Information about the various conversion programs in this tree

HOW TO BUILD

Move **shared.subproj** into the relevant folder (e.g. **Convert_RTF**), open the project file in the Convert folder, and choose build (the debug target will *not* install any subdirectories in the app wrapper). All should work just fine...

NOTE: I tend to prefer to do my coding in the New Century Schoolbook typeface. If tabs etc look strange in the source files, try that instead.

QUESTIONS

If you have any questions, please just email me at davidjohn@kira.net.netcom.com. I may not have the time to answer big complex questions, but I can at least offer pointers and suggestions if I can't.

DIRECTORY CONTENTS

shared.subproj contains code used by all the Convert programs
hex2bin contains an executable used by to produce some tests for Convert PICT.
Extractors is source code for Extract PICT & Extract FONT. These are in Macintosh MPW C.
Convert_FONT contains the source for Convert FONT
Convert_MacPaint is the source for Convert MacPaint
Convert_PICT is the source for (surprise) Convert PICT
Convert_RTF is the source for Convert RTF
Convert_TEXT is the source for Convert TEXT

Shared.subproj

Now, There are a number of classes that all the converters use. These are stored in shared.subproj. The intent was to use a link from each project to point to this folder. Alas, Project Builder in 3.1 does not like links. So, as things stand at the moment, this folder must be copied or moved into each Convert folder before compiling that converter. This is a kludge, I admit. It may or may not be better than that which I used before. I'm sure there's a better way, but I've not had time to do it yet. On the bright side, it used to be ickier. =)

Some of the classes defined here have .rtf files that document them. However, I make no promises that

these files are up to date. They should give you a pretty good idea though.

Here's a description of the relevant classes:

common is a pair of c and h files that define the data types used throughout the converter (you probably won't like them. People seem to like `long' and `int' and `char' a lot, whereas I don't. I tend to prefer a different way of viewing the data types. Oh well). The c file provides routines for dealing with some of these (e.g. allocating and deallocating a CString type).

ResultObject. The intention here was to allow any object to return more than one value by storing it, and allowing the caller to call back and get the result value(s). This was an experiment, and I would not do it these days. These projects grew up on top of this class faster than I'd expected. However, since everything is a subclass of this class, it is too much to remove it just now. I am sure a noticeable performance gain could be gotten by removing it.

File is a very commonly used class, both directly and in subclasses. Simply, it wraps a set of method calls around an NXStream object. This was, originally, based on a FILE*, so there may still be references to this lying around.

PSFile and **TextFile** are both subclasses of File that are just good enough to be used in the Converters, but probably not general purpose enough. Basically, they provide shortcuts for writing out data to these particular types of files (e.g. writing integers as strings, or producing DSC comments, etc).

AbstractConverter. Every converter is a subclass of this class. This does very little save for setting up the basic methods that should be present in all the converters.

ConvertController. All the Convert applications, in addition to having a Converter Class (subclassed from AbstractConverter) also has a Controller class, which is subclassed from this class. This simply defines the basic path needed to open up a file and call the converter to get it converted. This is also probably the kludgiest class here. I don't like it much. Nevertheless, it's another important and often used class.

Convert.msg is simply the definition of the method call that can be made to a converter.

ProgressIndicator. This class, and the related psw file, are used to draw the circular indicator on the conversion progress window so that the current status is displayed

TextConverter is a class used in some cases, but not all. Like AbstractConverter and ConvertController, it is designed to be subclassed, and not to be used directly. It merely defines a set of standard method calls for calling an object that converts between character sets.

MacToNeXTText is a subclass of TextConverter which merely converts Macintosh text to NeXT text.

NeXTToMacText is another subclass of TextConverter that (surprise) converts NeXT text to Mac text.

hex2bin

As you'll discover soon, I've included many of the test files I use to test these converters. Many of the PICT test files are stored in an ascii format (see that section for more info), and I wrote this quick and dirty little application to process those files. Well, one day I seem to have been stupid, and managed to toss the .c file rather than the .o file into the recycler. I didn't notice this for months, though, and by then I'd expired all my backups that might have had it. Sigh. So, all you get is executable here, I fear. I've not bothered to write a new one yet, because this still meets my needs. (an, for some time. Maybe I should stop being employed? =)

hex2bin takes a text file containing `hexified' data on standard in, and writes out a file on standard out that is the binary equivalent. If the file contains 0001020A, this would write out a binary file with the null byte, ascii 1, ascii 2, and ascii 10 in it. It's smarter than just this, though. If it finds a % character, it interprets it as a line comment, and ignores the rest of the line. It also ignores white space (and probably non-hex characters too). Furthermore, when it sees a (, it assumes that an ascii string follows and is terminated by). It writes this out as a Pascal style string (length of string first byte, followed by the string). This was added because PICT files have Pascal strings in them, and I didn't want to have to enter the string as a set of hex digits, and I didn't want to compute the length, so this does it for me. I think that's all it does, but it's been a while, so if any of those pict tests look to be doing something else, then this must support it. =)

Extractors

There are two Macintosh programs which turn up in Convert PICT.app and Convert FONT.app which extract data from Macintosh files. The source for these is in extractors. They use the same source file, but simply use a compile define flag to invoke the conditionally compiled parts of each.

Convert apps in general

The layout of all five convert directories is very similar, so I'll describe all the common features:

Unless otherwise noted, all the .**tiff** files are for the app and the file types it understands.

There are the usual NeXTSTEP app related files (.**iconheader**, ...**_main.m**, **Makefile**, **PB.project**).

English.lproj will always contain the .nib for the app, and the **Help** folder with all the online help text.

The **Makefile.postamble** and **Makefile.preamble** are simply used for getting all the right files, if needed, into the app wrapper and getting it renamed. NS 3.x added a lot of stuff that allowed these to be pared down, and I haven't looked to see if I tossed out all that I could. (that is, these may be strangely organized to your mind)

Testing contains some of the files I have used to test the app.

Additionally, each app will have a subclass of **ConvertController** and **AbstractConverter** in it. The AbstractConverter subclass is the core of the conversion process, while the ConvertController subclass is the user-interface controller and general administrator class.

Convert FONT

PreFontConverter is the subclass of ConvertController.

FontConverter is the subclass of AbstractConverter.

The **Testing** folder contains a couple Mac fonts that have been extracted with the Extract FONT app. The names are of the form fontName-##, where -number is the point size of the original font.

MacTypes.h simply provided typedefs for certain Macintosh types used when dealing with the Mac data.

Convert MacPaint

macpaintController is the subclass of ConvertController.

macpaintConverter is the subclass of AbstractConverter.

The **Testing** folder contains several MacPaint documents.

The **PSstuff** folder contains the routines used to display the finished eps image, both the packed and the unpacked form.

Convert PICT

PictController is the subclass of ConvertController.

PictConverter is the subclass of AbstractConverter.

PICTFile is a subclass of File (see the shared.subproj section above) and provides crude methods for extracting certain PICT data structures from a PICT file.

RegionConverter is logically part of the PictConverter. It was moved into a separate class partially to keep things organized (it is relatively large) and partially because it could operate pretty independantly of the rest of the code. It simply has the purpose of converting a PICT region data structure to a set of coordinates for the PS code to later digest

service_specification is used to define the conversion services Convert PICT provides..

CommentedPSCode is a folder that holds all the PS code that Convert PICT copies into a converted file. That is, a PICT file gets turned into a bunch of calls to PS procedures that are defined in the files in this folder. So, the converted file contains the routines from these files that were used, and the data converted from the PICT file Put another way: This is a library of PS routines that are included in each

file converted by Convert PICT. Note that **UncommentedPSCode** is created by the makefile and the **makeUncommented** script in **CommentedPSCode**.

Testing contains a large number of tests (relative to the other converters) for testing Convert PICT. It contains three subdirectories. **pict_tests** contains a number of pict files which can be tested with (I can provide more doc about the region tests if you care). **built_pict** is simply an empty folder which will hold tests that are created from the following folder. **hex_tests** contains a lot of files with extensions .hex. Opening any one will reveal an ascii file containing comments (% is the comment character) whitespace and hexadecimal digits. Each file defines a PICT image. These were created because I wanted *exact* control over what I was testing, and so using a draw program, for instance, to generate test files would not have been adequate. Each file should have some minimal commenting inside to let you know what is going on. however. You really have to have read Inside Mac volume 5, at least, to be familiar with the opcodes used here and their arguments. Also, I figured out the region structure by looking at it (Apple doesn't document it so far as I know), so the best documentation I can offer about its structure is the comments to the **RegionConverter** object =(. To turn these into useful PICT files for conversion, you must use the program **hex2bin**, described above. If you have it in your path, you can use the scripts **BuildTest** and **BuildAll** in this folder to run the tests through the program and put them into **pict_tests**.

Convert RTF

rtfController is the subclass of ConvertController.

rtfConverter is the subclass of AbstractConverter.

rtfFile is a subclass of File (see the shared.subproj section above) and provides crude methods for reading and writing rtf file elements. Despite its good points, I'm sure this could be improved upon dramatically

FontEntry is a class used to help the **rtfConverter** keep track of what fonts it has processed so far

rtfToken is a class used to keep track of a single rtf `token'. Used in **rtfConverter** anf **rtfFile**, and **FontEntry**.

MacToNeXTRTF is a class used to convert text from an rtf file from a Mac to NeXT format (it deals with, say, converting characters to Symbol on the NeXT to preserve their look)

NeXTToMacRTF is a class used to convert text from an rtf file from a NeXT to Mac format.

Testing is a folder with an assortment of rtf files. Some are there to test particular features of the converter. Some not. Not much more can be said without documenting each.

Convert TEXT

CharController is the subclass of ConvertController.
This application does not have a subclass of AbstractConverter. The conversion is so simple that

CharController simply directly runs the appropriate TextConverter subclass. Therefore, it plays a few games so that it looks like both classes to folks that care.

CRLFToNeXTText is a class used to convert text with lines that end with CR and LF to NeXT text line endings.

NeXTToCRLFText is a class used to convert text with lines that end with LF characters to CR and LF line endings.

Testing is a folder with one test file. It is a text file with 256 bytes, from 0 to 255. This can be used to run through the converter and see that the expected changes were done. Note that this can get surprisingly complex at times (e.g. pretend it is a Mac file newly transferred, and now convert. the result may not look like your intuitions think it should)