The objects are swept into a newly created subpatcher, inlet and outlet objects are added as appropriate. Don't like what happened? You can undo it.

The inverse operation is also possible. Sometimes objects stuck in a subpatcher are bothersome when trying to manage two windows to keep track of everything. You can now bring objects in a subpatcher "home" to their parent patcher with the *De-encapsulate* feature.

- Select a subpatcher



- Choose **De-encapsulate** from the Edit menu.



The subpatcher disappears and its contents are placed in the parent patcher, preserving any existing connections. De-encapsulation can be undone too.

New from Clipboard and the Text Clipboard

Has someone ever sent you (or posted) a patch as text you wanted to try? Previously you had to copy the text, paste it into a file, save the file, switch to Max, locate the file, and open it. Now the procedure is simpler:

- Copy the text in your e-mail program

- Switch to (or launch) Max
- Choose **New from Clipboard** from the File menu

Voila, there's your patch in an untitled window. A window created via **New from Clipboard** is set as modified, so you will be reminded to save it when you close it or quit Max.

More generally, we've changed the way patch data is stored on the Clipboard. It is now always saved as text. This means you can do fun things like copy an object, paste it into a Text window in Max, modify the saved data (assuming you know what you're doing), Copy the modified text, and paste it back into a patcher. Perhaps you want to make a change to a large group of objects (for example, change every instance of **jit.qt.movie** to **242.film**). A text editor is an ideal way to do this, and now such an operation is relatively painless (however, you shouldn't do this kind of editing unless you know what you're doing—or can make a reasonable guess).

New from Clipboard is enabled if there is any text on the Clipboard, whether it is patch data or not. An analysis is performed on the data.

- If it is a patcher, it is treated as a file and a new patcher is created (assuming there are no problems with the text).
- If it is part of a patcher (for example, if you've just copied a couple of objects from a patcher, **New from Clipboard** creates a new patcher window and pastes them into it.
- If the clipboard just contains some random text, a new Text window is created.

Changes to Patcher Selection of Text Objects

Storing data from objects in a patcher that you copy as text exposed an ambiguity about selection in a patcher. When you click on an object box to select it, have you selected the box or the text inside the box? Previously, Max copied both the box and the text inside it, and pasted the thing that was appropriate depending on whether text was selected in a box when you chose Paste from the Edit menu.

However, this strategy doesn't work in a world where all the data is text and data can arrive from anywhere. Therefore, we've tightened the rules about selecting text in an object box and the new regime may take some getting used to. We think you'll eventually prefer the more standardized approach we've adopted. In addition, we've provided a couple of options you can play with to make the patcher selection behavior more or less standard.

Text objects in a patcher are the object box, message box, and comment. The remarks below about "text objects" apply to all three of these objects.

For Mac OS X users, the difference between selecting a text object as a whole and having its text selected for editing is the same as the difference between selecting a file in the Finder for performing some kind of operation (Open, Move to Trash etc.) versus selecting the file's name to change it.

The Way It Used to Work

To select a text object for editing, you used to be able to drag around the object or click directly on it. Visually, the object would show the text selected transparently as shown below.

However, when multiple objects were selected, the selection changed to a dark gray transparent overlay, as shown below.

There was a bug where you could get a single text object into this "selected as a whole" state by selecting two objects, then shift-selecting one of them to unselect it. The remaining object could not be edited by typing or clicking until you unselected it and clicked on it again.

The Way It Works Now (With Options)

When a box appears selected for text editing, and you copy or cut it, you copy the text inside of the box to the clipboard, not the data for recreating the box.

When the box is selected "as a whole" the box itself is copied to the clipboard (i.e., the data for recreating the box) so you could paste a copy of the object somewhere else.

Previous versions of Max did both types of copying when an object was selected for text editing. Sort of. Now, Max explicitly copies both the text inside the box and the box as a whole. When you have clicked inside an text object to select the text for editing, the text from an object you copied will be pasted. Otherwise the object as a whole will be pasted.

Clicking on a text object selects the text inside of it for editing. If you click and drag the box somewhere before releasing the button, it will *not* be enabled for text editing (at least visually).

Dragging *around* a box to select it does *not* select the text inside for editing (at least visually).

When the **Typing Automatically Edits Selected Box** option is on (which it is by default), typing in an unlocked patch will route key presses to the selected text object and enable it for editing *even if it is selected as-a-whole*. When it is off, the text *must* be visually selected (as shown below) or there must be an insertion point in the text before you can edit it by typing.

When the **Select Text on Click** option is on (which it is by default), clicking on a box without moving it immediately selects the text for text editing. When it is off, clicking on a box always selects it as-a-whole. Clicking on it again, selects all of the text, and clicking on it yet again moves the insertion point to the place you clicked. This behavior is consistent with the Mac OS X Finder.

A nice addition to all of this is the change to the role of the Enter key (on both Mac and Windows). The Enter key now enters and exits text editing mode for a text object. Let's say you have an object selected as a whole…

Press Enter and the box's contents are now ready to be edited as text.

Press Enter again and your changes (if any) are updated. The object is selected as a whole again. Pressing Enter moves between the two selection modes.

Note that Windows XP machines only have an Enter key while Mac OS X machines have separate Return and Enter keys. On Windows XP the Enter key functions as the Mac OS X Return key and Shift+Enter functions as the Mac OS X Enter key. So, on Windows XP pressing Enter adds a new line while editing the text of a text object, and pressing Shift+Enter toggles between text selection and box selection modes.

An Editing Options patcher has been added to the Options menu, for changing the settings of *Select Text on Click* and *Typing Automatically Edits Selected Box*.

New Max/MSP Objects

jstrigger The **jstrigger** object combines the **trigger** and **expr** objects using Javascript. Unlike either of these objects, you can create and output lists in response to input. The **jstrigger** object is less efficient than the **trigger** object, but if you are trying to express a complex sequence of operations, you may be able to express a solution without resorting to real Javascript, Java, or C code.

mxj~ The new **mxj~** object allows you to write signal processing code in Java. There are two Java classes you subclass in order to write your code, as well as an innovative dynamic patching system built into the implementation. Another thing that mere mortal programmers will appreciate is that if your code crashes, the **mxj~** object stops working, but Max continues to run—and even continues to process audio. For details on both **mxj** and **mxj~** refer to the new WritingExternalsInJava documentation found in the java-doc folder inside the Max/MSP application folder. In addition, you'll see illustrations of how to write DSP code in Java in the mxj~ examples folder inside the java-doc folder.

slide The **slide** object acts as a filter for numbers. For details, see the Max Reference Manual entry for the object.

Highlights of Changes to Max/MSP Objects

js and **jsui** The Javascript objects **js** and **jsui** can now execute in the high priority thread using the new immediate property of functions. In addition, new capabilities have been added for saving object state, defining attributes, and sending messages to named objects. For complete details, refer to the revised JavascriptInMax documentation.

mxj Java classes can now be loaded from the same directory as your patcher or any subdirectory.

pattrstorage Numerous improvements including support for attributes, support for ISO-Latin1 character sets, preset data editing, and a new message grab that sucks the data out of all the object's clients.