Documentation Set Guide

(Legacy)

Developer

Contents

Figures and Listings

Introduction

Important: This document may not represent best practices for current development. Links to downloads and other resources may no longer be valid.

Xcode, Apple's integrated development environment, includes documentation access features that allows users to easily search and view Apple's developer documentation. If your documentation is properly packaged, it can also take part in these features, and appear in Xcode's documentation and Quick Help windows.

This document explains how to package and build a documentation set for use with Xcode. If you have a developer-targeted software product, this document shows you how to integrate documentation for that product with the Xcode Documentation window. This document assumes that you have existing HTML or PDF documentation files; it does not describe how to write or generate these documentation files.

Before reading this document, you should be familiar with the documentation viewing and access features that are available in Xcode. For a complete description of these features and how to use them, see Documentation Access in *Xcode Workspace Guide*.

In this document the term **documentation producer** (or producer for short) identifies a person involved in creating documentation sets. The term **documentation user** (or user for short) refers to Xcode users who access documentation sets installed on their file systems using the Xcode Documentation window or Quick Help.

Organization of This Document

This document contains the following chapters:

- "Documentation Sets" (page 12) introduces the documentation set bundle and provides an overview of how to create a documentation set.
- "Creating Documentation Sets" (page 15) describes how to create and name the folder hierarchy for your documentation set bundle, as well as how to choose where to install that bundle.
- "Configuring Documentation Sets" (page 18) introduces the `Info.plist` and `Nodes.xml` files, which are required for any documentation set. This chapter shows how to use these files to describe the documentation set and its contents.

- "Supporting API Lookup in Documentation Sets" (page 36) shows how to create a tokens file to support API search in your documentation set.
- "Indexing Documentation Sets" (page 52) describes the `docsetutil` indexing tool and shows how to use it to index your documentation set, so that Xcode can access and display its contents.
- "Internationalizing Documentation Sets" (page 54) shows how to support multiple languages by localizing all or some of the contents of the documentation set bundle.
- "Acquiring Documentation Sets Through Web Feeds" (page 59) describes how to support automatic detection and downloading of documentation set updates, using an RSS or Atom feed.
- "Testing and Packaging Documentation Sets" (page 66) tells how to test documentation set indexes and whether Xcode can access and display your documentation. Also shows how to package your documentation set bundle as an XAR archive.
- "docsetutil Reference" (page 72) describes the command-line utility for creating, testing, and querying full-text and API indexes for a documentation set.
- "Documentation-Set Property List Key Reference" (page 75) describes the keys that Xcode recognizes.
- "Documentation-Set Nodes Schema Reference" (page 79) lists all of the elements supported in the nodes file format.
- "Documentation-Set Tokens Schema Reference" (page 88) lists all of the elements supported in the tokens file format.

See Also

In addition to the material in this document, you may find the following resource helpful:

- The AtomEnabled website provides information about the Atom format, which you'll find useful if you plan to provide Atom feeds for documentation sets.

Documentation Sets

The Xcode IDE includes a full-featured documentation viewer that lets you view the installed developer documentation. The Documentation window of Xcode, available starting in Xcode 1.0, provides integrated searching and viewing of Apple's developer documentation. Xcode also provides Quick Help, which is a lightweight window for displaying reference documentation within Xcode's text editor.

Starting with Xcode 3.0, you can integrate documentation for your own products into the Xcode Documentation window, by packaging your documentation as a documentation set. Users can take advantage of all the Xcode documentation-viewing features to search for and look at information in your documents.

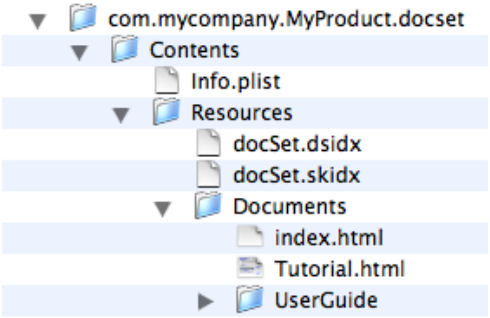
This chapter describes the documentation-set bundle and provides an overview of how to construct a documentation set. For a description of how documentation sets appear to the user and how users get them, see Documentation Access in *Xcode Workspace Guide*.

The Documentation Set Bundle

A **documentation set** is a standard OS X bundle. The structure of a documentation set follows the conventions described in *Bundle Programming Guide*. Therefore, a documentation set's content, including documentation files, can be localized.

The basic structure of an installed documentation set bundle is similar to that shown in Figure 1-1.

Figure 1-1 The structure of an installed documentation set bundle



The documentation set bundle contains the following (for the structure of a localized documentation set bundle, see "Internationalizing Documentation Sets" (page 54)):

- An information property list (`Info.plist`) file. This file describes the documentation set bundle. Xcode uses this file to:
  - Display the publisher name in documentation preferences
  - Display the documentation set name in the documentation set list in Documentation preferences
  - Display information about the availability of updates to the documentation set and optionally download and install those updates
- HTML documentation files. These must be placed inside the `Documents` directory.
- Generated index files. The `docSet.dsidx` and `docSet.skidx` files are binary files that describe the symbols and documents in the documentation set. Xcode uses the information in these files to:
  - Display the documentation set's contents in the Documentation window
  - Carry out full-text searches of the HTML documentation files
  - Associate symbol names with locations in the HTML documentation
  - Provide symbol information that is displayed in the Quick Help window

To generate the index files, you must also include several XML metadata files in your documentation set bundle. These files, which describe the contents of the documentation set, are:

- A `Nodes.xml` file. This file describes the structure of the documentation set. This file is required.
- One or more `Tokens.xml` files. These files (known as **tokens files**) associate symbol names with locations in the documentation and are used to create the symbol index for a documentation set. Although optional, you must include a tokens file to support fast API lookup.

These metadata files must reside within the documentation set bundle when you index the documentation set, but can then be removed.

Documentation Set Development Workflow

The basic steps for creating a documentation set are:

1. Organize documentation files into the bundle structure.

   The first step to building a documentation set is to create the directory hierarchy of the documentation set bundle and populate it with your HTML documentation files.

2. Describe the documentation set and its structure.

The next step in building a documentation set is to create an `Info.plist` file and `Nodes.xml` file for the documentation set. The `Info.plist` file describes the overall characteristics of the documentation set, such as its name and version number. The `Nodes.xml` file describes the structure of the documentation set.

3. Support API lookup.

   If you wish to support API lookup for your documentation set, you must also include one or more `Tokens.xml` files.

4. Localize the documentation set.

   If you provide documentation in more than one language, you can localize all or part of the documentation set bundle. This step is optional.

5. Index the documentation set.

   The `docsetutil` tool generates the full-text (`docSet.skidx`) and API (`docSet.dsidx`) indexes that Xcode uses to access your documentation.

6. Provide an RSS or Atom feed.

   You can take advantage of automatic documentation updates in Xcode by providing an RSS or Atom feed that publishes updated versions of your documentation set. This step is optional.

7. Test, package, and distribute the documentation set.

   When your documentation set is complete, you can test it in Xcode simply by placing it in one of the standard documentation locations. You can also use the `docsetutil` tool to test the generated index files. After you have made sure that the documentation set and its indexes are complete and correct, you can use `docsetutil` to package the documentation set bundle as an XAR archive for distribution. The `docsetutil` tool can also generate or update the Atom feed that advertises your documentation set to Xcode users.

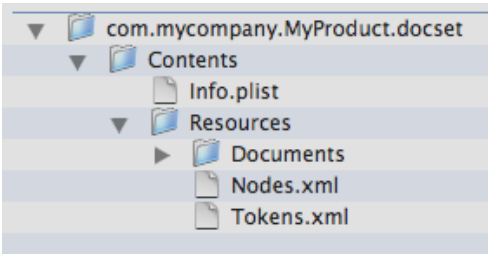
Creating Documentation Sets

A documentation set is a standard OS X bundle that packages a set of HTML (or PDF) documentation files, along with the information needed to access and display that documentation in the Xcode Documentation window. This chapter describes the directory hierarchy, naming conventions, and file-system locations of a documentation set bundle.

Organizing Documentation Set Bundles

The first step in creating a documentation set is to create and populate the folder hierarchy of the documentation set bundle. The documentation set structure that you create should look similar to that shown in Figure 2-1.

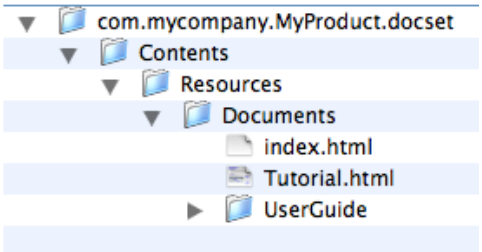
Figure 2-1 Creating the bundle hierarchy



You must place all of a documentation set's documentation files inside the `Resources/Documents` directory.

Within the `Documents` directory, you can use any arbitrary hierarchy to organize your documentation files. In this example (see Figure 2-2), the documentation set contains a `Tutorial.html` file and a directory of additional documentation files, `UserGuide`.

Figure 2-2 Organizing documentation files



Xcode and `docsetutil` indexing tool support both HTML and PDF documentation files. Your documentation set can contain files in either format.

The `Documents` directory, like any other bundle resource, can be localized. For the structure of a localized documentation set bundle, see "Internationalizing a Documentation Set Bundle" (page 56).

Naming Documentation Set Bundles

Because documentation sets are installed in standard locations, documentation set bundle names should be chosen to minimize the chance of conflicts. You should choose a name for each documentation set that conforms to the uniform type identifier (UTI)–style used for the `CFBundleIdentifier` property, as described in Property List Key Reference.

Documentation set bundle names must include the `docset` extension. So, for example, a documentation set bundle name would look like this:

```
com.mycompany.MyDocSet.docset
```

Note that the bundle name is separate from the displayed name, which is controlled by the `CFBundleName` property of the `Info.plist`.

Choosing Documentation Set Locations

When it first displays the Documentation window, Xcode scans for all installed documentation sets. It searches for these documentation sets in the following locations, listed in the order in which they are searched:

1. `<Xcode>/Documentation/DocSets`.

   In Xcode 3.0 through Xcode 4.2.1, users can install multiple versions of Xcode on a single computer. Xcode always searches for documentation sets within the `<Xcode>` directory in which the active Xcode binary resides. For example, if the user has installed Xcode 3.0 at `/Xcode 3.0`, Xcode searches for documentation sets at `/Xcode 3.0/Documentation/DocSets`. (The default location for Xcode is `/Developer`.)

2. `Applications/Xcode.app/Contents/Developer/Documentation/DocSets`.

   In Xcode 4.3 and later, Xcode has been repackaged as a single app bundle, eliminating the need for the `/Developer` directory. This location contains documentation sets packaged with the app.

3. `~/Library/Developer/Shared/Documentation/DocSets`. This location contains documentation sets specific to the user that is currently logged in to the system.

4. `/Library/Developer/Shared/Documentation/DocSets`. This location contains documentation sets viewable by all users of the current system.

5. `/Network/Library/Developer/Shared/Documentation/DocSets`. This location contains documentation sets used by a group of people on the same network.
6. `/System/Library/Developer/Shared/Documentation/DocSets`. You should not install files in the system domain.

If the same documentation set appears in more than one location, Xcode uses the most recent version. It they have the same version, then Xcode uses the first instance that it finds.

You should install your documentation set in the same file system domain as the software product that it documents. For example, if your product is installed in the local domain—say, at `/Applications/MyProduct`—your documentation set should be installed at `/Library/Developer/Shared/Documentation/DocSets`.

In some cases, you may want to install your documentation set in a location other than the standard documentation set locations. You can make sure that Xcode finds and accesses your documentation set by creating a symbolic link (symlink) to your documentation set and placing that symlink in one of the standard documentation locations.

After the initial installation, users can move any documentation set—with the exception of the Developer Tools documentation set provided by Apple—to any of the standard locations.

Configuring Documentation Sets

A documentation set bundle contains documentation content and indexes into that content. There are two files that identify a documentation set to Xcode and describe its content:

- `Info.plist`: Specifies the general characteristics of the documentation set, including its name, provider, bundle identifier, and so forth.

  This file must be contained in a documentation set; otherwise, Xcode doesn't recognize the bundle as a documentation set.

- `Nodes.xml`: Describes the structure and contents of the documentation set.

  This file is needed only to create the documentation set's index. It's not required for publication.

The following sections describe these two files and show how to use them to describe the documentation set.

Identifying Documentation Sets

Every documentation set needs to include an `Info.plist` file that identifies the documentation set.

You can create the `Info.plist` file manually, using the Property List Editor application (`<Xcode>/Applications/Utilities`), or programmatically, using the `NSDictionary` Cocoa class. This section assumes that you create the `Info.plist` file using Property List Editor.

When you launch Property List Editor, it automatically creates a file for you and populates it with a root element. You can add new child or sibling items, depending on the currently selected item. For more information, see *Property List Programming Guide*.

The `Info.plist` file follows the standard OS X conventions for property list files, described in *Runtime Configuration Guidelines*.

The property list keys that are relevant to documentation sets are described in "Documentation-Set Property List Key Reference" (page 75). However, in order for Xcode to properly recognize your documentation set, Apple recommends that you provide values for all the following keys in your `Info.plist` file, even those marked optional:

- **DocSetPublisherName.** The publisher name specifies the name of the publisher to which the documentation set belongs. It provides an umbrella under which multiple documentation sets from a single publisher can be grouped. In Xcode's Documentation Preferences pane, documentation sets are grouped under the publisher name, as provided by this key. This key is optional.
- **DocSetPublisherIdentifier.** Specifies the unique identifier for the publisher. All documentation sets that have the same publisher identifier are grouped under the same publisher name, even if the documentation sets are provided by different feeds. The identifier should be a reverse domain-name style string. For example, `com.mycompany.documentation`. This key is optional.
- **CFBundleName.** Specifies the name of the documentation set, as it appears under the publisher name in Documentation preferences.
- **CFBundleIdentifier.** A string that uniquely identifies the documentation set bundle. This should be a reverse domain-name style string. For example, `com.mycompany.MyDocSet`.

  Xcode uses this identifier to match an installed documentation set bundle to an entry in the Atom feed specified by the "DocSetFeedURL" (page 77) key. This value is assumed to be unique. As it searches for documentation sets on a user's file system, Xcode loads only sets with identifiers it hasn't loaded before. Therefore, if there are more than one documentation set with the same bundle identifier on a user's file system, only one of them (Either the one with the greatest version number or the first one found) is loaded.

A minimal `Info.plist` file might look like that shown in Listing 3-1; you can get output like this by saving the file as an XML plist and then opening it in a text editor.

Listing 3-1 A minimal `Info.plist` file

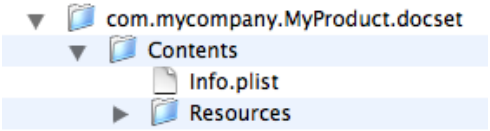
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
     <key>CFBundleName</key>
     <string>My Product Documentation</string>
     <key>CFBundleIdentifier</key>
     <string>com.mycompany.MyProduct.docset</string>
     <key>DocSetPublisherIdentifier</key>
     <string>com.mycompany.documentation</string>
     <key>DocSetPublisherName</key>
     <string>My Company</string>
</dict>
```



```
</plist>
```

When you are done creating the `Info.plist` file, place it in the documentation set bundle directly inside the `Contents` directory, as shown in Figure 3-1.

Figure 3-1 Placement of the `Info.plist` file



Describing Documentation Sets

To generate the necessary index files for your documentation set using the `docsetutil` tool, you must include a single nodes file (`Nodes.xml`). The **nodes file** describes the hierarchical structure of the documentation set.

The structure of a documentation set consists of one or more documentation nodes, each represented by a `Node` element. This section describes documentation nodes and shows how to construct a node definition. It then goes on to describe how to build a table of contents from one or more nodes. Finally, this section shows how to create a library of node definitions, independent of the structure of the table of contents, and reference those nodes.

> **Note:** You can use most any tool qualified to generate and manipulate XML to create the `Nodes.xml` file.

Defining a Documentation Node

A **documentation node** represents a single entry in the document hierarchy of the documentation set. A documentation node may contain a list of other documentation nodes, or subnodes. These lists of subnodes recursively define the structure of the documentation set.

A documentation node corresponds to a documentation file or a folder of files within the documentation set. Each documentation node is associated with a location that identifies the file to display when the user selects that node in the Documentation window "Documentation Set Development Workflow" (page 13).

A single documentation node is represented by the `Node` element. The key information provided by the `Node` element includes:

- The name of the documentation node.
- The type of the documentation node. If you do not explicitly specify the type of the node, Xcode assumes a default value.
- The location of the file or files that the node represents. The location tells Xcode what file to load when the user selects the documentation node and tells `docsetutil` which files to index.

The `Node` element also lists the documentation node's subnodes, if any. In addition, if you want to reference the node from elsewhere in the documentation set, you can assign a unique identifier to that node. The following sections show how to define a node element and specify its name, type, and location.

Naming a Documentation Node

Every documentation node must have a name, which is used to display the node in the path shown in the status field at the bottom of the Documentation window. To specify the name of a documentation node, use the `Name` element, as shown in Listing 3-2. The `Name` element is required.

Listing 3-2 Assigning a name to a node

```
<Node>
   <Name>A Documentation Node</Name>
</Node>
```

Specifying the Type of the Node

The type of the node tells Xcode and the indexing tool whether the node represents a single file or a folder of files.

There are three types of documentation nodes:

- **File nodes** represent a single file in the documentation set.
- **Folder nodes** represent a folder or folder hierarchy of documentation files.
- **Bundle nodes** represent a bundle-style folder hierarchy following the conventions of the `CFBundle` opaque type, described in *Bundle Programming Guide*. In this folder hierarchy, documentation content may be localized into multiple languages.

The type of a node determines how the node location is interpreted and how the documentation files at that location are indexed for full-text search by the `docsetutil` tool.

> **Note:** A node can represent a single HTML file, a single PDF file, or a folder of HTML files. It cannot represent a mix of the two.

You specify the type of a documentation node using the `type` attribute of the `Node` element. For example, Listing 3-3 shows how you might specify the type of a node representing a folder of files that comprise a single document in a documentation set.

Listing 3-3 Specifying the type of a node

```
<Node type="folder">
   <Name>My Document</Name>
</Node>
```

The `type` attribute is optional. If you do not specify the type of a node, it is assumed to be a file node.

Assigning an ID to a Node

As mentioned earlier, after you have defined a documentation node, you can reference that node from other locations in your `Nodes.xml` file or, if you choose to support API lookup, from a token definition in the tokens file, as described in "Supporting API Lookup in Documentation Sets" (page 36).

If you want to reference a given documentation node, however, you must assign that node a unique identifier. You assign an ID to a node using the `id` attribute of the `Node` element. The value assigned to this attribute can be any 32-bit integer value; however, it must be unique within the `Nodes.xml` file.

```
<Node id="5">
   <Name>A Documentation Node</Name>
</Node>
```

Specifying the Location of a Documentation Node

One of the most important pieces of information associated with a documentation node is the location of the documentation represented by that node. A documentation node's location tells Xcode and `docsetutil`:

- What page to load and display when the user selects the node from a list of search results
- Which file or files to index and include when performing full-text searches

The location of a documentation node consists of several parts, each of which is represented by its own element inside the `Node` element. The parts of a node's location are:

- `URL`: The **base URL** of the node. Use this element to specify an alternate location for nodes whose documentation files reside on the web, instead of in the installed documentation set bundle or the documentation set's *fallback web location* (see "Downloading and Indexing Web Content" (page 53)).

  You do not need to explicitly specify a URL. If you do not, the base URL for the node is assumed to be the `Documents` directory of the documentation set bundle.

- `Path`: The **path** to the page to display when the user selects the node or to the folder containing that page. This path is interpreted relative to the node's base URL.

  For bundle nodes, the path identifies the bundle folder.

- `File`: The **filename** of the file to display when the user selects the node. Xcode and `docsetutil` look for this file at the location specified by the node's base URL and path, if any.

  For bundle nodes, Xcode resolves this filename against the localizations available in the bundle associated with the node.

- `Anchor`: An optional **anchor** within the specified HTML file. When loading the node's landing page, Xcode scrolls to the location of this anchor.

When the user selects a node in the Xcode Documentation window, Xcode looks up the node and attempts to locate the file specified by that node. For file and folder nodes, Xcode attempts to load the page at `<url>/<path>/<filename>`. If no URL is explicitly specified, Xcode uses the default base URL—the Documents folder of the current documentation set. Either `<path>` or `<file>` or both may be empty, indicating that no `Path` or `File` element, respectively, exists for the node.

For bundle nodes, Xcode looks for the bundle at `<url>/<path>`. To find the file to load, it resolves the contents of the `File` element against the locations available within that bundle. For more information, see "Internationalizing Individual Documents" (page 54).

> **Note:** A node can specify a PDF file, instead of an HTML file, as its landing page. If it does so, however, only that PDF file is indexed, regardless of the node's type.

In addition to specifying the node landing page, the node location also specifies which file or files the `docsetutil` indexing tool indexes for full-text searches. The `docsetutil` tool interprets the node location differently, based on the type of the node:

- For file nodes, `docsetutil` indexes only the single file to which the node points—the node landing page.
- For folder nodes, `docsetutil` recursively searches for and indexes all HTML files within the folder pointed to by the node. The `docsetutil` tool assumes that this folder is at `<url>/<path>`. Again, if no base URL is explicitly specified, `docsetutil` uses the default value. If no path is specified, `<path>` is empty.

- For bundle nodes, `docsetutil` identifies all localized versions of the node's landing page inside the bundle at `<url>/<path>`. It indexes each directory of localized content inside this bundle.

  Note: In each case `docsetutil` skips any HTML files that contain the `NOINDEX` meta tag. For example:

  ```
  <meta name="ROBOTS" content="NOINDEX">
  ```

Important: The sets of documentation files represented by two separate documentation nodes must not overlap. For example, given a documentation node that represents a folder of HTML files, you must not define another documentation node to represent a file within that that folder. The `docsetutil` tool cannot handle overlapping nodes.

To illustrate some of the ways in which you might describe a node's location, imagine a fictional documentation set with the following contents:

Listing 3-4 Structure of a fictional documentation set

```
com.mycompany.MyProduct.docset
   Contents
      Resources
        Documents
          MyApplication
            index.html
            Tutorial
            ReleaseNotes.html
            UserGuide
              Contents
                 Resources
                   en.lproj
                   ja.lproj
            QuickReference.pdf          // Remote node
```

This documentation set contains four documents, each of which must be represented by a documentation node. These nodes are:

- `Tutorial` is a folder node. This directory contains the HTML files for a single document.
- `ReleaseNotes.html` is a file node.
- `UserGuide` is a bundle node.
- `QuickReference.pdf` is a node whose file resides on the web.

Listing 3-5 shows how you might describe the node representing the content in the `Tutorial` directory.

Listing 3-5 A folder node

```
<Node id="3" type="folder">
    <Name>Tutorial</Name>
    <Path>MyApplication/Tutorial</Path>
    <File>index.html</File>
</Node>
```

This node uses the `Path` element to specify the path to the folder containing the document's files. As no URL is specified, this path is interpreted relative to the documentation set's `Documents` directory. The `File` element indicates that the landing page for the documentation node—that is, the page that Xcode loads when the user selects this node—is a file named `index.html` inside the `Tutorial` directory. When `docsetutil` runs, this file and all other HTML files inside of the `Tutorial` directory are indexed for full-text search.

The user guide in the example documentation set is localized. You can represent this in the `Nodes.xml` file by creating a bundle node. The `Node` element describing the user guide might look something like that shown in Listing 3-6.

Listing 3-6 A bundle node

```
<Node id="2" type="bundle">
    <Name>User Guide</Name>
    <Path>MyApplication/UserGuide</Path>
    <File>index.html</File>
</Node>
```


> **Note:** This example shows a single localized documentation node within a documentation set. The entire documentation set itself can also be localized.

In this case, the `Path` element specifies only the path to the directory that contains the localized bundle structure; that is, it contains the path to the `UserGuide` directory. This directory's hierarchy must follow the `CFBundle` opaque type conventions described in *Bundle Programming Guide*.

The `File` element specifies the landing page to load, `index.html`. However, when the user accesses the documentation node that represents the user guide document, the actual file that Xcode displays depends on the user's language preferences. If, for example, the user's primary language is Japanese, Xcode looks for a file named `index.html` inside the `ja.lproj` directory.

The next node represents a single document, the `ReleaseNotes.html` file shown in Listing 3-4 (page 24). One way to describe this node is shown in Listing 3-7.

Listing 3-7 A file node

```
<Node id="4">
    <Name>Release Notes</Name>
    <Path>MyApplication</Path>
    <File>ReleaseNotes.html</File>
</Node>
```

Because the type of the node is not explicitly declared, it is assumed to be a file node. This means that only the file at `MyApplication/ReleaseNotes.html`, as specified by the `Path` and `File` elements, is indexed. Alternatively, you can simply use the `Path` element to specify the entire path to the file, as shown in Listing 3-8.

Listing 3-8 Another way to describe a file node

```
<Node id="4">
    <Name>Release Notes</Name>
    <Path>MyApplication/ReleaseNotes.html</Path>
</Node>
```

Documentation sets can include nodes whose files reside outside the installed documentation set bundle. This allows you to install a smaller subset of files onto the user's file system and still be able to access the other files.

As mentioned earlier, the `QuickReference.pdf` file in Listing 3-4 (page 24) is not actually present in the installed documentation set bundle on the user's file system but exists on the company's website. A node representing this file would look similar to that shown in Listing 3-9.

Listing 3-9 A node with web-based content

```
<Node id="6">
    <Name>My Application Quick Reference</Name>
    <URL>http://mycompany.com/Documentation/pdfs/QuickReference.pdf</URL>
</Node>
```

Here, the `URL` element specifies an alternate path for a PDF file that lives on the company's website.

Specifying Subnodes

Every node can have a list of **subnodes**, which are nodes that appear beneath the current node in the document hierarchy. For example, the fictional documentation set introduced in "Specifying the Location of a Documentation Node" (page 22) has four documents that describe the same application. Each of those documents is represented by a node. Assuming that they are grouped together, those nodes are subnodes of the node representing the application.

Use the `Subnodes` element to specify a node's list of subnodes. The `Subnodes` element contains one or more nodes, represented by a `Node` (or `NodeRef`) element. Listing 3-10 shows how the node representing the entry for My Application might appear; all the documents in this category appear as subnodes of that node.

Listing 3-10 Specifying subnodes

```
<Node>
   <Name>My Application</Name>
   <Path>MyApplication</Path>
   <File>index.html</File>
   <Subnodes>
      <Node id="2" type="bundle">
         <Name>User Guide</Name>
         <Path>MyApplication/UserGuide</Path>
         <File>index.html</File>
      </Node>
      <Node id="4">
         <Name>Release Notes</Name>
```



```
            <Path>MyApplication/ReleaseNotes.html</Path>
        </Node>
        <Node id="3" type="folder">
            <Name>Tutorial</Name>
            <Path>MyApplication/Tutorial</Path>
            <File>index.html</File>
        </Node>
        <Node id="6">
            <Name>My Application Quick Reference</Name>
            <URL>http://mycompany.com/Documentation/pdfs/QuickReference.pdf</URL>
        </Node>
    </Subnodes>
</Node>
```

Each of the documentation nodes in the `Subnodes` element can itself contain a list of subnodes.

Using the TOC Element

Previous sections show how to construct a documentation node using the `Node` element; this section describes how to turn a node definition into a table of contents. Although the Documentation window in Xcode 3.2 does not provide a browser view that displays a table of contents, it is a requirement that the `Nodes.xml` file contains a `TOC` element.

The primary purpose of the `Nodes.xml` file is to define the structure of the documentation set. The structure defines the path Xcode shows for the content that's currently displayed in the Documentation window.

The `TOC` element contains a single `Node` element—that is, a single documentation node—which represents the root node of the documentation set. The root node represents the topmost entry for the documentation set. It identifies the landing page of the entire documentation set.

The root node contains a list of its subnodes. As you saw in "Specifying Subnodes" (page 27), each of these subnodes can have its own list of subnodes. In this way, you can recursively define the documentation set's structure. Listing 3-11 shows how the TOC structure for the fictional documentation set introduced in "Specifying the Location of a Documentation Node" (page 22) might look.

Listing 3-11 Building the TOC structure

```
<TOC>
```



```
<Node>                                  <!-- Root node -->
   <Name>My Documentation Set</Name>
   <File>index.html</File>
   <Subnodes>
      <Node>
         <Name>My Application</Name>
         <Path>MyApplication</Path>
         <File>index.html</File>
         <Subnodes>
            <Node id="2" type="bundle">
               <Name>User Guide</Name>
               <Path>MyApplication/UserGuide</Path>
               <File>index.html</File>
            </Node>
            ...
            <!--Remaining node definitions here--!>
            ...
         </Subnodes>
      </Node>
   </Subnodes>
  </Node>
</TOC>
```

Referencing Documentation Nodes

After you define a documentation node in the `Nodes.xml` file, you can reference that node. You can use a node reference to:

- Make a single node or document appear multiple times in the documentation set's document hierarchy. Apple uses this feature to assign a document to multiple locations within the Reference Library.
- Easily import nodes into the TOC. As described in "Creating a Library of Node Definitions" (page 30), you can create a collection of node definitions, independent of the TOC. You can do the work of defining a documentation node once and simply link to it when constructing the document hierarchy later. This makes it easy to make changes to the structure.
- Associate documentation nodes with symbols in the `Tokens.xml` file.

A node reference is represented by a `NodeRef` element. This element has a single required attribute, `refid`, which is the unique identifier assigned to the referenced node. Thus, to reference a documentation node, you must have already assigned a value to the `id` attribute of the `Node` element representing that node's defining instance. Listing 3-12 (page 30) shows an example of a `NodeRef` definition.

Creating a Library of Node Definitions

In addition to the required `TOC` element, the `Nodes.xml` file can contain a collection, or library, of node definitions that exist independently of the document hierarchy. Having a separate library of documentation nodes makes it easy to make changes to your documentation set's TOC. Particularly for large documentation sets, it is often easier to define documentation nodes in the the library section of the `Nodes.xml` file, and then quickly construct or alter the structure in the `TOC` element by referencing the nodes you've already defined.

Use the `Library` element to create a **node definition library**. The `Library` element appears after the `TOC` element in the `Nodes.xml` file and contains one or more `Node` elements. For example, you can rewrite the `Nodes.xml` file for the documentation set introduced in "Specifying the Location of a Documentation Node" (page 22) so that the nodes representing the set's documents are defined in the library, and the TOC simply references those nodes. Listing 3-12 illustrates this configuration.

Listing 3-12 Creating a library of nodes

```
<TOC>
  <Node>
    <Name>My Documentation Set</Name>
    <File>index.html</File>
    <Subnodes>
      <Node>
        <Name>My Application</Name>
        <Path>MyApplication</Path>
        <File>index.html</File>
        <Subnodes>
          <NodeRef refid="2"/>
          <NodeRef refid="4"/> <!-- instead of defining node here, simply reference
library definition -->
          <NodeRef refid="3"/>
          <NodeRef refid="6"/>
        </Subnodes>
      </Node>
```



```
    </Subnodes>
  </Node>
</TOC>
<Library>
  <Node id="2" type="bundle">
    <Name>User Guide</Name>
    <Path>MyApplication/UserGuide</Path>
    <File>index.html</File>
  </Node>
  <Node id="4"> <!-- Definition of referenced node -->
    <Name>Release Notes</Name>
    <Path>MyApplication/ReleaseNotes.html</Path>
  </Node>
  <Node id="3" type="folder">
    <Name>Tutorial</Name>
    <Path>MyApplication/Tutorial</Path>
    <File>index.html</File>
  </Node>
  <Node id="6">
    <Name>My Application Quick Reference</Name>
    <URL>http://mycompany.com/Documentation/pdfs/QuickReference.pdf</URL>
  </Node>
</Library>
```

Minimal Nodes.xml File Example

A documentation set must contain a `Nodes.xml` file with at least one node, the root node. The root node identifies the landing page of the entire documentation set. Listing 3-13 shows an example of such a file.

Listing 3-13 A minimal `Nodes.xml` file

```
<?xml version="1.0" encoding="UTF-8"?>
<DocSetNodes version="1.0">     <!-- Root element -->
    <TOC>
        <Node type="folder">    <!-- Root node -->
            <Name>Root</Name>
```



```
            <Path>index.html</Path>
        </Node>
    </TOC>
</DocSetNodes>
```

A `Nodes.xml` file such as that shown in Listing 3-13 is sufficient, along with a minimal `Info.plist` file like that shown in Listing 3-1 (page 19) and a full set of documentation files, to build a documentation set that can be loaded by the Documentation window and that supports full-text search of its HTML-based documentation.

The root element of the `Nodes.xml` file is the `DocSetNodes` element. This element in turn contains a single `TOC` element, which describes the structure of the documentation set. The `TOC` element must have a single `Node` (or `NodeRef`) element as its child. This element defines the topmost entry of the documentation set.

Expanded Nodes.xml File Example

Expanding on the documentation set example shown in Listing 3-4 (page 24), imagine that that documentation set also includes documentation for an accompanying framework that exports an API for interfacing with your company's application. Listing 3-14 shows these documents and their accompanying HTML files.

Listing 3-14 An expanded documentation set

```
com.mycompany.MyProduct.docset
   Contents
      Resources
        Documents
          index.html
          MyApplication
            ...
          MyFramework
            index.html
            Reference
            Overview
```

In addition to the nodes for the `MyApplication` entry and its accompanying documents, described in "Specifying the Location of a Documentation Node" (page 22) and "Specifying Subnodes" (page 27), the `Nodes.xml` file must include the following nodes:

- **My Documentation Set:** The root node of the documentation set.
- **My Framework:** Umbrella group for documents targeted at readers using the corresponding framework to interface with the application. It is also a file node which corresponds to a single HTML file.
- **Overview:** File node that corresponds to a single HTML file.
- **Reference:** Folder node that corresponds to a folder of HTML files that comprise a single document.

Listing 3-15 shows how the entire `Nodes.xml` file for the expanded documentation set might look.

Listing 3-15 An expanded `Nodes.xml` file

```
<?xml version="1.0" encoding="UTF-8"?>
<DocSetNodes version="1.0">
    <TOC>
      <Node>
        <Name>My Documentation Set</Name>
        <File>index.html</File>
        <Subnodes>
          <Node>
            <Name>My Application</Name>
            <Path>MyApplication</Path>
            <File>index.html</File>
            <Subnodes>
              <Node id="2" type="bundle">
                <Name>User Guide</Name>
                <Path>MyApplication/UserGuide</Path>
                <File>index.html</File>
              </Node>
              <Node id="4">
                <Name>Release Notes</Name>
                <Path>MyApplication/ReleaseNotes.html</Path>
              </Node>
              <NodeRef refid="3"/>
              <NodeRef refid="6"/>
            </Subnodes>
          </Node>
          <Node>
```



```
            <Name>My Framework</Name>
            <Path>MyFramework</Path>
            <File>index.html</File>
            <Subnodes>
              <NodeRef refid="7"/>
              <NodeRef refid="5"/>
            </Subnodes>
          </Node>
        </Subnodes>
    </Node>
  </TOC>
  <Library>
    <Node id="3" type="folder">
      <Name>Tutorial</Name>
      <Path>MyApplication/Tutorial</Path>
      <File>index.html</File>
    </Node>
    <Node id="6">
      <Name>My Application Quick Reference</Name>
      <URL>http://mycompany.com/Documentation/pdfs/QuickReference.pdf</URL>
    </Node>
    <Node id="5" type="folder">
      <Name>Reference</Name>
      <Path>MyFramework/Reference</Path>
      <File>index.html</File>
    </Node>
    <Node id="7" type="file">
      <Name>Overview</Name>
      <Path>MyFramework/Overview</Path>
      <File>index.html</File>
    </Node>
  </Library>
</DocSetNodes>
```


The `TOC` element, contains the root node representing the documentation set. This root node has two subnodes, representing the entries for `My Application` and `My Framework`. Each of these two nodes in turn have their own lists of subnodes, which represent the documents in those categories.

In this case, the nodes representing the *Tutorial*, *Quick Reference*, and both of the framework-related documents are defined in the `Library` element and simply imported into the TOC using node references (represented by the `NodeRef` element). This makes it easy for the documentation provider to rearrange the structure of the documentation set without having to move or modify the canonical definition of these nodes.

Supporting API Lookup in Documentation Sets

One of the key features of the Xcode Documentation window is fast API search, the ability to quickly filter large lists of API symbols to find a particular symbol and its associated documentation. If your documentation set contains reference documentation for API symbols or other tokens, you can support fast API lookup for that documentation set by including one or more `Tokens.xml` files. The tokens file associates symbols or tokens with their primary reference documentation.

Documentation sets can have more than one tokens file; however, if a documentation set lacks a tokens file, Xcode supports only title and full-text searches in the Documentation window for that documentation set—fast API search is disabled.

A tokens file consists of a series of token definitions. Each token definition represents information about a single symbol. This chapter shows how to create a token definition to describe a symbol for lookup, how to provide information about that symbol for use with Quick Help, and how to organize large numbers of token definitions.

> **Note:** You can use any tool capable of generating and manipulating XML code to create the tokens file.

Defining a Symbol for Lookup

You associate a symbol with its reference documentation using the `Token` element. Because this element represents a single token definition, you can think of it as the building block of the tokens file. A **token** includes:

- A unique identifier representing the symbol described by the `Token` element
- The location of the symbol's primary reference documentation
- Summary information about the symbol (for display in Quick Help)
- Information about documents and symbols related to the symbol

This section shows how to define a token using the `Token` element.

Identifying Symbols

Every symbol that you describe in a tokens file must have a unique identifier, known as a **token identifier**. The information that uniquely identifies a symbol is the symbol's name, type (function, method, and so forth), scope, and language (C, C++, Objective-C, and so forth). There are two ways to specify a token identifier:

- By specifying each of the symbol's properties individually
- Using an identifier that conforms to the standard described in "Symbol Markers for HTML-Based Documentation" in *HeaderDoc User Guide*

> **Note:** These strings are informally known as **apple_ref** strings, because they begin with the prefix `//apple_ref.`

A token identifier is represented by the `TokenIdentifier` element. This element can contain either a combination of child elements—if you choose to identify the symbol by its individual properties—or a single string, if you choose to use the apple_ref convention to identify the symbol. These methods are described in greater detail in the following sections.

Defining Tokens Using Individual Properties

One way to specify a token identifier is to list the symbol's identifying properties—name, type, language, and scope—as individual subelements within the `TokenIdentifier` element.

> **Note:** The format described in this section is essentially a decomposed version of the apple_ref format described in "Defining Tokens Using apple_ref Identifiers" (page 38).

Use the following subelements of the `TokenIdentifier` element to specify each of the symbol's identifying properties separately:

- **Name**. The name of the symbol. This is required information.
- **APILanguage**. This is the programming language to which the symbol applies. Use this element only if your tokens file represents an API symbol.

  Apple defines a small number of values, described in *HeaderDoc User Guide*, for common languages. However, you can use any arbitrary string in this element. If you have symbols belonging to languages that are not covered by this specification, please contact Apple, so that Apple can define a value for that language that everyone can use.
- **Type**. The type—such as function, method, class, and so forth—of the symbol. Acceptable values are described in "Symbol Markers for HTML-Based Documentation." Arbitrary values are not allowed.

- **Scope**. The name of a namespace or container to which the symbol belongs. For example, the scope for most API symbols in object-oriented languages is the class or protocol in which the symbol is defined. API symbols that exist in a global namespace, such as data types or classes, do not have a scope.

For example, Listing 4-1 shows how you might construct the token identifier for the `NSArray` method `arrayWithContentsOfFile:` by listing each of its properties separately.

Listing 4-1 Specifying a token identifier with individual properties

```
<Token>
   <TokenIdentifier>
     <Name>arrayWithContentsOfFile:</Name>
     <APILanguage>occ</APILanguage>
     <Type>clm</Type>
     <Scope>NSArray</Scope>
   </TokenIdentifier>
</Token>
```

The contents of the token identifier for the `arrayWithContentsOfFile:` method are:

1. The `Name` element: Specifies the name of the method described by this token identifier, `arrayWithContentsOfFile:`.
2. The `APILanguage` element: Contains the string `occ`, which represents the Objective-C language, as defined in *HeaderDoc User Guide*.
3. The `Type` element: Contains the string, `clm`, which identifies the `arrayWithContentsOfFile:` symbol as a class method. This and other values for the `Type` element are defined in *HeaderDoc User Guide*.
4. The `Scope` element: Indicates that the scope of the `arrayWithContentsOfFile:` method is the `NSArray` class.

Defining Tokens Using apple_ref Identifiers

Another way to specify the token identifier for a symbol is to use a unique string that conforms to the specification in Symbol Markers for HTML-Based Documentation in *HeaderDoc User Guide*. These strings are known as **apple_ref** strings, because they begin with the prefix `//apple_ref`.

As with the technique described in "Defining Tokens Using Individual Properties" (page 37), an apple_ref string also uniquely identifies a symbol by listing the symbol's name, programmatic type, and programming language context. Where appropriate, you can also specify the symbol's containing scope, such as the name of the class in which a method is found.

Apple uses apple_ref strings in its own documentation sets to uniquely identify a symbol and mark the location of the symbol's primary documentation. For example, the primary documentation for the `OpenMovieStorage` C function is marked by embedding the associated apple_ref string as a named anchor in the HTML files, as in the following example:

```
<a name="//apple_ref/c/func/OpenMovieStorage">OpenMovieStorage</a>
```

You can use the apple_ref format to specify token identifiers in the `Tokens.xml` file. Using the `arrayWithContentsOfFile:` method as an example again, you would specify an apple_ref identifier as shown in Listing 4-2.

Listing 4-2 Specifying a token identifier using an apple_ref string

```
<Token>
<TokenIdentifier>//apple_ref/occ/clm/NSArray/arrayWithContentsOfFile:</TokenIdentifier>
</Token>
```

Associating Symbols with API Reference Documentation

The purpose of the tokens file is to associate symbols with a location within the documentation set. Therefore, after you have constructed a token identifier to uniquely identify a symbol, you need to specify the location of that symbol's documentation.

To associate a token with a location, you:

1. Identify the node that represents the symbol's documentation using the `NodeRef` element.

   This is useful when you've already defined a node that represents only the HTML files containing the symbol's documentation.

2. Specify the path to the HTML file containing the symbol's documentation, using the `Path` and `Anchor` elements, if the referenced node represents multiple symbols.

> **Note:** If you have a large documentation set in which multiple symbols are documented in a single HTML file, it can be redundant to list the location for each symbol separately. In this case, you can group the token definitions for all symbols that are documented in a single HTML file. In this way, you have to specify the location of the file only once. For more information, see "Grouping Tokens by File" (page 46).

Listing 4-3 shows how you might use the `NodeRef` element to specify the location of a symbol's documentation.

Listing 4-3 Referencing the documentation node for a token

```
<Token>
<TokenIdentifier>//apple_ref/occ/instm/NSArray/initWithArray:copyItems:</TokenIdentifier>
<NodeRef refid="22">
</Token>
```

The `refid` attribute of the `NodeRef` element identifies the documentation node that you are referencing. The value of this attribute should correspond to the value assigned to the target node's `id` attribute in the `Nodes.xml` file. This attribute is described further in "Defining a Documentation Node" (page 20).

When referring to nodes that represent multiple symbols, you need to identify the HTML file that contains the symbol's documentation using the `Path` and, if necessary, `Anchor` elements. The path specified is interpreted relative to the documentation set's `Documents` directory. Listing 4-4 shows how you might specify the path to the documentation for the `NSArray` instance method `initWithArray:copyItems:`. In this example, the `Anchor` element specifies the location of an anchor marking the beginning of the symbol's description.

Listing 4-4 Specifying the path to a symbol's documentation

```
<Token>
<TokenIdentifier>//apple_ref/occ/instm/NSArray/initWithArray:copyItems:</TokenIdentifier>
<NodeRef refid="15"/>
<Path>documentation/Cocoa/Reference/NSArray.html</Path>
<Anchor>initWithArray:copyItems:</Anchor>
</Token>
```


Providing Additional Information About a Symbol

One of the compelling features of Xcode's documentation integration is the ability to provide context-sensitive information. Quick Help is one example of this ability. The user can select a symbol in the code editor and see additional information about that symbol in Quick Help.

If your documentation set includes a `Tokens.xml` file to support API lookup, you can also supply additional information about the tokens described therein, to allow you to take full advantage of features such as Quick Help.

The `Token` element allows a number of subelements that let you provide additional information—such as a declaration statement, version information and so forth—for a symbol. These are described in further detail in the following sections.

Symbol Abstract and Declaration

Documentation users commonly want access to the declaration statement for a symbol and a brief summary of what that symbol does when they are actively coding. You can provide this information for a token using the `Declaration` and `Abstract` elements, respectively. For symbols that represent methods or functions, you can also provide a summary of the parameters and the return value.

Listing 4-5 shows how you might use these elements to provide additional information about the `NSArray count` method.

Listing 4-5 Specifying a summary and declaration

```
<Token>

<TokenIdentifier>//apple_ref/occ/instm/NSArray/indexOfObject:inRange:</TokenIdentifier>
   <Abstract>Returns the lowest index within a specified range whose corresponding
 array value is equal to a given object.</Abstract>
   <Declaration>- (NSUInteger)indexOfObject:(id)anObject
inRange:(NSRange)range</Declaration>
   <Parameters>
     <Parameter>
        <Name>anObject</Name><Abstract type="html">An object.</Abstract>
     </Parameter>
     <Parameter>
       <Name>range</Name><Abstract type="html">The range of indexes in the receiver
 within which to search for anObject.</Abstract>
     </Parameter>
```



```
    </Parameters>
    <ReturnValue><Abstract>The lowest index within range whose corresponding array
 value is equal to anObject</Abstract></ReturnValue
</Token>
```

Header File Information

For tokens that represent an API symbol, you can provide information about the header in which the symbol is declared, using the `DeclaredIn` element. You specify header information using this element either by calling the components of the header location out explicitly or by specifying it as a path.

For symbols that are part of a framework, you can use the `HeaderPath` and `FrameworkName` subelements to explicitly call out the header file and framework name. Doing so allows Xcode to present the path to the symbol's header file and the name of the framework that must be loaded to use the symbol as separate items.

Listing 4-6 shows how you would use this format to specify header file information for the `NSArray count` method.

Listing 4-6 Specifying header file information for a symbol

```
<Token>
<TokenIdentifier>//apple_ref/occ/instm/NSArray/count</TokenIdentifier>
<DeclaredIn>
<HeaderPath>/System/Library/Frameworks/Foundation.framework/Headers/NSArray.h</HeaderPath>
<FrameworkName>Foundation</FrameworkName>
</DeclaredIn>
</Token>
```

Alternatively, you can specify the path to the header file as a string directly within the `DeclaredIn` element. For example, if you don't want to specify the framework name separately or the header file isn't contained in a framework, you can restate the header file information for the `count` method as follows:

```
<DeclaredIn>/System/Library/Frameworks/Foundation.framework/Headers/NSArray.h</DeclaredIn>
```

Version Information

You can provide availability information for a symbol—that is, information about the versions of a software product in which the symbol appears—using the `Availability` element. With it, you can specify when the symbol was introduced, when it became deprecated, and when it was removed from later versions.

Providing Version Information

You can specify version numbers using the following subelements of the `Availability` element:

- `IntroducedInVersion`. The first version of the product in which the symbol appears. This information is required.
- `DeprecatedInVersion`. For symbols whose use is no longer recommended, the first version of the product in which the use of the symbol is deprecated.
- `RemovedAfterVersion`. For symbols that have been removed from later versions of the product, the last version in which the symbol still appears.

For symbols that are deprecated or removed from the distribution, you can also provide a brief statement about that symbol's status and other symbols that the user should use instead, using the `DeprecationSummary` element.

Listing 4-7 shows how you might provide version information for a token.

Listing 4-7 Specifying version information for a token

```
<Token>
  <TokenIdentifier>//apple_ref/occ/instm/NSArray/count</TokenIdentifier>
  <Availability distribution="OS X">
    <IntroducedInVersion>10.0</IntroducedInVersion>
    <DeprecatedInVersion>11.7</DeprecatedInVersion>
    <RemovedAfterVersion>11.9</RemovedAfterVersion>
    <DeprecationSummary>Replaced by newCount method.</DeprecationSummary>
  </Availability>
</Token>
```

Notice that version numbers are specified in the form `x.y.z`, where `x`, `y`, and `z` are integers: `x` specifies the major version number, `y` specifies the minor version number, and `z` specifies the maintenance version number. Only the major version number is required.

Naming the Product to Which the Version Applies

You must provide the name of the product to which the version information applies, using the `distribution` attribute. In the example shown in Listing 4-7 (page 43), the value of this attribute is "OS X." The `distribution` attribute is required.

Specifying Version Information for Multiple Architectures

If you have symbols whose version information is different for specific architectures, you can use the following attributes to contextualize the information in any of the version number elements—`IntroducedInVersion`, `DeprecatedInVersion`, and `RemovedAfterVersion`—for that architecture:

- `cputype`. This can be either `ppc` for the PowerPC architecture or `i386` for the Intel architecture.
- `bitsize`. This can be either `32` or `64`.

For example, imagine that the last version in which the `count` method appears differs for the PowerPC and Intel-based architectures. Listing 4-8 shows how you might specify the version information for this method.

Listing 4-8 Version information for multiple architectures

```
<Availability distribution="OS X">
     <IntroducedInVersion>10.0</IntroducedInVersion>
     <RemovedAfterVersion cputype="ppc">11.8</RemovedAfterVersion>
     <RemovedAfterVersion cputype="i386">11.9</RemovedAfterVersion>
</Availability>
```

Note that, in this case, you have multiple instances of the version element within a single `Availability` element. In the absence of architecture-specific version information, the availability information applies to all architectures. For example, Listing 4-8 specifies that the symbol was introduced in version 10.0 of the product for the `ppc32`, `ppc64`, `i386`, and `x86–64` architectures. After version 11.8 the symbol is not available for PowerPC architectures (both 32 bit and 64 bit). After version 11.9 the symbol is not available for any architecture.

Related Symbols, Documents, and Sample Code

Another type of information that users commonly want quick access to is a list of related resources, which they can consult to obtain further information about the symbol in question. You can provide this information for a symbol, using the following subelements of the `Token` element:

- `RelatedTokens`. Use this element to provide a list of other symbols that the reader may want to look at along with the current one.

  Each symbol in this list is identified using a `TokenIdentifier` element, as shown in Listing 4-9, and should correspond to a token definition in this or another tokens file in the documentation set.
- `RelatedDocuments`. Use this element to provide a list of documents, other than the symbol's primary reference document, that give further information about or discussion of the current symbol.

You specify an individual document in this list by referencing that document's node, using the `NodeRef` element.

- `RelatedSampleCode`. Use this element to provide a list of documents containing examples and sample code that use the current symbol.

  You specify an individual piece of sample code in this list by referencing that document's node, using the `NodeRef` element.

Listing 4-9 shows how you can use the elements described in this section to list related symbols and documents.

Listing 4-9 Specifying related symbols and documents

```
<Token>
  <TokenIdentifier>//apple_ref/occ/instm/NSArray/count</TokenIdentifier>
  <RelatedTokens>
    <TokenIdentifier>//apple_ref/occ/instm/NSArray/capacity</TokenIdentifier>
    <TokenIdentifier>
      <Name>objectAtIndex:</Name>
      <Scope>NSArray</Scope>
    </TokenIdentifier>
  </RelatedTokens>
  <RelatedDocuments>
    <NodeRef refid="17" />
  </RelatedDocuments>
  <RelatedSampleCode>
    <NodeRef refid="25" />
  </RelatedSampleCode>
</Token>
```

When using the `TokenIdentifier` element to reference an existing token, you need to specify only enough information to uniquely identify the token within your documentation set.

Managing Symbol Information in Large Documentation Sets

For large documentation sets, the number of token definitions in the tokens file can become unwieldy. There are several strategies you can use to reduce the amount of redundant information and make the large number of token definitions more manageable. You can:

- **Group token definitions according to the HTML file in which their associated documentation appears**. If your documentation set organizes the primary reference documentation for more than one symbol into a single HTML file, you can use this technique to avoid specifying the location of that file multiple times.
- **Define master lists of interrelated tokens**. If you have a group of symbols, each of which is related to all the other symbols in the group, you can simply specify a single list of interrelated tokens, rather than defining the list separately for each symbol.
- **Split symbol information into more than one tokens file**. A documentation set can have multiple tokens files; this allows you to split large sets of token definitions into more manageable chunks. You can separate symbol information based on the type of the information, programming language, or any other category that you choose.

The following sections describe these techniques in more detail.

Grouping Tokens by File

For documentation sets that document a large number of symbols, it often makes sense to combine the documentation for multiple symbols into a single HTML file. For example, you might group documentation for API symbols in an object-oriented language according to the class to which those symbols belong.

When you do this, however, you may find that you have to specify the location of an HTML file multiple times, once for each symbol documented in that file, if you use the approach described in "Associating Symbols with API Reference Documentation" (page 39).

To eliminate this redundant information, you can group symbols according to the file in which they appear. To do so, you wrap the token definitions—the `Token` elements—for each symbol described in a given HTML file within a single `File` element.

When you specify a token definition inside of a `File` element, you do not need to specify the location of the HTML file separately for each token. Instead, you specify the path to the HTML file once, using the `path` attribute of the `File` element, as shown in Listing 4-10.

Listing 4-10 Grouping tokens by file

```
<File path="documentation/Cocoa/Reference/NSArray.html">
  <Token>
    <TokenIdentifier>//apple_ref/occ/cl/NSArray</TokenIdentifier>
  </Token>
  <Token>
    <TokenIdentifier>
```



```
      <Name>arrayWithContentsOfFile:</Name>
      <Type>clm</Type>
      <Scope>NSArray</Scope>
      <APILanguage>occ</APILanguage>
    </TokenIdentifier>
    <Anchor>arrayWithContentsOfFile:</Anchor>
  </Token>
  <Token>
    <TokenIdentifier>//apple_ref/occ/instm/NSArray/count</TokenIdentifier>
    <Anchor>count</Anchor>
  </Token>
</File>
```

To make sure that the Documentation window scrolls directly to the location of a symbol's documentation when the user selects that symbol, you can specify an anchor location within the HTML file for that symbol's token definition, using the `Anchor` element.

Specifying Related Tokens

Although each symbol represented by a `Token` element can have its own list of related symbols, defining related symbols in this way only lets you specify a one-way relationship. The symbol described by the `Token` element is related to all of the symbols listed in the `RelatedTokens` subelement, but those symbols do not necessarily define the inverse relationship.

Often you will find that, for a given group of symbols dealing with a common area of functionality—hiding and showing a window, for example—the lists of related tokens for each symbol look very similar. If you list related tokens separately for each symbol in the group, you will end up repeating a lot of the same information more than once, bloating the tokens file and making the relationships hard to maintain.

You can instead take the simpler step of using the `RelatedTokens` element as a child of a `Tokens` element (the root of a tokens file) to create lists of interrelated symbols. Each symbol listed in this `RelatedTokens` element is related to all other symbols listed there. Listing 4-11 shows how you can use the `RelatedTokens` element as a subelement of the root element of the tokens file to specify a list of interrelated symbols.

Listing 4-11 Creating a list of related tokens

```
<Tokens>
<RelatedTokens title="Array Creation">
```



```
<TokenIdentifier>//apple_ref/occ/instm/NSArray/initWithArray:copyItems:</TokenIdentifier>
<TokenIdentifier>//apple_ref/occ/clm/NSArray/arrayWithContentsOfFile:</TokenIdentifier>
</RelatedTokens>
...
</Tokens>
```

Using this technique, rather than defining a list of related symbols for each individual token, you can eliminate a great deal of redundant information for large groups of interrelated symbols.

Using Multiple Tokens Files

A documentation set is not limited to a single tokens file. The number of symbols documented in a single documentation set is potentially very large. In addition, there is a wide variety of information (which may come from a number of different sources) that can be associated with each token. Allowing multiple tokens files makes it possible to split this information into more manageable chunks.

The `docsetutil` tool looks for and processes all available XML files that are located in the documentation set's `Resources` directory (or in the appropriate localized subfolder) and have names that start with the string "Tokens". For example, you could create tokens files that divide the definitions of tokens into groups based on their programming language. In this case, your documentation set might have `Tokens-C.xml` and `Tokens-Java.xml` files.

Another possible way to divide tokens is to gather various types of information about the tokens in separate files. For example, you could have a file of abstracts, `Tokens-abstracts.xml`, and a file of documentation locations, `Tokens-files.xml`.

When the information for a single token is split across multiple `Token` elements, within the same file or across different files, `docsetutil` attempts to merge all the information for a single token together. If every token is uniquely identified by its `TokenIdentifier` element, the information for each token can be merged successfully. If there are tokens with duplicate identifiers in the documentation set, however, information in the duplicate records may be assigned to the wrong token.

If there is a file named `Tokens.xml`, `docsetutil` always processes that file first, followed by the remaining XML files in a case-sensitive alphabetical order.

Example Tokens.xml File

The previous sections show how to construct individual token definitions, group token definitions according to file, and create lists of related tokens. Your tokens file can contain any number of these items—that is, of `Token`, `File`, and `RelatedTokens` elements—in any order. Listing 4-12 gives an example of a tokens file containing at least one of each of these items and shows how you might assemble them to create a complete tokens file. Note that the root element of the tokens file is the `Tokens` element.

Listing 4-12 An example tokens file

```
<?xml version="1.0" encoding="UTF-8"?>
<Tokens version="1.0"> <!-- Root element -->
<!-- The File element groups symbols that are documented in a common HTML file--!>
<File path="documentation/Cocoa/Reference/NSArray.html">
  <Token>
    <TokenIdentifier>//apple_ref/occ/cl/NSArray</TokenIdentifier>
  </Token>
  <Token>
    <TokenIdentifier>
      <Name>arrayWithContentsOfFile:</Name>
      <Type>clm</Type>
      <Scope>NSArray</Scope>
      <APILanguage>occ</APILanguage>
    </TokenIdentifier>
  </Token>
  <Token>
    <TokenIdentifier>//apple_ref/occ/instm/NSArray/count</TokenIdentifier>
    <Abstract>Returns the number of objects in the array.</Abstract>
    <Declaration>- (unsigned)count;</Declaration>
    <DeclaredIn>
<HeaderPath>/System/Library/Frameworks/Foundation.framework/Headers/NSArray.h</HeaderPath>
      <FrameworkName>Foundation</FrameworkName>
    </DeclaredIn>
    <Availability distribution="OS X">
      <IntroducedInVersion>10.0</IntroducedInVersion>
      <DeprecatedInVersion>11.7</DeprecatedInVersion>
      <RemovedAfterVersion>11.9</RemovedAfterVersion>
```



```
      <DeprecationSummary>Replaced by newCount method.</DeprecationSummary>
    </Availability>
    <RelatedTokens>
      <TokenIdentifier>//apple_ref/occ/instm/NSArray/capacity</TokenIdentifier>
      <TokenIdentifier>
        <Name>objectAtIndex:</Name>
        <Scope>NSArray</Scope>
      </TokenIdentifier>
    </RelatedTokens>
    <RelatedDocuments>
      <NodeRef refid="17" />
    </RelatedDocuments>
    <RelatedSampleCode>
      <NodeRef refid="25" />
    </RelatedSampleCode>
  </Token>
</File>
<!-- You can also list token definitions individually, specifying a location for
each -->!>
<Token>

<TokenIdentifier>//apple_ref/occ/instm/NSArray/initWithArray:copyItems:</TokenIdentifier>
  <Path>documentation/Cocoa/Reference/NSArray.html</Path>
  <Abstract>Initializes an instance from an array, optionally creating copies of
the objects.</Abstract>
  <Availability distribution="OS X">
    <IntroducedInVersion>10.2</IntroducedInVersion>
  </Availability>
</Token>
<Token>

<TokenIdentifier>//apple_ref/occ/instm/NSArray/initWithContentsOfFile:</TokenIdentifier>
  <NodeRef refid="22">
  <Availability distribution="OS X">
    <IntroducedInVersion>10.0</IntroducedInVersion/>
    <RemovedAfterVersion cputype="ppc">11.8</RemovedAfterVersion>
```



```
    <RemovedAfterVersion cputype="i386">11.9</RemovedAfterVersion>
  </Availability>
</Token>
<!-- If the same token identifier is used multiple times, the information in the
token definition is merged together --!>
<Token>

<TokenIdentifier>//apple_ref/occ/clm/NSArray/arrayWithContentsOfFile:</TokenIdentifier>
  <Availability distribution="OS X">
    <IntroducedInVersion>10.0</IntroducedInVersion>
  </Availability>
</Token>
<!-- Instead of defining related symbols for each individual token, use the
RelatedTokens element at the root level to define a set of interrelated symbols
--!>
<RelatedTokens title="Array Creation">

<TokenIdentifier>//apple_ref/occ/instm/NSArray/initWithArray:copyItems:</TokenIdentifier>

<TokenIdentifier>//apple_ref/occ/clm/NSArray/arrayWithContentsOfFile:</TokenIdentifier>
</RelatedTokens>
</Tokens>
```


Indexing Documentation Sets

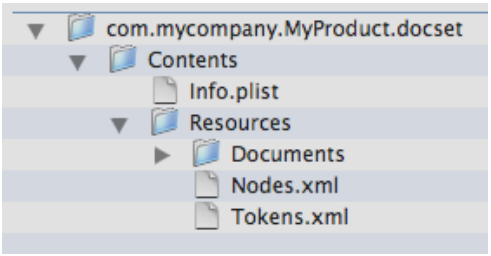
To make the contents of your documentation set accessible and searchable in Xcode, you must index the documentation set using the `docsetutil` command-line tool. You can find the `docsetutil` tool in the Xcode directory at `<Xcode>/usr/bin` by default. This chapter describes the `docsetutil` tool and how to use it to generate indexes.

Creating Indexes Using docsetutil

The `docsetutil` tool takes the information that you have provided about the structure and symbols in your documentation set and creates index files. These indexes are used by Xcode to search and access your documentation.

The `docsetutil` tool expects to find a `Nodes.xml` file and, if you support API lookup for your documentation set, one or more `Tokens.xml` files. Before you can index your documentation set, you must place these files inside the documentation set bundle. Figure 5-1 shows the structure of a typical documentation set bundle before running the indexing tool.

Figure 5-1 A documentation set bundle before indexing



To create full-text and API indexes for a self-contained documentation set, run the `docsetutil` tool from the command line, using a command such as the following:

```
<Xcode>/usr/bin/docsetutil index com.mycompany.MyProduct.docset
```

The `docsetutil` tool loads the `Nodes.xml` and `Tokens.xml` XML metadata files and generates a full-text index (`docSet.skidx`) and an API and document data store (`docSet.dsidx`). It places the generated index files in the Resources folder of the documentation set bundle.

The `docsetutil` tool provides several options, which let you specify a localization for indexing, an alternate location for remote content, and so forth. For the full list of indexing options, see "docsetutil Reference" (page 72).

Downloading and Indexing Web Content

If you have documentation set content located on the web, you must also index that content. The `docsetutil` tool provides options to help you index web content.

If you have specified a fallback web location for the entire documentation set bundle, using the `DocSetFallbackURL` property, you must index a local copy of the web content. Use the `–fallback` option to specify the location of the local copy of the web content. For example, the following command creates full-text and API indexes for a documentation set whose content resides in the documentation set bundle as well as on the web. The `CopyOfPublicWebsite` directory must correspond to the location indicated by the `DocSetFallbackURL` property in the documentation set's `Info.plist` file.

```
<Xcode>/usr/bin/docsetutil index com.mycompany.MyProduct.docset –fallback
/Documents/CopyOfPublicWebsite
```

If `docsetutil` doesn't find the documentation for a node in the documentation set bundle, it looks in the location specified using the `–fallback` option. For more on specifying an alternate web location using the `DocSetFallbackURL` property, see "DocSetFallbackURL" (page 76).

If your documentation set contains individual nodes that specify an Internet address as the location of their landing page, use the `–download` option to have `docsetutil` download and index those landing pages. For example, the following command generates indexes for a documentation set, downloading any web-based nodes:

```
<Xcode>/usr/bin/docsetutil index –download com.mycompany.MyProduct.docset
```

For any node in the nodes file that specifies a web address using the `URL` element, `docsetutil` downloads the node's landing page and includes it in the index. The `docsetutil` tool downloads only the specified landing page. If the node is a folder or bundle node, `docsetutil` does not download the entire folder of documentation represented by the node.

Internationalizing Documentation Sets

To internationalize a documentation set, you can localize all or part of the content in a documentation set bundle into more than one language. There are two ways to provide localized content in a documentation set:

1. For large documentation sets containing only a couple of internationalized documents, you can localize individual documentation nodes.
2. For documentation sets that have a larger percentage of their content internationalized, you can localize the whole documentation set bundle, including both the HTML files as well as the index files.

This chapter describes how to localize individual documents or the entire documentation set bundle. It also shows how to run `docsetutil` to create indexes for a particular locale. For information about product internationalization in OS X, see *Internationalization Programming Topics*.

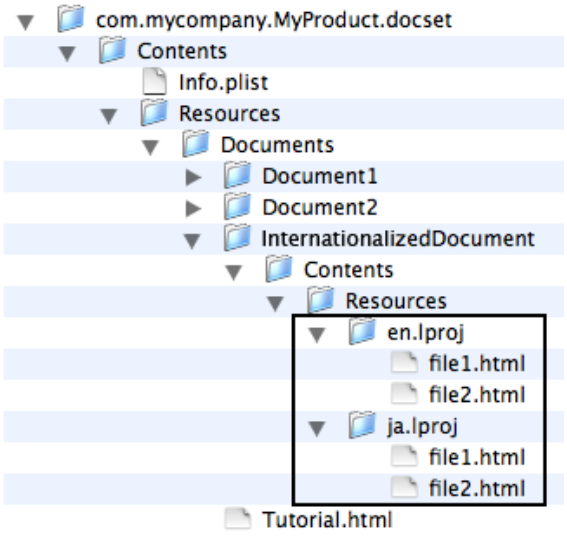
Internationalizing Individual Documents

If you need to internationalize only one or two documents in your documentation set, it may make sense to localize the individual documentation nodes that represent these documents. You can do so by:

1. Creating a file bundle to contain the HTML files for the localized document.
2. Defining a bundle node to represent the document in the `Nodes.xml` file.

For example, a documentation set that contains several documents, of which only one is internationalized, may be organized as shown in Figure 6-1.

Figure 6-1 Internationalized node in a documentation set



The internationalized document is available in Japanese and English; the `InternationalizedDocument` directory uses the standard bundle conventions to organize two subdirectories containing the localized HTML files. The `en.lproj` directory contains the English language versions of the HTML-based documentation files and the `ja.lproj` directory contains the Japanese language version of these same files. See *Internationalization Programming Topics* for more information on the structure of internationalized bundles.

Once you have the bundle hierarchy established, you need to define a `Node` element to represent the internationalized bundle. Returning to the example documentation set in Figure 6-1, you can represent a node with this structure using a `Node` definition such as the one shown in Listing 6-1.

Listing 6-1 An internationalized `Node` element

```
<Node type="bundle">
    <Name>My Internationalized Document</Name>
    <Path>InternationalizedDocument</Path>
    <File>file1.html</File>
</Node>
```


The `Path` element specifies the path—relative to the `Documents` directory in the documentation set—to the bundle containing the localized content. The `File` element specifies the name of the HTML file to load for that node; in this case, `file1.html`. When the user accesses the content represented by this node, Xcode determines which `.lproj` directory to load files from based on the user's preferred language, using the same process outlined in *Internationalization Programming Topics*.

When you index a documentation set with internationalized nodes, `docsetutil` indexes all available localizations for those nodes and includes that information in the `docSet.skidx` and `docSet.dsidx` files that it generates. If Xcode finds a search result in multiple localizations—that is, in localized versions of the same file—the Documentation window shows only the result corresponding to the user's preferred language.

Internationalizing a Documentation Set Bundle

Because it is a standard OS X bundle, you can use standard internationalization techniques to localize a documentation set bundle into one or more languages. Any of the documentation set resources can be localized, including the following:

- `Tokens.xml` files
- `Nodes.xml` file
- `Documents` directory
- Generated index files (`docSet.skidx` and `docSet.dsidx`)
- `Info.plist` property values

You can localize all of these resources, or just some of them. To localize a documentation set, you must:

1. Construct a standard localized bundle hierarchy.

   The `Resources` directory in the documentation set can contain one or more locale-specific folders, named using the `<locale>.lproj` convention. These subfolders can contain localized versions of the `Documents` directory. They can also optionally contain localized versions of the metadata files, the index files, or `InfoPlist.strings` files, which localize one or more values in the `Info.plist` file.

2. Optionally localize the nodes or tokens files.

   Although you do not need to localize the `Nodes.xml` and `Tokens.xml` files, doing so allows you to support searching in additional languages. That is, when users perform searches in the Documentation window, they will see localized titles of documents in the search results. Localizing the tokens files lets you provide token-related information—for features such as Quick Help—in multiple languages.

3. Localize the `Info.plist` file.

The `Info.plist` file also contains strings that appear in the user interface. To localize this content, place an `InfoPlist.strings` file with the localized string values in the appropriate locale-specific subdirectory, as described in "Strings Files."

4. Run `docsetutil` once for each locale you wish to support. "Creating Indexes for Specific Locales" (page 57) describes how to run docsetutil for a specific locale.

Creating Indexes for Specific Locales

By default, `docsetutil` produces global indexes and stores them at the top level of the `Resources` directory. To produce indexes for particular locales, you must run `docsetutil` with a specific locale, using the `–localization` option. For example, to produce a set of Japanese language indexes for the documentation set shown in Listing 6-2 (page 57), you would invoke the `docsetutil` tool like this:

```
<Xcode>/usr/bin/docsetutil index com.mycompany.MyProduct.docset –localization ja
```

The `docsetutil` tool generates indexes using the XML files and `Documents` directory for the specified locale and stores the indexes in the appropriate directory—`ja.lproj`, in this case.

You must run the `docsetutil` tool separately for each locale that you wish to support.

For example, a small documentation set with a handful of documents that are localized into both English and Japanese might have a bundle structure—before indexing—similar to that shown in Listing 6-2.

Listing 6-2 An internationalized documentation set before indexing

```
com.mycompany.MyProduct.docset
   Contents
      Resources
         en.lproj
            Nodes.xml
            Tokens.xml
            InfoPlist.strings
            Documents
               Document1
               Document2
               Document3.html
         ja.lproj
            Nodes.xml
```



```
            Tokens.xml
            InfoPlist.strings
            Documents
               Document1
               Document2
               Document3.html
```

After indexing with the `-localization ja` command-line argument, the resulting bundle would look like that shown in Listing 6-3.

Listing 6-3 An internationalized documentation set after indexing

```
com.mycompany.MyProducet.docset/
   Contents
      Resources
         en.lproj
            Nodes.xml
            Tokens.xml
            InfoPlist.strings
            Documents
               Document1
               Document2
               Document3.html
         ja.lproj
            docSet.skidx
            docSet.dsidx
            Nodes.xml
            Tokens.xml
            InfoPlist.strings
            Documents
               Document1
               Document2
               Document3.html
```


Acquiring Documentation Sets Through Web Feeds

Xcode supports automatic detection and download of documentation set updates. You can take advantage of this feature by providing an RSS or Atom feed (also known as **web feeds**) to publish content updates for your own documentation sets. Users can subscribe to your web feed from Xcode Documentation preferences. They can check for and download updates to installed documentation sets, or download new documentation sets.

Xcode uses the Publication Subscription framework to subscribe to documentation set feeds. It can recognize and support the same web feed formats supported by that framework. Currently, this includes RSS 1.0, RSS 2.0, and Atom. This chapter uses Atom in all of its explanations and examples. To learn more about the elements in an Atom feed, visit http://atomenabled.org/developers/syndication.

This chapter describes the key elements used to describe an Atom feed and a documentation set entry. It also includes an example Atom feed.

Specifying Feed Information

A feed consists of metadata, which provides general information about the feed and its contents, and a series of entries. Each entry represents a **documentation set update**.

The feed is represented by the standard Atom element, `Feed`. To describe a feed you must specify the following information:

- The publisher of the feed (`publisherName` and `publisherID`)
- The unique identifier of the feed (`id`)
- The title of the feed (`title`)
- The last time the feed was modified (`updated`)

As of Xcode 3.2, you have the option to use these elements on the entire feed:

`minimumXcodeVersion`: Specifies the earliest Xcode release that is to download the feed.

`maximumXcodeVersion`: Specifies the latest Xcode release that is to download the feed.

> **Note:** These are all standard Atom elements. You can learn more about these required elements at http://atomenabled.org/developers/syndication.

Listing 7-1 shows how you might use these elements to describe a feed. The Atom specification also defines other elements that let you specify additional information such as the author, copyright information and so forth.

Listing 7-1 Describing a feed

```
<feed xmlns="http://www.w3.org/2005/Atom"
  xmlns:docset="http://developer.apple.com/rss/docset_extensions"
  xml:lang="en">
  <docset:publisherName>Apple</docset:publisherName>
  <docset:publisherID>com.apple.adc.documentation</docset:publisherID>
  <id>http://developer.apple.com/</id>
  <title type="text">Apple Developer Documentation</title>
  <updated>2006-03-24T12:00:00Z</updated>
  ...
</feed>
```

Specifying a Documentation Set Entry

As you learned in the previous section, each entry in the web feed corresponds to a single documentation set download. You can publish updates to multiple documentation sets with a single web feed.

You specify an entry using the standard Atom element, Entry. For each entry, or documentation set, you must specify:

- The unique identifier for the entry (`id`). This can be any unique uniform resource name (URN).
- The name of the documentation set (`title`). This must match the value of the documentation set's `CFBundleName` property.
- The last time the entry was modified (`updated`).
- A link to the documentation set download (`link`).
- The documentation set identifier (`docset:identifier`). This must match the value of the documentation set's `CFBundleIdentifier` property.

- The documentation set version (`docset:version`). This must match the value of the documentation set's `CFBundleVersion` property.

The first four elements are standard Atom elements, described further at http://atomenabled.org/developers/syndication. You can also include other elements defined by the Atom specification.

The `identifier` and `version` elements are custom elements defined by the `docset` namespace. The `docset` namespace also defines several optional elements:

- `minimumXcodeVersion`: Specifies the earliest Xcode release that is to download the documentation set.
- `maximumXcodeVersion`: Specifies the latest Xcode release that is to download the documentation set.
- `signer`: Specifies the distinguished name of the signer of the documentation set certificate.
- `issuer`: Specifies the distinguished name of the issuer of the documentation set certificate.

Listing 7-2 shows how you can use the elements described in this section to create a web feed entry for a documentation set.

Listing 7-2 Describing a documentation set entry

```
<entry>
  <id>tag:apple.com/CoreRef/2004</id>
  <title type="text">Core Reference Library</title>
  <summary type="text">Includes reference for Carbon, Cocoa, and other
frameworks.</summary>
  <updated>2006-03-24T12:00:00Z</updated>
  <link rel="enclosure" type="application/octet-stream"
href="http://developer.apple.com/docsets/CoreRef2004.xar"/>

<docset:identifier>com.apple.ADC_Reference_Library.CoreReference</docset:identifier>
  <docset:version>200.4</docset:version>
  <docset:minimumXcodeVersion>3.0</docset:minimumXcodeVersion>
  <docset:signer>CN=ADC DocSet Update,O=Apple Inc.,OU=Apple Developer
Connection,C=US</docset:signer>
  <docset:issuer>CN=ADC DocSet Update,O=Apple Inc.,OU=Apple Developer
Connection,C=US</docset:issuer>
</entry>
```


The Documentation Set Acquisition Process

New or updated documentation sets are downloaded by Xcode as XAR archives. The documentation set contained in the archive must pass the Xcode security checks described later in this section before Xcode can place the documentation set in the user's filesystem.

To determine whether an installed documentation set needs to be updated when examining its feed, Xcode performs the following tasks:

1. For each installed documentation set, finds the matching (`docset:identifier`) entry with the highest version number (`docset:version`) that has the same `signer` and `issuer` values and is compatible with the running version of Xcode (`minimumXcodeVersion` and `maximumXcodeVersion`.
2. Compares the version number of the entry with the version number of the installed documentation set to determine if the version number of the entry is higher than the version number of the documentation set. If it is, then the installed documentation set needs to be updated.

When a documentation set that hasn't been installed on the user's computer becomes available, the user can download the documentation set by pressing the Get button for that documentation set in Documentation preferences. When an update to an installed documentation set is available, Xcode will update it automatically unless the user opts to do so manually.

To ensure that the documentation set acquisition process is not used as a mechanism to introduce malicious software into a user's computer, Xcode performs several security checks before unarchiving documentation set archives and placing them in the user's filesystem. Among these checks are:

- **Documentation set archive signature verification.** Xcode verifies that the signature used in incoming documentation sets matches the signature of the corresponding web feed or the documentation set being updated.
- **Quarantine of incoming documentation sets.** Before installing an incoming documentation set, Xcode places it in the user's private temporary directory in `/var/folders` and marks it as quarantined.
- **Nonexecutable code verification.** Xcode ensures that the incoming documentation set does not contain executable code.
- **Administrator authentication.** When installing new documentation sets in privileged locations for the first time or when updating installed documentation sets in privileged locations, Xcode may request administrator authentication before carrying out the install or update process. (When the user has already approved the acquisition of a signed documentation set, further signed updates of the same documentation set do not require administrator authentication.)

The following sections describe the tasks Xcode performs when getting new documentation sets or updating installed documentation sets.

Getting Documentation Sets

When the incoming documentation set has not been installed in any of the documentation set locations (that is, its identifier doesn't match the identifier of any installed documentation set), the user can click the Get button next to the documentation set's name in the documentation set list to install the new documentation set. (The documentation set list is in Documentation preferences.) After the user clicks the button, Xcode performs the following tasks:

1. **Administrator authentication.** If the documentation set will be installed into a privileged location, Xcode will request administrator authentication. If the user is unable to authenticate as one of the computer's administrators, Xcode terminates the install process.

   When the documentation set is to be placed in a user-writable location, no authentication is performed.

2. **Signature verification.** If the web feed provides a signature, Xcode determines whether the signature is valid and whether it matches the incoming documentation set's signature. If either of these tests is not passed, Xcode terminates the install process.

3. **Installation.** If a documentation set from the same publisher of the incoming documentation set is installed in any documentation set location, Xcode installs the incoming documentation set in one of those locations. Otherwise, Xcode places the documentation set in `/Library/Developer/Shared/Documentation/DocSets`.

4. **End.** Xcode ends the process.

Updating Documentation Sets

An update occurs when the identifier of the incoming documentation set matches the identifier of an **installed documentation set** (a documentation set in one of the documentation set locations). Xcode replaces the installed documentation set with the incoming one. Xcode performs the following tasks during an update:

1. If the installed documentation set bundle is owned by the `_devdocs` system user and the incoming documentation set is signed:

   a. **Signature verification.** Xcode determines whether the signature in the incoming documentation set is valid and whether it matches the installed documentation set's signature. If either of these tests is not passed, Xcode terminates the process.

   b. **Update.** Xcode replaces the installed documentation set with the incoming one.

   c. **End.** Xcode ends the process.

2. If the installed documentation set bundle is owned by the `_devdocs` system user and the incoming documentation set is not signed:

   a. **Administrator authentication.** If the user is unable to authenticate as one of the computer's administrators, Xcode terminates the process.

   b. **Update.** Xcode replaces the installed documentation set with the incoming one.

c. **End.** Xcode ends the process.

3. If the installed documentation set bundle is writable by the user:
   a. **Update.** Xcode replaces the existing documentation set with the incoming one.
   b. **End.** Xcode ends the process.
4. Otherwise, Xcode terminates the process.

An Example Atom Feed

Listing 7-3 shows the entire specification for a sample Atom feed. This feed includes two documentation sets, Core Reference Library and PDF 2.0 reference material.

Listing 7-3 Example Atom feed

```
<feed xmlns="http://www.w3.org/2005/Atom"
  xmlns:docset="http://developer.apple.com/rss/docset_extensions"
  xml:lang="en">
  <id>http://developer.apple.com/</id>
  <title type="text">Apple Developer Documentation</title>
  <updated>2006-03-24T12:00:00Z</updated>
  <author>
    <name>Apple Developer Publications</name>
    <uri>http://developer.apple.com/</uri>
  </author>
  <rights>Copyright (c) 2007, Apple Inc.</rights>
  <link rel="self" href="ADCDocSets.atom" />
  <entry>
    <id>http://developer.apple.com/docsets/corereflib1.0</id>
    <title type="text">Core Reference Library</title>
   <summary type="text">Includes reference for Cocoa and other frameworks.</summary>
    <updated>2006-03-24T12:00:00Z</updated>
    <!-- Link to actual download. This is required. -->
    <link rel="enclosure" type="application/octet-stream"
href="corereflib1.0/corereflib1.0.xar"/>

<docset:identifier>com.apple.ADC_Reference_Library.CoreReference</docset:identifier>
```



```
    <docset:version>1.0</docset:version>
    <docset:minimumXcodeVersion>3.0</docset:minimumXcodeVersion>
  </entry>
  <entry>

<id>tag:developer.apple.com,2008-04-23:com.apple.ADC_Reference_Library.JavaReference/17</id>
    <title type="text">Java Library</title>
    <summary type="text">Java Library (v17)</summary>
    <updated>2009-01-05T08:51:17-07:00</updated>
    <link rel="enclosure" type="application/octet-stream"
href="java/javaref17.xar"></link>

<docset:identifier>com.apple.ADC_Reference_Library.JavaReference</docset:identifier>
    <docset:version>17</docset:version>
    <docset:minimumXcodeVersion>3.1</docset:minimumXcodeVersion>
  </entry>
</feed>
```


Testing and Packaging Documentation Sets

The final steps in building a documentation set are to ensure that Xcode can correctly access, display, and search the contents of the documentation set, and to package the documentation set bundle for distribution.

This chapter describes how to use the `docsetutil` tool to test the functionality and contents of your documentation set's indexes, how to test whether Xcode can access and display your documentation, and how to package your documentation set bundle as an XAR archive.

Querying and Testing Indexes

Because Xcode relies on the indexes to find and access the content of a documentation set, you should test those indexes thoroughly. The same `docsetutil` tool that you used to index your documentation set content also lets you test and validate the generated index files.

The following sections show how you can use the `docsetutil` tool to compare the contents of the indexes to the contents of the local documentation set bundle, test the search results returned by the indexes, and print out index contents.

Comparing the Contents of the Documentation Set Bundle and its Indexes

One of the simplest tests you can perform on your documentation set is to validate that the generated indexes match the content of the documentation set bundle. You can do this using the `docsetutil validate` command, as in this example:

```
> <Xcode>/usr/bin/docsetutil validate com.mycompany.MyProduct.docset
```

The `docsetutil` tool warns of any discrepancies between the files listed in the indexes and the files in the local copy of the documentation set bundle. The tool detects any nonexistent files referenced in the nodes and tokens files or files that disappeared after the indexes were built.

Verifying Indexes

Another useful feature of the `docsetutil` tool is the ability to print the contents of a documentation set's indexes. By examining the contents of your documentation set's indexes, you can verify that they contain the expected number of nodes and tokens, and that the document hierarchy that they describe is correct.

The `docsetutil` tool can print:

- The total number of nodes in the documentation set
- The node hierarchy; that is, all or part of the TOC of the documentation set
- The total number of documents in the full-text index
- The path to each node in the full-text index
- The total number of tokens in the API index
- The identifiers of each of the tokens in the API index

Use the `docsetutil dump` command to print the contents of your documentation set's indexes. For example, the following command prints the contents of the test documentation set's indexes:

```
> <Xcode>/usr/bin/docsetutil dump com.mycompany.MyProduct.docset
```

The output generated by this command looks similar to that shown in Listing 8-1.

Listing 8-1 Dumping the contents of a documentation set's indexes

```
Documentation set contains 9 nodes in hierarchy
My Doc Set
  IB
    IB Reference
  Performance
    Performance Tools Tutorial
  Xcode
    User Guide
    Tutorial
    Build Settings

Full text index contains 614 documents
docset://__DEFAULT__
  IBInspector_class
```



```
  XcodeBuildSettingRef
  XcodeQuickTour
  XcodeUserGuide
  InstrumentsQuickTour
count = 6

API index contains 9 tokens
Objective-C/instm/IBInspector/document
Objective-C/instm/IBInspector/inspectedObjects
Objective-C/instm/IBInspector/inspectedObjectsController
Objective-C/instm/IBInspector/label
Objective-C/instm/IBInspector/refresh
Objective-C/clm/IBInspector/sharedInstance
Objective-C/clm/IBInspector/supportsMultipleObjectInspection
Objective-C/instm/IBInspector/view
Objective-C/instm/IBInspector/viewNibName
```

Notice that `docsetutil` prints the node hierarchy, followed by the number of documents in the full-text index. Following that, `docsetutil` prints the path to each node in the full-text index. These paths are grouped according to the root where the nodes are found. The `_DEFAULT_` root represents the `Documents` directory of the documentation set bundle. If you identified additional roots by specifying a fallback root using the `-fallback` option, they are listed under the `_FALLBACK_` root. If you specified files to download and include using the `-download` option, these also appear in the output of the `docsetutil dump` command, along with the paths to the nodes at that location.

By default, `docsetutil` prints only the paths to the nodes that are explicitly specified in the `Nodes.xml` file. However, folder and bundle nodes typically represent multiple HTML files. You can also have `docsetutil` print the paths to each of the individually indexed files in a node using the `-text-depth` option.

The contents of the indexes for very large documentation sets can be difficult to parse. You can test smaller portions of the indexes by restricting the output of the `docsetutil dump` command. You do this in the following ways:

- Use the `-skip-text` and `-skip-api` options to skip either the full-text or API indexes when printing out the index contents.
- Use the `-toc-depth` option to limit the depth of the document hierarchy printed by `docsetutil`.

- Use the `–text–depth` option to limit the amount of information printed about the documentation set's nodes and their contents.
- Use the `–node` option to dump only the contents of the specified node and its subnodes.

For example, the following command prints only the document hierarchy of the documentation set:

```
> <Xcode>/usr/bin/docsetutil dump com.mycompany.MyProduct.docset –skip–text –skip–api
```

For more on these options, see "docsetutil Reference" (page 72).

Querying Indexes

You can also use the `docsetutil` tool to test the search results returned by your documentation set's indexes. The command `docsetutil search` searches for a particular string in the indexes of the specified documentation set. This command has one required option, `–query`, which specifies the search string. For example, the following command searches for the word "trace" in a documentation set:

```
> <Xcode>/usr/bin/docsetutil search –query trace com.mycompany.MyProduct.docset
```

The output of the `docsetutil` tool looks similar to that shown in Listing 8-2. It notes the maximum rank of the matching results and lists the normalized score and path of each matching HTML page.

Listing 8-2 Sample output from `docsetutil search`

```
Max score: 90.330826
 Score   Path
 0.129 InstrumentsQuickTour/Built–InInstruments/chapter_6_section_6.html
 0.166
XcodeUserGuide/Contents/Resources/en.lproj/06_02_db_set_up_debug/chapter_40_section_10.html
...
 1.000 InstrumentsQuickTour/WorkingWithInstruments/chapter_3_section_4.html
```

By default, `docsetutil search` queries both the full-text and API indexes. However, it is often easier to query and test a single index at a time. Use the `–skip–text` and `–skip–api` flags to disable the full-text or API indexes, respectively.

For example, the following command queries a documentation set's API index for information on the `CreateWindow` symbol:


```
> <Xcode>/usr/bin/docsetutil search com.mycompany.MyProduct.docset –query
CreateWindow –skip–text
```

You can also limit the scope of the search to a particular node and its subnodes, using the `–node` option, as described in "docsetutil Reference" (page 72). The following command searches for the string "trace" in only those documents that are part of the Xcode node and its subnodes:

```
> <Xcode>/usr/bin/docsetutil search –query trace  –node Xcode
com.mycompany.MyProduct.docset
```

Testing Display and Navigation

In addition to testing the content in the indexes, you should make sure that your documentation set displays properly in the Xcode Documentation window. To do so, simply drop the documentation set bundle into its default installation location.

After placing the documentation set bundle in its default installation location, launch Xcode, and open the Documentation preferences. (You may have to quit and restart Xcode if it was already running when you installed the documentation set bundle.) Check to make sure that that the documentation set appears in the documentation sets list. You should also check the Home pop-up menu that's in the Documentation window toolbar (in Xcode 3.2 and later) and see whether your documentation set appears in it. When you choose your documentation set from the menu, the Documentation window should display the root node page.

Finally, in the Documentation window, perform a variety of searches to ensure that all the search types supported by your documentation set work as expected and return reasonable results.

Packaging Documentation Sets

The last thing you need to do before distributing your documentation set is to package it. Using the `docsetutil package` command, you can package your documentation set as an XAR archive.

For example, the following command packages a documentation set and places it on the Desktop:

```
> <Xcode>/usr/bin/docsetutil package –output ~/Desktop/MyProductDocSet.xar
com.mycompany.MyProduct.docset
```


If you do not specify an output path, `docsetutil` places the archive in the same folder as your documentation set bundle. You can also sign the package using the `-signid` option and generate (or update) an Atom feed for the documentation set using the `-atom` option. For more information on these options, see "docsetutil Reference" (page 72).

> **Note:** The `Nodes.xml` and `Tokens.xml` files do not need to be included in the documentation set bundle for distribution to users; they are required only for indexing and can be removed from the documentation set before distribution.

docsetutil Reference

Apple provides the command-line tool `docsetutil` to help you create, test, and query full-text and API indexes for your documentation set. To use the tool, open Terminal and enter the following:

```
docsetutil <verb> [options] <doc_set_path>
```

The `<doc_set_path>` argument specifies the file system path to the documentation set bundle and is required for all operations using `docsetutil`.

You can use `docsetutil` to:

- Generate full-text and API indexes
- Search for a particular string in the indexes
- Validate the indexes
- Print index content
- Generate a downloadable update

The `docsetutil` tool supports the following common options, which you can use with any of the available verbs:

- `–localization <locale>`. The locale to use when operating on the documentation set. Use this option to index or search localized documentation set content. The `<locale>` argument should be one of the standard locale designations, as described in "Language and Locale Designations."

  When you specify a locale for indexing, the `docsetutil` tool uses the most appropriate metadata files for the specified locale to create the index files. It stores the generated index files in a `<locale>.lproj` subdirectory of the `Resources` directory. Without this option, `docsetutil` creates global indexes, stored at the top level of the `Resources` directory, although it uses the most appropriate metadata files for the current user's preferred language.
- `–verbose`. Print additional information about the operation.
- `–debug`. Print additional debugging information about the tool and any errors.

These are the available verbs and their corresponding options:

- `help`. Print a list of the `docsetutil` tool's command-line options.

- `index`. Generate indexes for the specified documentation set. These are this verb's options:

  `-fallback <path>`. The path to a local copy of the web content associated with the documentation set. This is the web content at the location specified by the documentation set's "DocSetFallbackURL" (page 76) property. If the documentation set splits its content between the locally installed documentation set bundle and additional files on the web, you must create a local copy of the web content for indexing. When a documentation node—as listed in the `Nodes.xml` file—is not found in the documentation set bundle, the `docsetutil` tool looks at the location specified by the fallback argument and uses the files it finds there for the full-text index.

  `-node <node_path>`. (Documentation-set debugging) The path to the documentation node that you want to index. This option indexes the specified node and any subnodes it might have. `<node_path>` is a colon-separated string of node names, such as `Core Reference:Cocoa`. The root node is optional. Node names must match exactly the `Name` element in the nodes file.

  `-download`. Downloads and indexes the landing page associated with any node that specifies an Internet address as its location using the URL element. The `docsetutil` tool downloads only the node's landing page. It does not download the folder or folder hierarchy of content associated with a folder or bundle node.

  `-skip-text`, `-skip-api`. Prevents the creation of either the full-text or API indexes, respectively. By default, the indexer creates both the full-text and API indexes. You can use these flags to disable one of these indexes.

- `search`. Query the existing indexes for the given string and report any matches. This allows you to test indexes that you have previously generated. These are this verb's options:

  `-query <string>`. The string to search for in the specified indexes. This option is required.

  `-node <node_path>`. The path to the documentation node that you want to search. This is useful for limiting the scope of your search to a particular node in the documentation set, and any subnodes that it might have. `<node_path>` is a colon-separated string of node names, such as `Core Reference:Cocoa`. The root node is optional.

  `-skip-text`, `-skip-api`. Skip either the full-text or API indexes, respectively. By default, `docsetutil` searches both the full-text and API indexes. You can use these flags to remove either of these indexes from the search, which is useful for limiting the scope of your search during testing.

- `dump`. Print the contents of existing indexes. This is useful for testing the indexes and ensuring that all expected information is present. This command also prints useful statistics about these indexes, such as the total number of nodes and tokens in them. This verb's options are:

  `-node <node_path>`. The path to the documentation node whose contents you want to print. Use this option to limit the scope of the printed information. The `docsetutil` tool prints the contents of the specified node, including its subnodes. `<node_path>` is a colon-separated string of node names, such as `Core Reference:Cocoa`. The root node is optional.

`–toc–depth <toc_levels>`. The number of levels of the documentation hierarchy that you want to print. You can use this option to restrict the printout of the documentation set's table of contents to a particular depth.

`–text–depth <text_levels>`. The number of levels of information you want `docsetutil` to print from the full-text index. Possible values are:

- `0`: `docsetutil` prints only the names of the root locations. The root locations include the location of the local documentation set bundle (the default root), any location specified by the `DocSetFallbackURL` property, and any other base URL locations specified by individual nodes in the `Nodes.xml` file.
- `1`: (Default) `docsetutil` prints the root locations and the relative path to each of the nodes found at those root locations.
- `2`: `docsetutil` prints the root locations, the path to each of the nodes therein, and the path to each HTML file indexed for each node.

`–skip–text`, `–skip–api`. Skip either the full-text or API indexes, respectively. By default, `docsetutil` prints the contents of both the full-text and API indexes. You can use these flags to disable one or both of these indexes. This is useful for limiting the size of the printout during testing. If you use both of these flags together, `docsetutil` prints only the document hierarchy.

- `validate`. Compare the contents of the indexes to the contents of the documentation set bundle and report any discrepancies.
- `package`. Package the documentation set bundle as an XAR archive. Options:

  `–output <package_path>`. The pathname of the archive to create. If no output path is specified, docsetutil places the XAR archive in the same directory as the documentation set bundle. The archive's name, in this case, is the same as the documentation set bundle name, but with the extension `xar` instead of `docset`.

  `–signid <identity_name>`. The identity to use when adding a digital signature to the XAR archive. `docsetutil` looks through the user's keychains for an identity with this name.

  `–atom <atom_path>`. The path of the Atom feed to update with a new entry (or update an existing entry with the same version number) populated with all the available metadata about the documentation set and, if used, its signing identity. If the file does not exist, an Atom feed file is created.

  `–download–url <URL>`. The URL at which the package is placed. This URL is used to create the feed entry. If not specified, a placeholder is put into the feed entry and you need to edit the entry before publishing it.

Documentation-Set Property List Key Reference

This chapter describes the property list keys that you can use in the `Info.plist` file for your documentation set bundle. It lists only those keys that have particular relevance to documentation sets. For a list of the core OS X property list keys, see "Property List Key Reference."

CFBundleDevelopmentRegion

The default region for the bundle. This is typically the language in which the documentation set is developed. Xcode uses the language specified by this key as the default language if a documentation set resource cannot be found for the user's preferred region or language.

This is a standard OS X property list key. See Property List Key Reference for more information.

CFBundleIdentifier

A reverse-domain name style string that uniquely identifies the documentation set bundle, such as `com.apple.ADC_Reference_Library.CoreReference`. This string should be globally unique. This string should not change between different versions of the same documentation set.

Xcode uses the identifier string to match an installed documentation set with a specific entry in the RSS/Atom feed specified by the feed URL.

This is a standard OS X property list key. See Property List Key Reference for more information.

CFBundleName

The name of the documentation set. Xcode uses this name to identify the documentation set to the user. It appears in Xcode Documentation preferences. This string can be localized.

This is a standard OS X property list key. See Property List Key Reference for more information.

CFBundleVersion

The version number of the documentation set bundle. This value is compared to the version number for the documentation set with the same identifier in the web feed to determine whether a new version is available for download.

This is a standard OS X property list key. See Property List Key Reference for more information.

DocSetFallbackURL

An optional URL pointing to a parallel location on the web where the contents of the documentation set's Documents folder can be found. When browsing or searching the installed documentation set, if Xcode cannot find a file in the local installation, it tries to load the file from the web site specified by this key.

For example, if Xcode is trying to access a file at the path—relative to the documentation set's `Documents` directory—`PlugIns/MyPlugIn.html` and cannot find that file, it checks to see if the documentation set specifies an alternate fallback URL. If that fallback URL exists—say, for example `http://mycompany.com/Documentation/MyNiftyDocSet`—Xcode looks for the missing file at `http://mycompany.com/Documentation/MyNiftyDocSet/PlugIns/MyPlugIn.html`.

If you have a large documentation set, you can use a fallback website to reduce its footprint. You can install only the essential parts of your documentation in the local copy of the documentation set bundle, while less commonly used documents may exist only on your website.

> **Note:** Using the DocSetFallbackURL key to establish a remote root is not the same as using the URL element to specify a remote location for a documentation node's content. The DocSetFallbackURL specifies an alternate location for the entire documentation set. Xcode checks this alternate location only after it checks the local copy of the bundle.
>
> The URL element lets you specify a remote location for a single documentation node. Xcode always checks this location.

DocSetFeedName

A short name describing the overall group of documentation to which this documentation set belongs. This name normally does not appear to the Xcode user, although Xcode uses this name as the publisher name if you do not set an explicit publisher name.

This display name is typically a name identifying the provider of the documentation set, such as "Apple." This string can be localized.

DocSetFeedURL

The URL of an RSS/Atom feed that Xcode can use to learn about updates or additional documentation sets available from the same publisher. A single publisher can provide multiple RSS/Atom feeds, with each feed providing information about multiple, related documentation sets. Any given documentation set should be listed in only one of these feeds.

DocSetPublisherIdentifier

A string specifying the unique identifier for the publisher.

For example: `com.mycompany.myproduct.documentation`

All documentation sets that have the same publisher identifier are grouped under the same publisher name, even if the documentation sets are provided by different feeds. If missing, Xcode uses the `DocSetFeedURL` as the identifier, in which case the feed is treated as its own publisher.

DocSetPublisherName

A string specifying the display name for the publisher.

For example: `My Company`

If missing, Xcode uses the `DocSetFeedName`. You can localize this string.

DocSetCertificateSigner

A string specifying the distinguished name of the signer of the documentation set certificate.

For example: `C=US,O=Apple Computer\, Inc.,OU=mac.com,CN=myMacAcct`.

DocSetCertificateIssuer

A string specifying the distinguished name of the issuer of the documentation set certificate.

For example: `C=US,O=Apple Computer\, Inc.,OU=Apple Computer Certificate Authority,CN=Apple .Mac Certificate Authority`.

NSHumanReadableCopyright

A copyright string. This string is displayed in the Info window for the documentation set.

This is a standard OS X property list key. See Property List Key Reference for more information.

Documentation-Set Nodes Schema Reference

This chapter describes the element structure of the `Nodes.xml` (nodes) file used in documentation sets.

The schema for the nodes file, `NodesSchema.rng`, is located in the DocSetAccess framework in `<Xcode>/Library/PrivateFrameworks/DocSetAccess.framework`.

Listing C-1 lists the topology of the elements of a nodes file.

Listing C-1 Nodes.xml element topology

```
DocSetNodes
   TOC
      Node
         Name
         URL
         Path
         File
         Anchor
         Subnodes
            Node
            NodeRef
      NodeRef
         Subnodes
   Library
      Node
```

DocSetNodes

Root element of the nodes file.

```
DocSetNodes [version]
   TOC
```

```
    Library
```

Attributes

Name	Type	Description
`version`	Decimal	The version number of the `NodesSchema.rng` file. The only supported version is 1.0.

Subelements

Cardinality	Element
1	"TOC" (page 80)
`0..1`	"Library" (page 86)

TOC

Defines the document hierarchy.

```
TOC []
    Node|NodeRef
```

Attributes

None.

Subelements

Cardinality	Element	Usage
1	"Node" (page 81) or "NodeRef" (page 83)	Although this element must contain a `Name` element, the Documentation window actually displays the localized documentation set name (specified by the `CFBundleName` property) for this root node. The documentation set name is displayed in the Home pop-up menu in the Documentation window.

Node

Represents a single node in the document hierarchy.

```
Node [ id,type,isPrimaryTOCNode,noindex]
    Name
    URL
    Path
    File
    Anchor
    Subnodes
```

A node represents a file or a group of files within the documentation set. A node is associated with a location in the HTML files, specified by a combination of its `URL`, `Path`, `File` and `Anchor` subelements.

See "Specifying Subnodes" (page 27) for usage details.

Attributes

Name	Type	Description
`id`	Integer	*Optional*. This unique—within the documentation set—identifier allows the node to be referenced elsewhere in the `Nodes.xml` or `Tokens.xml` files.

Name	Type	Description
`type`	String	*Optional*. Specifies the node type. Values: `file` (default), `folder`, `bundle`.
`isPrimaryTOCNode`	Boolean	*Optional*. Indicates whether the node represents the primary TOC location of the document the node represents. This is relevant only if this node appears in the node hierarchy multiple times. Default: `false`.
`noindex`	Boolean	*Optional*. Species whether the node is excluded from the documentation set's indexes and from any searches. Default: `false`.

Subelements

Cardinality	Element	Usage
`1`	"Name" (page 82)	Specifies the node's name, which appears in the Documentation window.
`0..1`	"URL" (page 84)	Specifies the location of the files the node represents.
`0..1`	"Path" (page 85)	Specifies the location of the files the node represents.
`0..1`	"File" (page 85)	Specifies the name of the file the node represents.
`0..1`	"Anchor" (page 86)	Specifies a scroll-to location within the node's landing page.
`0..1`	"Subnodes" (page 83)	Defines a group of subnodes.

Name

Specifies a name.

```
Name [] {string}
```

Attributes

None.

Content

String.

Subnodes

Defines a list of nodes.

```
Subnodes []
    Node,NodeRef
```

Attributes

None.

Subelements

Cardinality	Element
1..*	"Node" (page 81)
	"NodeRef" (page 83)

NodeRef

Refers to a node defined elsewhere in the nodes file.

```
NodeRef [refid, isPrimaryTOCNode]
    Subnodes
```

You can also use this element in tokens files to associate a symbol with a documentation node. This is useful when a `Node` element describing the symbol's reference documentation already exists.

When it appears within a `Subnodes` or `TOC` element, a `NodeRef` is treated as if the `Node` element it references was itself listed there. This allows a document to be listed multiple times in the document hierarchy of the documentation set.

Important: If multiple `NodeRef` elements define subnodes for the same `Node`, only one set of subnodes are used; which one is used is undefined.

Attributes

Name	Type	Description
`refid`	Integer	Specifies the `id` of the referenced node.
`isPrimaryTOCNode`	Boolean	*Optional*. Indicates whether the node represents the primary TOC location of the document the node represents. Default: `false`.

Subelements

Cardinality	Element	Usage
`0..1`	"Subnodes" (page 83)	Defines a group of subnodes. These subnodes are used only when the node identified by the `refid` attribute does not contain subnodes.

URL

Specifies the base location of the node's documentation as a URL.

```
URL [] {URL}
```

When used alone, the `URL` element is interpreted as the full path to the file to load when the user selects the node. See "Specifying the Location of a Documentation Node" (page 22) for details about using this element with other `Node` subelements.

Attributes

None.

Content

URL.

Path

Specifies the path to the file or directory associated with a node.

```
Path [] {filepath}
```

The Path element can specify:

- The relative path to the node's landing page.
- A subpath to the directory containing the node's landing page.

See "Specifying the Location of a Documentation Node" (page 22) for details about using this element with other `Node` subelements.

Attributes

None.

Content

Filepath.

File

Specifies a filename.

```
File [] {normalizedString}
```

See "Specifying the Location of a Documentation Node" (page 22) for details about using this element with other `Node` subelements.

Attributes

None.

Content

Normalized string.

Anchor

Identifies a scroll-to-here location within a node's landing page.

```
Anchor [] {normalizedString}
```

See "Specifying the Location of a Documentation Node" (page 22) for details about using this element with other `Node` subelements.

Attributes

None.

Content

Normalized string.

Library

Defines a library of nodes.

```
Library []
    Node
```

See "Creating a Library of Node Definitions" (page 30) for usage details.

Attributes

None.

Subelements

Cardinality	Element
1..*	"Node" (page 81)

Documentation-Set Tokens Schema Reference

This chapter describes the element structure of the `Tokens.xml` (tokens) file used in documentation sets.

The schema for the nodes file, `TokensSchema.rng`, is located in the DocSetAccess framework in `<Xcode>/Library/PrivateFrameworks/DocSetAccess.framework`.

Listing D-1 lists the topology of the elements of a nodes file.

Listing D-1 `Tokens.xml` element topology

```
Tokens
   Token
      TokenIdentifier
         Name
         APILanguage
         Type
         Scope
      Path
      NodeRef
      Anchor
      Abstract
      Declaration
      Parameters
          Parameter
             Name
             Abstract
      ReturnValue
      DeclaredIn
         HeaderPath
         FrameworkName
      Availability
         IntroducedInVersion
         RemovedAfterVersion
```

```
            DeprecatedInVersion
            DeprecationSummary
         RelatedTokens
            TokenIdentifier
         RelatedDocuments
            NodeRef
            URL
         RelatedSampleCode
            NodeRef
            URL
      File
         Token
      RelatedTokens
         TokenIdentifier
```

Tokens

The root element of a tokens file.

```
Tokens [version]
   Token
   File
   RelatedTokens
```

Attributes

Name	Type	Description
`version`	Decimal	The version number of the `NodesSchema.rng` file. The only supported version is 1.0.

Subelements

Cardinality	Element	Para
1..*	"Token" (page 90)	Specifies a symbol.
	"File" (page 85)	Identifies an HTML file and defines set of symbols that are documented in that file. See "Grouping Tokens by File" (page 46) for details.
	"RelatedTokens" (page 103)	Specifies a set of symbols in which each symbol is related to every other symbol in the set. See "Specifying Related Tokens" (page 47) for details.

Token

Describes a single symbol, or token.

```
Token []
    TokenIdentifier
    Path
    NodeRef
    Anchor
    Abstract
    Parameters
    ReturnValue
    Declaration
    DeclaredIn
    Availability
    RelatedTokens
    RelatedDocuments
    RelatedSampleCode
```

This element:

- Associates a symbol with its primary reference documentation.
- Supplies additional information—such as availability, declaration, and so forth—about the symbol.

For usage information, see "Defining a Symbol for Lookup" (page 36).

Attributes

None.

Subelements

Cardinality	Element	Usage
1	"TokenIdentifier" (page 92)	Identifies the symbol.
0..1	"Path" (page 100)	Specifies the path to the HTML file containing the primary documentation. Must not be used when the token is located within a `File` element.
0..1	"NodeRef" (page 96)	References the node representing the primary documentation for the symbol. Must not be used when the token is located within a `File` element. Can be used alone or in conjunction with a "Path" (page 100) element (see "Associating Symbols with API Reference Documentation" (page 39) for details).
0..1	"Anchor" (page 95)	If you have a single HTML file that contains the primary reference documentation for more than one token, use this element to specify the location within that file of a particular token's description.
0..1	"Abstract" (page 95)	Provides a summary or brief description of the symbol.
0..1	"Declaration" (page 96)	Specifies the symbol's declaration statement.
0..1	"Parameters" (page 97)	Specifies the list of parameters, if any, passed to the symbol.
0..1	"ReturnValue" (page 98)	Specifies the value returned by the symbol, if any.
0..1	"DeclaredIn" (page 98)	Specifies the header and framework in which the symbol is declared.
0..1	"Availability" (page 99)	Specifies the product versions and computer architectures in which the token appears.

Cardinality	Element	Usage
0..1	"RelatedTokens" (page 103)	Specifies a set of symbols that are related to the token. The relationship is one way; there's no inverse relationship from those symbols to this one.
0..1	"RelatedDocuments" (page 104)	Specifies a list of documents that contain further information about the symbol. A token identifies its primary reference documentation through the "Path" (page 100), "NodeRef" (page 96), or "File" (page 105) elements; do not use this element to specify the primary reference documentation.
0..1	"RelatedSampleCode" (page 105)	Specifies a list of documents containing sample code that showcase the symbol's usage.

TokenIdentifier

Uniquely identifies a symbol or token.

```
TokenIdentifier [] {tokenizedString}
    Name
    APILanguage
    Type
    Scope
```

There are two ways to specify a token identifier (use only one):

1. Using the `Name`, `Type`, `APILanguage`, and `Scope` subelements to individually specify the token's properties ("Subelements" (page 93)).
2. Using an identifier that conforms to the apple_ref convention ("Content" (page 93)), described in "Symbol Markers for HTML-Based Documentation" in *HeaderDoc User Guide*.

For more information, see "Identifying Symbols" (page 37).

Attributes

None.

Subelements

Each subelement specifies a specific component of a symbol identifier. See "Defining Tokens Using Individual Properties" (page 37) for details.

Cardinality	Element	Usage
1	"Name" (page 93)	Specifies the name of the symbol. For example: `arrayWithContentsOfFile:`.
`0..1`	"APILanguage" (page 94)	Specifies the programming language in which the symbol is defined. For example: `occ`.
1	"Type" (page 94)	Specifies the symbol's type. For example: `clm`.
`0..1`	"Scope" (page 94)	Specifies the scope within which the symbol is defined. For example: `NSArray`.

Content

Tokenized string. This string identifies a symbol. For example: `//apple_ref/occ/clm/NSArray/arrayWithContentsOfFile:`. See "Defining Tokens Using apple_ref Identifiers" (page 38) for details.

Name

Specifies a name.

```
Name [] {string}
```

Attributes

None.

Content

String.

APILanguage

Species a programming language.

```
APILanguage [] {string}
```

Attributes

None.

Content

String.

Type

Specifies a symbol type.

```
Type [] {string}
```

Attributes

None.

Content

String. Valid values are described in “Symbol Markers for HTML-Based Documentation” in *HeaderDoc User Guide*.

Scope

Specifies a programming scope (namespace or container).

```
Scope [] {string}
```


Attributes

None.

Content

String.

Abstract

Specifies a summary or brief description.

```
Abstract [type] {HTMLCode|string}
```

This element lets you provide a summary description, usually one sentence.

When using HTML content, it must be valid; this includes double-escaping of entities. You can include links or basic HTML formatting. Hyperlinks with relative paths are resolved relative to the `Documents` directory.

Attributes

Name	Type	Description
`type`	String	*Optional*. The type of the content. Values: `"text"` (default), `"html"`.

Content

HTML code or string.

Anchor

Specifies the name of an anchor in an HTML file.

```
Anchor [] {normalizedString}
```

Attributes

None.

Content

Normalized string.

NodeRef

References a node defined in the nodes file.

```
NodeRef [refid]
```

Attributes

Name	Type	Description
`refid`	Integer	Specifies the `id` of the referenced node.

Declaration

Specifies a symbol's declaration statement.

```
Declaration [type] {HTMLCode|string}
```

When using HTML content, it must be valid; this includes double-escaping of entities. You can include links or basic HTML formatting. Hyperlinks with relative paths are resolved relative to the `Documents` directory.

Use an HTML PRE element to ensure the line breaks and indentations are preserved when the content is displayed.

Attributes

Name	Type	Description
`type`	String	*Optional*. The type of the content. Values: `"text"` (default), `"html"`.

Content

HTML code or string.

Parameters

Specifies the list of parameters that can be passed to the symbol, if any.

```
Parameters
    Parameter
```

`Parameters` encloses one or more `Parameter` elements. Each parameter element encloses the parameter name followed by an abstract.

Subelements

Cardinality	Element	Usage
1..*	"Parameter" (page 97)	Specifies a parameter that can be passed to the symbol.

Parameter

Specifies a parameter that can be passed to the symbol, if any.

```
Parameter
    Name
    Abstract
```

Subelements

Cardinality	Element	Usage
1	"Name" (page 93)	Specifies the parameter name.
1	"Abstract" (page 95)	Specifies a short description of the parameter.

ReturnValue

Specifies the value returned by the symbol, if any.

```
ReturnValue
    Abstract
```

Subelements

Cardinality	Element	Usage
1	"Abstract" (page 95)	Provides a short description of the value returned by the symbol.

DeclaredIn

Specifies the header and framework in which the symbol is declared, for tokens that describe an API symbol.

```
// Usage 1:
DeclaredIn []
   HeaderPath
   FrameworkName
// Usage 2:
DeclaredIn [] {filepath}
```

This element supports two usage patterns:

1. You can specify the file path to the header in which the symbol is declared and the name of the framework that must be loaded to use that symbol separately, using the `HeaderPath` and `FrameworkName` elements, respectively. See "Subelements" (page 99) for details.
2. If the API is not part of a framework, you can specify the path to the header as a string, directly within the `DeclaredIn` element. See "Content" (page 99) for details.

Subelements

Cardinality	Element	Usage
1	"HeaderPath" (page 100)	Specifies the pathname of the symbol's header file.
0..1	"FrameworkName" (page 101)	Specifies the name of the symbol's framework. Needed only when the symbol is part of a framework.

Content

Pathname. The pathname of the symbol's header file.

Availability

Specifies availability information related to a product and computer architecture.

```
Availability [distribution]
    IntroducedInVersion
    RemovedAfterVersion
    DeprecatedInVersion
    DeprecationSummary
```

See "Version Information" (page 42) for usage details.

Attributes

Name	Type	Description
distribution	String	Specifies the name of a product, such as OS X.

Subelements

Cardinality	Element	Usage
1..*	"IntroducedInVersion" (page 101)	Specifies a product version and computer architecture in which a symbol was introduced.

Cardinality	Element	Usage
`0..*`	"RemovedAfterVersion" (page 102)	Specifies a product version and computer architecture in which a symbol was last available.
`0..*`	"DeprecatedInVersion" (page 102)	Specifies a product version and computer architecture in which a symbol was deprecated.
`0..1`	"DeprecationSummary" (page 103)	Provides information about a symbol whose usage is not recommended.

Path

Specifies a filepath.

```
Path [] {filepath}
```

Attributes

None.

Content

Filepath.

HeaderPath

Species the pathname of a header file.

```
HeaderPath [] {pathname}
```

Attributes

None.

Content

Pathname.

FrameworkName

Specifies the name of a framework.

```
FrameworkName [] {string}
```

Attributes

None.

Content

String.

IntroducedInVersion

Specifies an introduced-in version number.

```
IntroducedInVersion [cputype,bitsize] {threeTupleNumber}
```

Attributes

Name	Type	Description
`cputype`	String	Specifies the CPU type to which this version information applies. Values: `ppc` (PowerPC), `i386` (Intel). When unspecified, the version applies to both CPU types.
`bitsize`	Integer	Specifies the CPU bitsize to which this version information applies. Values: `32` (32 bit), `64` (64 bit). When unspecified, the version number applies to both CPU bitsizes.

Content

Period-separated, three-tuple number in the form `x.y.z`, where `x`, `y`, and `z` are integers. Only the major version number—`x`—is required.

DeprecatedInVersion

Specifies a deprecated-in version number.

```
DeprecatedInVersion [cputype,bitsize] {threeTupleNumber}
```

Attributes

Name	Type	Description
`cputype`	String	Specifies the CPU type to which this version information applies. Values: `ppc` (PowerPC), `i386` (Intel). When unspecified, the version applies to both CPU types.
`bitsize`	Integer	Specifies the CPU bitsize to which this version information applies. Values: `32` (32 bit), `64` (64 bit). When unspecified, the version number applies to both CPU bitsizes.

Content

Period-separated, three-tuple number in the form `x.y.z`, where `x`, `y`, and `z` are integers. Only the major version number—`x`—is required.

RemovedAfterVersion

Specifies a removed-after version number.

```
RemovedAfterVersion [cputype,bitsize] {threeTupleNumber}
```

Attributes

Name	Type	Description
`cputype`	String	Specifies the CPU type to which this version information applies. Values: `ppc` (PowerPC), `i386` (Intel). When unspecified, the version applies to both CPU types.

Name	Type	Description
`bitsize`	Integer	Specifies the CPU bitsize to which this version information applies. Values: `32` (32 bit), `64` (64 bit). When unspecified, the version number applies to both CPU bitsizes.

Content

Period-separated, three-tuple number in the form `x.y.z`, where `x`, `y`, and `z` are integers. Only the major version number—`x`—is required.

DeprecationSummary

Specifies summary information about a symbol whose usage is not recommended.

Use this element to provide additional information about other symbols or technologies that the user should use instead.

```
DeprecationSummary [type] {HTMLCode|String}
```

When using HTML content, it must be valid; this includes double-escaping of entities. You can include links or basic HTML formatting. Hyperlinks with relative paths are resolved relative to the `Documents` directory.

Attributes

Name	Type	Description
`type`	String	*Optional*. The type of the content. Values: `"text"` (default), `"html"`.

Content

HTML code or string.

RelatedTokens

Defines a list of tokens.


```
RelatedTokens [title]
    TokenIdentifier
```

Attributes

Name	Type	Description
title	String	Specifies a label for the token list. (Optional)

Subelements

Cardinality	Element	Usage
1..*	"TokenIdentifier" (page 92)	Identifies a token defined in the documentation set.

RelatedDocuments

Defines a list of documents.

```
RelatedDocuments []
    NodeRef, URL
```

Attributes

None.

Subelements

Cardinality	Element	Usage
1..*	"NodeRef" (page 96)	References a node defined in the nodes file.
	URL	Absolute URL to a document outside the documentation set.

RelatedSampleCode

Defines a list of documents containing sample code.

```
RelatedSampleCode []
   NodeRef, URL
```

Attributes

None.

Subelements

Cardinality	Element	Usage
1..*	"NodeRef" (page 96)	References a node defined in the nodes file.
	URL	Absolute URL to a document outside the documentation set.

File

Identifies an HTML file and a set of tokens that are documented in that file.

```
File [path,noderef]
   Token
```

When you use the File element to group token definitions, the individual Token elements inside of the File element cannot contain Path or NodeRef elements.

Attributes

Name	Type	Description
path	Filepath	Specifies the path to the HTML file that documents the tokens. See "Grouping Tokens by File" (page 46) for details.
noderef	Integer	Specifies the id of the node to associate with the tokens this element specifies. See "Node" (page 81) for more information.

Subelements

Cardinality	Element	Usage
1..*	"Token" (page 90)	Specifies a token that is documented in the HTML file.

URL

Identifies the location of a document that is outside of the documentation set.

```
URL [] {URL}
```

Attributes

None.

Content

URL.

Document Revision History

This table describes the changes to *Documentation Set Guide*.

Date	Notes
2009-05-05	Updated Xcode installation location information. Updated for Xcode 3.2. Added new keys "DocSetPublisherIdentifier" (page 77) and "DocSetPublisherName" (page 77). Added new keys for feeds. See "Specifying Feed Information" (page 59). Added new token element information to "Documentation-Set Tokens Schema Reference" (page 88). Made numerous changes throughout the document due to modifications in the Documentation window and Xcode Documentation preferences user interfaces.
2009-01-06	Made minor corrections. Corrected typos in Listing 7-1 (page 60) and Listing 7-3 (page 64).
2007-10-31	New document that describes how to integrate third-party documentation with the Xcode Documentation window.

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