How to Use Templates

Here's how to apply templates in Word and LibreOffice.

Applying NSP_template_010721_AU.dotm in Word 2003 and Earlier Versions

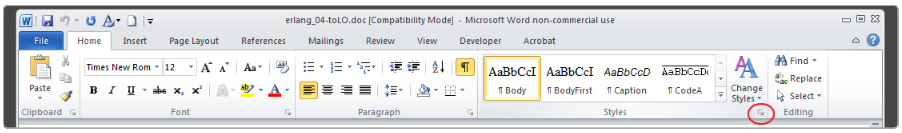
You can open a new untitled document with the template attached by double-clicking the template file itself (*NSP_template_01072021_AU.dotm*). This file also contains examples and explanations of the styles we use.

To apply the template to an existing document:

1. Open your chapter.
2. Go to **Tools ▸ Templates and Add-ins**.
3. Click the **Attach** button and browse to *NSP_template_01072021_AU.dotm*.
4. Check the box next to the words *Automatically update document styles*, then click **OK**.
5. Go to **Format ▸ Styles and Formatting**. The Styles and Formatting pane should open at the right side of your document window. (The Styles and Formatting pane is available in Word 2002 and later versions.)
6. In the drop-down menu at the bottom of the Styles and Formatting pane, choose **Show: Formatting in use** or **Show: All Styles**. Apply No Starch Press styles as necessary by selecting text and clicking the styles in the Styles and Formatting pane. If the version of Word you're using doesn't offer this Styles and Formatting pane, you can also select and apply styles using the drop-down menu on the toolbar.

Viewing Paragraph Styles

You can view paragraph style labels in the margin of your workspace by doing the following:



1. Select **View ▸ Normal**.
2. Go to **Tools ▸ Options**. Under the View tab, near the bottom, there is a setting called Style Area Width. In the measurement box, enter 1.0” (or more). A Style Area should appear to the left of your text, like a margin. The name of the paragraph style for each paragraph will be visible.

Applying NSP_template_01072021_AU.dotm in Word 2007 or later

You can open a new untitled document with the template attached by double-clicking the template file itself (*NSP_template_01072021_AU.dotm*). This file also contains examples and explanations of the styles we use.

To apply the template to an existing document, do the following:

1. Open your chapter.
2. From the **Office Button** menu (2007) or the **File** tab menu (2010), choose **Word Options** or **Options**.
3. Choose **Add-Ins** from the left side of the Word Options dialog.
4. On the right side of the window, near the bottom, choose **Templates** from the **Manage** drop-down list. Click **Go**.
5. Click the **Attach** button and browse to *NSP_template_01072021_AU.dotm*. Click **Open**.
6. Check the box next to the words *Automatically update document styles*, then click **OK**.

To show the Styles pane, navigate to the Home tab and click the small diagonal arrow at the bottom-right corner of the Styles gallery. The Styles pane will appear on the right side of your document.

Applying NSP_template_02072021_AU.ott in LibreOffice

1. Open LibreOffice.
2. Press CTRL-O and navigate to and select *NSP_template_02072021_AU.ott*. (The template is in OOo 2.x format.)
3. Choose **File ▸ Templates ▸ Save**. Type **`NSP_template_02072021_AU.ott`** into the Name field at the top of the window. Make sure *My Templates* is highlighted, then click **OK**.
4. Open your chapter.
5. Press F11 to open the Styles and Formatting window.
6. Click the button at the top right of the Styles and Formatting window (it has a + sign and a drop-down arrow), and select **Load Styles**. The Load Styles window should pop up (it is the same as the window you saw in step 2). If you are not saving the template, click the **From File...** button, navigate to your saved template, and click **OK**. If you've saved the template, be sure that *My Templates* and *1* are highlighted, then click **OK**.
7. Return to the Styles and Formatting window. Click the drop-down arrow at the very bottom (likely next to the word *Automatic*) and select **All Styles**. You should now see all the No Starch Press styles. If you want to see only the No Starch Press styles, choose **Custom Styles**.

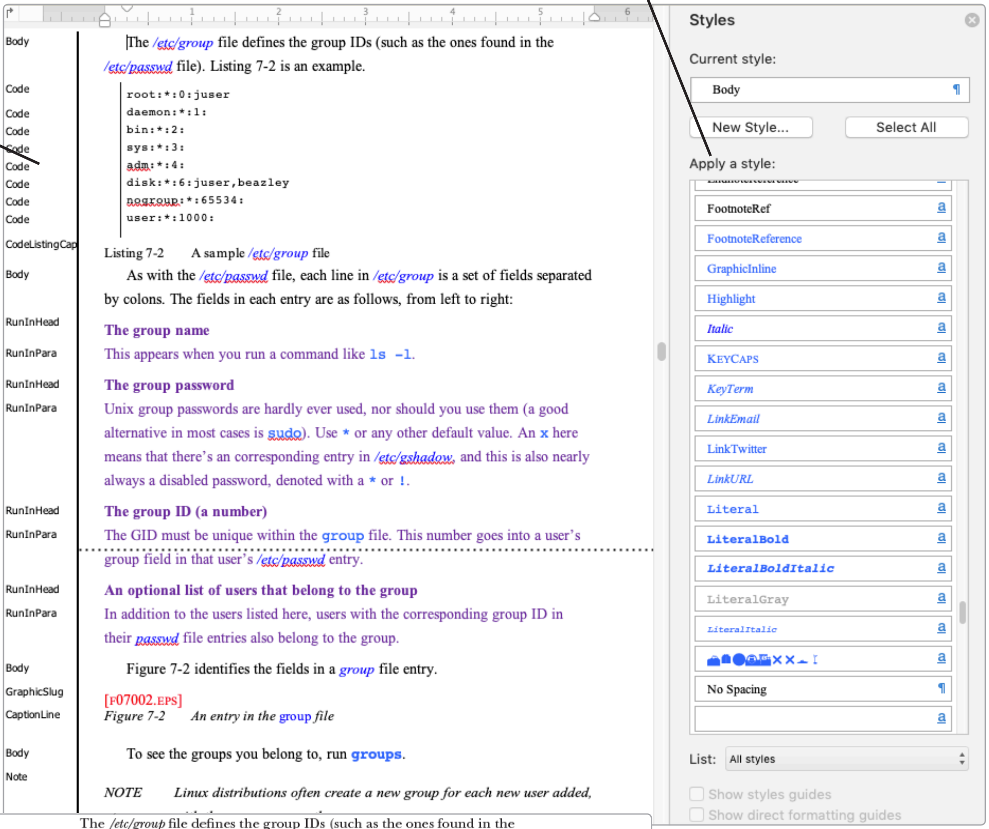
Here are some tips:

- Consider docking the Styles and Formatting window to one side of your OOo workspace. Just drag it off to the very edge and it should dock there.
- Once you've applied some styles (headings and such), you can choose Applied Styles from the drop-down menu at the bottom of the Styles and Formatting window to show only applied styles and reduce the clutter.
- Click the uppercase "A" icon to show character styles (Bold and such).

Word's Styles and Formatting pane
showing formatting in use

Style Area showing paragraph styles in Word

Character styles are shown in blue, and xrefs will appear in red until the final print PDF.



The /etc/group file defines the group IDs (such as the ones found in the /etc/passwd file). Listing 7-2 is an example.

```
root:*:0:juser
daemon:*:1:
bin:*:2:
sys:*:3:
adm:*:4:
disk:*:6:juser,beazley
nogroup:*:65534:
user:*:1000:
```

Listing 7-2 A sample /etc/group file

As with the /etc/passwd file, each line in /etc/group is a set of fields separated by colons. The fields in each entry are as follows, from left to right:

The group name

This appears when you run a command like ls -l.

The group password

Unix group passwords are hardly ever used, nor should you use them (a good alternative in most cases is sudo). Use * or any other default value. An x here means that there's an corresponding entry in /etc/gshadow, and this is also nearly always a disabled password, denoted with a * or !.

The group ID (a number)

The GID must be unique within the group file. This number goes into a user's group field in that user's /etc/passwd entry.

An optional list of users that belong to the group

In addition to the users listed here, users with the corresponding group ID in their passwd file entries also belong to the group.

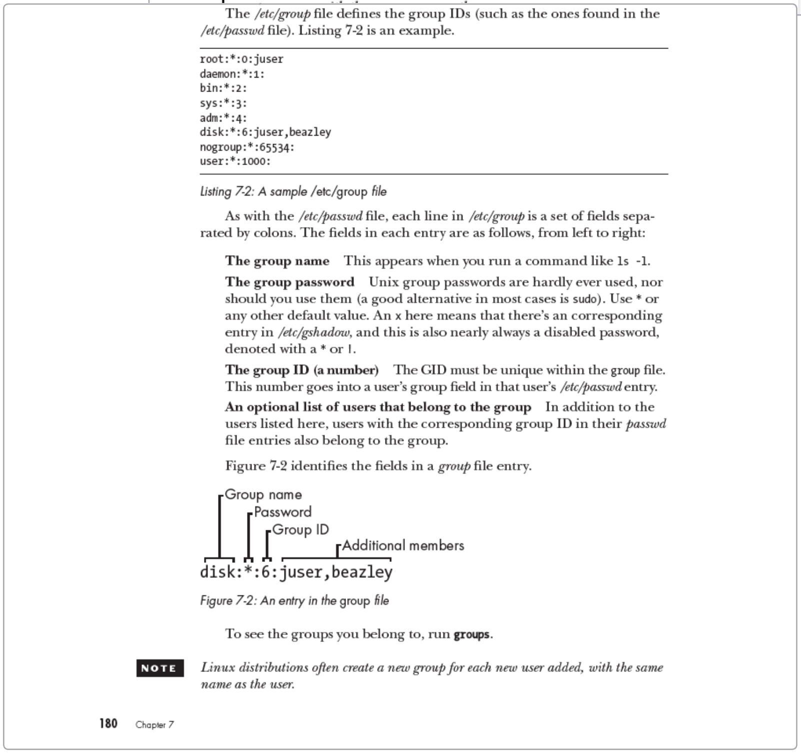
Figure 7-2 identifies the fields in a group file entry.

[F07002.EPS]

Figure 7-2 An entry in the group file

To see the groups you belong to, run groups.

NOTE Linux distributions often create a new group for each new user added,



The /etc/group file defines the group IDs (such as the ones found in the /etc/passwd file). Listing 7-2 is an example.

```
root:*:0:juser
daemon:*:1:
bin:*:2:
sys:*:3:
adm:*:4:
disk:*:6:juser,beazley
nogroup:*:65534:
user:*:1000:
```

Listing 7-2: A sample /etc/group file

As with the /etc/passwd file, each line in /etc/group is a set of fields separated by colons. The fields in each entry are as follows, from left to right:

The group name This appears when you run a command like ls -l.

The group password Unix group passwords are hardly ever used, nor should you use them (a good alternative in most cases is sudo). Use * or any other default value. An x here means that there's an corresponding entry in /etc/gshadow, and this is also nearly always a disabled password, denoted with a * or !.

The group ID (a number) The GID must be unique within the group file. This number goes into a user's group field in that user's /etc/passwd entry.

An optional list of users that belong to the group In addition to the users listed here, users with the corresponding group ID in their passwd file entries also belong to the group.

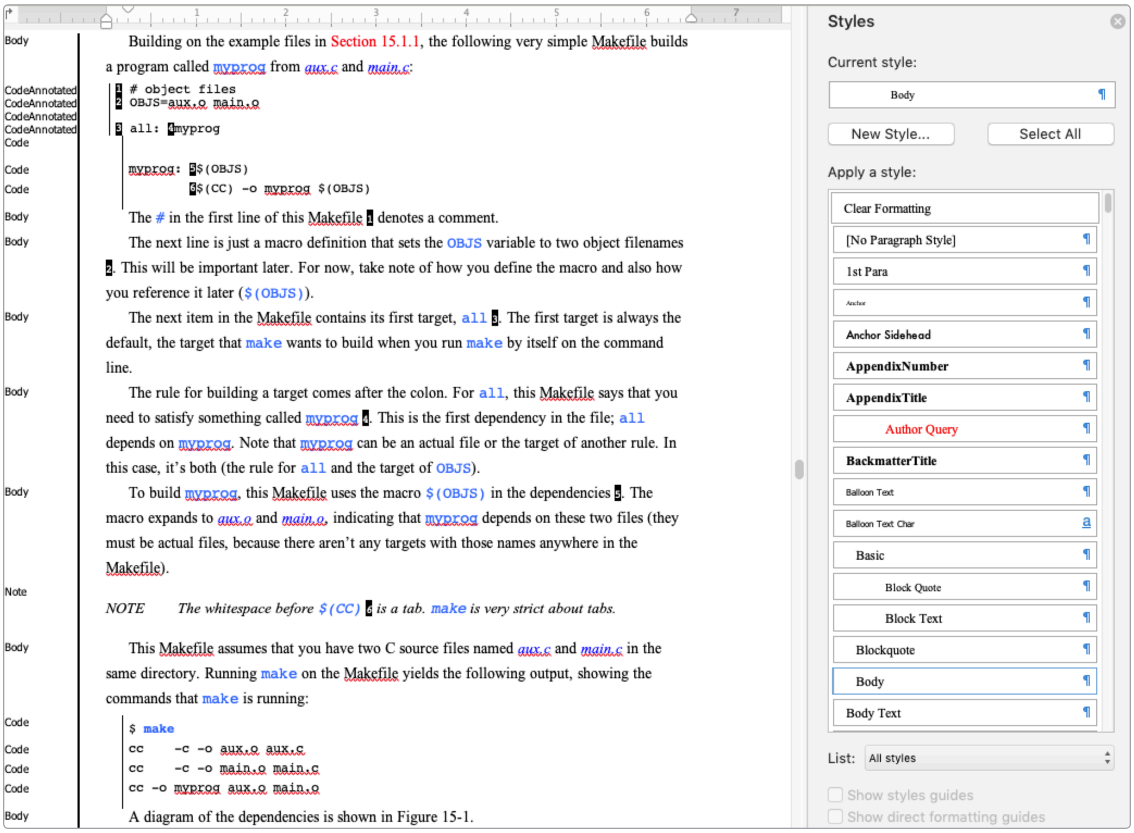
Figure 7-2 identifies the fields in a group file entry.

Figure 7-2: An entry in the group file

To see the groups you belong to, run **groups**.

NOTE *Linux distributions often create a new group for each new user added, with the same name as the user.*

180 Chapter 7



Body Building on the example files in Section 15.1.1, the following very simple Makefile builds a program called myprog from aux.c and main.c:

```
CodeAnnotated  1 # object files
CodeAnnotated  2 OBJS=aux.o main.o
CodeAnnotated
CodeAnnotated  3 all: 4myprog
Code
Code           myprog: 5$(OBJS)
Code                   6$(CC) -o myprog $(OBJS)
```

Body The # in the first line of this Makefile 1 denotes a comment.

Body The next line is just a macro definition that sets the OBJS variable to two object filenames 2. This will be important later. For now, take note of how you define the macro and also how you reference it later ($(OBJS)).

Body The next item in the Makefile contains its first target, all 3. The first target is always the default, the target that make wants to build when you run make by itself on the command line.

Body The rule for building a target comes after the colon. For all, this Makefile says that you need to satisfy something called myprog 4. This is the first dependency in the file; all depends on myprog. Note that myprog can be an actual file or the target of another rule. In this case, it's both (the rule for all and the target of OBJS).

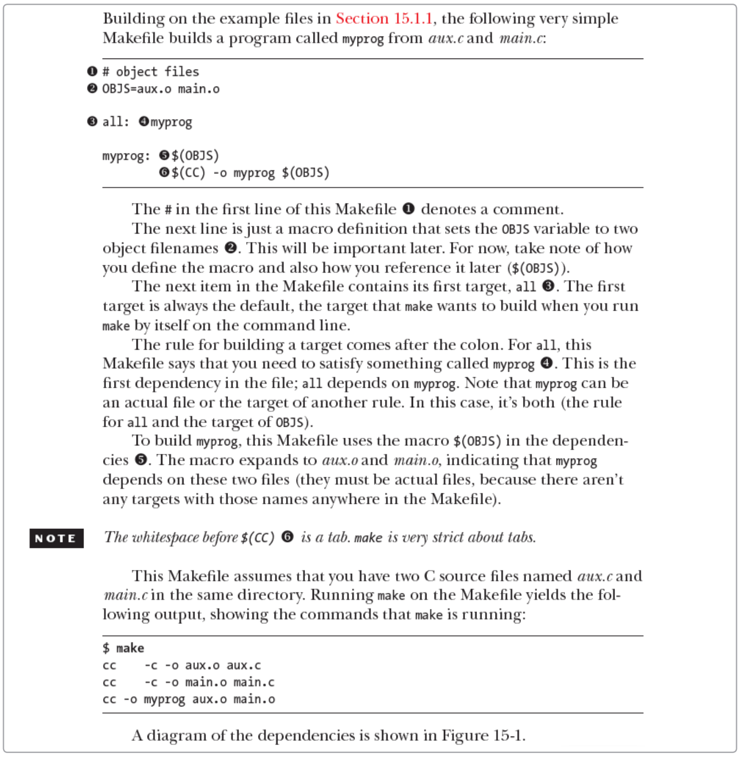
Body To build myprog, this Makefile uses the macro $(OBJS) in the dependencies 5. The macro expands to aux.o and main.o, indicating that myprog depends on these two files (they must be actual files, because there aren't any targets with those names anywhere in the Makefile).

Note *NOTE The whitespace before $(CC) 6 is a tab. make is very strict about tabs.*

Body This Makefile assumes that you have two C source files named aux.c and main.c in the same directory. Running make on the Makefile yields the following output, showing the commands that make is running:

```
Code  $ make
Code  cc    -c -o aux.o aux.c
Code  cc    -c -o main.o main.c
Code  cc -o myprog aux.o main.o
```

Body A diagram of the dependencies is shown in Figure 15-1.



Building on the example files in Section 15.1.1, the following very simple Makefile builds a program called myprog from *aux.c* and *main.c*:

```
❶ # object files
❷ OBJS=aux.o main.o

❸ all: ❹myprog

  myprog: ❺$(OBJS)
         ❻$(CC) -o myprog $(OBJS)
```

The # in the first line of this Makefile ❶ denotes a comment.

The next line is just a macro definition that sets the OBJS variable to two object filenames ❷. This will be important later. For now, take note of how you define the macro and also how you reference it later ($(OBJS)).

The next item in the Makefile contains its first target, all ❸. The first target is always the default, the target that make wants to build when you run make by itself on the command line.

The rule for building a target comes after the colon. For all, this Makefile says that you need to satisfy something called myprog ❹. This is the first dependency in the file; all depends on myprog. Note that myprog can be an actual file or the target of another rule. In this case, it's both (the rule for all and the target of OBJS).

To build myprog, this Makefile uses the macro $(OBJS) in the dependencies ❺. The macro expands to *aux.o* and *main.o*, indicating that myprog depends on these two files (they must be actual files, because there aren't any targets with those names anywhere in the Makefile).

NOTE *The whitespace before $(CC) ❻ is a tab. make is very strict about tabs.*

This Makefile assumes that you have two C source files named *aux.c* and *main.c* in the same directory. Running make on the Makefile yields the following output, showing the commands that make is running:

```
$ make
cc    -c -o aux.o aux.c
cc    -c -o main.o main.c
cc -o myprog aux.o main.o
```

A diagram of the dependencies is shown in Figure 15-1.

Examples of manuscript pages in Word and the same content in layout (this page and previous page)