Dictionary Services Programming Guide

Developer

Contents

Figures, Tables, and Listings

Introduction

Dictionary Services lets you create your own custom dictionaries that users can access through the Dictionary application. You also use these services to access dictionaries programatically and to support user access to dictionary look-up through a contextual menu.

You should read this document if you want to create a dictionary for users to view in the Dictionary application or access a dictionary programmatically.

Organization of This Document

This document is organized into the following chapters:

- Dictionary User Interface and Markup (page 6) describes the user interface to dictionaries and defines the elements and attributes that you use to markup dictionary entries.
- Creating Dictionaries (page 18) explains how to modify the files provided in the Dictionary development Kit to create and build your own dictionary. It also provides sample dictionaries, shows how to create a Japanese dictionary, and describes implementing advanced features.
- Accessing Dictionaries (page 32) shows how to use the Dictionary Services programming interface to access active dictionaries.

See Also

Dictionary Services Reference describes the functions available in the programming interface.

Dictionary User Interface and Markup

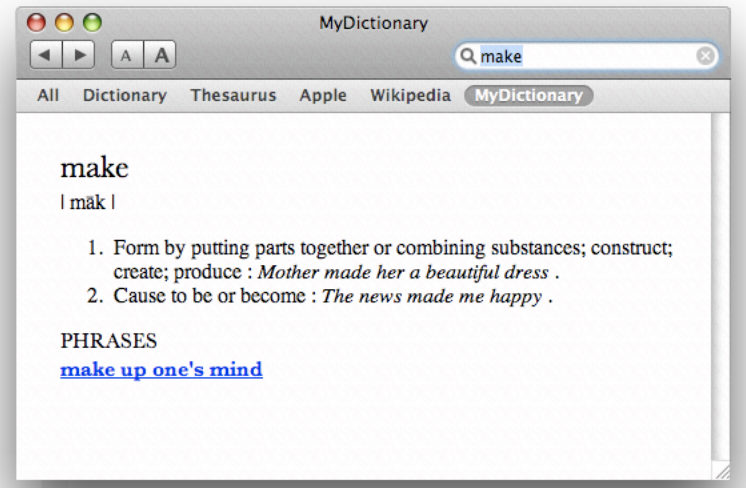
This chapter describes the user interface for Dictionary Services and defines the elements and attributes that you use to mark up dictionary content. The dictionary schema conforms to the RELAX NG definition (http://www.relaxng.org/). You define the dictionary structure by marking up the content using XHTML tags and tags that are defined by Dictionary Services.

Dictionary User Interface

Dictionaries are useful references that serve many purposes. The most typical is for users to look-up the meaning of a word in a particular language. However you can create content for a thesaurus, bilingual dictionaries (such as English-Japanese), in-house glossaries, and professional dictionaries (such as legal, medical, and technical). OS X includes the Dictionary application to provide users with digital access to content that is bundled according to the instructions provided in this book.

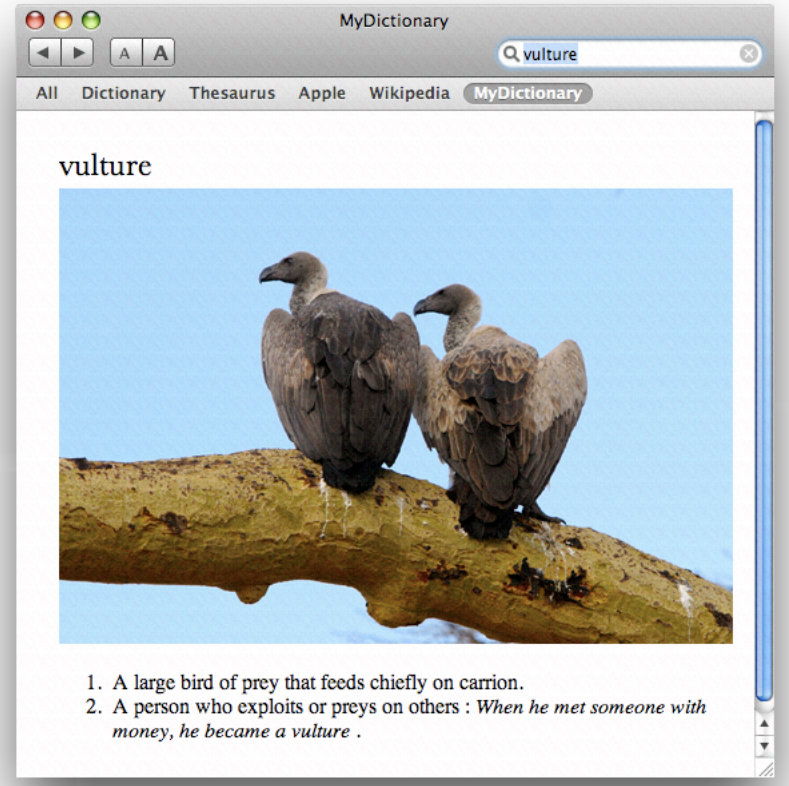
Figure 1-1 shows an entry from the MyDictionary dictionary for the word *make*. The MyDictionary dictionary was created using the Dictionary Development Kit. (You can download the kit in "Auxiliary Tools for Xcode - February 2012" from Downloads for Apple Developers.) As you read the rest of this chapter, you'll see how to markup text using XHTML to create the entry, pronunciation, definitions, usage examples, phrases, and so on that appear in this figure.

Figure 1-1 A dictionary entry



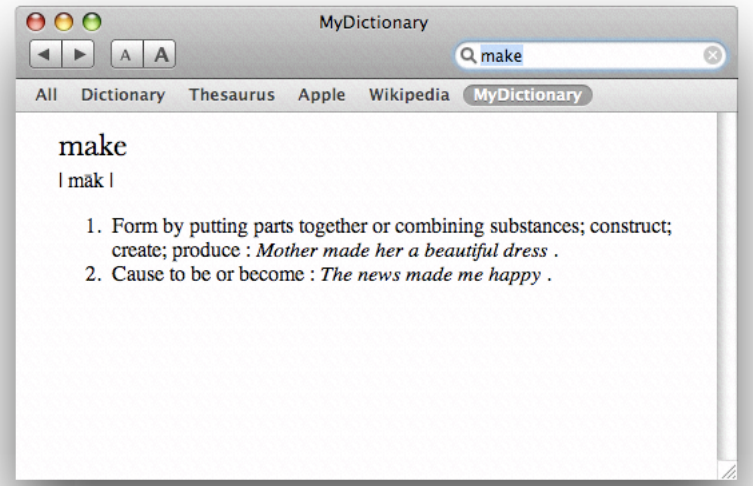
A dictionary entry is not limited to text. You can include other resources such as images (see Figure 1-2) , movies, sounds, and hyperlinks to webpages.

Figure 1-2 A dictionary entry that displays an image



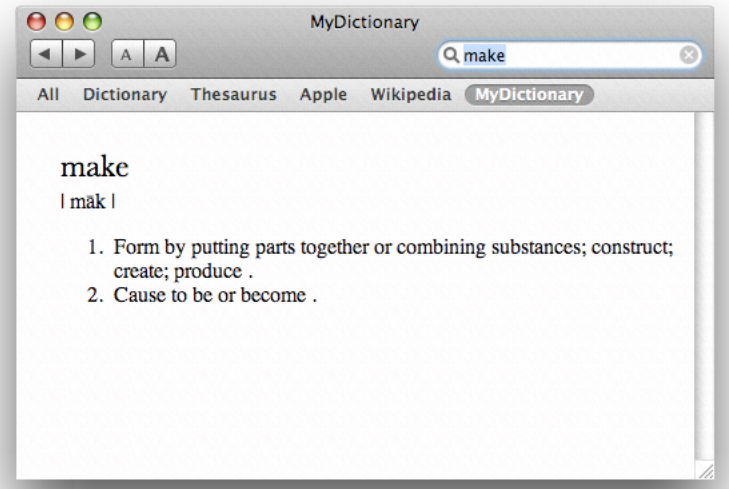
You can structure a dictionary to respect parental control user settings. (Parental controls are set in System Preferences by administrator users for restricted users.) By using the appropriate tag, as you'll see later in this chapter, you can suppress entries or particular portions of an entry. Figure 1-3 shows the same entry shown in Figure 1-1 (page 7), but as seen for a user that has parental controls turned on. The phrases portion of the entry is marked with a parental controls tag that suppresses its display.

Figure 1-3 Parental controls can suppress phrases



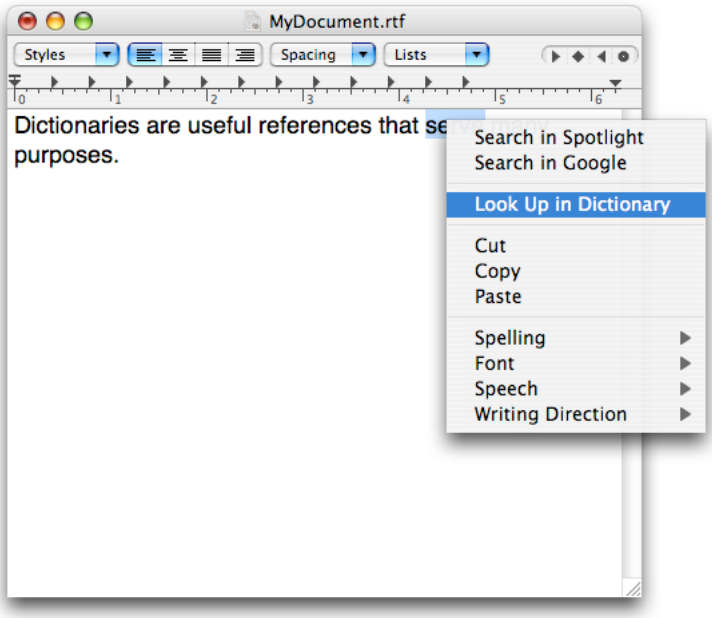
You can also markup entries with priority tags to control the depth of the information that's shown. Figure 1-4 shows the same entry as Figure 1-1 (page 7), but this time both parental controls and priority settings suppress content. The usage sentences shown in Figure 1-1 (page 7) are tagged with a priority setting that causes them not to be displayed in Figure 1-4.

Figure 1-4 Priority settings and parental controls limit an entry



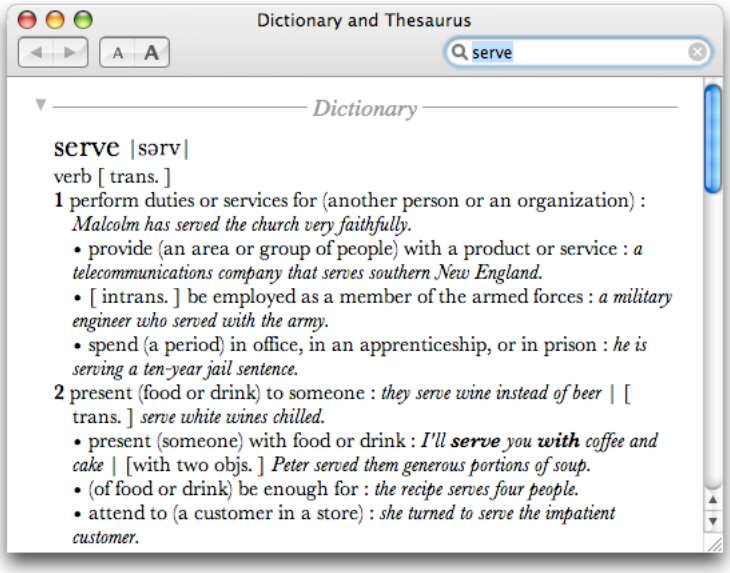
In addition to using the Dictionary application, users can access dictionary entries from within an application by control-clicking selected text. Then, in the contextual menu that appears, the user can choose Look Up in Dictionary, as shown in Figure 1-5.

Figure 1-5 The dictionary look-up command in the contextual menu



The Dictionary window opens to show the results of searching for the selected text in the active dictionaries. See Figure 1-6.

Figure 1-6 The Dictionary window



Dictionary Markup

You define the structure of a dictionary by marking up its content with XHTML and Dictionary Services markup items (elements, attributes, processing instructions, comments, and so forth).

- XHTML. You can use any XHTML markup except for those defined in the Structure module. You can find the DTD for XHTML modules on this website:

  http://www.w3.org/TR/xhtml-modularization/dtd_module_defs.html#a_dtd_module_defs

- Dictionary Services. Some markup defines the structure of the dictionary. Other markup attaches a specific functionality or semantics to XHTML elements. See Dictionary Services Markup (page 12).

Dictionary Services Markup

This section defines the markup items that are specific to Dictionary Services. The default namespace refers to XHTML. The namespace prefix `d` identifies elements and attributes that are specific to Dictionary Services.

Dictionary Content

`%dic.content` defines an element that can be used in various places.

It `%Flow.mix;` as XHTML does. The `Flow.mix` tag includes all text content, block, and inline elements.

```
<!ENTITY % dic.content
    "( #PCDATA | %Flow.mix; )*"
>

<!-- %Flow.mix; includes all text content, block and inline -->
<!ENTITY % Flow.mix
    "%Heading.class;
    | %List.class;
    | %Block.class;
    | %Inline.class;
    %Misc.class;"
>
```

Root Dictionary Element

`d:dictionary` defines the root element of a dictionary that contains one or more entries.

```
<!ELEMENT d:dictionary ( d:entry+ ) >
```

Entry

`d:entry` contains one or more index and a body part, and constructs the logical structure of a dictionary entry.

```
<!ELEMENT d:entry ( d:index+, %dic.content;) >
<!ATTLIST d:entry
    id                    ID        #REQUIRED
    d:title            NMTOKEN    #REQUIRED
    d:parental-control    ( 1 )        #IMPLIED
>
```

The attributes are:

- `id` ID of the entry. The value should be unique in the dictionary.
- `d:title` The title is the text that represents the entry. It should represent the entry or be the canonical form of the entry. You can use this text as the title of the window that displays the entry. You can also use it to fill the search text field of the Dictionary application.
- `d:parental-control` Parental control level indicates a restricted part of the entry. The value of this attribute should be "1". When parental control function is active, the entries that have this attribute are not searched and do not appear in the search result list.

Index

`d:index` defines information to extract from an entry and use to build the index. The index element should appear prior to other elements in the entry.

```
<!ELEMENT d:index EMPTY >
<!ATTLIST    d:index
    d:value    NMTOKEN    #REQUIRED
    d:title    NMTOKEN    #REQUIRED
    d:parental-control    NMTOKEN    #IMPLIED
    d:anchor    NMTOKEN    #IMPLIED
    d:yomi    NMTOKEN    #IMPLIED
>
```

The attributes are:

- `d:value` The search key text of for the entry.
- `d:title` The text that is displayed on the search result list. It can be used as the title of the window that displays the entry.
- `d:parental-control` Removes the title from search result list. See Parental Controls and Priority (page 15).
- `d:anchor` Highlights a specific part in an entry, such as the explanation of an idiomatic phrase.
- `d:yomi` Used only for Japanese dictionaries.

For example, the following entries for "make" indicates that a search using either *make*, *makes*, or *made* will return make as a result.

```
<d:index d:value="make"  d:title="make"/>
<d:index d:value="makes" d:title="makes"/>
<d:index d:value="made"  d:title="made"/>
<d:index d:value="make it" d:title="make it" d:parental-control="1"
d:anchor="xpointer(//*[@id='make_it'])"/>
```

Gaiji

`d:gi` defines characters that are not available in Unicode or for which the platform does not have glyph data. You can also use this element to specify a font that contains special characters. Use the `gi` element like an inline element.

```
<!ENTITY % inline.extra "| d:gi">
<!ELEMENT d:gi (#PCDATA) >
<!ATTLIST    d:gi
    d:set             NMTOKEN    #IMPLIED
    d:name            NMTOKEN    #IMPLIED
    d:ps-font-name    NMTOKEN    #IMPLIED
>
```

The attributes are:

- `d:set` The glyph set name. Only one is supported, "AdobeJapan1".
- `d:name` CID number (Character Identifier)

- `d:ps-font-name` The PostScript font name.

For example:

```
<d:gi d:set="AdobeJapan1" d:name="6930">□</d:gi>  # One of the character variants
<d:gi d:ps-font-name="Webdings">&#xF06A;</d:gi>   # Airplane symbol
```

Highlighting

You can highlight a portion of an entry using the `d:anchor` attribute of the `d:index` tag. The value of `d:anchor` must be an XPath expression that specifies the block in the entry to highlight. The `id` that you supply must be unique.

For example the `d:anchor` attribute in the following:

```
<d:entry id="make_1" d:title="make">
    ...
<d:index d:value="make it" d:parental-control="1"
d:anchor="xpointer(//*[@id='make_it'])"/>
```

causes the this part of the entry to be highlighted:

```
<div id="make_it"><b>make it</b> : succeed in something; survive.</div>
```

Parental Controls and Priority

`d:parental-control` identifies content that should not be shown for users that have parental controls enabled. (Parental controls are enabled in System Preferences.)

```
<!ATTLIST div d:parental-control NUMBER #IMPLIED >
<!ATTLIST span d:parental-control NUMBER #IMPLIED >
<!ATTLIST entry d:parental-control NUMBER #IMPLIED >
<!ATTLIST d:index d:parental-control NUMBER #IMPLIED >
```

The value is always `1`.

The `d:entry` tag can have this attribute. Entries that have this attribute don't appear on the search result list.

`d:priority` hides part of the content to allow for briefer content. It's typically used for results displayed in the Dictionary window.

```
<!ATTLIST div d:priority NUMBER #IMPLIED >
<!ATTLIST span d:priority NUMBER #IMPLIED >
```

The value can be in the range of `0` to `9`, inclusive. Content that has a priority value greater than `1` is not displayed in the Dictionary window. The behavior is defined only for `0` and `2`. All other values are reserved for future use.

Dictionary Services applies the following rules to content with `d:parental-control` and `d:priority` tags. (Note that these rules are part of the RELAX NG definition.)

- If you don't specify a parental control or priority tag for an element, the value is assumed to be `0`.
- An element that doesn't have a parental control or priority tag inherits the value of its parent element.
- A child element can have a parental control or priority tag whose value is equal to or greater than the value of that of its parent. If a child element has a parental control or priority tag whose value is smaller than its parent, the value of the parent applies. In other words, if the parent element isn't shown, neither is the child element. But if a parent element is shown, you can hide the child element.

Pronunciation

`pr` marks the pronunciation of the entry. You don't supply a `value` attribute because all values are acceptable.

```
<!ATTLIST div d:priority NMTOKEN #IMPLIED >
<!ATTLIST span d:priority NMTOKEN #IMPLIED >
```

For example:

```
<h1>make</h1>
<span class="syntax"><span d:pr="1">| māk |</span></span>
```

You can use this tag to switch pronunciation notation according to dictionary-specific preferences.

Reference Link

You can implement a hypertext link as follows:

```
<h3>PHRASES</h3>
    ...
    <h4><a href="x-dictionary:r:make_up_ones_mind"><b>make up one's
mind</b></a></h4>
```

When the user clicks the link, it jumps to the following dictionary entry whose id is `make_up_ones_mind`:

```
<d:entry id="make_up_ones_mind" d:title="make up one's mind" d:parental-control="1">
    <d:index d:value="make up one's mind"/>
    <h1>make up one's mind</h1>
    <ul>
        <li>
        make a decision.
        </li>
    </ul>
</d:entry>
```

For other variations, see URI Scheme (page 17).

URI Scheme

`x-dictionary:` is an URI scheme that describes cross references between entries in dictionaries. It is used in tag such as `<a href="x-dictionary:r:another_id">`.

The `x-dictionary:` URI contains three elements separated by colons as the general form—target selector, target text, and dictionary bundle ID. The target selector must be either `d` (for definition) or `r` (for reference). Use `d` if you want to search definitions of the following key text. Use `r` if you want to refer to the entry specified by the reference ID which must be unique to each dictionary.

```
x-dictionary:d:key_text:dict_bundle_id
x-dictionary:r:reference_id:dict_bundle_id
```

The dictionary bundle ID can be omitted in both forms, as shown in the following lines. If it is omitted, Dictionary Services searches the target text in all active dictionaries.

```
x-dictionary:d:key_text
x-dictionary:r:reference_id
```

Yomi

The `d:yomi` attribute marks Japanese Yomi.

For details on Yomi and marking Yomi content in a Japanese dictionary, see Creating a Japanese Dictionary (page 29).

Creating Dictionaries

Prior to building a dictionary, you need to create your dictionary source file, prepare a style sheet, edit the property list file, and add any resources needed by your dictionary. This chapter explains how to perform these tasks, provides instructions for building your dictionary, and shows a few simple examples. You'll also find out how to create a Japanese dictionary, set up preferences, and add front and back matter.

Preparing Source Data and Building a Dictionary

Before you start preparing your source data, copy the `project_template` folder from the development kit to the directory that you use for code development. Then, follow the instructions described in the following sections:

1. Preparing Dictionary Contents (page 18)
2. Editing the Property List File (page 19)
3. Adding Resources Needed by Your Dictionary (page 20)
4. Preparing the Makefile (page 21)

Preparing Dictionary Contents

Take a look at the `MyDictionary.xml` file provided in the project template. Your dictionary should follow this form, using UTF-8 encoding. You can change the file name to something other than `MyDictionary`, but if you change the name, you must edit the `DICT_SRC_PATH` variable in the makefile.

Dictionary User Interface and Markup (page 6)) describes the XML schema you should use to develop a dictionary. The schema uses the RELAX NG schema language, which is described on this website:

http://www.relaxng.org/

You should validate your dictionary source prior to building a new dictionary You can use RELAX NG validators programs, which are available from this website:

http://www.relaxng.org/#validators

You can also validate XML using jing as follows:

```
$ java –jar <path to jing.jar> <schema definition> <XML to validate>
```

From the project_template folder, with jing located in ../jing/, the command line is as follows:

```
$ java –jar ../jing/bin/jing.jar
../documents/DictionarySchema/AppleDictionarySchema.rng MyDictionary.xml
```

For more information on jing, see:

http://www.thaiopensource.com/relaxng/jing.html

Preparing a Style Sheet

You can prepare a style sheet to use for the contents of the dictionary by editing the `MyDictionary.css` file provided with the project template. If you change the name of this css file, you need to edit the `CSS_PATH` variable in the makefile.

You should minimally edit the style sheet. The Dictionary application and the Dictionary window control use their own style definitions to ensure that the contents fit the display. For best results, do not specify an absolute font or font size.

Editing the Property List File

The property list file for a dictionary is an an XML text file. The project template contains an example file—`MyInfo.plist`—whose contents are shown in Listing 2-1. You can edit this file so that it contains entries appropriate for your dictionary. Table 2-1 (page 20) explains the values that you need to provide for your dictionary.

> **Note:** The property list cannot use binary format. If you need to, you can convert a binary property list file to an XML text file using the `plutil` command.

Listing 2-1 An example of a property list file for a dictionary

```
<key>CFBundleDevelopmentRegion</key>
<string>English</string>
<key>CFBundleDisplayName</key>
<string>My Dictionary</string>
<key>CFBundleIdentifier</key>
<string>com.apple.dictionary.MySample</string>
```

```
<key>CFBundleName</key>
<string>MyDictionary</string>
<key>CFBundleShortVersionString</key>
<string>1.0</string>
<key>DCSDictionaryCopyright</key>
<string>Copyright (c) Apple Computer, Inc.</string>
<key>DCSDictionaryManufacturerName</key>
<string>Apple Computer, Inc.</string>
```

If you change the name of the property list file, you must edit the `PLIST_PATH` variable in the makefile.

Table 2-1 Keys and values for the dictionary property list file

Key	Value
`CFBundleDevelopmentRegion`	A region
`CFBundleDisplayName`	The full display name of the dictionary. The default is to use the file system name.
`CFBundleIdentifier`	The identifier of the dictionary bundle; specify a unique ID.
`CFBundleName`	The short display name of the dictionary.
`CFBundleShortVersionString`	The version of the dictionary.
`DCSDictionaryCopyright`	The copyright notice of the dictionary.
`DCSDictionaryManufacturerName`	The manufacturer name of the dictionary.

Adding Resources Needed by Your Dictionary

You must place any resources (for example, images) that your dictionary needs in the OtherResources folder in the `project_template` folder. When you build the dictionary, the resources are copied into the built dictionary.

For example, if your dictionary uses an image file name `test.png`, you need to place it in the following location:

`project_template/OtherResources/Images/test.png`

The Images folder is copied into the dictionary. When the dictionary needs the image, it uses to the relative path—`Images/test.png`—which is written to the XML file during the build process.

Preparing the Makefile

Name your dictionary and then edit the `DICT_NAME` variable in the makefile. This name is used as the folder name for the dictionary. For example, when `DICT_NAME = "My Dictionary"`, the built dictionary is `My Dictionary.dictionary`.

If you change the location of the Dictionary Development Kit from `/Developer/Extras/Dictionary Development Kit`, you must modify the `DICT_BUILD_TOOL_DIR` variable in the makefile to reflect the change.

Building the Dictionary

To build your dictionary, follow these steps:

1. Launch the Terminal application.
2. Use the `cd` command to change to the appropriate location:

   ```
   /Developer/Extras/Dictionary Development Kit
   ```

3. Enter `make`.
4. After the make process finishes successfully, type make install to copy the new dictionary to:

   ```
   ~/Library/Dictionaries/
   ```

After running the `make install` command, you can delete all the intermediate files in the objects folder. In the Terminal window, enter make clean to remove the objects folder.

Now you can launch the Dictionary application and test your new dictionary.

As you can see from looking at the makefile, the building process uses a script `build_dict.sh`, located in `/Developer/Extras/Dictionary Development Kit/bin`. This script takes 4 arguments—`dictionary_name`, `dictionary_source_path`, `StyleSheet_path`, and `InfoPlist_path`. It builds a new directory in the `/objects` folder.

A One-Word Dictionary Example

Yomi shows a simple example of a dictionary that contains an entry for the word *make*. It looks like Figure 1-1 when opened using the Dictionary application.

Listing 2-2 A one-word dictionary

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<d:dictionary xmlns="http://www.w3.org/1999/xhtml"
xmlns:d="http://www.apple.com/DTDs/DictionaryService-1.0.rng">
<d:entry id="make_1">
    <d:index d:value="make" d:title="make"/>
    <d:index d:value="makes" d:title="makes (make)"/>
    <d:index d:value="made" d:title="made (make)"/>
    <d:index d:value="making" d:title="making (make)"/>
    <h1>make</h1><span class="syntax">| māk |</span>
    <div>
        <ol>
            <li>
               Form by putting parts together or combining substances; construct;
 create; produce
                <span d:priority="2"> : <i>Mother made her a beautiful dress</i>
                </span>
                .
            </li>
            <li>
                Cause to be or become
                <span d:priority="2"> : <i>The news made me happy</i>
                </span>
                .
            </li>
        </ol>
    </div>
    <div d:parental-control="1" d:priority="2">
        <h3>PHRASES</h3>
        <div id="make_it"><b>make it</b> : succeed in something; survive.</div>
        <h4><a href="x-dictionary:r:make_up_ones_mind"><b>make up one's
mind</b></a></h4>
    </div>
</d:entry>
</d:dictionary>
```

An Acronym Dictionary Example

Listing 2-3 shows how to tag the following content to create an acronym dictionary.

- LDAP, Lightweight Directory Access Protocol
- MIDI, Musical Instrument Digital Interface
- XML, Extensible Markup Language

Listing 2-3 An acronym dictionary

```
<?xml version="1.0" encoding="UTF-8"?>
<d:dictionary xmlns="http://www.w3.org/1999/xhtml"
xmlns:d="http://www.apple.com/DTDs/DictionaryService-1.0.rng">
<d:entry id="ldap">
    <d:index d:value="LDAP" d:title="LDAP"/>
    <h1>LDAP</h1>
    <p>Lightweight Directory Access Protocol</p>
</d:entry>
<d:entry id="midi">
    <d:index d:value="MIDI" d:title="MIDI"/>
    <h1>MIDI</h1>
    <p>Musical Instrument Digital Interface</p>
</d:entry>
<d:entry id="xml">
    <d:index d:value="XML" d:title="XML"/>
    <h1>XML</h1>
    <p>Extensible Markup Language</p>
</d:entry>
</d:dictionary>
```

Adding Front and Back Matter to a Dictionary

You can add front and back matter to a dictionary by following these steps:

1. Prepare an Entry for the Front and Back Matter (page 24)
2. Modify the Info.plist File (page 24)
3. Add an Index Entry (Optional) (page 24)

You can view the front and back matter for a dictionary using the Dictionary application. In the Go menu, choose Front/Back Matter.

Prepare an Entry for the Front and Back Matter

In the XML file for the dictionary, you need to specify a front-back matter entry using an `id` attribute whose value is set to `front_back_matter` as shown in the following simple example:

```
<d:entry id="front_back_matter" d:title="Front/Back Matter">
    <h1><b>My Dictionary</b></h1>
    <h2>Front/Back Matter</h2>
    <p>
        This is a front matter page of the sample dictionary.<br/>
    </p>

    ...

</d:entry>
```

Modify the Info.plist File

You must modify the property list for the dictionary by adding the `DCSDictionaryFrontMatterReferenceID` key to the `Info.plist` file. The associated value is a string that specifies the `id` value you used in the XML file. The following shows the string used in Prepare an Entry for the Front and Back Matter (page 24).

```
<key>DCSDictionaryFrontMatterReferenceID</key>
<string>front_back_matter</string>
```

Add an Index Entry (Optional)

This example does not use a `<d:index>` element, which means that the page won't show up in a search. If you want the front and back matter to show in a search, add the`<d:index>` element. Otherwise, users can view the front and back matters by choosing Go > Front/Back Matter from within the Dictionary application.

Setting Up Preferences for a Dictionary

You have the option to set up preferences for a dictionary. Users access the preferences settings from within the Dictionary application by choosing Dictionary > Preferences. For example, the New Oxford American Dictionary provided with OS X v10.5 allows users to choose from among three phonetic notations—U.S. English (Diacritical), U.S. English (IPA), or British English (IPA).

This section shows how you can set up preferences for your dictionary. You can find the files associated with this example in the Dictionary Development Kit located in:

`/Developer/Extras/Dictionary Development Kit/samples/`

To implement dictionary-specific preferences, follow these steps:

1. Modify the Dictionary Contents Appropriately (page 25).
2. Prepare an XSLT File to Apply to Dictionary Entries (page 26).
3. Implement the Preferences User interface (page 27).
4. Modify the Info.plist File (page 28).

Modify the Dictionary Contents Appropriately

You need to modify the contents to support the preferences that you set up. For example, if you want to allow the user to choose from among three phonetic notations, you need to provide the three phonetic notations for each entry in our dictionary. The following example shows three phonetic notations for the word `make`.

```
<d:entry id="make_1" d:title="make">
    ...
    <h1>make</h1>
    <span class="syntax">
        <span d:pr="US">| māk |</span>
        <span d:pr="US_IPA">| meɪk |</span>
        <span d:pr="UK_IPA">| meɪk |</span>
    </span>
    ...
</d:entry>
```

Note that the XML does not specify which phonetic notation to show. You'll do that in the next section by creating an XSLT file.

Prepare an XSLT File to Apply to Dictionary Entries

The XSLT files contains instructions that Dictionary Services applies to each entry before displaying the it. In this example, you need to provide instructions to remove the unused phonetic notation. For this, use the `$pronunciation` variable, which is a global variable supply by the Dictionary application, as shown in the following example.

> **Note:** You need to place the XSLT file in the Other Resources folder of your dictionary project.

```
<xsl:template match="*[@d:pr='US']">
    <xsl:if test="$pronunciation = '0'">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()" />
        </xsl:copy>
    </xsl:if>
</xsl:template>

<xsl:template match="*[@d:pr='IPA']">
    <xsl:if test="$pronunciation = '1'">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()" />
        </xsl:copy>
    </xsl:if>
    <xsl:if test="$pronunciation = '2'">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()" />
        </xsl:copy>
    </xsl:if>
</xsl:template>

<xsl:template match="*[@d:pr='US_IPA']">
    <xsl:if test="$pronunciation = '1'">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()" />
        </xsl:copy>
```

```
    </xsl:if>
</xsl:template>

<xsl:template match="*[@d:pr='UK_IPA']">
    <xsl:if test="$pronunciation = '2'">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()" />
        </xsl:copy>
    </xsl:if>
</xsl:template>
```

Implement the Preferences User interface

You need to provide an XHTML file that specifies the user interface to present in the Dictionary Preferences window in the Dictionary application. The following example shows how to set up preferences for phonetic notation. The user will see three choices displayed as radio buttons. After making a selection, Dictionary Services saves the it and passes the selection to the XSLT instructions. The instructions are then applied to the dictionary entries.

Note: You need to place the XHTML file in the Other Resources folder of your dictionary project.

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    </head>
    <body>
        <div id="copyright"></div>
        <hr />
        <div class="query">
            <input type="hidden" name="version" value="1" />
        </div>
        <div class="query">
            Pronunciation:<br />
            <input type="radio" name="pronunciation" value="0">US English
(Diacritical)</input><br />
```

```
            <input type="radio" name="pronunciation" value="1">US English
(IPA)</input><br />
            <input type="radio" name="pronunciation" value="2">British English
(IPA)</input><br />
        </div>
    </body>
</html>
```

Modify the Info.plist File

You must add keys to the `Info.plist` file to indication that the dictionary has its own preferences. The values for this example are shown in the following property list entry. You need to change the values to ones that are appropriate for your dictionary. The values expected for each of the keys are shown in Table 2-2 (page 28).

```
<key>DCSDictionaryDefaultPrefs</key>
    <dict>
        <key>pronunciation</key>
        <string>0</string>
        <key>version</key>
        <string>1</string>
    </dict>
    <key>DCSDictionaryPrefsHTML</key>
    <string>MyDictionary_prefs.html</string>
    <key>DCSDictionaryXSL</key>
    <string>MyDictionary.xsl</string>
```

Table 2-2 Values for dictionary preferences keys

Key	Value
`DCSDictionaryDefaultPrefs`	None. This key specifies that key-value pairs for the default values follow.
`DCSDictionaryPrefsHTML`	The XHTML file name
`DCSDictionaryXSL`	The XSLT file name

Creating a Japanese Dictionary

The Japanese language has multiple character classes—Hiragana, Katakana, and Kanji. Because of this, Japanese words often have multiple representations. Kanji uses ideographic characters; Hiragana and Katakana use phonograms. (Foreign words are often represented using Katakana. Roman characters also are used.) A Kanji word has its Yomi that represents its reading. Hiragana or Katakana usually describe Yomi.

> **Note:** **Yomi** means the reading of a character, word, phrase, or sentence. For example, the word "□ □" has "□" as its Yomi.

Entries in many Japanese word dictionaries are sorted by Yomi. In such dictionaries, the user enters Yomi to search for words. Using Yomi for Hiragana and Katakana is not an issue because these are phonograms, and there aren't many of them. But for Kanji, it is an issue because the same Yomi can represent different Kanji. For example the Yomi "□" represents many words: □□,□□,□□,... □□,□□, ,

A user who searches for "□" gets these results:

□□ □□ □

□□ □□ □

□□ □□ □

□□ □□ □

□□ □□ □

□□ □□ □

□□ □□ □

...

□□ □□ □ □

□□ □□ □ □

...

The user can choose the appropriate entry using the additional Kanji part.

If the search word is "□ □", the results are:

□□ □□ □□ □

□□ □□ □□ □

Here are search results for "□". The Yomi □ and □ are added to choose appropriate entry.

□□ □□ □ □ □

□□ □□ □ □ □

You need to mark Yomi content in your Japanese dictionary using the `d:yomi` attribute. The Dictionary application displays the `d:title` and `d:yomi` in the appropriate order in the search results list. When the user searches using Hiragana, the Yomi appears before the Kanji. When the user searches using Kanji, the Yomi is added.

For the words in a Japanese dictionary that have multiple representations, the value associated with the `d:title` element can be different from the value associated with the `d:value` element. To ensure that users can find what they search for, use the `d:yomi` attribute with the `d:index` element.

In a Japanese dictionary, a Kanji entry has both Kanji and Yomi as its `d:value` attribute of the `d:index` element. The `d:title` markup in its ordinary form is:

```
<d:entry ... d:title="□□">
    <d:index d:value="□□□" d:title="□□"/>
    <d:index d:value="□□" d:title="□□"/>
    ...
</d:entry>
```

You can add Yomi using the `d:yomi` attribute as shown:

```
<d:entry ... d:title="□□">
    <d:index d:value="□□□" d:title="□□" d:yomi="□□□"/>
    <d:index d:value="□□" d:title="□□" d:yomi="□□□"/>
    ...
</d:entry>
<d:entry ... d:title="□□">
    <d:index d:value="□□□" d:title="□□" d:yomi="□□□"/>
    <d:index d:value="□□" d:title="□□" d:yomi="□□□"/>
    ...
</d:entry>
```

The Dictionary application uses `d:yomi` and `d:title` to add supplementary information either for searching by Yomi or searching by Kanji. You can omit `d:title` when it has the same value as `d:value`.

The Dictionary Development Kit contains the source for a sample Japanese dictionary (see `/Developer/Extras/Dictionary Development Kit/samples/`). The dictionary contains entries for various cases, including the following:

- A word that is written using multiple, mixed character classes.
- A foreign word that has both Katakana and Roman representations.

Accessing Dictionaries

Although the Dictionary Services programming interface provides only a few functions, these functions provide two valuable services. You can:

- Search for and obtain definitions from active dictionaries
- Display a Dictionary window that shows results of a dictionary search.

Getting a Definition

There are two steps you need to perform to get a definition:

1. Call the function `DCSGetTermRangeInString` to obtain the range in a dictionary that contains the definition for a word or phrase.
2. Pass that range to the function `DCSCopyTextDefinition` to get the definition associated with the text range and word or phrase. Dictionary Services returns the definition as a `CFString` object, which is plain text.

An alternative to getting the definition returned to you is to let Dictionary Services obtain and display the search results for you. In this case, instead of calling `DCSCopyTextDefinition`, you would pass the text range to the `HIDictionaryWindowShow` function.

Supporting Contextual Lookup

You may want to use the `HIDictionaryWindowShow` function to support contextual look-up in any application that provides text editing. For example, you could set up your application to respond as follows when the user control-clicks a text selection:

- Get the text selection and supply that as the `textString` parameter. If you use a `CFString` object for this text, you also need to specify a Core Text font to use to display the text. Otherwise, Dictionary Services uses the attributes provided.
- Determine the range of the selected text and provide that as the `selectionRange` parameter.
- Determine the origin of the typographic baseline of the selected text and provide that as the `textOrigin` parameter.

For More Information

Dictionary Services Reference describes functions and their parameters in detail.

Document Revision History

This table describes the changes to *Dictionary Services Programming Guide*.

Date	Notes
2007-05-30	New document that explains how to create a dictionary and access it programmatically.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010