1 Introduction

The present document intends to provide a brief overview of the Apple® Uniform Type Identifier (UTI) system and recommendations for usage with TEX document types. It is mainly intended as a reference for developers who will be writing TEX-related software on Mac OS X.

1.1 What is a UTI?

A UTI is a string which represents an abstract "type," or class of entities, to use Apple® terminology,[1] and file entities will be the focus of this discussion (*e.g.*, TEX files that sit on your hard drive). The UTI describes the file type uniquely, and is merely a way of tagging it so that the system and other software can understand how to handle it. For example, a plain text file is identified as `public.plain-text`, which is a UTI that is defined by Apple® in the operating system. Only Apple® is allowed to create UTIs in the `public` domain. In general, the owner is advised to use a reverse-DNS prefix, such as `com.adobe` as in the `com.adobe.pdf` UTI assigned to the PDF type.

1.1.1 Inheritance

One of the distinguishing concepts of the UTI type system is inheritance, whereby a file is declared to descend from a root type. For the simple cases, this is relatively obvious; for instance, it seems clear that `public.plain-text` would descend from `public.text`. However, UTIs also support multiple inheritance, so a file can descend from otherwise unrelated content types. For example, OmniGraffle[2] files have a tree as shown in Figure 1. Notice how `public.text` descends from `public.data` and

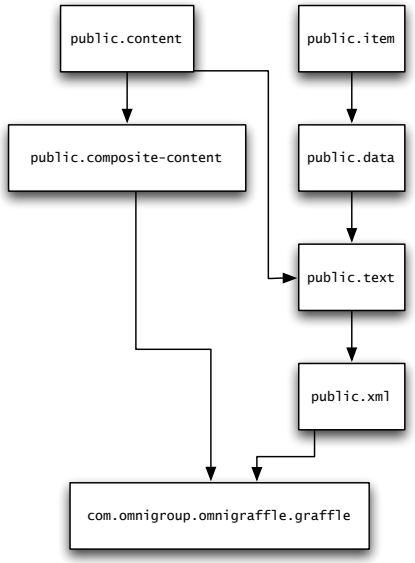

Figure 1: OmniGraffle UTI inheritance tree

`public.content`; in essence, the former indicates its representation in the system (a byte stream), and the latter its function (user-viewable content). This also means that an application which can open any

[1] `http://developer.apple.com/library/mac/#DOCUMENTATION/FileManagement/Conceptual/understanding_utis/understand_utis_conc/understand_utis_conc.html`

[2] `http://www.omnigroup.com/products/omnigraffle/`

one of these types can open an OmniGraffle file, so you can open it in your favorite text editor or XML editor, and view some representation of the data.

1.2 How are they used?

The main use of a UTI that we are concerned with is to allow applications to declare which file type(s) they can open. For instance, an application could claim to open any `public.image` file, or it could only claim to handle `public.jpeg` if its functionality was more limited. For proprietary file types (*viz.*, those not declared by Apple®), developers must include a definition of the file type and its identifying characteristics.

In addition, the UTI is used to determine which Spotlight importer to use to search the content of a file, and it is used by Quick Look to determine which plugin to use for previewing that file or creating icons. Likewise, Finder uses the type to assign an icon to the file, if you are not using Quick Look to create icons dynamically.

1.2.1 Association

A common misconception is that a UTI can be assigned to a specific file or file type. This is not the case. Rather, Mac OS X infers the UTI of a file using metadata associated with that file, such as its name or the HFS type/creator code embedded in the file. For data transmitted over a network, the MIME type may also be used (though it's not clear if Apple® actually uses it at this time).

Although a UTI is not the same as a file extension such as the ".pdf" typically found on the end of an Adobe® PDF file name, current versions of Mac OS X consider the extension as the primary form of metadata, overriding HFS type and creator code information if present.[3] This causes some confusion among users and developers alike, and is possibly the weakest point in Mac OS X's usage of UTIs, as there is now no way to resolve conflicts between files that have the same extension but contain different data types. For instance, a ".img" file may be a disk image, or it may be a raster image file, but when you double-click on that file to open it in Finder, the system will attempt to open it in the default application for that UTI. Since the UTI is based on the file's extension, it may well open in the wrong application! You can usually work around this by opening a file from within a given application, but this is not possible for Spotlight and Quick Look plugins.[4,5]

1.2.2 Declaration

A UTI is typically declared in the Info.plist file of an application bundle on Mac OS X, as in this fragment from Apple® documentation:[6]

```
<key>UTExportedTypeDeclarations</key>
        <array>
            <dict>
                <key>UTTypeIdentifier</key>
                <string>public.jpeg</string>
                <key>UTTypeReferenceURL</key>
                <string>http://www.w3.org/Graphics/JPEG/</string>
                <key>UTTypeDescription</key>
                <string>JPEG image</string>
```

[3] http://tidbits.com/article/10537

[4] http://lists.apple.com/archives/quicklook-dev/2008/Feb/msg00005.html

[5] http://lists.apple.com/archives/quicklook-dev/2008/Feb/msg00006.html

[6] http://developer.apple.com/library/mac/#DOCUMENTATION/FileManagement/Conceptual/understanding_utis/understand_utis_declare/understand_utis_declare.html

```
            <key>UTTypeIconFile</key>
            <string>public.jpeg.icns</string>
            <key>UTTypeConformsTo</key>
            <array>
                <string>public.image</string>
                <string>public.data</string>
            </array>
            <key>UTTypeTagSpecification</key>
            <dict>
                <key>com.apple.ostype</key>
                <string>JPEG</string>
                <key>public.filename-extension</key>
                <array>
                    <string>jpeg</string>
                    <string>jpg</string>
                </array>
                <key>public.mime-type</key>
                <string>image/jpeg</string>
            </dict>
        </dict>
    </array>
```

There are at least two important points to make here.

1. The definition includes a significant amount of ancillary data, such as a URL pointing to the reference documentation.

2. Multiple tags are defined, including an HFS type code ('JPEG'), two extensions (".jpeg" and ".jpg"), and the MIME type ("image/jpeg").

You can define a UTI with less information, but it's best to include as much as possible, since it serves as a reference for users or developers.

Actually using the UTI is another matter, and depends on the type of bundle (standalone application, Quick Look plugin, Spotlight plugin) you are developing. Refer to the appropriate Apple® documentation for details, though it's something of a mess, due to rampant duplication of information. The Info.plist documentation[7] is a reasonable starting point. *Be very careful if you mix UTI and classic type declarations in an NSDocument-based application!* Support for UTIs varies across operating system versions, and some rather bizarre problems can occur.[8]

1.2.3 Conflicts

The preceding example uses the `UTExportedTypeDeclarations` key to declare the UTI. A `UTImportedTypeDeclarations` key is also available, and is recommended when you are declaring a type that you do not own, or when the owning application may not be present on the system. For instance, if you wrote an application that could read OmniGraffle files, you would declare the `com.omnigroup.omnigraffle.graffle` UTI in a `UTImportedTypeDeclarations` section of your application's Info.plist file. This would allow the system to associate your application with OmniGraffle files when OmniGraffle itself is not installed, but would not override Omni's `UTExportedTypeDeclarations` when OmniGraffle is installed.

[7] `http://developer.apple.com/library/mac/#documentation/General/Reference/InfoPlistKeyReference/Introduction/Introduction.html`

[8] `http://www.omnigroup.com/mailman/archive/macosx-dev/2007-November/060638.html`

There are some additional subtleties not covered in the official documentation. For example, if two applications declare a different UTI for the same tag, both using `UTImportedTypeDeclarations`, this can lead to non-deterministic behavior and failure to open files.[9] Such conflicts are exceedingly painful to debug, particularly for users who have no idea why their file icons are randomly changing, or the wrong application is being used to open their files.

1.2.4 Debugging

There are several useful tools for debugging problems with file types and UTIs. The first is **mdls(1)**, which can be used from the command line to determine which UTI is associated with a given file. For example, here is the output for the OmniGraffle file used to create Figure 1:

```
$ mdls UTIDiagram.graffle
kMDItemContentCreationDate     = 2011-10-29 00:12:42 +0000
kMDItemContentModificationDate = 2011-10-29 00:17:37 +0000
kMDItemContentType             = "com.omnigroup.omnigraffle.graffle"
kMDItemContentTypeTree         = (
    "com.omnigroup.omnigraffle.graffle",
    "public.composite-content",
    "public.content",
    "public.xml",
    "public.text",
    "public.data",
    "public.item"
)
kMDItemDateAdded               = 2011-10-29 00:17:37 +0000
kMDItemDisplayName             = "UTIDiagram.graffle"
kMDItemFSContentChangeDate     = 2011-10-29 00:17:37 +0000
kMDItemFSCreationDate          = 2011-10-29 00:12:42 +0000
kMDItemFSCreatorCode           = ""
kMDItemFSFinderFlags           = 1024
kMDItemFSHasCustomIcon         = 1
kMDItemFSInvisible             = 0
kMDItemFSIsExtensionHidden     = 0
kMDItemFSIsStationery          = 0
kMDItemFSLabel                 = 0
kMDItemFSName                  = "UTIDiagram.graffle"
kMDItemFSNodeCount             = 132986
kMDItemFSOwnerGroupID          = 80
kMDItemFSOwnerUserID           = 501
kMDItemFSSize                  = 132986
kMDItemFSTypeCode              = ""
kMDItemKind                    = "OmniGraffle document"
kMDItemLastUsedDate            = 2011-10-29 00:12:43 +0000
kMDItemLogicalSize             = 132986
kMDItemPhysicalSize            = 135168
kMDItemUseCount                = 2
```

[9] `http://www.omnigroup.com/mailman/archive/macosx-dev/2009-May/062244.html`

```
    kMDItemUsedDates                = (
        "2011-10-28 07:00:00 +0000"
    )
```

The most relevant parts for our discussion are `kMDItemContentTypeTree`, `kMDItemContentType`, and `kMDItemKind`. These correspond to `UTTypeConformsTo`, `UTTypeIdentifier`, and `UTTypeDescription`, respectively. You still may wonder where these come from, though, and **mdls(1)** doesn't help with that!

For further debugging, you need to use the **lsregister** tool, located at /System/Library/Frameworks/CoreServices.framework/Frameworks/LaunchServices.framework/Support/lsregister (in Mac OS X 10.7; may be different in previous systems). Note that you must use the full path, as this command is not available in the shell's default search path. Use the `-help` option to see a summary of usage, but the most useful option is `-dump`, which dumps the contents of the registration database to standard output as plain text. For the OmniGraffle document, we find the following:

```
type id:            29764
uti:            com.omnigroup.omnigraffle.graffle
description:    OmniGraffle document
flags:          exported  active  trusted
icon:
conforms to:    public.composite-content, public.xml
tags:           .graffle
```

From this, we discover that it is an "exported" UTI, that it has multiple conformance, and a single tag. The same information could be found by examining OmniGraffle's Info.plist at `OmniGraffle.app/Contents/Info.plist`, but this is very useful if you are trying to figure out which applications are claiming a type or tag.

For debugging Quick Look plugins and Spotlight importers, use **qlmanage(1)** and **mdimport(1)**, respectively. Note that their debug options are very useful, as they can tell you exactly which plugin is being used. Launch Services will commonly end up using an old plugin, even one that is in the trash, if you are actively developing. This can lead to much frustration with the system.

2 TEX and UTI

We have probably spent an excessive amount of time introducing UTIs, when a simple link to the relevant Apple® documentation would suffice. However, the official documentation does not discuss conflicting UTI definitions or some of the other issues raised here. In this section, we discuss the motivation for writing this document, and our hope that developers will agree to adopt common set of UTIs for TEX and its ancillary documents. At present, we have competing definitions, largely because no standard exists, and there is no central clearinghouse for non-proprietary UTI definitions.

2.1 Prefix

In the case of TEX and ancillary files, a "public" UTI seems appropriate, since it has been available to the public for many years. However, the UTType.h header[10] on Mac OS X contains the following comment:

[10] To quickly open this header file in Terminal, you can use `open -h UTType.h`. Likewise, you can use `open -h UTCoreTypes.h` to see the system definitions of various UTIs and their named constants.

```
/*
  Types which are standard or not controlled by any one organization
  are declared in the "public" domain. Currently, public types may
  be declared only by Apple.
*/
```

This requires us to declare a non-public UTI for a public type, since we are not Apple®, and only Apple® can mandate a UTI for a type that it does not own. We understand the reasoning behind Apple® reserving the public domain for their own use on Mac OS X, but find it ironic nonetheless.

We propose the prefix of `org.tug.tex` for TEX and related file types on Mac OS X. Alternatives include `edu.stanford.knuth` or `comp.text.tex`. We do not have any claim over either of these, and more importantly, we cannot establish a `UTTypeReferenceURL` at either of them. A better alternative is `public-texmf`, which is a clearly neutral option, and would be acceptable with any domain. This would require changing the UTI definitions of all shipping applications that we are aware of[11], but that is a one-time inconvenience to developers and users (assuming that users update to the latest version).

2.2 Suffixes

The set of suffixes is determined by the number of file types used and generated by TEX and associated programs such as BibTEX; in general, these will be filename extensions. When this topic was discussed on the MacTEX Technical Working Group mailing list, a large list of extensions was developed, and it was soon evident that it would not be reasonable to declare all of them as TeX file types. A non-exhaustive list is presented here for illustration: .tex, .ltx, .ctx, .bib, .bst, .aux, .dvi, .xdv, .sty, .cls, .bbl, .blg, .brf, .fmt, .idx, .ilg, .ind, .ini, .mp, .toc, .log, .out, .synctex, .mem, .tfm, .vf, .pl, .vpl, .enc, .map, .lig, .tui, .tuo, .tub, .mps, .mpo, .mpy, .mpx, .rlg, .rlx. Of these, not all are guaranteed to represent a TEX-related file; .ini and .log being two notable examples.

We propose that suffixes be added only as required, in order to avoid conflicting UTIs (see Section 1.2.3). In this context, "required" means that your application requires that file for its functionality, or provides some unique capability (such as syntax highlighting). Further, we suggest that obviously ambiguous tags such as .ini and .log never be associated with TEX-related UTIs.

3 Proposed Base Types

Table 1 describes our proposed set of types that would be suitable for most applications that handle TEX-related files on Mac OS X. We consider these types to be unambiguously related to TEX, and they are required by one or more currently shipping applications on Mac OS X.

Note that .tex is commonly used as an extension for multiple file types, so there is no guarantee that a .tex file is not LATEX, rather than plain TEX. Since Launch Services considers the extension to be the primary metadata in this case, the opening application or bundle will have to determine the actual type by sniffing the content. This may present some difficulty in mapping internal types used by NSDocument to your UTIs.

[11] BibDesk and TeX Live Utility use `org.tug.tex`

Table 1: Proposed type definitions for typical applications, separated by usage.

	UTI[a]	Description[b]	Parent[c]	Extension[d]	MIME Type[e]	HFS[f]
Input	org.tug.tex	TeX Document	public.plain-text	.tex	application/x-tex	TEXT
	org.tug.tex.context	ConTeXt Document	org.tug.tex	.ctx	text/plain	TEXT
	org.tug.tex.latex	LaTeX Document	org.tug.tex	.ltx	application/x-latex	TEXT
	org.tug.tex.bibtex	BibTeX Database	public.plain-text	.bib	text/plain	TEXT
System	org.tug.tex.latex.style	LaTeX Style	org.tug.tex.latex	.sty	text/plain	TEXT
	org.tug.tex.latex.class	LaTeX Class	org.tug.tex.latex	.cls	text/plain	TEXT
	org.tug.tex.latex.dtx	LaTeX Docfile	org.tug.tex.latex	.dtx	text/plain	TEXT
	org.tug.tex.bibtex.style	BibTeX Style	public.plain-text	.bst	text/plain	TEXT
Output	org.tug.tex.dvi	DVI Document	public.data	.dvi	application/x-dvi	DDVI ODVI ∗DVI DVI2
	org.tug.tex.xdv	XDV Document	public.data	.xdv	application/octet-stream	
Intermediate	org.tug.tex.aux	LaTeX Auxiliary	public.plain-text	.aux	text/plain	TEXT

[a] UTTypeIdentifier
[b] UTTypeDescription
[c] UTTypeConformsTo
[d] public.filename-extension
[e] public.mime-type
[f] com.apple.ostype

4 Adding to the list

The list of UTIs in Table 1 is hardly comprehensive. We anticipate a need to add more UTIs, but it is not yet clear which ones will be required. There are three ways to proceed here:

1. Define an exhaustive list of all possible UTIs
2. Define a subset, and require developers to use their own domain for others
3. Define a subset, and allow developers to add to our list

The first is not practical, and the second defeats our purpose of unifying UTIs for the TEX community. We choose the third option, and provide guidelines for creating new UTIs. If a developer needs a UTI for a type not on this list, there is no need to wait for us to "approve" its addition. However, if you create a new `org.tug.tex` UTI, please inform us via e-mail at mactex@tug.org and provide a definition of the UTI.

The usage categories (input/system/output/intermediate) are only cosmetic, as a user can certainly have a .sty file as part of the input for a project. The intermediate files are files which are output as part of the typesetting process and can be deleted with no loss of user-entered data.

4.1 UTI creation guidelines

When adding a new UTI, first consider that the primary usage is by Launch Services. The UTI is used to determine which application(s) can open a given file, which Spotlight importer should be used for that file, and which Quick Look plugin should be used to preview it. If you need one or more of those, then you should add a new UTI. We hope that the guidelines here will result in compatible choices, assuming multiple developers were to independently add the same type. The `UTTypeReferenceURL` can point to any reasonable URL that describes the file format, hopefully as specifically as possible.

4.2 Inheritance

We recommend that you use `public.plain-text` for plain text files, and `public.data` for any binary data files. For the UTTypeDescription, use the form "Suffix Document," so that a file with extension .foo would be a "Foo Document." If the .foo file is produced by BibTEX, use `org.tug.tex.bibtex` as the prefix, and extension as the suffix, resulting in a UTI of `org.tug.tex.bibtex.foo`. The principle we follow here is that inheritance is based on functionality you would need as a developer. It is rare that you would want a BibTEX database to be handled in the same way as a TEX document, and the reverse is true as well. If `org.tug.tex.bibtex` conformed to `org.tug.tex`, an application that claimed to handle BibTEX files could also be called upon to handle plain TEX files, and that is not generally what we want. On the other hand, a developer is free to create `com.foosoft.mytype` which inherits from all `org.tug.tex` UTIs. We just don't want to require that by default.

4.3 Tags

In general, the HFS type code (if it even exists) is largely irrelevant, but should be added if it is known. Unless a well-known MIME type exists, use application/octet-stream for binary data such as the XETEX .xdv files, and use text/plain for plain text data.

4.4 Example property list

If your PDF viewer supports embedded files (Preview.app on Mac OS X does not), you can extract a sample Info.plist with complete examples of our proposed UTIs by opening the following annotation.