




# Mac OS X Server

Podcast Producer  
Workflow Tutorial  
For Version 10.5 Leopard

 Apple Inc.  
© 2008 Apple Inc. All rights reserved.

The owner or authorized user of a valid copy of Mac OS X Server software may reproduce this publication for the purpose of learning to use such software. No part of this publication may be reproduced or transmitted for commercial purposes, such as selling copies of this publication or for providing paid-for support services.

Every effort has been made to ensure that the information in this manual is accurate. Apple Inc. is not responsible for printing or clerical errors.

Apple  
1 Infinite Loop  
Cupertino CA 95014-2084  
408-996-1010  
[www.apple.com](http://www.apple.com)

The Apple logo is a trademark of Apple Inc., registered in the U.S. and other countries. Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

Apple, the Apple logo, iCal, iChat, iTunes, Leopard, Mac, the Mac logo, Macintosh, Mac OS, QuickTime, Xgrid, and Xserve are trademarks of Apple Inc., registered in the U.S. and other countries. Finder is a trademark of Apple Inc.

Adobe and PostScript are trademarks of Adobe Systems Incorporated.

UNIX is a registered trademark of The Open Group.

Other company and product names mentioned herein are trademarks of their respective companies. Mention of third-party products is for informational purposes only and constitutes neither an endorsement nor a recommendation. Apple assumes no responsibility with regard to the performance of these products.

019-1253/2008-04-25

# Contents

<b>Preface</b>	<b>7 About This Guide</b>
	7 What's in This Guide
	8 Using Onscreen Help
	8 Mac OS X Server Administration Guides
	10 Viewing PDF Guides Onscreen
	10 Printing PDF Guides
	10 Getting Documentation Updates
	11 Getting Additional Information
<b>Chapter 1</b>	<b>13 Introduction</b>
	13 Before You Start
	14 The Workflow at a Glance
	15 Downloading the Workflow Tutorial Files
	16 Additional Sources of Information
<b>Chapter 2</b>	<b>17 About Workflow Templates</b>
	17 Workflow Variables
	18 Workflow Tasks
	18 Contents of a Workflow Template
	19 About the art.rb Script
<b>Chapter 3</b>	<b>21 Hello World!</b>
	21 The Workflow at a Glance
	21 Workflow 1 Setup Summary
	22 Setting Up Workflow 1
	26 Validating the Workflow
	27 Deploying the Workflow
	27 Testing the Workflow
<b>Chapter 4</b>	<b>29 Encoding Media</b>
	29 The Workflow at a Glance
	30 Workflow 2 Setup Summary
	30 Setting Up Workflow 2
	33 Getting Information About the pcstaction Command

	35	Deploying the Workflow
<b>Chapter 5</b>	<b>37</b>	<b>Running Tasks in Parallel</b>
	37	The Workflow at a Glance
	38	Workflow 3 Setup Summary
	38	Setting Up Workflow 3
	41	Deploying the Workflow
<b>Chapter 6</b>	<b>43</b>	<b>Modifying the Mail Template</b>
	43	The Workflow at a Glance
	44	Workflow 4 Setup Summary
	44	Setting Up Workflow 4
	49	Deploying the Workflow
<b>Chapter 7</b>	<b>51</b>	<b>Posting to the Group Blog</b>
	51	The Workflow at a Glance
	52	Workflow 5 Setup Summary
	52	Setting Up Workflow 5
	55	Writing a Blog Template
	56	Updating the Mail Template
	57	Deploying the Workflow
<b>Chapter 8</b>	<b>59</b>	<b>Watermarking</b>
	59	The Workflow at a Glance
	60	Workflow 6 Setup Summary
	60	Setting Up Workflow 6
	61	Deploying the Workflow
<b>Chapter 9</b>	<b>63</b>	<b>Adding Quartz Composer Effects</b>
	64	The Workflow at a Glance
	64	About the Lower Third Composition
	65	Workflow 7 Setup Summary
	65	Setting Up Workflow 7
	67	Deploying the Workflow
<b>Chapter 10</b>	<b>69</b>	<b>Adding an Introductory Video</b>
	70	The Workflow at a Glance
	70	About the Intro Movie
	71	Workflow 8 Setup Summary
	71	Setting Up Workflow 8
	72	Deploying the Workflow
<b>Chapter 11</b>	<b>73</b>	<b>Dynamic Titling</b>
	74	The Workflow at a Glance

	75	Workflow 9 Setup Summary
	77	Deploying the Workflow
<b>Chapter 12</b>	<b>79</b>	<b>Calling Command-Line Tools</b>
	80	The Workflow at a Glance
	81	Workflow 10 Setup Summary
	81	Setting Up Workflow 10
	85	Deploying the Workflow
<b>Chapter 13</b>	<b>87</b>	<b>Integrating with Custom Scripts</b>
	88	The Workflow at a Glance
	89	Workflow 11 Setup Summary
	89	Setting Up Workflow 11
	93	Deploying the Workflow
<b>Chapter 14</b>	<b>95</b>	<b>Tips for Designing and Developing Workflows</b>
	95	Create a Graph of the Workflow
	95	Start Small
	96	Use an All-in-One Setup for Workflow Development
	96	Validate Your Workflow Before Running It
	96	Check Metadata
	99	Keep Temporary Files
<b>Chapter 15</b>	<b>101</b>	<b>Debugging Workflows</b>
	101	Using Xgrid Admin
	101	Using the xgrid Command
	101	Monitoring Xgrid Tasks
	101	Debugging Failing Commands



# About This Guide

## Use this tutorial to learn how to develop and customize Podcast Producer workflows.

*Podcast Producer Workflow Tutorial* takes you step by step through the process of developing a workflow incrementally.

### What's in This Guide

This guide includes the following chapters:

- Chapter 1, "Introduction," provides information you should know before going through this tutorial.
- Chapter 2, "About Workflow Templates," provides an overview of workflow templates.
- Chapter 3, "Hello World!," describes how to create a simple workflow that sends email containing the message Hello World! to a list of recipients.
- Chapter 4, "Encoding Media," describes how to encode movies to play on iPods.
- Chapter 5, "Running Tasks in Parallel," describes how to configure workflow tasks to run in parallel.
- Chapter 6, "Modifying the Mail Template," describes how to modify a mail template, which generates properly formatted mail messages.
- Chapter 7, "Posting to the Group Blog," describes how to post podcasts to group blogs.
- Chapter 8, "Watermarking," describes how to add a watermark to movies.
- Chapter 9, "Adding Quartz Composer Effects," describes how to add the Lower Third Quartz Composer effect to movies.
- Chapter 10, "Adding an Introductory Video," describes how to add an introductory movie to the movie being processed by the workflow.
- Chapter 11, "Dynamic Titling," describes how to dynamically add titles to movies.
- Chapter 12, "Calling Command-Line Tools," describes how to access command-line tools from workflows using the `pcastaction` command.
- Chapter 13, "Integrating with Custom Scripts," describes how to add custom functionality to workflows through shell scripts.

- Chapter 14, “Tips for Designing and Developing Workflows,” provides tips for designing and developing Podcast Producer workflows.
- Chapter 15, “Debugging Workflows,” describes tools and techniques for debugging workflows.

**Note:** Because Apple periodically releases new versions and updates to its software, images shown in this book may be different from what you see on your screen.

## Using Onscreen Help

You can get task instructions onscreen in the Help Viewer application while you’re managing Leopard Server. You can view help on a server or an administrator computer. (An administrator computer is a Mac OS X computer with Leopard Server administration software installed on it.)

### To get help for an advanced configuration of Leopard Server:

- Open Server Admin or Workgroup Manager and then:
  - Use the Help menu to search for a task you want to perform.
  - Choose Help > Server Admin Help or Help > Workgroup Manager Help to browse and search the help topics.

The onscreen help contains instructions taken from *Server Administration* and other advanced administration guides described in “Mac OS X Server Administration Guides,” next.

### To see the most recent server help topics:

- Make sure the server or administrator computer is connected to the Internet while you’re getting help.

Help Viewer automatically retrieves and caches the most recent server help topics from the Internet. When not connected to the Internet, Help Viewer displays cached help topics.

## Mac OS X Server Administration Guides

*Getting Started* covers basic installation and initial setup methods for an advanced configuration of Leopard Server as well as for a standard or workgroup configuration. An advanced guide, *Server Administration*, covers advanced planning, installation, setup, and more. A suite of additional guides, listed below, covers advanced planning, setup, and management of individual services. You can get these guides in PDF format from the Mac OS X Server documentation website:



This guide ...	tells you how to:
<i>Getting Started and Mac OS X Server Worksheet</i>	Install Mac OS X Server and set it up for the first time.
<i>Command-Line Administration</i>	Install, set up, and manage Mac OS X Server using UNIX command-line tools and configuration files.
<i>File Services Administration</i>	Share selected server volumes or folders among server clients using the AFP, NFS, FTP, and SMB protocols.
<i>iCal Service Administration</i>	Set up and manage iCal shared calendar service.
<i>iChat Service Administration</i>	Set up and manage iChat instant messaging service.
<i>Mac OS X Security Configuration</i>	Make Mac OS X computers (clients) more secure, as required by enterprise and government customers.
<i>Mac OS X Server Security Configuration</i>	Make Mac OS X Server and the computer it's installed on more secure, as required by enterprise and government customers.
<i>Mail Service Administration</i>	Set up and manage IMAP, POP, and SMTP mail services on the server.
<i>Network Services Administration</i>	Set up, configure, and administer DHCP, DNS, VPN, NTP, IP firewall, NAT, and RADIUS services on the server.
<i>Open Directory Administration</i>	Set up and manage directory and authentication services, and configure clients to access directory services.
<i>Podcast Producer Administration</i>	Set up and manage Podcast Producer service to record, process, and distribute podcasts.
<i>Print Service Administration</i>	Host shared printers and manage their associated queues and print jobs.
<i>QuickTime Streaming and Broadcasting Administration</i>	Capture and encode QuickTime content. Set up and manage QuickTime streaming service to deliver media streams live or on demand.
<i>Server Administration</i>	Perform advanced installation and setup of server software, and manage options that apply to multiple services or to the server as a whole.
<i>System Imaging and Software Update Administration</i>	Use NetBoot, NetInstall, and Software Update to automate the management of operating system and other software used by client computers.
<i>Upgrading and Migrating</i>	Use data and service settings from an earlier version of Mac OS X Server or Windows NT.
<i>User Management</i>	Create and manage user accounts, groups, and computers. Set up managed preferences for Mac OS X clients.
<i>Web Technologies Administration</i>	Set up and manage web technologies, including web, blog, webmail, wiki, MySQL, PHP, Ruby on Rails, and WebDAV.
<i>Xgrid Administration and High Performance Computing</i>	Set up and manage computational clusters of Xserve systems and Mac computers.
<i>Mac OS X Server Glossary</i>	Learn about terms used for server and storage products.

## Viewing PDF Guides Onscreen

While reading the PDF version of a guide onscreen:

- Show bookmarks to see the guide's outline, and click a bookmark to jump to the corresponding section.
- Search for a word or phrase to see a list of places where it appears in the document. Click a listed place to see the page where it occurs.
- Click a cross-reference to jump to the referenced section. Click a web link to visit the website in your browser.

## Printing PDF Guides

If you want to print a guide, you can take these steps to save paper and ink:

- Save ink or toner by not printing the cover page.
- Save color ink on a color printer by looking in the panes of the Print dialog for an option to print in grays or black and white.
- Reduce the bulk of the printed document and save paper by printing more than one page per sheet of paper. In the Print dialog, change Scale to 115% (155% for *Getting Started*). Then choose Layout from the untitled pop-up menu. If your printer supports two-sided (duplex) printing, select one of the Two-Sided options. Otherwise, choose 2 from the Pages per Sheet pop-up menu, and optionally choose Single Hairline from the Border menu. (If you're using Mac OS X v10.4 or earlier, the Scale setting is in the Page Setup dialog and the Layout settings are in the Print dialog.)

You may want to enlarge the printed pages even if you don't print double sided, because the PDF page size is smaller than standard printer paper. In the Print dialog or Page Setup dialog, try changing Scale to 115% (155% for *Getting Started*, which has CD-size pages).

## Getting Documentation Updates

Periodically, Apple posts revised help pages and new editions of guides. Some revised help pages update the latest editions of the guides.

- To view new onscreen help topics for a server application, make sure your server or administrator computer is connected to the Internet and click "Latest help topics" or "Staying current" in the main help page for the application.
- To download the latest guides in PDF format, go to the Mac OS X Server documentation website:  
[www.apple.com/server/documentation](http://www.apple.com/server/documentation)

## Getting Additional Information

For more information, consult these resources:

- *Read Me documents*—important updates and special information. Look for them on the server discs.
- *Mac OS X Server website* ([www.apple.com/server/macosx](http://www.apple.com/server/macosx))—gateway to extensive product and technology information.
- *Mac OS X Server Support website* ([www.apple.com/support/macosxserver](http://www.apple.com/support/macosxserver))—access to hundreds of articles from Apple’s support organization.
- *Apple Training website* ([www.apple.com/training](http://www.apple.com/training))—instructor-led and self-paced courses for honing your server administration skills.
- *Apple Discussions website* ([discussions.apple.com](http://discussions.apple.com))—a way to share questions, knowledge, and advice with other administrators.
- *Apple Mailing Lists website* ([www.lists.apple.com](http://www.lists.apple.com))—subscribe to mailing lists so you can communicate with other administrators using email.



Before you start working on this tutorial, read this chapter to get more information about the tutorial.

This tutorial takes you step by step through creating a Podcast Producer workflow.

Rather than creating the workflow in one shot, this tutorial starts by stepping you through creating a simple Hello World! workflow in the next chapter. Then, in every subsequent chapter, this tutorial shows how to expand the workflow until it is complete.

This incremental approach to building a workflow helps you understand the different aspects of workflow development and shows you the recommended way to build a workflow.

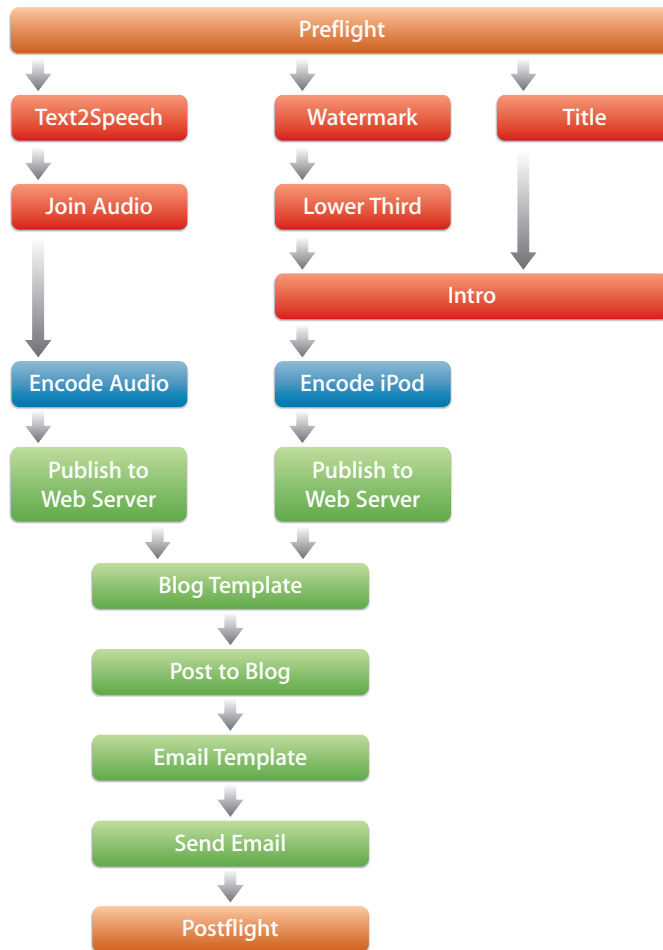
## Before You Start

Before you start working through this tutorial, do the following:

- Set up a Podcast Producer server, as described in *Podcast Producer Administration*.
- Familiarize yourself with workflows and how they work, as described in *Podcast Producer Administration*.
- Verify that your Podcast Producer server is set up properly, by using Podcast Capture on a Mac OS X v10.5 client computer to submit video using some of the default workflows that ship with Podcast Producer.

## The Workflow at a Glance

The following task dependency graph shows you the order of task execution and task dependencies in the workflow you'll be building in this tutorial.



The following is a description of the tasks you'll be adding to the workflow.

Task	Description
Preflight	Runs the <code>preflight</code> script specified in Server Admin > Podcast Producer > Settings > Properties. The goal of this script is to allow you to perform any necessary actions before running the other tasks in the workflow. This task is the <i>start</i> node of your task dependency graph.
Text2Speech	Converts the title of the input movie to speech, which is later added to an audio-only version of the movie.
Watermark	Adds a watermark to the input movie.

Task	Description
Lower Third	Applies the lower third Quartz Composer composition to the watermarked movie.
Title	Renders the title of the submitted movie into a predefined intro movie.
Join Audio	Adds the audio produced by the Text2Speech task to the audio version of the submitted video.
Intro	Adds the intro video containing the title of the movie to the watermarked version of the submitted movie.
Encode Audio	Encodes the audio before it is published.
Encode iPod	Encodes the movie that will be published for playing on iPods.
Publish to Web Server	Publishes the audio and video produced by the workflow into a Web server.
Blog Template	Customizes the template of the blog where the audio and video will be posted.
Post to Blog	Posts the audio and video to a blog.
Email Template	Customizes the template of the email that will be sent to notify users about the availability of the podcasts.
Send Email	Sends notification email.
Postflight	Runs the <code>postflight</code> script specified in Server Admin > Podcast Producer > Settings > Properties. The goal of this script is to allow you to perform any necessary actions after running the main workflow tasks. This task is the <i>end</i> node of your task dependency graph. The default <code>postflight</code> script deletes all temporary files created by the workflow.

## Downloading the Workflow Tutorial Files

You can download the finished workflow bundles and the Lower Third.qtz Quartz composition file used in this tutorial from <http://connect.apple.com>.

### To download the tutorial files:

- 1 Go to <http://connect.apple.com>.
- 2 If you have an Apple Developer Connection (ADC) account, enter your Apple ID and password in the relevant fields and click Login.
- 3 If you don't have an ADC account, click Join Now to create a new account, and then log in.
- 4 Click Downloads.
- 5 In the Downloads list on the right, click Mac OS X Server.
- 6 In the Podcast Producer Workflow Tutorial section, click Podcast Producer Workflow Tutorial (Disk Image) to download the tutorial files.
- 7 Open the disk image and store the files on your hard drive.

The workflow bundles are:

- Workflow 0.pwf
- Workflow 1.pwf
- Workflow 2.pwf
- Workflow 3.pwf
- Workflow 4.pwf
- Workflow 5.pwf
- Workflow 6.pwf
- Workflow 7.pwf
- Workflow 8.pwf
- Workflow 9.pwf
- Workflow 10.pwf
- Workflow 11.pwf
- Workflow Final.pwf

You can use the finished workflows to copy content to the workflows you develop in this tutorial to save time and minimize typing mistakes.

## Additional Sources of Information

For information about Quartz Composer, see:

[http://developer.apple.com/documentation/GraphicsImaging/Conceptual/QuartzComposer/qc\\_intro/chapter\\_1\\_section\\_1.html](http://developer.apple.com/documentation/GraphicsImaging/Conceptual/QuartzComposer/qc_intro/chapter_1_section_1.html)

For information about the concept of bundles in Mac OS X, see:

<http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFBundles/CFBundles.html>

For information about YAML Ain't Markup Language (YAML™), see:

<http://www.yaml.org>

For information about Embedded Ruby (ERB), see:

<http://ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html>



## This chapter provides an overview of workflow templates.

As described in *Podcast Producer Administration*, a workflow template is an Xgrid batch job specification. It tells the Xgrid controller which tasks to execute and the order of their execution.

An Xgrid batch job specification is a property list (plist) formatted in ASCII or XML. Podcast Producer workflows ship in XML format, because it's the most common format. However, in this tutorial you'll use the ASCII format when developing workflows, because it's easier to read.

For more information about plist formats, see <http://developer.apple.com/documentation/Cocoa/Conceptual/PropertyLists/PropertyLists.html>.

The name of the workflow template is always `template.plist` and the template is in the Resources folder of the workflow bundle (for example, `Workflow 1.pwf/Contents/Resources/`).

## Workflow Variables

The template includes workflow variables (strings enclosed between pairs of `$$` characters). The Podcast Producer server replaces these variables with their corresponding values at runtime.

Some of the variables used in workflows represent workflow properties generated at runtime by Podcast Producer. The other variables represent workflow properties whose values you can set using Server Admin.

The values of all variables used in a workflow, whether they represent those properties generated at runtime or the properties defined in Server Admin, are defined in the `properties.plist` file, a temporary file that Podcast Producer stores in the Recording folder for a job submission.

## Workflow Tasks

A workflow task consists of a UNIX shell script and the arguments the script takes, as in the following example, which describes the `preflight` task:

```
preflight = {
  // The UNIX shell script to run
  command = "/usr/bin/pcastaction";
  // The arguments to pass to the script
  arguments = (
    preflight,
    "--basedir=${Base Directory}"
  );
};
```

In this example, the `pcastaction` shell script provided by Podcast Producer runs the `preflight` shell script and passes it an argument specifying the location of the Podcast Producer base directory. The equivalent shell command looks like:

```
/usr/bin/pcastaction preflight --basedir /Volumes/Data/SFS/Recordings/
3D994F4E-549A-41CF-BC62-F8E2DADB90B1
```

In this example, the based directory (represented by `${Base Directory}`) is `/Volumes/Data/SFS/Recordings/3D994F4E-549A-41CF-BC62-F8E2DADB90B1`.

Although you can redefine the task to run the `preflight` script directly, it's better to run it through `pcastaction`. The `pcastaction` script wraps the `preflight` script and knows how to set the relevant environment variables and specify the working directory. In addition, `pcastaction` knows how to translate error codes and correctly pass them to workflow tasks. This is true not only for `preflight`, but also for all the other subcommands of `pcastaction`.

## Contents of a Workflow Template

Each workflow template contains the following entries:

Key	Description
<code>name</code>	Specifies the name of the Xgrid job.
<code>notificationEmail</code>	Specifies the Xgrid job notification email. The Xgrid controller sends notification email to the address specified by this key whenever the job changes state (Running, Failure, or Finished).
<code>artSpecifications</code>	Specifies the arguments that are passed to the Agent Ranking Tool (ART) script used by the Xgrid controller to determine the machines on which the Xgrid job can run.

Key	Description
artConditions	<p>Specifies the conditions for running the job on an Xgrid agent.</p> <p>In the workflows that ship with Mac OS X Server v10.5, the value of this key is <code>{0 = {artEqual = 1; }; }</code>.</p> <p>This condition indicates that the Xgrid job can run only on an Xgrid agent where running the <code>art.rb</code> script (in Workflow 1.pwf/Contents/Resources/Tools/) returns 1.</p> <p>For every condition specified by this key, the number on the left side (in this case 0) must map to one of the arguments specified by the <code>artSpecifications</code> key.</p>
	<p style="text-align: center;">template.plist</p> <pre> ... artSpecifications = {   0 = {     artPath = "\$Workflow Resource Path\$/Tools/art.rb";     artArguments = ("\$\$Shared Filesystem\$\$", "\$\$Server UUID\$\$");   }; }; artConditions = {0 = {artEqual = 1; }; }; ... </pre>
	<p>In most cases, you won't need to change this key. For more information, see <i>Podcast Producer Administration</i>.</p>
taskSpecifications	Specifies the tasks that are executed by the workflow.

## About the art.rb Script

The `art.rb` Ruby script (in Workflow 1.pwf/Contents/Resources/Tools/) determines whether to run a task on an Xgrid agent. This is called Xgrid scoring. The `art.rb` script used in the default Podcast Producer workflows runs tasks on Xgrid agents running Mac OS X v10.5 or Mac OS X Server v10.5.

**Important:** To avoid unexpected results, do not edit the `art.rb` file unless you know how to perform Xgrid scoring. To learn more about Xgrid scoring, see the `xgrid` man page.

The default `art.rb` script contains the following Ruby code:

```
#!/usr/bin/env ruby

ENV['PATH'] = '/bin:/usr/bin'

begin
  # Check for Leopard.
  os_vers = `/usr/bin/sw_vers -productVersion`.chomp
  if (!os_vers.match("10.5"))
    raise "Requires Leopard"
  end
end
```

```

# Check shared filesystem.
shared_filesystem = ARGV[0]
server_uuid = ARGV[1]

if (!shared_filesystem || !server_uuid)
  raise "Usage: art.rb <shared_filesystem> <server_uuid>"
end

if (!File.exist?(shared_filesystem))
  raise "Shared file system does not exist"
end

info_path = File.join(shared_filesystem, "pcastserverd.plist")
if (!File.exist?(info_path))
  raise "#{info_path} does not exist"
end

info = IO.read(info_path)
if (!info.match(server_uuid))
  raise "#{info_path} does not contain #{server_uuid}"
end

puts "{ artScore = 1; }" # Success
rescue => boom
puts "{ artScore = 0; reason = \"#{boom}\"; }" # Failure
end

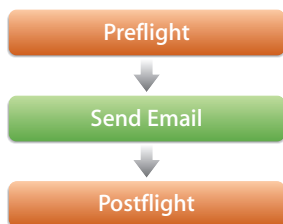
```

In this chapter, you'll learn how to create a simple workflow.

In this chapter, you'll create the Hello World! workflow, which sends email containing the message Hello World! to a list of recipients.

## The Workflow at a Glance

The Hello World! workflow consists of the following tasks:



## Workflow 1 Setup Summary

To create Workflow 1, you'll perform the following tasks:

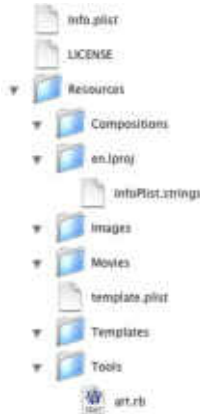
- 1 Create the workflow bundle (page 22).
- 2 Configure localized workflow metadata (see page 22).
- 3 Configure general workflow metadata (page 23).
- 4 Modify license information (page 24).
- 5 Configure the workflow template (page 24).
- 6 Configure workflow properties in Server Admin (page 26).

## Setting Up Workflow 1

### Step 1: Create the Workflow Bundle

The first step when creating a workflow is to create the workflow bundle in which you'll store the workflow files.

The following illustration shows the contents of the workflow bundle you'll be creating.



You can either create the workflow bundle from scratch or copy an existing bundle and modify it to suit your needs. The latter approach is preferable because it is faster.

#### To create the Hello World! bundle:

- 1 If you haven't downloaded the finished workflow bundle, see "Downloading the Workflow Tutorial Files" on page 15 for downloading instructions.
- 2 Make a copy of Workflow 0.pwf.  
This is a starter bundle that you can use to create a new workflow.
- 3 Rename the bundle to Workflow 1.pwf.

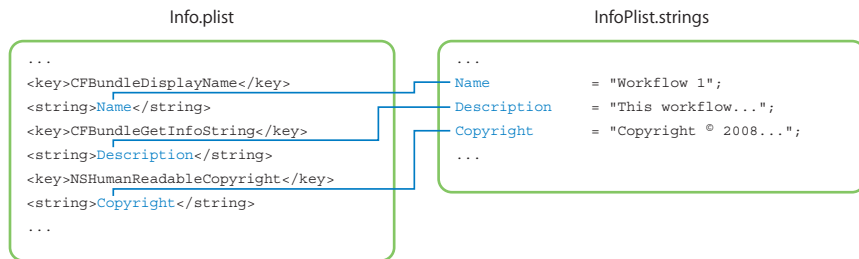
### Step 2: Configure Localized Workflow Metadata

The InfoPlist.strings file (in Workflow 1.pwf/Contents/Resources/en.lproj/) contains the localized versions of keys in the Info.plist file (in Workflow 1.pwf/Contents/).

Each entry in InfoPlist.strings ends with a semicolon and consists of key-value pairs separated by equal signs. For example, the key in the following entry is `Name` and the value is "Workflow 1".

```
Name = "Workflow 1";
```

Each key in InfoPlist.strings must map to a key in Info.plist. For example, as shown below, the `Name` key in InfoPlist.strings maps to the `CFBundleDisplayName` key in Info.plist. The value of the `CFBundleDisplayName` key is equal to the name of the corresponding key in InfoPlist.strings.



In addition to key-value pairs, you can add comments to InfoPlist.strings, which is always recommended. If you add comments, enclose them between the `/*` and `*/` characters. Or use a pair of `//` characters in front of each line of comments.

#### To configure workflow localization information:

- 1 Open InfoPlist.strings.

This file stores the localized versions of keys in the Info.plist file (in Workflow 1.pwf/Contents/).

- 2 Replace the contents of the Info.plist file with the following entries:

```
Copyright = "Copyright © 2008, Apple Inc., All Rights Reserved.";
Name      = "Workflow 1";
Description = "This workflow sends an email.";
```

The values of the three keys appear in Server Admin and Podcast Capture when you request more information about a workflow.

- 3 Save the file.

#### Step 3: Configure General Workflow Metadata

In addition to configuring the localized workflow information, you should also configure the general workflow information, which is stored in the Info.plist file.

#### To configure general workflow information:

- 1 Open Info.plist (in Workflow 1.pwf/Contents/) using Property List Editor.
- 2 Set the value of the `CFBundleName` key to Workflow 1.

The `CFBundleName` key defines the name of the workflow, which appears in Server Admin and Podcast Capture.

- 3 Set the value of the `CFBundleIdentifier` key to `com.apple.PodcastProducer.workflow.Workflow_1`.

- 4 Save the Info.plist file.

#### Step 4: Modify License Information

Because you're using an existing workflow that was provided by Apple, you don't need to modify the licence information contained in the LICENSE file (in Workflow 1.pwf/Contents/).

However, when you create your own workflow, make sure you replace the copyright information contained in the file with your own.

#### Step 5: Configure the Workflow Template

The most important step when creating a workflow is to create the workflow template and define the tasks that the workflow will run.

To configure the workflow template:

- 1 Open the workflow template template.plist.
- 2 Delete the contents of the file.
- 3 Add content to the workflow template.

```
(
  {
    name = "$$Xgrid Job Name$$";

    notificationEmail = "$$Administrator Email Address$$";

    artSpecifications = {
      0 = {
        artPath = "$$Workflow Resource Path$$/Tools/art.rb";
        artArguments = ("$$Shared Filesystem$$", "$$Server UUID$$");
      };
    };

    artConditions = {0 = {artEqual = 1; }; };

    taskSpecifications
      // Define the preflight task.
      // In this case, the workflow uses the default preflight script,
      // which doesn't do anything.
    preflight = {

      // Specify the command to run.
      command = "/usr/bin/pcastaction";

      arguments = (

        // Run the preflight script specified in Server Admin
        // Podcast Producer > Settings > Properties.
        preflight,
```



```

        // All pcastaction commands take the --basedir argument.
        // This argument defines the directory where all the
        // work happens and where temporary files are stored.
        // This directory is located in the Recordings folder on
        // the shared file system.
        "--basedir=$$Base Directory$$"
    );
};

// Define the email task. This task sends an email to the
// specified email address.
email = {
    // The email runs only after the preflight task has
    // successfully completed running. If there is no dependency,
    // the task runs as soon as possible.
    dependsOnTasks = (preflight);
    command = "/usr/bin/pcastaction";
    arguments = (
        // Run the mail subcommand of pcastaction.
        mail,
        "--basedir=$$Base Directory$$",
        // Specify the mail sender.
        "--from=$$User Email Address$$",
        // Specify the mail recipients.
        "--to=$$Audience Email List$$, $$User Email Address$$,
        $$Administrator Email Address$$",
        // Specify the address of the outgoing SMTP mail server.
        "--smtp=$$SMTP Server$$",
        // Set the mail's subject to Hello World!.
        "--body=Subject:Hello World!"
    );
};

// Define the postflight task.
// The default postflight script erases all temporary files.
postflight = {
    dependsOnTasks = (email);
    command = "/usr/bin/pcastaction";
    arguments = (
        postflight,
        "--basedir=$$Base Directory$$"
    );
};

};
}
)

```

#### 4 Save and close the template.plist file.

## Step 6: Configure workflow properties in Server Admin

Before you can deploy the workflow, you need to define the values of the customizable workflow properties used in the workflow template. These are the properties that you can set using Server Admin:

- Audience Email List—A list of SMS recipients of new podcast announcements from Podcast Producer.
- SMTP Server—The host name of an SMTP mail server used by workflows to send mail notifications.

**To define variable values:**

- 1 Open Server Admin.
- 2 In the Computers and Services list, select Podcast Producer.
- 3 Click Settings.
- 4 Click Properties.
- 5 Set the values for the following variables:
  - Audience Email List
  - SMTP Server
- 6 Click Save.

The other variables used in the workflow represent metadata that Podcast Producer generates at run time:

- Administrator Email Address
- Base Directory
- Workflow Resource Path
- Shared Filesystem
- Server UUID
- User Email Address
- Xgrid Job Name

## Validating the Workflow

Podcast Producer provides the `pcastconfig` command located at `/usr/bin/`, which lets you validate workflows before running them.

**To validate Workflow 1:**

- Enter the following command:

```
$ sudo /usr/bin/pcastconfig --validate_workflow_at_path /Library/  
PodcastProducer/Workflows/Workflow\ 1.pwf  
Workflow: /Library/PodcastProducer/Workflows/Workflow 1.pwf is VALID
```

## Deploying the Workflow

To deploy the Hello World! workflow, you must make sure that it is accessible from your test system.

### To deploy the Hello World! workflow:

- 1 Move Workflow 1.pwf to /Library/PodcastProducer/Workflows/ on your server.
- 2 Open Server Admin.
- 3 In the Computers and Services list, select Podcast Producer.
- 4 Click Workflows.
- 5 Select the workflow in the Workflow list.
- 6 Click “Allow access to *workflow name* for all users and groups.”

## Testing the Workflow

After deploying the Hello World! workflow, test it.

### To test the Hello World! workflow:

- 1 On a Mac running Mac OS X v10.5 that’s connected to the same network as your Podcast Producer server, launch Podcast Capture (in /Applications/Utilities/).
- 2 Log in to the Podcast Producer server as pcastuser.
- 3 Select File and choose a very small QuickTime movie.  
  
Even though your workflow doesn’t process QuickTime movies, the only way you can run a workflow is by submitting a recording and submitting audio, video, or screen activity, or by uploading a QuickTime movie for processing.
- 4 From the Workflow pop-up menu, select the Hello World! workflow and give the movie a title and description.
- 5 Click Done.

After the movie is uploaded to the shared file system, Podcast Producer executes the workflow, which in this case doesn’t do anything with the movie.

After a short period of time, the mail recipients you specified in Server Admin should receive a mail message with Hello World! in the subject field. The body of the message should be empty.

If you don’t receive mail notification, go over the instructions in this chapter to make sure that you haven’t missed a step.

**Important:** In this chapter, for simplicity, you didn’t apply the proper formatting to the mail message. In some cases, an improperly formatted mail message might get lost because a mail server won’t be able to parse it correctly. Later in this tutorial, you’ll learn the proper way to format mail messages.

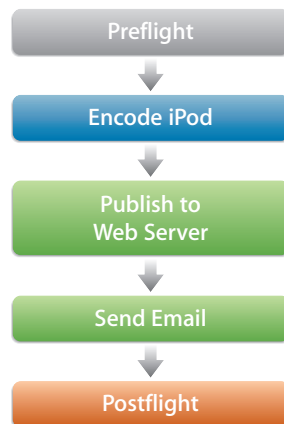


## Create a workflow that encodes and publishes media.

In this chapter, you'll expand the Hello World! workflow by adding two tasks to it to encode and publish submitted media.

### The Workflow at a Glance

The Encoding Media workflow consists of the following tasks:



In this chapter, you'll expand on Workflow 1 by:

- Adding the following tasks:
  - Encode iPod
  - Publish to Web Server
- Modifying the Send Email task

## Workflow 2 Setup Summary

Having successfully deployed and tested the Hello World! workflow, you can use it as the starting point for the workflow you'll create in this chapter. The new workflow expands on the Hello World! workflow by adding two new tasks.

To edit property lists, use your favorite text editor or Property List Editor.

To create Workflow 2, you'll perform the following tasks:

- 1 Create the workflow bundle (page 30).
- 2 Configure the workflow metadata (see page 30).
- 3 Configure the workflow template (page 31).
- 4 Configure workflow properties in Server Admin (page 33).

## Setting Up Workflow 2

### Step 1: Create the Workflow Bundle

- Make a copy of /Library/PodcastProducer/Workflows/Workflow 1.pwf and change the name to Workflow 2.pwf.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

#### To configure the workflow metadata:

- 1 Open the Workflow 2.pwf bundle (by Control-clicking the bundle and choosing Show Package Contents).
- 2 Open Info.plist in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_2`.
  - Set `CFBundleName` to `Workflow 2`.
- 3 Open InfoPlist.strings and set the Name and Description keys as follows:

```
Name = "Workflow 2";  
Description = "Here we added encoding the input video to the iPod format.";
```

### Step 3: Configure the Workflow Template

- 1 Open the workflow template `template.plist`.
- 2 In the `taskSpecifications` section of the workflow template, add the `encode_ipod` task after the `preflight` task:

```
// Take input movie and transcode it to an iPod Video format.
encode_ipod = {
    // This task runs as soon as the preflight script finishes running.
    dependsOnTasks = (preflight);

    command = "/usr/bin/pcastaction";
    arguments = (
        // Run the encode subcommand of pcastaction.
        // For more information, run "pcastaction help encode."
        encode,
        "--basedir=$$Base Directory$$",

        // Path to input file (relative to the Base Directory)
        "--input=$$Content File Basename$$$$Content File Extension$$",

        // Path to output file (relative to the Base Directory)
        "--output=$$Content File Basename$$-ipod.m4v",

        // Specify the encoding setting to use (iPod format).
        "--encoder=ipod"
        );
    };
```

You can place the task anywhere in the `taskSpecifications` section. The Xgrid controller determines the execution order based on task dependency.

- 3 In the `taskSpecifications` section of the workflow template, add the `publish_ipod` task after the `encode_ipod` task:

```
// Copy the encoded video file produced by encode_ipod to the Web server.
publish_ipod = {
    // This task runs right after encode_ipod finishes running.
    dependsOnTasks = (encode_ipod);
    command = "/usr/bin/pcastaction";
    arguments = (
        // Run the publish subcommand of pcastaction.
        // For more information, run "pcastaction help publish."
        publish,
        "--basedir=$$Base Directory$$",

        // Web Doc Root
        "--web_root=$$Web Document Root$$",

        // Web URL
        "--web_url=$$Web URL$$",

        // Date of posting
    );
```

```

        "--date=${Date_YYYY-MM-DD}",

        // Podcast title
        "--title=${Title}",

        // String representing the published format
        "--format=ipod",

        // MIME type of movie being published
        "--type=video/mp4",

        // File to be pushed to the Web Server (copied to Web Doc Root)
        "--file=${Content File Basename}-ipod.m4v",

        // The YAML file containing information about published podcast
        "--outfile=ipod_publish_description_file.yaml"
    );
};

```

The `publish_ipod` task writes out a YAML file containing information about the published file, including the full URL to the content. For more information about YAML, see [www.yaml.org](http://www.yaml.org).

#### 4 Modify the `email` task as follows:

```

email = {
    // Change dependency. Now, this task executes after publish_ipod is done.
    dependsOnTasks = (publish_ipod);

    command = "/usr/bin/pcastaction";
    arguments = (
        mail,
        "--basedir=${Base Directory}",
        "--from=${User Email Address}",
        "--to=${Audience Email List}, ${User Email Address},
            ${Administrator Email Address}",
        "--smtp=${SMTP Server}",

        // Specify message's contents.
        "--body_file=ipod_publish_description_file.yaml"
    );
};

```

For now, just mail the contents of the YAML file. However, later in this guide, you'll learn how to properly format mail messages. Mail messages must be properly escaped to be valid. If not, some mail servers might not process the message, which causes the task to fail.

#### 5 Save and close the `template.plist` file.



#### Step 4: Configure workflow properties in Server Admin

Before you can deploy the workflow, define the values of the workflow properties used in the workflow template. For Workflow 2, you need to define the values of the `$$Web Document Root$$` and `$$Web URL$$` variables.

##### To define variable values:

- 1 In Server Admin, set values for the following workflow properties:
  - Web Document Root—The web server root folder on the shared file system where new podcast streams are published.
  - Web URL—The base URL of the web server where podcasts are published.
- 2 Click Save.

Podcast Producer gets the values of other `$$` variables from the metadata stored in the `properties.plist` file.

## Getting Information About the `pcastaction` Command

To get information about the `pcastaction` command, see its man page:

```
$ man pcastaction
```

```
...
```

```
SYNOPSIS
```

```
pcastaction command [options] [args]
```

```
OVERVIEW
```

```
The pcastaction tool is a tool built to facilitate writing Podcast Producer workflows. It wraps a large set of commands needed for creating video or audio podcasts. Actions include annotating QuickTime movies, video editing, sending emails, posting blog entries.
```

```
The available subcommands are:
```

```
unpack
shell
preflight
postflight
encode
annotate
qceffect
watermark
title
merge
iTunes
iTunesU
mail
archive
publish
groupblog
template
```

```
approval
```

Run ``pcastaction help`` to access the built-in tool documentation. You can also run ``pcastaction <subcommand> help`` for more information about the different commands.

To get information about `pcastaction` subcommands, use the following command:

```
pcastaction help subcommand
```

For example, to display information about the `encode` subcommand, enter:

```
$ pcastaction help encode
```

```
...
```

```
Podcast Producer action task, version 0.1
```

```
...
```

```
encode: transforms the input file to the output file with the specified
encoder usage: encode --basedir=BASEDIR --input=INPUT --output=OUTPUT
--encoder=ENCODER
```

```
the available encoders are:
```

```
3gpp2_ezmovie
3gpp2_release_A
3gpp2_release_A_hint
3gpp2_release_A_hint_server
3gpp_mobile_mp4
3gpp_mobile_mp4_hint
3gpp_mobile_mp4_hint_server
3gpp_release_5
3gpp_release_5_hint
3gpp_release_5_hint_server
amc_ezmovie
appletv
avi
avi_1cd
avi_2cd
h264
h264_hint
h264_hint_server
iphone
iphone_cellular
ipod
mp4_audio_high
mp4_audio_low
mp4_high
mp4_lan
mp4_low
mp4_low_stream
mp4_medium
```

mp4\_medium\_stream

## Deploying the Workflow

Validate, deploy, and test Workflow 2 to make sure it runs successfully, and then proceed to the next chapter.

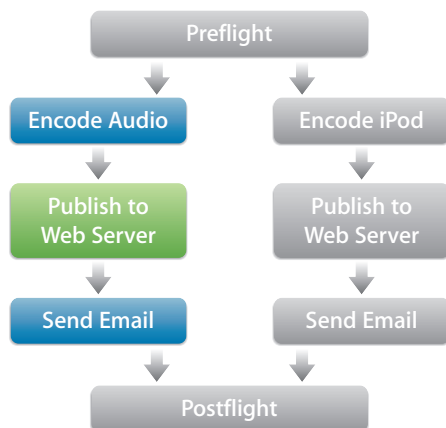


## Create a workflow that encodes and publishes audio and video in parallel.

In this chapter, you'll create a workflow based on the workflow from the last chapter. You'll add to it three tasks that run in parallel with similar tasks already in the workflow.

### The Workflow at a Glance

The Running Tasks in Parallel workflow consists of the following tasks:



The three new tasks produce an audio-only version of the submitted QuickTime movie, publish the resulting audio podcast to the web, and send notification email.

## Workflow 3 Setup Summary

In the previous chapter you created Workflow 2, which encodes the submitted video, publishes the video on a web server, and sends notification email.

Workflow 2, like Workflow 1 (the Hello World! workflow), defines a list of tasks to run sequentially. However, one of the powerful features of workflow templates or Xgrid jobs is the ability to run tasks in parallel by distributing tasks among Xgrid agents if enough cores are available. Running tasks in parallel reduces the amount of time needed to run workflows and publish podcasts.

To create Workflow 3, you'll perform the following tasks:

- 1 Create the workflow bundle (page 38).
- 2 Configure the workflow metadata (see page 38).
- 3 Configure the workflow template (page 39).

## Setting Up Workflow 3

### Step 1: Create the Workflow Bundle

- Make a copy of `/Library/PodcastProducer/Workflows/Workflow 2.pwf` and change the name to `Workflow 3.pwf`.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

To configure the workflow metadata:

- 1 Open the `Workflow 3.pwf` bundle.
- 2 Open `Info.plist` in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_3`.
  - Set `CFBundleName` to `Workflow 3`.
- 3 Open `InfoPlist.strings` and set the `Name` and `Description` keys as follows:

```
Name = "Workflow 3";  
Description = "Here we added encoding the video to an audio-only format.";
```

### Step 3: Configure the Workflow Template

- 1 Open the workflow template `template.plist`.
- 2 In the `taskSpecifications` section of the workflow template, modify the `email` task as follows:

```
// Change the name of the task.
email_ipod = {
    dependsOnTasks = (publish_ipod);
    command = "/usr/bin/pcastaction";
    arguments = (
        mail,
        "--basedir=$$Base Directory$$",
        "--from=$$User Email Address$$",
        "--to=$$Audience Email List$$, $$User Email Address$$,
            $$Administrator Email Address$$",
        "--smtp=$$SMTP Server$$",
        "--body_file=ipod_publish_description_file.yaml"
    );
};
```

- 3 In the `taskSpecifications` section of the workflow template, add the `encode_audio` task after the `email_ipod` task:

```
// Take the input movie and transcode it to an audio-only format.
encode_audio = {
    // This task, like encode_ipod, also depends on the preflight script.
    // If your Xgrid has enough open CPUs, it can schedule this encoding in
    // parallel with the encode_ipod task.
    dependsOnTasks = (preflight);

    command = "/usr/bin/pcastaction";
    arguments = (
        // The arguments here are very similar to those of the encode_ipod
        // task, except for the encoding format (mp4_audio_high).
        encode,

        "--basedir=$$Base Directory$$",
        "--input=$$Content File Basename$$$$Content File Extension$$",
        "--output=$$Content File Basename$$-audio.m4a",

        // The encoding setting to use (MPEG-4 high quality audio format)
        "--encoder=mp4_audio_high"
    );
};
```

- 4 In the `taskSpecifications` section of the workflow template, add the `publish_audio` task after the `encode_audio` task:

```
// Copy the encoded audio file produced by encode_audio to the Web server.
publish_audio = {
  dependsOnTasks = (encode_audio);
  command = "/usr/bin/pcastaction";
  arguments = (
    publish,
    "--basedir=$$Base Directory$$",
    "--web_root=$$Web Document Root$$",
    "--web_url=$$Web URL$$",
    "--date=$$Date_YYYY-MM-DD$$",
    "--title=$$Title$$",
    // String representing the published format
    "--format=audio",
    // MIME type of audio being published
    "--type=audio/mp4a-latm",
    // File to be pushed to the Web Server (copied to Web Doc Root)
    "--file=$$Content File Basename$$-audio.m4a",
    // The YAML file containing information about published podcast
    "--outfile=audio_publish_description_file.yaml"
  );
};
```

- 5 In the `taskSpecifications` section of the workflow template, add the `email_audio` task after the `publish_audio` task:

```
// Send a mail notification when the audio podcast is available online.
email_audio = {
  dependsOnTasks = (publish_audio);
  command = "/usr/bin/pcastaction";
  arguments = (
    mail,
    "--basedir=$$Base Directory$$",
    "--from=$$User Email Address$$",
    "--to=$$Audience Email List$$, $$User Email Address$$,
      $$Administrator Email Address$$",
    "--smtp=$$SMTP Server$$",

    // Specify the message's contents.
    "--body_file=audio_publish_description_file.yaml"
  );
};
```



- 6 In the `taskSpecifications` section of the workflow template, modify the `postflight` task as follows:

```
postflight = {  
    // This task now depends on both mailing tasks  
    dependsOnTasks = (email_audio, email_ipod);  
    command = "/usr/bin/pcastaction";  
    arguments = (  
        postflight,  
        "--basedir=${Base Directory}"  
    );  
};
```

- 7 Save and close the `template.plist` file.

## Deploying the Workflow

Validate, deploy, and test Workflow 3 to make sure it runs without a problem, and then proceed to the next chapter.



## Create a workflow that properly formats emails.

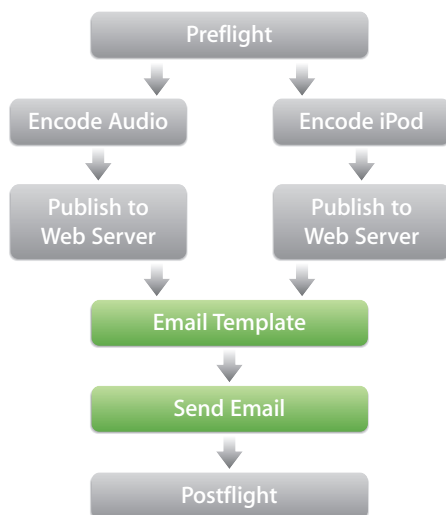
In the previous three chapters, for simplicity, you sent email notifications without proper formatting. Now you'll learn how to use the `template` subcommand of `pcastaction` to properly format mail messages, by creating a new workflow based on Workflow 3.

**Important:** An incorrectly formatted message can get lost or end up in the wrong mailbox.

You'll also learn how to send one mail message with links to the podcasts, instead of sending one message for every produced media file as you did in the previous chapter.

## The Workflow at a Glance

The Mail Template workflow (Workflow 4) consists of the following tasks:



In this workflow, you'll create two new tasks:

- Email Template—Applies the correct formatting to the mail message
- Send Email—Sends one message to every recipient

## Workflow 4 Setup Summary

Workflow 4 is almost identical to Workflow 3, except for the new Email Template task and the consolidation of the two mail tasks into one task.

To create Workflow 4, you'll perform the following tasks:

- 1 Create the workflow bundle (page 44).
- 2 Configure the workflow metadata (see page 44).
- 3 Configure the workflow template (page 45).
- 4 Write the mail template (page 46).

## Setting Up Workflow 4

### Step 1: Create the Workflow Bundle

- Make a copy of /Library/PodcastProducer/Workflows/Workflow 3.pwf and change the name to Workflow4.pwf.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

**To configure the workflow metadata:**

- 1 Open the Workflow 4.pwf bundle.
- 2 Open Info.plist in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_4`.
  - Set `CFBundleName` to `Workflow 4`.
- 3 Open InfoPlist.strings and set the Name and Description keys as follows:

```
Name = "Workflow 4";  
Description = "Here we use an email template to send one email with the  
information gathered from the previous tasks.";
```

### Step 3: Configure the Workflow Template

- 1 Open the workflow template template.plist.
- 2 In the `taskSpecifications` section of the workflow template, add the following task after the `publish_audio` task.

```
// Generate a mail body text file from the template ERB file (mail.txt.erb).
// This task interprets the ERB file in order to generate the mail.txt file.
template_mail = {
  dependsOnTasks = (publish_audio, publish_ipod);
  command = "/usr/bin/pcastaction";
  arguments = (
    // Run the template subcommand of pcastaction
    template,
    "--basedir=$$Base Directory$$",
    // Path to ERB template file to be interpreted
    "--template=$$Workflow Resource Path$$/Templates/mail.txt.erb",
    // Path where the resulting mail file will be written
    "--output=mail.txt"
  );
};
```

- 3 In the `taskSpecifications` section of the workflow template, add the following task after the `template_mail` task:

```
// The new email task is very similar to the previous ones except for its
// dependency and that we now send the generated body file
email = {
  dependsOnTasks = (template_mail);
  command = "/usr/bin/pcastaction";
  arguments = (
    mail,
    "--basedir=$$Base Directory$$",
    "--from=$$User Email Address$$",
    "--to=$$Audience Email List$$, $$User Email Address$$, $$Administrator
        Email Address$$",
    "--smtp=$$SMTP Server$$",
    // Send the mail.txt file generated by the template file
    "--body_file=mail.txt"
  );
};
```

- 4 In the `taskSpecifications` section of the workflow template, delete the `email_ipod` and `email_audio` tasks.

- 5 In the `taskSpecifications` section of the workflow template, modify the `postflight` task as follows:

```
postflight = {
  // This task now depends on both mailing tasks
  dependsOnTasks = (email);
  command = "/usr/bin/pcastaction";
  arguments = (
    postflight,
    "--basedir=${Base Directory}"
  );
};
```

- 6 Save and close the `template.plist` file.

#### Step 4: Write the Mail Template

To properly format mail messages, you write an Embedded Ruby (ERB) template file, which generates a formatted mail message that you can send out to recipients.

After you write the file, you'll add a task that takes this file as input and formats the mail notification message using the embedded Ruby commands.

In an ERB file:

- Lines starting with `##` are comments.
- Lines starting with `%` are Ruby code.
- A Ruby expression is enclosed between `<%=` and `%>`.

For more information about ERB, see:

<http://ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html>

When writing the mail template, you'll make calls to helper methods defined in the `pcastaction` Ruby script located in `/usr/bin/`. Before you start writing the mail template, open `pcastaction` in your favorite text editor and take a look at the `ERBContext` module, where the helper methods are defined.

The `ERBContext` module exposes helper methods that you can call from outside of the script. In the mail template you create, you'll use the following helper methods:

Method	Description
<code>quotify</code>	Adds quotation marks around the input string: <pre>def self.quotify(string)   "'" + string + "'" end</pre>
<code>braketify</code>	Adds angle brackets around the input string: <pre>def self.braketify(string)   '&lt;' + string + '&gt;' end</pre>

Method	Description
emailify	Creates a properly formatted mail message from the input strings: <pre>def self.emailify(name, address)   quotify(name) + ' ' + bracketify(address) end</pre>
localized_time	Produces a time value formatted based on you locale: <pre>def self.localized_time(seconds_since_epoch)   prepare_for_localizing unless @@time_formatter   @@time_formatter.stringFromDate(OSX::NSDate.d ateWithTimeIntervalSince1970(seconds_since_ep och)).to_s end</pre>
localized_date	Produces a date formatted based on you locale: <pre>def self.localized_date(seconds_since_epoch)   prepare_for_localizing unless @@date_formatter   @@date_formatter.stringFromDate (OSX::NSDate.dateWithTimeIntervalSince1970 (seconds_since_epoch)).to_s end</pre>

In addition to using helper methods, you can access workflow properties stored by Podcast Producer from the mail script using the `properties` keyword.

For example, the following code snippet returns the full name of the Podcast Producer administrator:

```
properties["Administrator Email Address"]
```

### To write the mail template:

- 1 Using your favorite text editor, create the `mail.txt.erb` file and save it in the workflow's Templates folder (Workflow 4.pwf/Contents/Resources/Templates/).
- 2 Define variables by adding the following lines of Ruby code to `mail.txt.erb`:

```
% administrator_full_name = (properties["Administrator Full Name"] || "")
% administrator_email_address = (properties["Administrator Email Address"]
  || "")
% audience_email_address = (properties["Audience Email List"] || "")
% user_full_name = (properties["User Full Name"] || "")
% user_email_address = (properties["User Email Address"] || "")
% title = (properties["Title"] || "")
% seconds_since_epoch = (properties["Recording Started At"] || "").to_i
```

These lines of code use helper methods defined in the `template` subcommand of `pcastaction` to retrieve property values from the `properties.plist` file. This is the file containing all the metadata describing a workflow submission.

After receiving a workflow job submission, Podcast Producer generates the `properties.plist` file and stores it in the recording folder it created for the workflow submission.

- 3 Add the following lines of Ruby code, which use the YAML library to read the YAML files generated by the publishing tasks:

```
% ipod_file_url =
  ((YAML::load_file("ipod_publish_description_file.yaml")[:url] rescue
   nil) || "")
% audio_file_url =
  ((YAML::load_file("audio_publish_description_file.yaml")[:url] rescue
   nil) || "")
```

- 4 Add the following lines of Ruby code, which use helper methods defined in the ERBContext of `pcastaction` to correctly format the different values needed to construct the email's header by adding the relevant escape characters:

```
% from_addresses = [emailify(administrator_full_name,
  administrator_email_address)]
% subject_value = "New Episode: #{quotify(title)}"
% to_addresses = [audience_email_address, emailify(user_full_name,
  user_email_address), emailify(administrator_full_name,
  administrator_email_address)]
%
<%= encoded_header_with_addresses("From", from_addresses) %>
<%= encoded_header_with_value("Subject", subject_value) %>
<%= encoded_header_with_addresses("To", to_addresses) %>
%
```

The subject value specified in this template (`subject_value`) appears in the Subject field.

- 5 Add the body of the mail message:

```
A new Podcast episode has been posted!

<%= quotify(title) %> was recorded at
  <%= localized_time(seconds_since_epoch) %> on
  <%= localized_date(seconds_since_epoch) %>.

Download the episode in the following formats:
iPod video: <%= braketify(ipod_file_url) %>
iPod audio: <%= braketify(audio_file_url) %>
```

This email was sent by Podcast Producer.

- 6 Save and close the `mail.txt.erb` file.



## Deploying the Workflow

Validate, deploy, and test Workflow 4 to make sure it runs successfully, and then proceed to the next chapter.

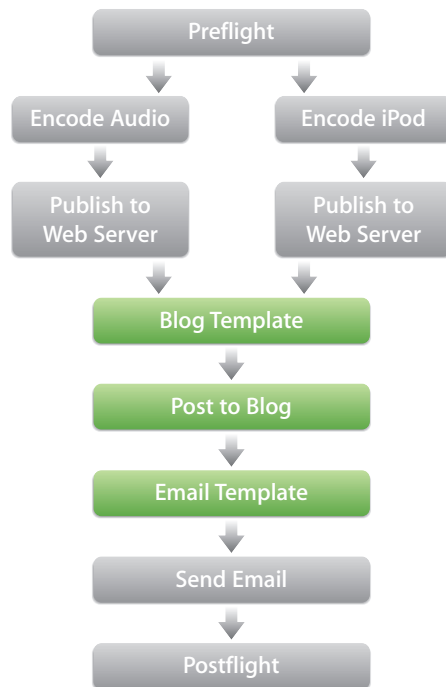


## Add to the workflow the ability to post to a group blog.

In this chapter you'll modify the mail template to include a link to the blog posting and add a task to the workflow to post to the blog.

### The Workflow at a Glance

The Group Blog workflow (Workflow 5) consists of the following tasks:



The Blog Template and Post to Blog tasks publish the podcasts to a blog. The Email Template task sends notification email that includes a link to the blog posting.

## Workflow 5 Setup Summary

Workflow 5 builds on Workflow 4 by adding the tasks that make it possible to post processed content to a blog and create an RSS feed.

To create Workflow 5, you'll perform the following tasks:

- 1 Create the workflow bundle (page 52).
- 2 Configure the workflow metadata (see page 52).
- 3 Configure the workflow template (page 52).
- 4 Configure workflow properties in Server Admin (page 55).

## Setting Up Workflow 5

### Step 1: Create the Workflow Bundle

- Make a copy of `/Library/PodcastProducer/Workflows/Workflow 4.pwf` and change the name to `Workflow 5.pwf`.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

To configure the workflow metadata:

- 1 Open the `Workflow 5.pwf` bundle.
- 2 Open `Info.plist` in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_5`.
  - Set `CFBundleName` to `Workflow 5`.
- 3 Open `InfoPlist.strings` and set the `Name` and `Description` keys as follows:

```
Name = "Workflow 5";
Description = "Here we added posting to the group's weblog (we are also
              using a template for the posting).";
```

### Step 3: Configure the Workflow Template

- 1 Open the workflow template `template.plist`.
- 2 In the `taskSpecifications` section of the workflow template, modify the `publish_ipod` task as follows:

```
publish_ipod = {
    dependsOnTasks = (encode_ipod);
    command = "/usr/bin/pcastaction";
    arguments = (
```

```

publish,
"--basedir= $$Base Directory$$",
"--web_root= $$Web Document Root$$",
"--web_url= $$Web URL$$",
"--date= $$Date_YYYY-MM-DD$$",
"--title= $$Title$$",
"--format=ipod",
"--type=video/mp4",
"--file= $$Content File Basename$$-ipod.m4v",
"--outfile=ipod_publish_description_file.yaml",

// Create a poster image of the video
"--create_poster_image"
);
};

```

- 3 In the `taskSpecifications` section of the workflow template, add the `template_groupblog` task after the `publish_audio` task:

```

template_groupblog = {
  dependsOnTasks = (publish_ipod, publish_audio);
  command = "/usr/bin/pcastaction";
  arguments = (
    template,
    --basedir= $$Base Directory$$,
    // Path to ERB template file to be interpreted
    --template= $$Workflow Resource Path$$/Templates/groupblog.html.erb,
    // Path to where the resulting file is written, relative to basedir.
    --output=groupblog.html
  );
};

```

- 4 In the `taskSpecifications` section of the workflow template, add the `groupblog` task after the `publish_audio` task.

```

// Take the HTML file generated by template_groupblog and post an entry to
// the Group's blog.
groupblog = {
  dependsOnTasks = (template_groupblog);
  command = "/usr/bin/pcastaction";
  arguments = (
    // Use the groupblog subcommand of pcastaction.
    groupblog,
    --basedir= $$Base Directory$$,

    // URL of server hosting the Group blog
    --server_url= $$Groups Web Server URL$$,
    --group= $$Group Short Name$$, // Group's shortname

    // Shortname of group administrator
    --username= $$Groups Administrator Username$$,

```

```

// URL of Podcast Producer server
"--pcast_server=${Podcast Producer URL}$",

// One time password for credentials to post to blog
"--otp=##Groups Administrator Username:Groups Administrator
Password##",
// The shortname of the user account that the group administrator
// will use for posting content.
"--authorize=${User Short Name}$",

"--title=${Title}$",

// HTML file generated by the ERB template
"--content_file=groupblog.html",

"--enclosure_description_file=ipod_publish_description_file.yaml",
"--enclosure_description_file=audio_publish_description_file.yaml",

// Generate an outfile (group_blog_entry_url.txt if nothing is
// specified)
"--outfile"
);
};

```

- 5 In the `taskSpecifications` section of the workflow template, modify the dependency of the `template_mail` task as follows:

```

template_mail = {

// Because the mail notification will includes a line to the blog,
// this task should run only after the blog is posted.
dependsOnTasks = (groupblog);

command = "/usr/bin/pcastaction";
arguments = (
    template,
    "--basedir=${Base Directory}$",
    "--template=${Workflow Resource Path$}/Templates/mail.txt.erb",
    "--output=mail.txt"
);
};

```

- 6 Save and close the `template.plist` file.

#### Step 4: Configure workflow properties in Server Admin

Before you can deploy the workflow, you need to define the values of the workflow properties you specified in the workflow template.

##### To define variable values:

- 1 In Server Admin, set the values for the following workflow variables:
  - Groups Web Server URL—The base URL of the web server where group blog announcements about new podcasts are published.
  - Group Short Name—The short name of the group that receives blog announcements about new podcasts from Podcast Producer.
  - Groups Administrator Username—The short name of the administrator user who is the group's owner and can post blogs on behalf of normal users. This user must also be an administrator on the system running the Web service used by Podcast Producer to post blogs.
- 2 Click Save.

## Writing a Blog Template

Like the mail template you created in the previous chapter, the blog template you're about to create is an ERB file in which you can embed Ruby code.

The blog template will define variables whose values are other variables and methods defined in the `pcastaction` Ruby script.

##### To write the blog template:

- 1 Using your favorite text editor, create the `groupblog.html.erb` file and save it in the workflow's Templates folder (`Workflow 5.pwf/Contents/Resources/Templates/`).
- 2 Define the following variables by adding the following lines of Ruby code to `groupblog.html.erb`:

```
% description = (properties["Description"] || "")
% ipod_publish = ((YAML::load_file("ipod_publish_description_file.yaml")
  rescue nil) || {})
% ipod_publish_poster_url = (ipod_publish[:poster_url] || "")
% ipod_publish_url = (ipod_publish[:url] || "")
%
```

- 3 Add the following lines of Ruby code, which use the helper methods defined in the `pcastaction` script, to define the posting dimensions of the video on the blog (the maximum width is 480 pixels):

```
% ipod_publish_width = (ipod_publish[:width].to_i <= 480 ?
  ipod_publish[:width].to_i : 480).to_s
% ipod_publish_height = (ipod_publish[:width].to_i <= 480 ?
  ipod_publish[:height].to_i : (480.0/
  ipod_publish[:width].to_f*ipod_publish[:height].to_f).to_i).to_s
%
```

- 4 Add the following lines of Ruby code, which use the helper methods defined in the `pcastaction` script, to define the HTML block that will be posted to the blog:

```
<%= h(description) %>
<br>
<br>
<img src=<%= quotify(ipod_publish_poster_url) %> alt=<%=
  quotify(ipod_publish_url) %> width=<%= quotify(ipod_publish_width) %>
  height=<%= quotify(ipod_publish_height) %> class="aligncenter
  posterimg" />
```

- 5 Save and close the `groupblog.html.erb` file.

## Updating the Mail Template

Update the `mail.txt.erb` file you created in the last chapter, as follows:

```
%
% administrator_full_name = (properties["Administrator Full Name"] || "")
% administrator_email_address = (properties["Administrator Email Address"]
  || "")
% audience_email_address = (properties["Audience Email List"] || "")
% user_full_name = (properties["User Full Name"] || "")
% user_email_address = (properties["User Email Address"] || "")
% title = (properties["Title"] || "")
% seconds_since_epoch = (properties["Recording Started At"] || "").to_i
%
%# Define variables for use in group blog posting and RSS feed generation.
%# When the workflow posts to the blog, the blog automatically generates an
%# RSS feed.
%
% group_short_name = (properties["Group Short Name"] || "")
% group_full_name = (properties["Group Full Name"] || "")
%
%# The groupblog task generates a text file containing the URL of the post
% group_blog_entry_url = ((IO.read("group_blog_entry_url.txt").chomp rescue
  nil) || "")
%
% groups_web_server_url = (properties["Groups Web Server URL"] || "")
% groups_blog_url = groups_web_server_url + '/' + group_short_name + '/blog/
%
% blog_feed_url = groups_blog_url.gsub(/^http/, 'itpc') + 'index.its'
% ipod_blog_feed_url = blog_feed_url + "?mediaTypeTag=ipod"
% audio_blog_feed_url = blog_feed_url + "?mediaTypeTag=audio"
%
% ipod_file_url =
  ((YAML::load_file("ipod_publish_description_file.yaml")[:url] rescue
  nil) || "")
% audio_file_url =
  ((YAML::load_file("audio_publish_description_file.yaml")[:url] rescue
  nil) || "")
```



```

% from_addresses = [emailify(administrator_full_name,
    administrator_email_address)]
%
%# Modify the mail's subject to contain a reference to the group name.
% subject_value = "New Episode in #{group_full_name} Wiki group:
    #{quotify(title)}"
%
% to_addresses = [audience_email_address, emailify(user_full_name,
    user_email_address), emailify(administrator_full_name,
    administrator_email_address)]
%
<%= encoded_header_with_addresses("From", from_addresses) %>
<%= encoded_header_with_value("Subject", subject_value) %>
<%= encoded_header_with_addresses("To", to_addresses) %>
New Podcast Episode in <%= group_full_name %> Wiki group

<%= quotify(title) %> was recorded at <%=
    localized_time(seconds_since_epoch) %> on <%=
    localized_date(seconds_since_epoch) %>:
<%= braketify(group_blog_entry_url) %>

%# Add a link to the blog
This episode has been published to the <%= group_full_name %> blog:
<%= braketify(groups_blog_url) %>

%# Add links to the RSS feeds to allow recipients to directly subscribe
%# to the feed from the mail message.
Subscribe in iTunes:
iPod video: <%= braketify(ipod_blog_feed_url) %>
iPod audio: <%= braketify(audio_blog_feed_url) %>

Download the episode in the following formats:
iPod video: <%= braketify(ipod_file_url) %>
iPod audio: <%= braketify(audio_file_url) %>

This email was sent by Podcast Producer.

```

## Deploying the Workflow

Deploy Workflow 5 and test it to make sure it runs without a problem, and then proceed to the next chapter.

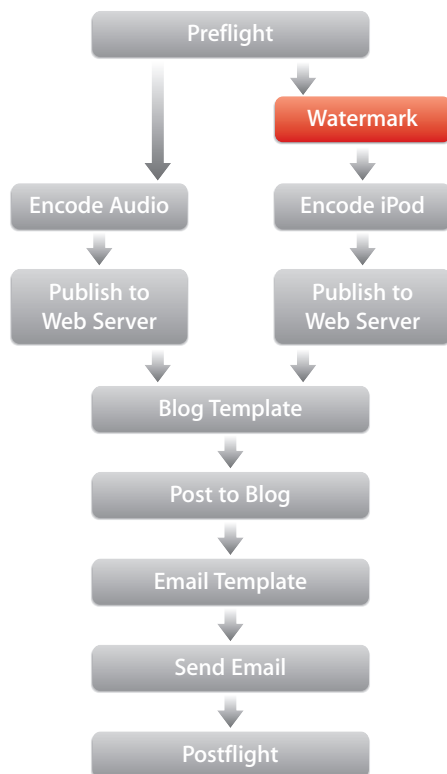


Add a task that superimposes a watermark on the input movie.

To add a watermark, you'll use the `watermark` subcommand of `pcastaction`.

## The Workflow at a Glance

The Watermarking workflow (Workflow 6) consists of the following tasks:



## Workflow 6 Setup Summary

Workflow 6 builds on Workflow 5 by adding a task that adds a watermark to the submitted video, and by modifying the dependency of the `encode_ipod` task that processes the watermarked video.

To create Workflow 6, you'll perform the following tasks:

- 1 Create the workflow bundle (page 60).
- 2 Configure the workflow metadata (see page 60).
- 3 Create a watermark (see page 60).
- 4 Configure the workflow template (page 60).

## Setting Up Workflow 6

### Step 1: Create the Workflow Bundle

- Make a copy of `/Library/PodcastProducer/Workflows/Workflow 5.pwf` and change the name to `Workflow 6.pwf`.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

To configure the workflow metadata:

- 1 Open the `Workflow 6.pwf` bundle.
- 2 Open `Info.plist` in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_6`.
  - Set `CFBundleName` to `Workflow 6`.
- 3 Open `InfoPlist.strings` and set the `Name` and `Description` keys as follows:

```
Name = "Workflow 6";  
Description = "Here we added the watermarking of the video.";
```

### Step 3: Create a Watermark

- Create a watermark and save it as `Watermark.png` in `Workflow 6.pwf/Contents/Resources/Images/`.

Alternatively, use the `Watermark.png` file provided with the finished `Workflow 6.pwf` bundle.

### Step 4: Configure the Workflow Template

- 1 Open the workflow template `template.plist`.

- 2 In the `taskSpecifications` section of the workflow template, add the `watermark` task after the `preflight` task:

```
// Water-mark the input video
watermark = {
  dependsOnTasks = (preflight);
  command = "/usr/bin/pcastaction";
  arguments = (
    watermark,
    "--basedir=$$Base Directory$$",

    // Path to the watermark image.
    "--watermark=$$Workflow Resource Path$$/Images/Watermark.png",

    // Path to the input QuickTime movie file.
    "--input=$$Content File Basename$$$$Content File Extension$$",

    // Path where the output QT reference movie will be written.
    // This is a reference movie and not a flat movie file.
    "--output=$$Content File Basename$$-watermarked.mov"
  );
};
```

- 3 In the `taskSpecifications` section of the workflow template, modify the `encode_ipod` task as follows:

```
encode_ipod = {
  // Wait for the watermark task to complete before running
  dependsOnTasks = (watermark);
  command = "/usr/bin/pcastaction";
  arguments = (
    encode,
    "--basedir=$$Base Directory$$",

    // The input movie is now the watermarked video.
    "--input=$$Content File Basename$$-watermarked.mov",

    "--output=$$Content File Basename$$-ipod.m4v",
    "--encoder=ipod"
  );
};
```

- 4 Save and close the template.plist file.

## Deploying the Workflow

Deploy Workflow 6 and test it to make sure it runs without a problem, and then proceed to the next chapter.

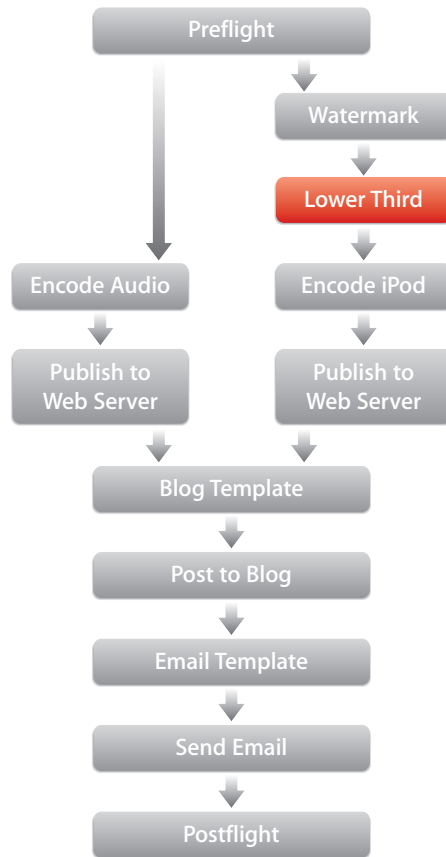


## Add the Lower Third Quartz Composer effect to the watermarked version of the submitted movie.

To add a Quartz Composer effect, you'll define it in Quartz Composer. For your convenience, workflow tutorial files that you have downloaded earlier include the Lower Third.qtz Quartz composition, which you'll use to add lower third titling.

## The Workflow at a Glance

The Lower Third workflow (Workflow 7) consists of the following tasks:



Only the new Lower Third task is added.

## About the Lower Third Composition

The Lower Third.qtz is produced using Quartz Composer and defines a title bar that can be placed over a movie.

The `qceffect` subcommand of the `pcastaction` command provides an option for applying a Lower Third Quartz Composer composition (in this case Third.qtz) to the input movie. In addition, the `qceffect` subcommand takes other input (in this example, the title of the podcast and the name of the author) and passes them to predefined ports on the Lower Third.qtz composition.

The Lower Third.qtz composition takes the input text and applies it to the composition, which is then rendered onto the movie.



## Workflow 7 Setup Summary

Workflow 7 builds on Workflow 6 by adding a task that adds a lower third title bar to the video.

To create Workflow 7, you'll perform the following tasks:

- 1 Create the workflow bundle (page 65).
- 2 Configure the workflow metadata (see page 65).
- 3 Add the Lower Third composition to the workflow (see page 65).
- 4 Configure the workflow template (page 65).

## Setting Up Workflow 7

### Step 1: Create the Workflow Bundle

- Make a copy of `/Library/PodcastProducer/Workflows/Workflow 6.pwf` and change the name to `Workflow 7.pwf`.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

To configure the workflow metadata:

- 1 Open the `Workflow 7.pwf` bundle.
- 2 Open `Info.plist` in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_7`.
  - Set `CFBundleName` to `Workflow 7`.
- 3 Open `InfoPlist.strings` and set the Name and Description keys as follows:

```
Name = "Workflow 7";  
Description = "Here we added the lower third titling to the video.";
```

### Step 3: Add the Lower Third Composition to the Workflow

- Add the lower third effect Quartz Composer (QC) effect (`Lower Third.qtz`) to the Composition repository (`/Library/Compositions/`) on all the rendering Xgrid agent systems.

### Step 4: Configure the Workflow Template

- 1 Open the workflow template `template.plist`.
- 2 In the `taskSpecifications` section of the workflow template, add the `lower_third` task after the watermark task:

```

// Add lower third titling to the watermarked video using a QC effect
lower_third = {
  dependsOnTasks = (watermark);
  command = "/usr/bin/pcastaction";
  arguments = (

    // Use the qceffect subcommand of pcastaction to apply the QC effect
    qceffect,

    "--basedir=$$Base Directory$$",

    // Pass the repository identifier for the composition to use
    "--composition=lower third",

    "--input=$$Content File Basename$$-watermarked.mov",

    // Path where the output QuickTime reference movie will be written.
    // This is a reference movie and not a flat movie file.
    "--output=$$Content File Basename$$-lower_third.mov",

    // Arguments following the "--" argument are passed to the QC effect
    "--",

    // Pass to the Title port of the composition...
    "--Title",

    // ...the title of the episode.
    "$$Title$$",

    // Pass to the Author port of the composition...
    "--Author",

    // ...the full user name of the submitter
    "$$User Full Name$$"
  );
};

```

- 3 In the `taskSpecifications` section of the workflow template, modify the `encode_ipod` task as follows:

```

encode_ipod = {
  // Wait for the lower_third task to complete before running.
  dependsOnTasks = (lower_third);
  command = "/usr/bin/pcastaction";
  arguments = (
    encode,
    "--basedir=$$Base Directory$$",

    // The input movie is now the watermarked video.
    "--input=$$Content File Basename$$-lower_third.mov",
  );
};

```

```
        "--output=${Content} File Basename${-ipod}.m4v",  
        "--encoder=ipod"  
    );  
};
```

- 4 Save and close the template.plist file.

## Deploying the Workflow

Deploy Workflow 7 and test it to make sure it runs without a problem, and then proceed to the next chapter.

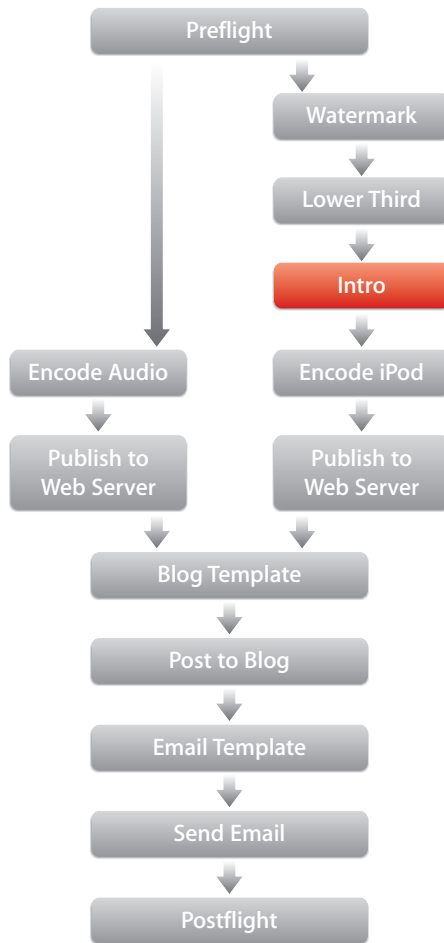


Add a task that adds an intro movie to the titled version of the submitted movie.

In this chapter you'll use the `merge` subcommand of `pcastaction` to merge an intro movie with the titled movie.

## The Workflow at a Glance

The Adding an Introductory Video workflow (Workflow 8) consists of the following tasks:



## About the Intro Movie

The workflow tutorial files include an intro movie that you can use to add to Workflow 8. You can also use your own intro movie.

The intro movie (Introduction.mov) is included with the finished workflow tutorial files in Workflow 8.pwf/Contents/Resources/Movies/. It's about 20 seconds long, and the last 10 seconds consist of black frames on which you'll later superimpose a title.

## Workflow 8 Setup Summary

Workflow 8 builds on Workflow 7 by adding a task that adds an intro movie at the beginning of the titled video.

To create Workflow 8, you'll perform the following tasks:

- 1 Create the workflow bundle (page 71).
- 2 Configure the workflow metadata (see page 71).
- 3 Configure the workflow template (page 71).

## Setting Up Workflow 8

### Step 1: Create the Workflow Bundle

- 1 Make a copy of `/Library/PodcastProducer/Workflows/Workflow 7.pwf` and change the name to `Workflow 8.pwf`.

### Step 2: Configure the Workflow Metadata

- 1 Open the `Workflow 8.pwf` bundle.
- 2 Open `Info.plist` in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_8`.
  - Set `CFBundleName` to `Workflow 8`.
- 3 Open `InfoPlist.strings` and set the `Name` and `Description` keys as follows:

```
Name = "Workflow 8";
Description = "Here we added an introduction bumper movie.";
```

- 4 Copy the intro movie (`Introduction.mov`) to `Workflow 8.pwf/Contents/Resources/Movies/`.

### Step 3: Configure the Workflow Template

- 1 Open the workflow template `template.plist`.
- 2 In the `taskSpecifications` section of the workflow template, add the `intro` task after the `lower_third` task:

```
// Step 2: Add the task that merges the introduction movie with the
// watermarked content video using a Quartz Composer Transition (abides to
// the Image transition protocol)
```

```
intro = {
    dependsOnTasks = (lower_third);
    command = "/usr/bin/pcastaction";

    arguments = (
        merge,
        "--basedir=$$Base Directory$$",
```

```

// First movie to merge.
"--input1=$$Workflow Resource Path$$/Movies/Introduction.mov",

// Second movie to merge.
"--input2=$$Content File Basename$$-lower_third.mov",

// Quartz Composer transition between the first and second movies.
// This can either be an absolute filepath to a composition or
// a QC Repository identifier (begins with a '/'). The identifier is
// case-sensitive.
"--transition=$$Quartz Composer Intro Transition$$",

// How long in seconds the transition should last.
"--duration=$$Intro Transition Duration$$",

// Path to where the output reference movie will be written.
"--output=$$Content File Basename$$-watermarked-intro.mov",
);
};

```

- 3 In the `taskSpecifications` section of the workflow template, modify the `encode_ipod` task as follows:

```

encode_ipod = {
    // Wait for the intro task to complete before running.
    dependsOnTasks = (intro);
    command = "/usr/bin/pcastaction";
    arguments = (
        encode,
        "--basedir=$$Base Directory$$",

        // The input movie is now the titled video with the added intro movie.
        "--input=$$Content File Basename$$-watermarked-intro.mov",

        "--output=$$Content File Basename$$-ipod.m4v",
        "--encoder=ipod"
    );
};

```

- 4 Save and close the `template.plist` file.

## Deploying the Workflow

Deploy Workflow 8 and test it to make sure it runs without a problem, and then proceed to the next chapter.

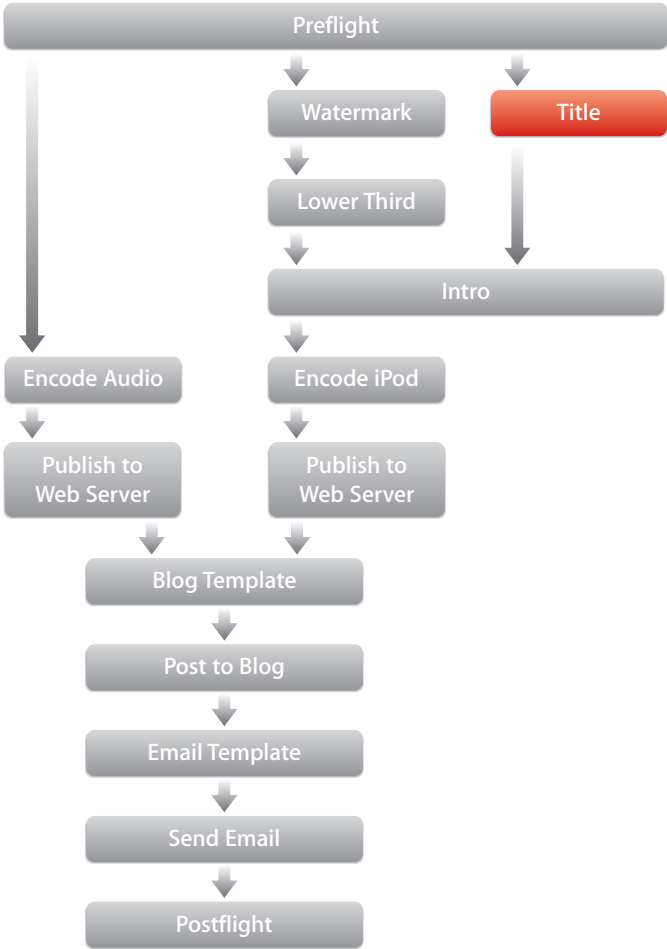


## Add a task for dynamically adding a title to the movie.

In this chapter, you'll use the `title` subcommand of `pcastaction` to dynamically add a title to the intro movie.

# The Workflow at a Glance

The Dynamic Titling workflow (Workflow 9) consists of the following tasks:



## Workflow 9 Setup Summary

Workflow 9 builds on Workflow 8 by adding a task that adds a title to the video.

To create Workflow 9, you'll perform the following tasks:

- 1 Create the workflow bundle (page 75).
- 2 Configure the workflow metadata (see page 75).
- 3 Configure the workflow template (page 75).
- 4 Configure workflow properties in Server Admin (page 77).

### Step 1: Create the Workflow Bundle

- Make a copy of `/Library/PodcastProducer/Workflows/Workflow 8.pwf` and change the name to `Workflow 9.pwf`.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

To configure the workflow metadata:

- 1 Open the `Workflow 9.pwf` bundle.
- 2 Open `Info.plist` in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_9`.
  - Set `CFBundleName` to `Workflow 9`.
- 3 Open `InfoPlist.strings` and set the `Name` and `Description` keys as follows:

```
Name = "Workflow 9";
Description = "Here we added the titling of the video.";
```

### Step 3: Configure the Workflow Template

- 1 Open the workflow template `template.plist`.
- 2 In the `taskSpecifications` section of the workflow template, add the `title` task after the `lower_third` task:

```
title = {
    dependsOnTasks = (preflight);
    command = "/usr/bin/pcastaction";
    arguments = (
        title,
        "--basedir=$$Base Directory$$",

        // Movie on top of which to add the title.
        "--input=$$Workflow Resource Path$$/Movies/Introduction.mov",
```

```

// Titling composition to use.
"--composition=$$Titling Composition$$",

// The title to add.
"--title=$$Title$$",

// Author's name to add to the intro movie.
"--author=$$User Full Name$$",

// By default, the title task adds the date to the input movie.
// However, to overwrite the date, use the --date option.
//"--date=Today",

// Organization
"--organization=$$Organization$$",

// Offset (in sec) at which the titling starts.
// In this case, the title is added after the first 8.25 seconds.
"--offset=8.25",

// Total duration of the output movie in seconds.
"--duration=20",

// Path to the output file.
"--output=Introduction.mov"
);
};

```

- 3 In the `taskSpecifications` section of the workflow template, modify the `intro` task as follows:

```

intro = {
  // Wait for the title and lower-third tasks to complete before running.
  dependsOnTasks = (title, lower_third);

  command = "/usr/bin/pcastaction";
  arguments = (
    merge,
    "--basedir=$$Base Directory$$",

    // This task now takes the titled movie generated by the title task.
    "--input1=Introduction.mov",

    "--input2=$$Content File Basename$$-lower_third.mov",
    "--transition=$$Quartz Composer Intro Transition$$",
    "--duration=$$Intro Transition Duration$$",
    "--output=$$Content File Basename$$-watermarked-titled-intro.mov",
  );
};

```

- 4 Save and close the template.plist file.

**Step 4: Configure workflow properties in Server Admin**

Before you can deploy the workflow, you need to define the values of the variables you specified in the workflow template.

**To define variable values:**

- 1 In Server Admin, set the values for the following workflow variables:
  - Titling Composition—The Quartz Composer composition used by the workflow to add a title to the video.
  - Organization—The name of the organization to be added to the introduction video.
- 2 Click Save.

## Deploying the Workflow

Deploy Workflow 9 and test it to make sure it runs without a problem, and then proceed to the next chapter.

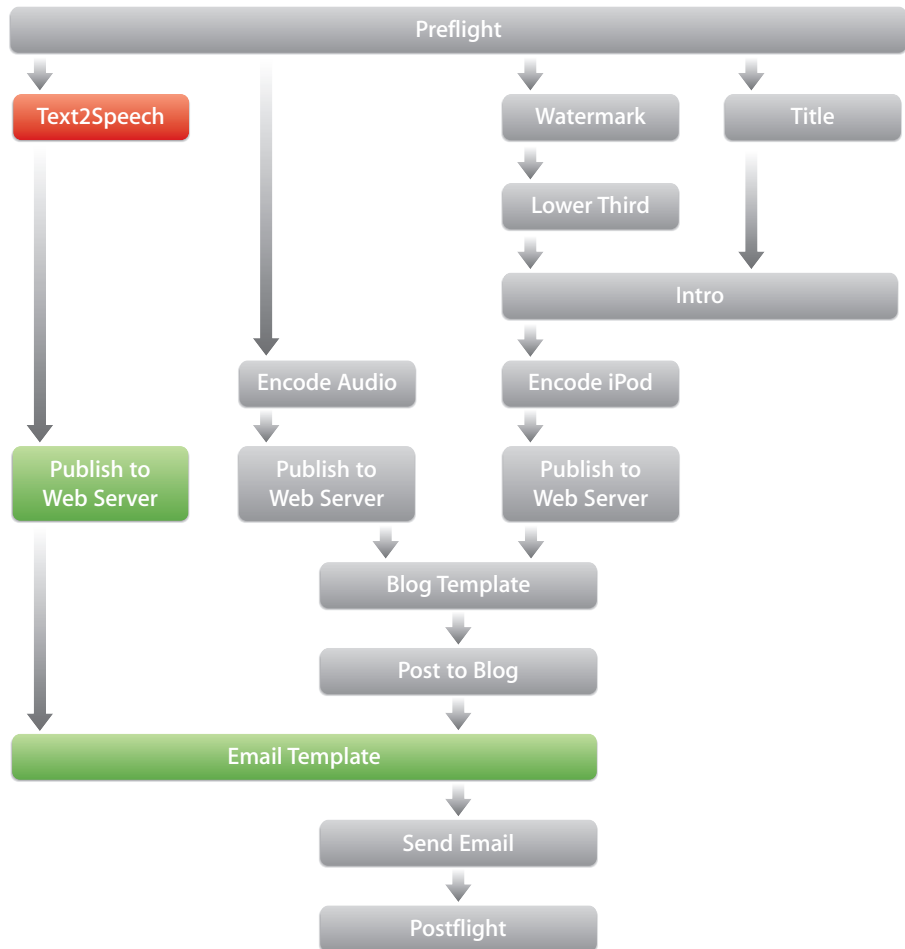


In this chapter, you'll learn how to run command-line tools from a workflow.

In this chapter, you'll use the `pcastaction` command to run command-line tools that add workflow functionality not available in `pcastaction`.

## The Workflow at a Glance

The Command-Line Tools workflow (Workflow 10) consists of the following tasks:



In this workflow, you'll create two new tasks:

- Text2Speech—Uses the `say` command-line tool to read aloud the title and author strings
- Publish to Web Server—Publishes the audio file generated by the Text2Speech task



## Workflow 10 Setup Summary

Workflow 10 builds on Workflow 9 by adding tasks that convert text to speech and publish the resulting audio.

To create Workflow 10, you'll perform the following tasks:

- 1 Create the workflow bundle (page 81).
- 2 Configure the workflow metadata (page 81).
- 3 Configure the workflow metadata (see page 81).
- 4 Modify the mail template (page 83).

## Setting Up Workflow 10

### Step 1: Create the Workflow Bundle

- Make a copy of `/Library/PodcastProducer/Workflows/Workflow 9.pwf` and change the name to `Workflow 10.pwf`.

### Step 2: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

To configure the workflow metadata:

- 1 Open the `Workflow 10.pwf` bundle.
- 2 Open `Info.plist` in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_10`.
  - Set `CFBundleName` to `Workflow 10`.
- 3 Open `InfoPlist.strings` and set the `Name` and `Description` keys as follows:

```
Name = "Workflow 10";
Description = "Here we added a task that reads out the title and author
              using Text2Speech.";
```

### Step 3: Configure the Workflow Template

Configure the workflow template as follows:

- Add a task that converts the podcast title and author information to speech.
- Add a task that publishes the audio of the spoken text.
- Delete the `publish_say` task, as it is no longer needed.
- Modify the dependency of the `template_mail` task.

## To configure the workflow template:

- 1 Open the workflow template `template.plist`.
- 2 In the `taskSpecifications` section of the workflow template, add the `say` task after the `publish_ipod` task:

```
// Generate an audio version of the podcast's title and author.
say = {
    dependsOnTasks = (preflight);
    command = "/usr/bin/pcastaction";
    arguments = (

        // The shell subcommand of pcastaction allows you to run commands in
        // an environment controlled by pcastaction. This minimizes the
        // possibility of running into problems, which can happen when you run
        // commands directly.
        shell,
        "--basedir=$$Base Directory$$",

        // What follows "--" is the command and its arguments.
        "--",

        // Run say, the text2speech tool (For more details, see its man page).
        "/usr/bin/say",

        // Tells 'say' to output an aiff file.
        "-o",
        // Path of the output file.
        "title.aiff",
        // Text to be spoken.
        "Welcome to $$Title$$ presented by $$User Full Name$$"
    );
};
```

This task runs the shell command `say`, which converts the input text to speech. For example, if the title of the presentation is `Workflow Design` and the name of the author is `Anna Johnson`, this task runs the following command:

```
usr/bin/say -o title.aiff "Welcome to Workflow Design presented by Anna
Johnson"
```

This command will read aloud the following:

Welcome to Workflow Design presented by Anna Johnson

- 3 In the `taskSpecifications` section of the workflow template, add the `publish_say` task after the `say` task.

```
// Publish the generated audio file to the Web server.
// This task is extremely similar to the other publishing tasks.
publish_say = {
    dependsOnTasks = (say);
```

```

command = "/usr/bin/pcastaction";
arguments = (
    publish,
    "--basedir= $$Base Directory $$",
    "--web_root= $$Web Document Root $$",
    "--web_url= $$Web URL $$",
    "--date= $$Date_YYYY-MM-DD $$",
    "--title= $$Title $$",
    "--format=audio",

    // String representing the published format
    "--type=audio/x-aiff",

    // MIME type of audio file we are publishing
    "--file=title.aiff",

    "--outfile=say_publish_description_file.yaml"
);
};

```

- 4 In the `taskSpecifications` section of the workflow template, modify the dependency of the `template_mail` task as indicated in the workflow diagram.

```

template_mail = {
    // Wait until groupblog and publish_say finish running
    dependsOnTasks = (groupblog, publish_say);

    command = "/usr/bin/pcastaction";
    arguments = (
        template,
        "--basedir= $$Base Directory $$",
        "--template= $$Workflow Resource Path $$/Templates/mail.txt.erb",
        "--output=mail.txt"
    );
};

```

- 5 Save and close the `template.plist` file.

#### Step 4: Modify the Mail Template

- Modify the mail template `mail.txt.erb` to include a link to the audio file of the spoken text:

```

% administrator_full_name = (properties["Administrator Full Name"] || "")
% administrator_email_address = (properties["Administrator Email Address"]
    || "")
% audience_email_address = (properties["Audience Email List"] || "")
% user_full_name = (properties["User Full Name"] || "")
% user_email_address = (properties["User Email Address"] || "")
% group_short_name = (properties["Group Short Name"] || "")
% group_full_name = (properties["Group Full Name"] || "")
% title = (properties["Title"] || "")

```

```

% group_blog_entry_url = ((IO.read("group_blog_entry_url.txt").chomp rescue
  nil) || "")
% seconds_since_epoch = (properties["Recording Started At"] || "").to_i
% groups_web_server_url = (properties["Groups Web Server URL"] || "")
% groups_blog_url = groups_web_server_url + '/' + group_short_name + '/blog/'

% blog_feed_url = groups_blog_url.gsub(/^http/, 'itpc') + 'index.its'
% ipod_blog_feed_url = blog_feed_url + "?mediaTypeTag=ipod"
% audio_blog_feed_url = blog_feed_url + "?mediaTypeTag=audio"
% ipod_file_url =
  ((YAML::load_file("ipod_publish_description_file.yaml")[:url] rescue
    nil) || "")
% audio_file_url =
  ((YAML::load_file("audio_publish_description_file.yaml")[:url] rescue
    nil) || "")
%
%# Define a variable that contains the URL to the audio file with the
  test2speech audio file generated by the 'say' task.
% title_audio_url =
  ((YAML::load_file("say_publish_description_file.yaml")[:url] rescue
    nil) || "")
%
% from_addresses = [emailify(administrator_full_name,
  administrator_email_address)]
% subject_value = "New Episode in #{group_full_name} Wiki group:
  #{quotify(title)}"
% to_addresses = [audience_email_address, emailify(user_full_name,
  user_email_address), emailify(administrator_full_name,
  administrator_email_address)]
%
<%= encoded_header_with_addresses("From", from_addresses) %>
<%= encoded_header_with_value("Subject", subject_value) %>
<%= encoded_header_with_addresses("To", to_addresses) %>

New Podcast Episode in <%= group_full_name %> Wiki group

<%= quotify(title) %> was recorded at <%=
  localized_time(seconds_since_epoch) %> on <%=
  localized_date(seconds_since_epoch) %>:
<%= braketyfy(group_blog_entry_url) %>

%# Add a link to the audio file containing the text2speech of the title
Listen to the title here: <%= braketyfy(title_audio_url) %>

This episode has been published to the <%= group_full_name %> blog:
<%= braketyfy(groups_blog_url) %>

Subscribe in iTunes:
iPod video: <%= braketyfy(ipod_blog_feed_url) %>
iPod audio: <%= braketyfy(audio_blog_feed_url) %>

```

Download the episode:

```
iPod video: <%= braketify(ipod_file_url) %>  
iPod audio: <%= braketify(audio_file_url) %>
```

This email was sent by Podcast Producer.

## Deploying the Workflow

Validate, deploy, and test Workflow 10 to verify that it runs successfully. Play the podcasts and verify that you can listen to the title and author information being spoken at the beginning of the podcast.

When you're finished, proceed to the next chapter.

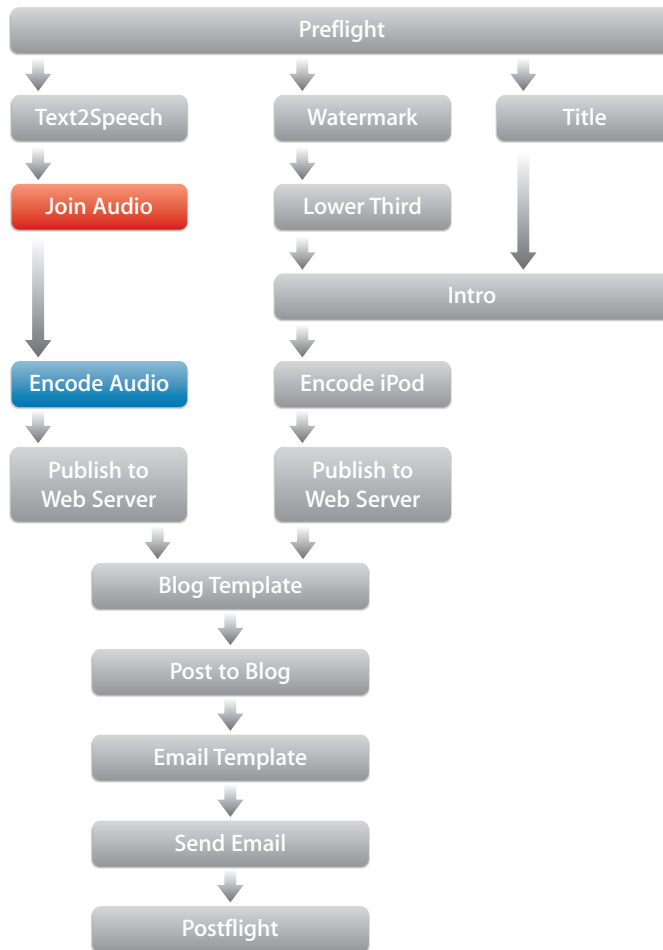


In this chapter, you'll learn how to add custom functionality to the workflow through scripts.

In this chapter, you'll use a custom script to join the audio version of the title and author information to the audio of the submitted movie.

## The Workflow at a Glance

The Integrating with Custom Scripts workflow (Workflow 11) consists of the following tasks:



In this workflow, the Join Audio task takes the file generated by the Text2Speech task and merges it with the audio of the submitted movie. The task uses a Ruby script and the Ruby Cocoa bridge to call QTKit and join the two movies with a gap inserted between them.



## Workflow 11 Setup Summary

Workflow 11 builds on Workflow 10 by adding a task that joins the spoken audio with the audio of the submitted movie.

To create Workflow 11, you'll perform the following tasks:

- 1 Create the workflow bundle (page 89).
- 2 Write a Ruby script that joins two QuickTime movies (page 89).
- 3 Configure the workflow metadata (see page 91).
- 4 Configure the workflow template (page 91).
- 5 Restore the mail template (page 93).

## Setting Up Workflow 11

### Step 1: Create the Workflow Bundle

- Make a copy of `/Library/PodcastProducer/Workflows/Workflow 10.pwf` and change the name to `Workflow 11.pwf`.

### Step 2: Write a Ruby Script that Joins Two QuickTime Movies

- Write the following Ruby script to join the spoken version of the title and author information with the audio of the submitted movie. Save the script as `qtjoin.rb` in `Workflow 11.pwf/Contents/Resources/Tools/`.

This script can join any two QuickTime files, not just audio files.

```
# Import the Ruby Cocoa bridge.
# For more information about Ruby Cocoa,
# see http://rubycocoa.sourceforge.net/HomePage.
require 'osx/cocoa'

# Load the QTKit framework.
OSX.require_framework 'QTKit'

# Print version helper method.
def print_version
  $stderr.puts "qtjoin, version 1.0"
  $stderr.puts "Copyright 2008 Apple, Inc"
  $stderr.puts
end

# Print usage helper method.
def print_usage
  print_version
  $stderr.puts "Takes two movies and joins them with an optional gap in the
  middle"
```

```

    $stderr.puts "usage: qtjoin first_movie_path second_movie_path
    output_movie_path [gap]"
    $stderr.puts "          qtjoin movie1.mov movie2.mov output.mov 1"
    $stderr.puts
end

# Check to see if there are enough arguments.
if ARGV.size < 3 || ARGV.size > 4
    print_usage
    exit(-1)
end

# Assign better names to the arguments.
first_input_path = ARGV[0]
second_input_path = ARGV[1]
output_path = ARGV[2]
gap = (ARGV.size > 3 ? ARGV[3].to_f : 0.0)

# Load the first movie.
first_input_movie, error =
    OSX::QTMovie.movieWithFile_error(first_input_path)
# Check to see if the first movie opened successfully.
if first_input_movie == nil or error != nil
    $stderr.puts "could not load first movie"
    exit(1)
end

# Load the second movie.
second_input_movie, error =
    OSX::QTMovie.movieWithFile_error(second_input_path)

# Check to see if the second movie opened successfully.
if second_input_movie == nil or error != nil
    $stderr.puts "could not load second movie"
    exit(1)
end

# Create the output movie.
output_movie = OSX::QTMovie.movie

# Make the output movie editable.
output_movie.setAttribute_forKey(true, OSX::QTMovieEditableAttribute)

# Determine time to insert the inter-movie gap and create its duration.
zero_time = OSX::QTMakeTime(0, first_input_movie.duration.timeScale)
gap_duration = OSX::QTMakeTimeWithTimeInterval(gap)

# Define the time ranges for the first and second input movies.
first_input_movie_time_range = OSX::QTMakeTimeRange(zero_time,
    first_input_movie.duration)

```

```

second_input_movie_time_range = OSX::QTMakeTimeRange(zero_time,
    second_input_movie.duration)

# Insert the two movies into the output movie at the correct times.
output_movie.insertSegmentOfMovie_timeRange_atTime(first_input_movie,
    first_input_movie_time_range, output_movie.duration)
output_movie.insertSegmentOfMovie_timeRange_atTime(second_input_movie,
    second_input_movie_time_range,
    OSX::QTTimeIncrement(output_movie.duration, gap_duration))

# Write the output movie to file.
success = output_movie.writeToFile_withAttributes(output_path, nil)

# Perform error checking.
unless success
    $stderr.puts "could not save movie"
    exit(1)
end

exit(0)

```

### Step 3: Configure the Workflow Metadata

Configure the following workflow metadata:

- Workflow bundle identifier
- Workflow bundle name
- Workflow name
- Workflow description

To configure the workflow metadata:

- 1 Open the Workflow 11.pwf bundle.
- 2 Open Info.plist in Property List Editor and do the following:
  - Set `CFBundleIdentifier` to `com.apple.PodcastProducer.workflow.Workflow_11`
  - Set `CFBundleName` to `Workflow 11`
- 3 Open InfoPlist.strings and set the Name and Description keys as follows:

```

Name = "Workflow 11";
Description = "Here we added a task that reads out the title and author
    using Text2Speech.";

```

### Step 4: Configure the Workflow Template

Configure the workflow template as follows:

- Add a task to join the audio of the podcast title and author information to the audio of the podcast.
- Delete the `publish_say` task, which is no longer needed.
- Modify the dependencies of the `encode_audio` and `template_mail` tasks.

### To configure the workflow template:

- 1 Open the workflow template template.plist.
- 2 In the `taskSpecifications` section of the workflow template, add the `join_audio` task after the `say` task.

```
// Call the qtjoin.rb script to join the text2speech audio file and the
    input
// movie.
join_audio = {
    dependsOnTasks = (say);
    command = "/usr/bin/pcastaction";
    arguments = (
        shell,
        "--basedir=$$Base Directory$$",

        "--",
        // Path to the qtjoin.rb script.
        "$$Workflow Resource Path$$/Tools/qtjoin.rb",

        // Path to the text2speech file.
        "title.aiff",

        // Path to the second file (input movie).
        "$$Content File Basename$$$$Content File Extension$$",

        // Path where the output file will be written to.
        "$$Content File Basename$$-withAudioTitle.mov",

        // Duration, in seconds, of the gap.
        "1"
    );
};
```

- 3 In the `taskSpecifications` section of the workflow template, modify the dependency of the `encode` task as indicated in the workflow diagram.

```
encode_audio = {
    dependsOnTasks = (join_audio);
    command = "/usr/bin/pcastaction";
    arguments = (
        encode,
        "--basedir=$$Base Directory$$",
        "--input=$$Content File Basename$$$$Content File Extension$$",
        "--output=$$Content File Basename$$-ipod.m4v",
        "--encoder=mp4_audio_high"
    );
};
```

- 4 In the `taskSpecifications` section of the workflow template, remove the `publish_say` task, which is no longer needed.

- 5 In the `taskSpecifications` section of the workflow template, modify the dependency of the `template_mail` task as indicated in the workflow diagram.

```
template_mail = {
  dependsOnTasks = (groupblog);
  command = "/usr/bin/pcastaction";
  arguments = (
    template,
    "--basedir=${Base Directory}",
    "--template=${Workflow Resource Path}/Templates/mail.txt.erb",
    "--output=mail.txt"
  );
};
```

- 6 Save and close the `template.plist` file.

### Step 5: Restore the Mail Template

You need to restore to the mail template used in Workflow 8, now that you have removed the `Text2Speech` task.

**To restore the mail template:**

- Replace the mail template in Workflow 11 (`Workflow 11.pwf/Contents/Resources/Templates/mail.txt.erb`) with the template used in Workflow 8 (`Workflow 8.pwf/Contents/Resources/Templates/mail.txt.erb`).

## Deploying the Workflow

Validate, deploy, and test Workflow 11 to make sure it runs successfully.

You have now completed the workflow. The next two chapters discuss general workflow design and troubleshooting tips.



## Learn how to better design and develop Podcast Producer workflows.

Use the tips in this chapter to improve and speed up the design and development of Podcast Producer workflows.

### Create a Graph of the Workflow

The first step when designing a workflow is to create a graph of the tasks you want the workflow to execute.

Use a flowcharting tool to draw the graph, because it gives you the flexibility you need to easily modify your graphs. When making the graph, make sure you use arrows to indicate dependencies, as in the examples shown throughout this tutorial.

After you finalize the design, start implementing the workflow design.

### Start Small

Workflows can quickly grow bigger and more complicated due to task interdependencies. As a result, if you attempt creating the workflow in one iteration, design or implementation errors can complicate the debugging process.

Instead of implementing your workflow design all at once, you should implement your workflow design as you have done in this tutorial—incrementally.

Go back to your graph and create a subgraph consisting of the first few tasks. Then, create more graphs by adding more tasks to every subsequent graph, just as you did in this tutorial, until the implementation is complete.

Every time you implement a subset of the original graph, deploy it and test it to make sure that it works as intended. If the workflow completes successfully, then expand it by adding one or two tasks at a time.

In this way, you can progress slowly but surely until the workflow is completed. Testing and debugging will be limited to a few tasks at time and will go much faster.

## Use an All-in-One Setup for Workflow Development

To simplify and speed up workflow development, use one computer running Mac OS X Server v10.5 to provide Podcast Producer services and all other services needed by your workflow.

By using a single computer for workflow development, you eliminate the dependency on other services running on other computers outside your control. Otherwise, if even just one service is down, you won't be able to deploy and test workflows until the service is back in operation.

## Validate Your Workflow Before Running It

Podcast Producer provides a way for you to validate your workflow before running it.

To validate a workflow, enter the following command:

```
$ pcastconfig --validate_workflow_at_path path_to_workflow
```

For example, to validate the Hello World! workflow, enter:

```
$ pcastconfig --validate_workflow_at_path /Library/PodcastProducer/  
Workflows/Workflow\ 1.pwf  
Workflow: /Library/PodcastProducer/Workflows/Workflow 1.pwf is VALID
```

For more information about `pcastconfig`, see its man page.

## Check Metadata

In addition to the key/value pairs defined in Server Admin, Podcast Producer stores metadata about a submitted job in the `properties.plist` file.

The `properties.plist` file contains all the metadata used by a workflow, and is useful for troubleshooting values and discovering what metadata is available for workflow development.

Look at a `properties.plist` file from a previous workflow submission to see all the metadata you can use.

Podcast Producer stores this file in the Recordings folder on the shared file system.

In a workflow template, you can access the value of every key in the `properties.plist` file by surrounding the key with `$$` characters. For example, to access the value of the `Author` key, use the `$$Author$$` variable to represent the value of the key.



The following is an example of the contents of a properties.plist file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
    www.example.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Administrator Email Address</key>
    <string>pcpadmin@sam.example.com</string>
    <key>Administrator Full Name</key>
    <string>pcpadmin</string>
    <key>Administrator Short Name</key>
    <string>pcpadmin</string>
    <key>Approval Email List</key>
    <string>pcpadmin@sam.example.com</string>
    <key>Approval Folder</key>
    <string>/Volumes/Data/SFS//Approval/</string>
    <key>Archive Root</key>
    <string>/Volumes/Data/SFS//Archive</string>
    <key>Audience Email List</key>
    <string>pcpuser@sam.example.com</string>
    <key>Audience SMS List</key>
    <string>pcpuser@sam.example.com</string>
    <key>Author</key>
    <string>sam</string>
    <key>Base Directory</key>
    <string>/Volumes/Data/SFS/Recordings/3D994F4E-549A-41CF-BC62-
        F8E2DADB90B1</string>
    <key>Comment</key>
    <string>no comment</string>
    <key>Content File Basename</key>
    <string>apple-iphone-how_to_640x496</string>
    <key>Content File Extension</key>
    <string>.mov</string>
    <key>Content File Name</key>
    <string>apple-iphone-how_to_640x496.mov</string>
    <key>Copyright</key>
    <string>Copyright (c) 2007 Apple Inc.</string>
    <key>Date</key>
    <string>20080212-142310</string>
    <key>Date_YYYY-MM-DD</key>
    <string>2008-02-12</string>
    <key>Description</key>
    <string>Auto-generated metadata for Blog with streaming and intro at:
        20080212-142310</string>
    <key>Exit Transition Duration</key>
    <string>0.75</string>
    <key>Exit Video Path</key>
    <string>/Volumes/Data/SFS//Caches/Resources/Movies/Exit.mov</string>
    <key>Global Resource Path</key>
    <string>/Volumes/Data/SFS/Caches/Resources</string>
```

```

<key>Group Full Name</key>
<string>Podcasts</string>
<key>Group Short Name</key>
<string>podcasts</string>
<key>Groups Administrator Username</key>
<string>pcpadmin</string>
<key>Groups Web Server URL</key>
<string>http://sam.example.com/groups</string>
<key>Intro Transition Duration</key>
<string>3</string>
<key>Introduction Video Path</key>
<string>/Volumes/Data/SFS//Caches/Resources/Movies/Introduction.mov</
string>
<key>Keywords</key>
<string>Blog with streaming and intro 20080212-142310 test3</string>
<key>Notification Language</key>
<string>English</string>
<key>Organization</key>
<string>Example Organization</string>
<key>Podcast Producer URL</key>
<string>https://sam.example.com:8170/podcastproducer</string>
<key>Postflight Script Path</key>
<string>/Volumes/Data/SFS//Caches/Resources/Tools/postflight_script</
string>
<key>Preflight Script Path</key>
<string>/Volumes/Data/SFS//Caches/Resources/Tools/preflight_script</
string>
<key>QTSS URL</key>
<string>rtsp://sam.example.com/PodcastProducer</string>
<key>Quartz Composer Exit Transition</key>
<string>/dissolve</string>
<key>Quartz Composer Filter</key>
<string>/sepia</string>
<key>Quartz Composer Intro Transition</key>
<string>/swing</string>
<key>QuickTime Streaming Media Root</key>
<string>/Volumes/Data/SFS//Streams</string>
<key>Recording Started At</key>
<string>1202854992</string>
<key>Recording Stopped At</key>
<string>1202854992</string>
<key>SMTP Server</key>
<string>sam.example.com</string>
<key>Server UUID</key>
<string>85E4CEA7-00BF-470A-A558-E254D8F30524</string>
<key>Shared Filesystem</key>
<string>/Volumes/Data/SFS/</string>
<key>Title</key>
<string>shipping Blog with streaming and intro (1)</string>
<key>Titling Composition</key>

```

```

<string>/Volumes/Data/SFS//Caches/Resources/Compositions/FlyingTitle.qtz</
  string>
<key>User Email Address</key>
<string>pcpuser@sam.example.com</string>
<key>User Full Name</key>
<string>pcpuser</string>
<key>User Home Directory</key>
<string>99</string>
<key>User ID</key>
<string>1025</string>
<key>User Short Name</key>
<string>pcpuser</string>
<key>Watermark Image</key>
<string>/Volumes/Data/SFS//Caches/Resources/Images/Watermark.png</string>
<key>Web Document Root</key>
<string>/Volumes/Data/SFS//Podcasts</string>
<key>Web URL</key>
<string>http://sam.example.com/Podcasts</string>
<key>Workflow Bundle Path</key>
<string>/Volumes/Data/SFS/Caches/Workflows/Blog with streaming and
  intro.pwf</string>
<key>Workflow Resource Path</key>
<string>/Volumes/Data/SFS/Caches/Workflows/Blog with streaming and
  intro.pwf/Contents/Resources</string>
<key>Xgrid Job Name</key>
<string>shipping Blog with streaming and intro (1) by pcpuser (Blog with
  streaming and intro)</string>
<key>iTunes U Posting Credentials</key>

  <string>Administrator@urn:mace:itunesu.com:sites:podcastproducer.exampl
    e.com</string>
<key>iTunes U Site URL</key>
<string>podcastproducer.example.com</string>
<key>iTunes U Tab ID</key>
<string>1234567890.12345678901</string>
</dict>
</plist>

```

## Keep Temporary Files

When Podcast Producer receives a workflow job for processing, it creates a temporary directory for the job on the shared file system (for example, `/Volumes/Data/SFS/Recordings/3D994F4E-549A-41CF-BC62-F8E2DADB90B1/`) where it stores the temporary files generated while processing the workflow.

To save space, the default postflight script that ships with Podcast Producer erases all the temporary files it generates. However, some of these temporary files (for example, `properties.plist`) contain information that can be useful when debugging workflows.

To keep the temporary files, set the Postflight property in Server Admin (Podcast Producer > Settings > Properties) to `/usr/bin/true`. The `true` script returns a true (positive) value and does not erase the temporary files.

Then, after you finish troubleshooting a workflow and are ready to deploy it, you can set the Postflight property in Server Admin back to the default value of `/System/Library/PodcastProducer/Resources/Tools/postflight_script` or whatever other postflight script you want to run. After the workflow is executed, Podcast Producer deletes the temporary files.

## Learn the tools and techniques available for debugging workflows.

This chapter discusses various options for debugging workflows.

### Using Xgrid Admin

Xgrid Admin lets you monitor the progress of your workflows and provides information about Xgrid jobs, including the Xgrid identifier. Double-clicking a job displays more information about it.

Using Xgrid Admin, you can detect problems and take corrective actions promptly.

You can pause a job and restart it, but you can't rerun it if it fails.

### Using the `xgrid` Command

In addition, to Xgrid Admin, you can also use the `xgrid` command to display all the information you need about an Xgrid job by specifying its Xgrid job identifier.

```
$ xgrid -h FQDN -auth Kerberos -job results -id job_identifier [-tid task_identifier]
```

```
$ xgrid -h FQDN -auth Kerberos -job specification -id job_identifier
```

### Monitoring Xgrid Tasks

Each Xgrid agent adds information about running Xgrid tasks to the system log and to `/Library/Logs/Xgrid/xgridagentd.log`. To monitor Xgrid tasks, check these logs for information. The `xgridagentd.log` file provides more detail than the system log.

### Debugging Failing Commands

If a task fails, one way to debug it is to run it in the Recording folder on the shared file system used by Podcast Producer.

You can copy the command from the system log.

**To run the command in the Recording folder:**

- 1 Start an SSH session:

```
$ ssh pcastxgrid@localhost
```

- 2 Change the directory to the Recording folder:

```
$ cd path_to_Recording_folder
```

- 3 Copy the command in question from the system log and run it:

```
$ pcastaction command arguments
```

Executing tasks from the command line recreates the context in which the tasks are run in Xgrid. This allows you to zero in on why tasks are failing. In addition, you get more information about the running tasks, including additional error messages that you won't see elsewhere, by reading stdout from the command line.