Documentation Addendum for Max/MSP 4.3.1

Revised 9 January 2004

This document summarizes the changes made between versions 4.2 and 4.3.1 of Max/MSP. This document provides information and corrections that are not yet in the other Max/MSP Documentation. Most of the changes are relatively minor but they may affect you. In general, the changes were made to support Max/MSP as a cross-platform environment that, as much as possible, works identically on Mac OS X and Windows XP.

MaxMSP 4.3.1 for Windows XP now supports building standalone applications. In addition, building standalone applications with MaxMSP 4.3.1 for OS X has been updated for improved OS X support and cross platform consistency with Windows XP. More details on these changes are provided below.

Cross-Platform Collective Format

We've implemented a new collective format that adds some much-requested features, and can be read by both Mac and Windows. The previous collective format is still supported on Macintosh OS X (and is still used for making plug-ins), but should be considered obsolete and support for it will be dropped in the next major Max/MSP release. The new format is extensible and will allow us to add more features into the future, such as encryption and data compression.

The Collective Editor, activated via the "File | Build Collective / Application …" menu item, is now a patcher instead of a dialog box. This can make it more convenient to develop a collective script while examining your patchers.

Select the collective file format you want to build using the "Save As" dialog which is shown after clicking the Build button in the Collective Editor. On Macintosh OS X use the pop-up menu labeled Format. On Windows XP use the drop down box labeled "Save as type". Note that on Windows XP the option labeled "mx@c Files (*.mxf)" is for the Cross-Platform Collective Format and the option labeled "APPL Files (*.exe)" is for building an Application.

The old collective format was based on Mac OS resources, which means the resulting files had "resource forks" that are not readable on Windows XP. Even Mac OS X has trouble with them. The new format uses only the "data" part of a file. The way Max works with the new format is pretty slick: objects that want to read "files" out of a collective have no idea these files are not in a separate file. This allows audio files to be read directly from a collective by the MSP buffer~ object, and even spooled from a collective by the sfplay~ object. If you are a developer of Max/MSP external objects and are interested in updating your external object to work with these files, please consult the sysfile API in the forthcoming external SDK.

Quicktime movies cannot be read directly out of a collective. MP3 files can be included, but they are copied to a temporary file before imported into buffer~. Similarly, some graphics files imported by Quicktime can be included in the collective, but require copying to a temporary file before they can be handled. This was also true of the previous collective file format, by the way.

Another advantage of the new collective file format is that external objects for both platforms can be included and objects for the wrong platform will be ignored. However, the procedure for doing this now is somewhat awkward. You'll have to create a collective on one platform with the needed external objects in it, copy it to the platform, and include the collective in another collective built on the second platform. Note that if the collective contains only Mac externals and is read on Windows, the Windows version will act as if it contains no externals at all. And vice versa, of course. This means that if you are only using standard external objects, a standard runtime Max/MSP installation will work to open the files on a platform other than the one on which they were created.

If you are going be giving Mac-created collectives to Windows XP users, you are strongly encouraged to use the .mxf file extension. Otherwise, it won't be possible for Windows XP users to open the files.

Standalone Applications

You can use Max/MSP to build a standalone application. This is now supported on Windows XP as well as Macintosh OS X. The application building process is very similar on both platforms, and is very similar to building a collective but instead of choosing a collective file format in the Collective Editor's Save As dialog an application format is chosen.

On both Macintosh OS X and Windows XP a standalone application consists of a folder containing a modified version of the Max Runtime and a collection of supporting files. This folder is a bundle on Macintosh OS X and therefore appears as a single file to the user. On Windows XP this folder is a normal folder. The details for each platform are discussed below.

Macintosh OS X Standalone Applications

To create a standalone application choose Application from the pop-up menu in the save file dialog instead of Max Collective or Max Old Format Collective. The new scheme for standalones incorporates the Mac OS X Bundle feature. In this scheme, a folder that ends in ".app" is treated as a bundle, hiding its interior contents. Double-clicking on the folder's icon (which is actually, by default, a generic application icon) launches an application file contained within.

The application builder in Max/MSP creates an arrangement of files and folders for your standalone shown in the following listing (the name YourApplication is given when you save the file):

YourApplication.app [note: the .app is not shown in the Finder]
- Contents [folder]
  - Info.plist
  - MacOS

```
        YourApplication  [ actually Max/MSP Runtime ]
        YourApplication.mxf
            [ the new format collective containing your patches ]
        support
            [ audio and MIDI support files ]
    Resources
        [ custom icon file goes here ]
```

Important Note: You can look at the contents of your bundle by holding down the control key while clicking on the bundle's icon in the Finder and choosing Show Package Contents from the pop-up menu.

Windows XP Standalone Applications

To create a standalone application choose "APPL Files (*.exe)" from the drop down list in the Save As dialog instead of "mx@c Files (*.mxf). A standalone application on Windows XP consists of a folder containing all supporting files. The application builder in Max/MSP creates an arrangement of files and folders for your standalone shown in the following listing (the name YourApplication is given when you save the file):

```
YourApplication  [ folder ]
    YourApplication.exe [modified MaxRT.exe – launch this to launch your app ]
    YourApplication.mxf [ new format collective containing your patches ]
    msvcr70.dll
        [ Microsoft C Runtime Library used by MaxRT.exe and externals ]
    support  [ folder]
        ad  [ folder containing MSP audio driver objects ]
        mididrivers  [ folder containing Max midi driver object ]
        MaxAPI.dll  [ Max API for external objects ]
        MaxAudio.dll  [ MSP library ]
        MaxQuicktime.dll  [ Max QT interface ]
        YourApplication.rsr [ Mac style Resources for your application ]
        asintppc.dll  [ Support DLL needed for Max ]
        asiport.rsr  [ Support resources needed for Max ]
        asifont.map  [ Support file needed for Max ]
```

The standalone Object

In order to set options for standalone applications, you place a standalone object in your top-level patcher, and then bring up its inspector by selecting the object in an unlocked patcher and choosing Get Info from the Object menu. Two options at the bottom of the standalone inspector window are new: Audio Support and MIDI Support. When checked, these copy all the necessary files to handle audio and MIDI in your application to the support folder in your standalone application, as shown in the above listings for Macintosh OS X and Windows XP respectively.

Custom Icons and the Property List for Macintosh OS X

In order to use a custom icon for your standalone application, follow these steps:

1. Create a property list file for your application. This is a text file containing an XML specification of information related to your application, such as its icons and version number. You can use the Property List compiler in Code Warrior (or some other tool, such as Apple’s Property List editor) to create the file (you need to create Info.plist, not a resource). For an example, open an application package in the Applications folder, for example, Safari, by control-clicking on Safari’s icon and choosing Show Package Contents from the pop-up menu. Open the Contents folder and you will see the Info.plist file.

2. Change the icons referenced in the file to the name of an icon file you will create. Specifically, for the application icon, you need to change this line:

   key "CFBundleIconFile" value string "" (property list compiler source)

   or

   <key>CFBundleTypeIconFile</key> <string>file.icns</string> (XML)

   The text should read:

   key "CFBundleIconFile" value string "YourApplication.icns" (where YourApplication.icns is the name of the icon file you will add to the bundle)

   or

   <key>CFBundleTypeIconFile</key> <string>YourApplication.icns</string>

3. You may want to change other information in this file, such as the name of your application and its version number:

   key "CFBundleName" value string "Your Application"

   or

   <key> CFBundleName </key> <string>Your Application </string>

4. To make an icon file, you can use Apple’s IconComposer (included in the Developer applications folder when you install the developer tools), or a third-party icon development tool such as the IconBuilder plug-in for Adobe Photoshop from Icon Factory.

5. To include your property list and icon files in the standalone, you’ll add them to the collective script, but the technique for adding each file is slightly different. For the property list file, first ensure that the filename ends in .plist,

then click the Include File button and choose the .plist file. For the icon file, click the Include File button and choose the file, then replace the word include with appicon. Here is an example:

```
include Disk:/MyAppFiles/Info.plist
appicon Disk:/MyAppFiles/Icon/MyApplication.icns
```

6. When building your application, the application builder will copy the Info.plist file to the correct location in your package (the Contents folder), then it will copy an icon file specified with the appicon keyword to the Resources folder.

7. You will likely need to restart your computer before the standalone's icon will show up.

Custom Icons and Splash Screens for Windows XP

In order to create a custom icon for your standalone application, follow these steps:

1. Create your ICON using the Windows ".ico" format.
2. Using the Collective Editor add a line to your Collective Script with the following syntax: appicon *filename*
   Hint: you can use the "Include File" button to choose the .ICO file and then change the "include" command to "appicon".

In order to create a custom splash screen for your standalone application, follow these steps:

1. Create your splash screen using the Windows ".bmp" format.
2. Using the Collective Editor add a line to your Collective Script with the following syntax: appsplash *filename*
   Hint: you can use the "Include File" button to choose the .ICO file and then change the "include" command to "appsplash".

The Search Path in Standalone Applications

It may be important for developers of standalones to know the order in which Max searches for files; it is slightly different than the normal search order. In addition, the meaning of the two searching options in the standalone inspector has changed slightly with the introduction of the new standalone format. This information is relatively advanced and will probably not be of much interest to users who are not developing standalone applications.

On Macintosh OS X the Utilize Search Path option, when checked, does the following for the search path of a standalone:

1. Adds all of the folders inside of the folder containing the standalone application (i.e., the Contents folder and all of its subfolders). The folder containing the application's package is not included. In other words, if you put your application in the Mac OS X Applications folder, the Applications folder is not included in the search path.

2. Adds the Cycling ’74 folder (/Library/Application Support/Cycling ’74) if it exists.

On Windows XP the Utilize Search Path option, when checked, does the following for the search path of a standalone:

1. Adds all of the folders inside of the application folder (i.e. the folder containing YourApplication.exe).
2. Adds the Cycling '74 folder (c:\Program Files\Common Files\Cycling '74\) if it exists.

When Utilize Search Path is not checked, the only folder(s) added to the search path are the support folder inside the standalone application’s folder, and any of its subfolders.

The order in which folders will be searched is as follows:

1. The collective file (and any other open collective files)
2. The support folder
3. The folder containing the application and its subfolders (optional, if Utilize Search Path is checked)
4. The Cycling ’74 folder (optional, if Utilize Search Path is checked)

The Search for Missing Files option is slightly different from the Utilize Search Path option. It prevents Max from looking for any files that are not in open collectives, including the support folder, which means that you cannot uncheck Search for Missing Files and have either audio or MIDI support. This means that Search for Missing Files is now of limited usefulness.

The former role of Search for Missing Files was for testing the collective to make sure you were including all of the files you need. With the advent of the support folder and its ability to contain audio and MIDI files, this testing role now falls mainly to the Utilize Search Path option. Utilize Search Path specifically allows you to check whether any files in the Cycling ’74 folder are needed by your standalone application: if, after turning off Utilize Search Path, you see errors indicating “no such object” or “can’t find files” in the Max window, you know you aren’t properly including all of the supporting files you need.

A tip that may help sort out path problems: Put ; max paths in a message box in your patch so you can click on it when the standalone is running. The paths message prints out the file paths currently in use.

File Sniffing and Type Information

What happens if you send a Max patcher called “foo” with no extension to a Windows user? On Windows, a file extension tells the system what kind of file format you have. But on Macintosh, the extension has always been optional, and “file type metadata” codes have been used to identify the file’s format.

Max now investigates the contents of files when they have neither extensions nor file type information. And in some cases, an extension isn't enough: traditionally, Max users have used the .pat extension for both text and binary files. We've now changed the definition of .pat from Max binary only to either Max binary or text. Since .pat is ambiguous, Max looks at the contents of the file to determine what format it's in.

By default, Max still writes files with Mac OS type information in them. But you can now turn off the writing of the type code with a message to the max object.

The message notypeinfo 1 to the max object (e.g., triggering ; max notypeinfo 1 in a message box) causes Max not to add Mac OS filetype information to files it writes out. The notypeinfo 0 message restrores the default behavior.

Text-Based Preferences

Your Max preferences (such as your default font used in the patcher window and the settings in the File Preferences window) and MSP preferences (such as the current audio device being used) are now stored as fairly readable text files in a special folder, instead of in an undocumented and unreadable format.

In addition to the advantage that they are harder to corrupt, the text-based preferences files can be edited by hand if you need to change your preferences without using Max/MSP to do it. The preferences files consist of messages to the max object (and the dsp object for MSP). Many of these messages can also be used while working in the program.

On OS X, you'll find the preferences files in the Max 4 Preferences folder inside the Preferences folder inside the Library folder of your home directory.

On Windows XP you'll find the preferences files in the folder:
C:\Documents and Settings*Username*\Application Data\Cycling '74\Max 4 Preferences Files\
Note one: the above *Username* will be your username, such as "rob".
Note two: the above folder is hidden by default. To see it you will need to turn on viewing of hidden folders in the Windows XP file explorer.

Inside the Max 4 Preferences folder you'll find a Recent Items folder, which contains aliases to all of the files you've opened with Max recently. You can double-click on them to open these files, if you want.

Font Mapping

Files saved by version 4.3 contain font names as well as Mac-specific font numbers. The Windows version can still read older files with numbered fonts, but since these numbers are both platform- and computer-specific, the font names are used to enhance portability.

The inclusion of font names in patcher files has been done in a way that will be ignored when older versions of the software load the patcher.

In the init folder, there is a file containing font mappings as messages to the max object. We map, for example, the Mac OS Geneva font to the generic “Sans Serif” font when saving a patcher. And we map the Sans Serif font back to Geneva when reading a patcher. This allows you to pick the “best” sans serif on a computer that may not have the fonts that were present on the machine that originally created the file.

File Path Syntax

Max has always allowed you to specify full or partial file path specifications as arguments to messages to objects that read files. In version 4.3, we’ve moved away from the Mac-centric colon as a file path separator. Paths with colons still work, but we now encourage you to use a cross-platform slash notation that should be familiar from URLs. Objects such as filepath and absolutepath output this new path style so you’ll notice the slashes have replaced colons in the File Preferences window.

A full path looks like this:

Volume:/Directory/File

A path relative to the Max application folder looks like this:

./Directory/File

A new feature is a path relative to the Macintosh OS X folder /Library/Application Support/Cycling ’74 or Window XP folder c:\Program Files\Common Files\Cycling '74\:

C74:/Directory/File

For more information see the Files topic of the Max documentation.

MIDI Setup Changes

The MIDI Setup window now contains checkboxes for enabling and disabling MIDI output devices as well as input devices. This feature was added primarily for Windows XP users, but may be of some use to Mac users. You could use it to shut off output to a particular MIDI device quickly. If you have many MIDI objects outputting to the device turning off the device’s checkbox is akin to yanking the MIDI cable. Much easier than hunting down each MIDI output object.

On Windows XP this is useful to prevent opening the midi output for the Microsoft GS Wavetable Software Synthesizer. Opening this midi synth will result in the Microsoft synth opening the default audio device. If this is an ASIO device that does not support multiple clients it can prevent MSP from being able to open the audio device.

Trace

We’ve tried to make the Trace feature more robust. One difference you will notice is that whenever a patcher is destroyed, Trace is immediately aborted, whether or not the destroyed patcher was involved in the trace. Choose Enable from the Trace menu to keep tracing.

New Object List

The Max Object List file that was used to set the contents of the New Object List window has been replaced by two files: max-objectlist.txt and audio-objectlist.txt in the init folder. These files contain a series of oblist messages to the max object. You can add your own text files in the init folder with custom additions to the object list, just use the oblist message, i.e.:

```
max oblist Category1 Objectname1;
max oblist Category2 Objectname2;
```

The text files can be named anything you want. Max tries to evaluate any files it finds in the init folder.