

Business Process Automation with ProcessMaker 3.1

A Beginner's Guide

—

Dipo Majekodunmi

Apress®

www.allitebooks.com

Business Process Automation with ProcessMaker 3.1

A Beginner's Guide

Dipo Majekodunmi

Apress®

Business Process Automation with ProcessMaker 3.1: A Beginner's Guide

Dipo Majekodunmi
Lagos, Nigeria

ISBN-13 (pbk): 978-1-4842-3344-3
<https://doi.org/10.1007/978-1-4842-3345-0>

ISBN-13 (electronic): 978-1-4842-3345-0

Library of Congress Control Number: 2017962072

Copyright © 2018 by Dipo Majekodunmi

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image designed by Freepik

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Susan McDermott
Development Editor: Laura Berendson
Coordinating Editor: Rita Fernando
Copy Editor: James A. Compton, Compton Editorial Services

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484233443. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

Table of Contents

About the Author	xiii
Acknowledgments	xv
Foreword	xvii
Introduction	xxi
Chapter 1: An Introduction to Workflow and Business Process Management	1
What Is a Workflow?	1
A Sample Workflow	2
What Is a Business Process?.....	5
What Is BPM?	6
What Is ProcessMaker?	7
Chapter 2: Getting Started with ProcessMaker	9
Installation Steps	11
For Mac OS X Users.....	11
For Windows Users.....	20
The Bitnami Application Manager	32
The Welcome Screen	32
The Manage Servers Screen	33
The Server Events Screen	34
Exploring the ProcessMaker Interface	35
ProcessMaker Concepts	36
Chapter 3: The ProcessMaker Workflow Designer	39
Process List Actions	40
New	40
Edit	42

TABLE OF CONTENTS

- Status 42
- Export 42
- Delete and Delete Cases..... 43
- Import 43
- Category Filter and Search 44
- Debug 44
- Process List Columns..... 45
- Process Designer 47
 - Top Toolbar 47
 - Process Map Area..... 50
 - Shapes Toolbox..... 51
- Chapter 4: Modeling a Process..... 73**
 - Create a New Process..... 74
 - Add Tasks to the Process 75
 - Connecting Tasks in the Process 76
- Chapter 5: Making the Process Comprehensible..... 79**
 - The Shapes Toolbox, Continued 79
 - Data Elements 79
 - Pools and Lanes 81
 - Artifacts 82
 - Tying It All Together 83
 - Complete the Process Model 86
- Chapter 6: Building the Process 93**
 - Variables 94
 - Variable Name 96
 - Variable Type 97
 - Database Connection..... 99
 - SQL 99
 - Define Accepted Variable Values 99

Create the Variables.....	100
Dynaforms.....	101
Chapter 7: The Responsive Dynaform Designer.....	103
Dynaform Title.....	104
Dynaform Designer Menu	104
Save.....	104
Export	104
Import.....	105
Preview.....	105
Clear	105
Language.....	105
Close.....	106
Dynaform Control and Properties Panel.....	106
Web Controls	106
Properties	107
History of Use.....	107
Dynaform Container	107
Form Control Properties	108
Row Control Properties.....	109
Chapter 8: Dynaform Web Controls	111
Creating Variables from the Dynaform Designer.....	112
Textbox.....	114
Textarea	121
Dropdown.....	122
Checkbox	125
Checkgroup.....	127
Radio	128
Datetime.....	129
Repositioning a Row	135
Suggest.....	136

TABLE OF CONTENTS

- Dividing a Row 137
- Hidden..... 139
- Title and Subtitle..... 140
- Label 141
- Link 141
- Image 142
- File 143
- Multiple File Uploader 146
- Submit and Button 147
- Grid 150
 - Adding Controls to the Grid 151
 - Adding and Deleting Rows 153
 - Paging Records in the Grid 154
 - Modifying the Grid Layout 155
 - Validating Required Fields 157
 - Mathematical Functions in Grids 157
- Panel 159
- Subform 163
- Chapter 9: Adding Forms to the Process 167**
 - Building the Form..... 168
 - Adding Comments to the Form 175
 - Debugging Errors in JavaScript 180
 - Cloning the Form..... 183
 - Adding Approval Functionality..... 184
 - Approval without Code 184
 - Approval with Code..... 184
 - Another Variant of the Form 189
 - Assigning a Form to a Task 190
 - Default Steps in a Task: Assignment and Routing 191
 - Adding a Dynaform Step..... 192

Chapter 10: Administering Users in ProcessMaker	199
Users.....	201
Adding a New User	201
Editing a User	204
Disabling a User	204
Deleting a User	206
User Summary, Group and Authentication.....	206
Groups.....	206
Creating a Group.....	206
Editing a Group	207
Deleting a Group.....	207
Assigning Users to a Group	208
Assigning Groups to a User	209
Departments	211
Adding a New Department	211
Assigning Users to a Department.....	212
Setting a Department Manager	213
Deleting a Department	215
Roles	215
Default Roles	215
Creating New Roles.....	216
Viewing and Editing Role Permissions	216
Assigning Users to Roles.....	218
Authentication Sources.....	220
Setting Up an Authentication Source.....	220
Importing Users from an Authentication Source.....	222
User Experience	225
Changing the User Experience for a User.....	225
Chapter 11: Assigning Users to Tasks in a Process	229
Assigning Users and Groups	230
Cyclical Assignment.....	232

TABLE OF CONTENTS

- Manual Assignment 235
- Comparing Cyclical, Manual, and Value-Based Assignment 237
- Value-Based Assignment 241
- Reports To 245
- Self Service..... 247
- Self-Service Value-Based Assignment..... 251
- Chapter 12: Triggers 257**
 - Trigger Timing 258
 - Before a Step..... 258
 - After a Step..... 258
 - Before Assignment 258
 - Before Routing..... 259
 - After Routing 259
 - Case and System Variables..... 259
 - Case Variable Prefixes 260
 - System Variables 260
 - Variable Selector 262
 - Creating Triggers..... 262
 - Predefined Triggers 263
 - Custom Triggers..... 267
 - Copying Triggers..... 269
 - Testing the Triggers..... 270
 - Debugging Triggers 272
 - Enabling and Disabling Debug Mode..... 273
 - ProcessMaker Debugger 274
 - Identifying Errors 276
- Chapter 13: Input and Output Documents 277**
 - Input Documents..... 277
 - Creating an Input Document..... 278
 - Adding Input Documents to a Dynaform..... 280

Adding Input Documents as a Step	281
Viewing the Documents in the Document Management System	286
Output Documents	287
Creating an Output Document	287
Chapter 14: Completing the Process	301
Building the Additional Forms	302
Modifying the Imported Form	303
Clone the Form	307
Assign the Forms to Tasks	311
Define the Routing Rule	312
Configure Assignment Rules	312
Set Up Receipt Upload	313
Generate the Expense Report	314
Add Some Triggers	319
Test the Changes	322
Chapter 15: Enhancing the Process	331
Feedback	331
Finance Officers	331
Supervisors	332
Employees	332
Case Labels	333
Email Notifications	334
Using the Task Notification Property	334
Creating a Template for Email Notification	336
Using PMFSendMessage in a Trigger	338
Assign the Triggers to Tasks	342
Check that Email Sending is Configured	342
Prefilling Form Fields with Triggers	344
Setting Datetime Control Properties	347
Dynaform Logic in JavaScript	348

TABLE OF CONTENTS

- Case Permissions and Case Notes..... 350
- Escalating Unclaimed Cases..... 354
- Testing the Enhancements..... 356
- Chapter 16: Complex Routing with Gateways 363**
 - Exclusive (XOR) Gateway 365
 - Parallel (AND) Gateway 365
 - Sample Process..... 365
 - Testing the Process 375
 - The Inclusive (OR) Gateway 378
 - Cloning the Process..... 378
 - Changing the Gateway..... 379
 - Applying the Conditions..... 380
 - Testing the Process 381
 - Default Flow 383
- Chapter 17: Admin Features 385**
 - Settings..... 385
 - Logo..... 386
 - Email Servers 387
 - Calendar 387
 - Process Categories..... 389
 - Language..... 391
 - Skins..... 394
 - Environment 395
 - Cases List Cache Builder 395
 - Clear Cache 396
 - PM Tables 396
 - Login..... 401
 - Dashboards 402
 - System..... 404
 - System Information, Check PM Requirements and PHP Information 404

Plugins	404
Logs	406
Chapter 18: Going Mobile	407
ProcessMaker Mobile Apps.....	407
Install the App.....	407
Install ngrok for Remote Access.....	411
Configure Mobile App Settings	414
Create a Case	415
Deploying to Production.....	416
Chapter 19: Installing ProceMaker on a Cloud Server.....	419
Getting a DigitalOcean Account.....	419
Register for Your Account	419
Confirm Your Email Address	420
Account Verification.....	421
Creating Your Droplet (Virtual Private Server).....	423
Connecting to the Droplet	426
Using Mac or Linux.....	426
Using Windows	427
Installing ProcessMaker.....	430
Remove MariaDB	431
Install Apache	431
Install PHP 5.6	432
Install MySQL 5.5.X.....	433
Secure the MySQL Installation	436
Disable SELINUX.....	437
Enable Firewall and Open ProcessMaker ports.....	437
Download and Extract ProcessMaker Installer.....	438
Configure Apache Web Server	441
Complete the Installation.....	443
Take a Snapshot.....	448

TABLE OF CONTENTS

- Chapter 20: Deploying to Production..... 451**
 - Get a Free Domain Name 451
 - Set Up DNS..... 454
 - Install SSL Certificate..... 457
 - Create a non-Root Super User..... 457
 - Install the Required Software 457
 - Request a Certificate from Let's Encrypt..... 458
 - Deploying the Process 462
 - Configuring the Mobile App..... 463
 - What Next?..... 464
- Index..... 465**

About the Author

Dipo Majekodunmi is a Certified ProcessMaker Architect and Developer with 7 years of experience building and automating business processes using ProcessMaker. He has implemented ProcessMaker for a number of banks and financial service providers in Nigeria, integrating ProcessMaker with banking applications and other enterprise systems. His background as a business analyst gives him the unique ability to understand and address business needs through technology. Dipo is an AIIM Certified Information Professional and holds a Post Graduate Diploma in Advanced IT and Business Management from the University of Wales. He is the founder and managing partner at dipoleDIAMOND in Lagos, Nigeria, where he helps businesses leverage technology to solve problems.

Acknowledgments

I would like to thank Amos B. Batto, ProcessMaker Technical Documentation Writer & Forum Manager, for reviewing the code in the book and giving valuable feedback and insights to make this book better.

I have added some of his feedback about JavaScript in Chapter 9 verbatim in a section there labeled “Notes from Amos.”

Foreword

The rate of technological change in the workplace is increasing every day at a faster pace. Business leaders that hope to maintain a competitive edge for their businesses have no choice but to embrace this technological change. According to one study, two-thirds of the CEOs of Global 2000 companies will have digital transformation at the center of their corporate strategy by the end of 2017 (source: IDC).

Current and future CEOs will not have the luxury of letting their CIOs make key technology decisions alone. The role of the CIO and will grow in importance, but the CEO will be expected to take a more active role in technology decision-making. In more and more industries business strategy will become inextricably linked to technology strategy.

I interpret the concept of digital transformation as the merger of business strategy with technology strategy. In particular I see three important factors that contribute to digital transformation:

1. Technology
2. Data
3. Process

Some businesses will need to focus on just one of these factors to achieve true digital transformation. Others will need to focus on two or all three of these factors.

The digital transformation caused by technological change is often obvious. These technological changes are things like autonomous vehicles, RFID/IoT, and CRISPR for gene editing. These technologies are literally shaking industries to the ground. It is easy to see the transformative nature offered by these pure technology breakthroughs.

The transformation being caused by data is a little subtler. Businesses that can produce big data sets have the opportunity to create new business models based around the monetization of their big data. Google has already developed several billion-dollar businesses based off of all the search data they collect. Amazon and its clients are another big winner thanks to big data. Amazon's Mechanical Turk gathers big data sets by paying people to perform discrete tasks called HITs (Human Intelligence Tasks). The

FOREWORD

results of these HITs make platforms smarter and more valuable at things like image recognition that require lots of data to be smart enough to be valuable.

The third leg of digital transformation is process. Every type of software today manages some type of process. CRM and ERP systems have been slowly transforming most businesses for the past several decades. However, processes are changing at a faster rate today than they did just a few years ago. The effects of technology changes and changes caused by big data are rippling across all industries. The result is that more and more businesses need to automate processes even faster than before.

And once automated, the work is not done. There is no such thing as automate and forget. Critical business processes not only require faster automation; they also require faster reconfiguration.

ProcessMaker is a modern, intelligent Business Process Management and Workflow suite designed to connect systems and people to make processes run faster and smarter. With the ProcessMaker visual process designer, it is simple to build process apps that connect system APIs, web forms, data, and people into a single process.

Examples of some of the custom processes that our customers model and automate in ProcessMaker include the following:

- Credit Applications
- Change Order Requests
- Purchase Requests
- CAPEX Requests
- AFE Requests
- Employee On-boarding Processes
- System Access Requests
- New Product Development

We are very excited to be able to recommend to our users Dipo's *Business Process Automation with ProcessMaker 3.1: A Beginner's Guide*. I have personally had the chance to work with Dipo over several years, and I have always been extremely impressed by both his technical knowledge and his business acumen. I believe that to be able to write a truly useful guide to ProcessMaker and BPM, a writer must have knowledge in both

areas. Dipo has just that. Dipo's years of working in banking and other industries have given him deep first-hand experience implementing real processes for real businesses. I am sure that ProcessMaker users all around the world will benefit enormously from this guide.

So enjoy the guide, and happy process automation!

Brian S. Reale, Co-Founder ProcessMaker

Introduction

I first came across ProcessMaker in 2010 when researching for an alternative BPM application at work to replace the current one we used. Even though I had no previous experience working with the solution, I was able to build a process in a matter of hours. Within a week I had a demo ready to show my boss.

Over the past seven years, I have worked on automating more complex business processes, integrating ProcessMaker with enterprise applications and building custom plugins. When I'm passionately proselytizing ProcessMaker, my new converts are not always sure where to start. This book is my effort to help those new to ProcessMaker understand the concepts and get started automating business processes.

I have learned a lot writing this book and would appreciate your feedback about it. You can leave your feedback, comments, questions and suggestions on the Learning BPM blog: <https://learningbpm.com/books/beginners-guide-feedback>

While on the subject of the blog, I will also post new insights, tips, and tricks on the blog, and you can also suggest topics you would like to see covered.

Once again, thank you for purchasing this book and all the best in automating your business processes.

—Dipo Majekodunmi

CHAPTER 1

An Introduction to Workflow and Business Process Management

Every organization has its way of doing things: making purchases, attending to customers' and employees' requests, keeping track of inventory, and so on. More often than not, these tasks involve more than one person, and the need for someone in authority to approve the request or a person assigned the responsibility to carry out the task. This is often documented using forms or documents that are passed around as the task or request progresses. The documents contain the details of the request and signatures of those who have worked on it.

Usually, these documents are printed and filled out. When there are supporting documents such as receipts, invoices and so on, they are also attached to the document. When the request is completed, the documents are filed and stored for future references.

In other scenarios, to avoid printing documents and also save time when the person required to work on a task cannot be reached or works in a different location, the documents are scanned and emailed. The other party then prints it out, endorses the document, scans it and emails it back so that the work can continue.

What Is a Workflow?

The scenarios just described illustrate what can be referred to as a “*workflow*.” A quick [Google](https://www.google.com?q=What+is+a+workflow) search for “What is a workflow?” (<https://www.google.com?q=What+is+a+workflow>) gives us this definition: “the sequence of industrial, administrative, or other processes through which a piece of work passes from *initiation* to *completion*.”

There are many more complex definitions of what a workflow is, but I would rather we keep things simple. Another definition I like is from SearchCIO (<http://searchcio.techtarget.com/definition/workflow>), which defines a workflow as “the series of activities that are necessary to complete a task.” Putting it all together, we can think of a workflow as the various activities that must be carried out in a specific order to accomplish a set business/organizational objective.

A Sample Workflow

For example, a common activity in most organizations is cash requisition and reporting expenses; and this process can be regarded as a workflow. This workflow consists of a series of steps such as these:

1. An employee makes a request: *Initiation.*
2. Her supervisor approves the request.
3. Finance gives her an advance for the amount requested.
4. The employee makes the purchase or expense.
5. The employee reports the expense and attaches a receipt.
6. The supervisor approves the report.
7. Finance reimburses the employee or receives the balance of the advance.
8. Finance updates the accounting system with appropriate accounting entries.
9. The expense report is signed as treated by finance and filed: *Completion.*

The workflow just described can be documented in Cash Advance Requisition and Expense Retirement forms similar to the ones that follow.

MSB Corporation Expense Retirement			
Report Date			
Employee Name			
Department			
Expense Details			
Reason for Expenses			
Expense Breakdown			
S/N	Description	Receipt Attached (Y/N)	Amount
1			
2			
3			
Total Amount Spent			
Amount Advanced			
Amount to be reimbursed		Amount to be refunded	
Approval			
Prepared By		Signature	
Approved By		Signature	
Reimbursement/Refund Details (Finance)			
Amount Refunded		<input type="checkbox"/> Cash <input type="checkbox"/> Cheque	
Amount Reimbursed		<input type="checkbox"/> Cash <input type="checkbox"/> Cheque	
Transaction Reference			
Processed by		Finance Officer Signature	

Sample expense retirement form

MSB Corporation Cash Advance Requisition			
Request Date			
Employee Name			
Department			
Expense Details			
Reason for Expenses			
Requested Amount			
Approval			
Requestor Name		Requestor Signature	
Approved By		Approver Signature	
Disbursement Details (Finance)			
Amount Advanced		Date Advanced	
Disbursed by		Finance Officer Signature	
Acknowledgement			
<p>I agree to account for this advance within ten working days of the date advanced as defined in the preceding section, either with adequate receipts, cash or a check for the balance made payable to MSB Corporation. I understand that my failure to account for advanced funds in full within sixty days will result in a Payroll deduction for the balance due. By signing below, I agree to allow MSB Corporation to make any such deductions from my pay.</p>			
Received by		Receiver's Signature	

Sample cash advance requisition form

What Is a Business Process?

Before we proceed further, I would like to also draw your attention to another term commonly used to describe workflows: *business processes*. SearchCIO defines a business process (<http://searchcio.techtarget.com/definition/business-process>) as “an activity or set of activities that will accomplish a specific organizational goal. *Business process management (BPM)* is a systematic approach to improving those processes.”

You can see that this definition of a business process is very similar to the definition of a workflow. The definition also introduces a new concept: *business process management*, which we will explore later. This shows that a workflow in our context is basically a business process, and this book will often use the terms interchangeably.

Returning to our example of the Expense Reporting business process, can you think of a number of problems that might be encountered in the day-to-day utilization of the process in your organization? A few that readily come to mind are these:

1. How do you ensure that employees fill out all required fields on the form?
2. How can you enforce that Finance does not treat any request without a properly filled out and approved form or avoid requests being sent by email that omit important details?
3. How do you ensure that supervisors do not approve advances above their authorized limits?
4. How can you easily track expenses based on approved budgets?
5. How do you make sure that reports must contain a receipt or other supporting documents?

Do any of these questions above resonate? If your organization is also environmentally conscious, a key concern would be reducing the amount of paper utilized. It is questions like these that business process management (BPM) attempts to address. So let us look at a couple of definitions of BPM.

What Is BPM?

According to AIIM (<http://www.aiim.org/What-is-BPM>), BPM refers to how we study, identify, change, and monitor business processes to ensure that they run smoothly and can be improved over time.

In a similar vein, SearchCIO describes BPM (<http://searchcio.techtarget.com/definition/business-process-management>) as “a systematic approach to making an organization’s workflow more effective, more efficient and more capable of adapting to an ever-changing environment.”

From this definition, we see that BPM is a way of improving our workflows or business processes—that is, standardizing them, eliminating inefficiencies, and positioning them to generate more business value. We can also see that it is not a one-off activity, but rather a cycle of activities:

1. Identify what objectives need to be achieved.
2. Define the tasks or activities required to achieve them.
3. Standardize these activities into a process.
4. Utilize the process within the organization.
5. Observe how the process is being used.
6. Identify the bottlenecks and problem areas.
7. Redesign the process to eliminate the bottlenecks and remove the identified problems.

We then repeat the cycle of observation, identification, and modification adapting to the changing demands of the organization and its stakeholders while ensuring that the best value is delivered by the business process.

BPM often consists of two parts, the methodology and the software. What we have described is the methodology, and you will agree that doing this without any form of automation would be a herculean task. BPM software enables us to automate the process of defining and creating workflows, monitoring and optimizing them.

Early in my professional career, I was fortunate to work in an organization that utilized BPM software in its day-to-day operations. While I knew that it made the organization and its employees more efficient and productive, I did not really grasp the magnitude of the benefits until I worked with other organizations that did not use BPM.

Valuable human hours were spent on mundane tasks such as capturing data from paper forms into Excel to generate management reports, verifying that forms were filled correctly, or searching storage archives looking for supporting documents for an approved request so they can be scanned and emailed to auditors, to mention a few.

You might be wondering, “all this sounds good, but BPM software is way too expensive for us to afford and too complex for our team to master,” and my response will be “you are both right and wrong.” A lot of BPM solutions are quite expensive, costing hundreds of thousands of dollars and requiring very complex procedures to set up and build processes with.

However, there are also free or open source solutions that are quite easy to set up and can be used to begin the first steps toward building and optimizing your organization’s business processes without costing an arm and a leg. One such solution is ProcessMaker, and my goal is to help you see how you can use it to build and automate your first business process, deploy it to your organization, and become proficient in implementing BPM in your organization.

ProcessMaker also has a paid enterprise edition that comes with support and additional features, but we will be focusing on the free open source edition, which I think is more than enough to help any organization start its BPM journey.

What Is ProcessMaker?

ProcessMaker is an open source workflow and business process management solution developed by ProcessMaker Inc., previously known as Colosa Inc. ProcessMaker allows you to model and build your business processes and workflows easily using an intuitive and easy-to-learn designer. It also allows you to run these processes with a user-friendly interface for the users within your organization. ProcessMaker helps businesses improve their efficiency by automating the flow of data, forms, and information across the organization.

I first discovered ProcessMaker in 2010, and one of the first things that attracted me to it as my preferred choice of a BPM solution was the simplicity and how quickly I was able to build a process without any previous training. Over the years, the product has improved significantly, with new features added with every new release.

ProcessMaker is a web-based application, meaning that all your end users require only a modern browser to access the application. ProcessMaker also comes with mobile apps for iOS and Android, allowing you to access your business processes on the go.


The application is built on the AMP (Apache, MySQL, and PHP) stack and runs on both Unix and Windows operating systems. ProcessMaker also comes with a full-featured REST API, which makes integration with other applications a breeze.

Rather than bore you with technical jargon, I think the best way to learn about ProcessMaker is to get our hands dirty, exploring it and building something with it. Fortunately for us, the folks at Bitnami (<https://bitnami.com>) have put together a Bitnami ProcessMaker Stack for the three common operating systems: Windows, MacOS (also called OS X), and Linux, allowing us to dive into ProcessMaker without having to deal with setting up a web server and database ourselves on our systems.

According to the Bitnami documentation, a Bitnami Stack is

an integrated software bundle that includes a web application and all of its required components (web server, database, language runtime), so it is ready to run out of the box. The Stacks can be deployed as traditional Native Installers, Virtual Machine Images or Cloud Images.

In this chapter, we defined a workflow and learned about business processes and business process management in general. We also introduced ProcessMaker as a powerful open source BPM solution that you can use to automate your business processes. In the next chapter, we will install ProcessMaker using the Bitnami Stack installer, explore the interface, and introduce some basic ProcessMaker concepts.

 Bitnami Installations Bitnami installations are generally only used for testing and developing processes. A manual installation is recommended when using ProcessMaker in production, because Bitnami installations cannot be upgraded to later versions, and they are generally slower and use more resources than manual installations.

CHAPTER 2

Getting Started with ProcessMaker

The following steps will walk you through installing ProcessMaker on your Mac or Windows computer using the Bitnami installer. Again, the purpose of this installation is simply to get started quickly exploring the features of ProcessMaker. When deploying ProcessMaker to production in your organization, you should instead deploy it on a server either on-premise or in the cloud using the manual installation. Later chapters in this guide will show how to install ProcessMaker manually on a cloud server, secure it with SSL, and access it using the ProcessMaker mobile app.

To begin, head over to <https://bitnami.com/stack/processmaker/installer> to download the ProcessMaker Open Source Edition installer for your system.


Windows

Version	Size	Checksum		
 ProcessMaker Open Source Edition 3.1-0 (64-bit)	141 MB	show	Download	Recommended
ProcessMaker Open Source Edition 3.1-0	138 MB	show	Download	

OS X

Version	Size	Checksum		
 ProcessMaker Open Source Edition 3.1-0 (64-bit)	116 MB	show	Download	Recommended

Linux

Version	Size	Checksum		
 ProcessMaker Open Source Edition 3.1-0 (64-bit)	120 MB	show	Download	

Bitnami ProcessMaker Installer page

We will use the Open Source edition for this book. To learn more about the difference between the Open Source and Enterprise editions, you can visit http://wiki.processmaker.com/ProcessMaker_Enterprise_Editionv.3.0.

On the Bitnami ProcessMaker Installer page, click the Download button for your operating system. You will be presented with a prompt requesting you to log in to Bitnami. This is optional, and you can proceed by clicking the “No thanks, just take me to the download” link at the bottom of the pop-up modal. If you are interested in learning more about Bitnami, you can log in with any of the social media accounts displayed.

Installation Steps

Once the download is completed, double-click the downloaded installation file to begin the installation.

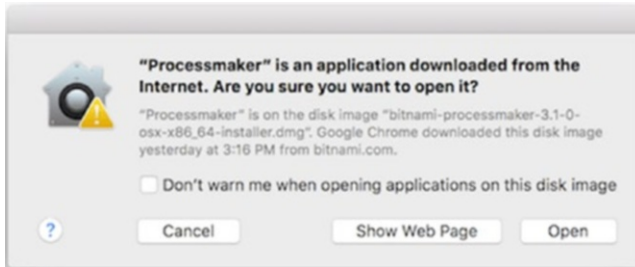
For Mac OS X Users

MacOS (OS X) users should take the following steps to install ProcessMaker:

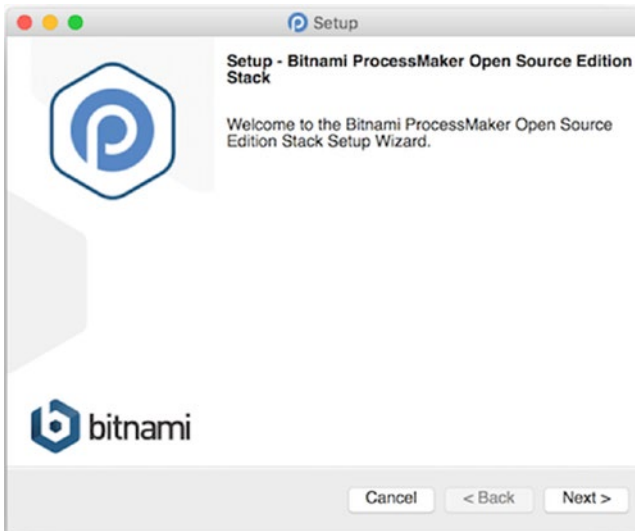
1. Double-click the `bitnami-processmaker-3.x-x-osx-x86_64-installer.dmg` file.
2. The following screen is displayed, requesting you to double-click to install.



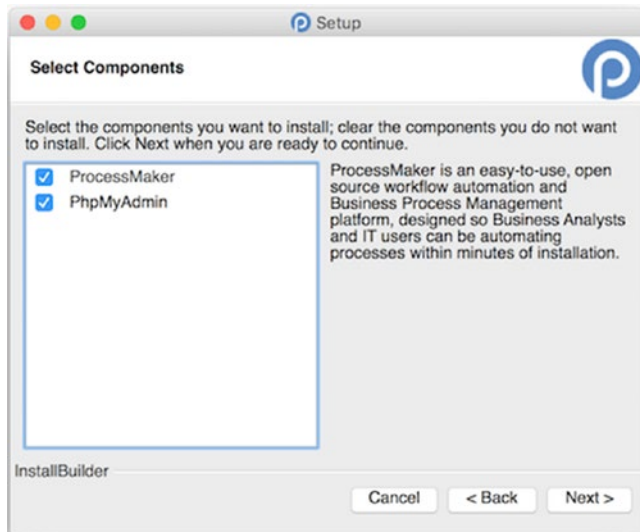
3. If you see a warning sign like the following, click Open to continue.



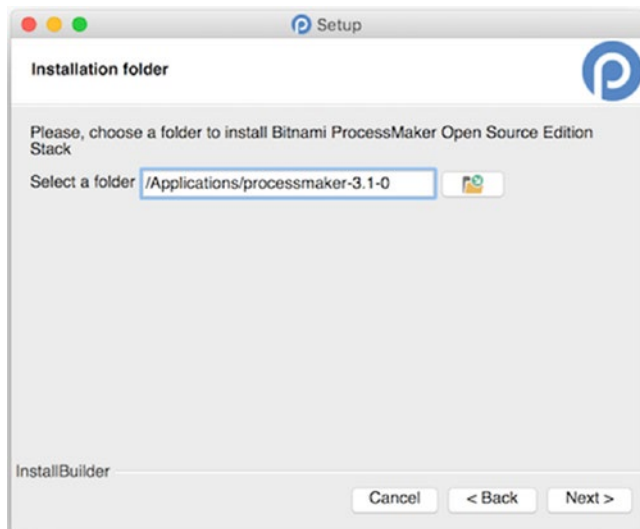
4. You should now see the ProcessMaker setup wizard. Click the Next button to walk through the wizard and complete the installation.



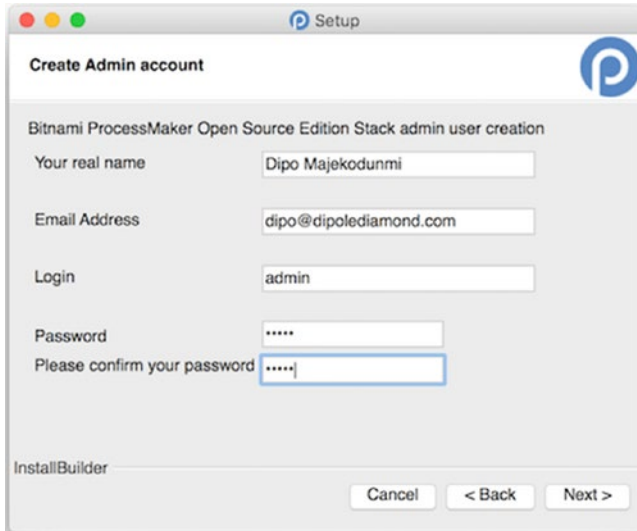
- a) In the next screen, Select Components, leave the ProcessMaker and PhpMyAdmin options checked. PhpMyAdmin is a web application that provides you with a graphical user interface (GUI) that will allow you manage the MySQL database that will be installed by the installer.



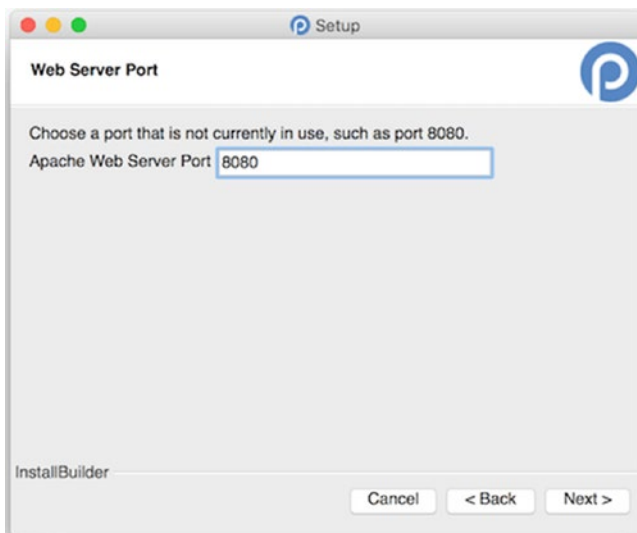
- b) In the next screen, Installation Folder, you can choose a location to install the application or leave the default selection.



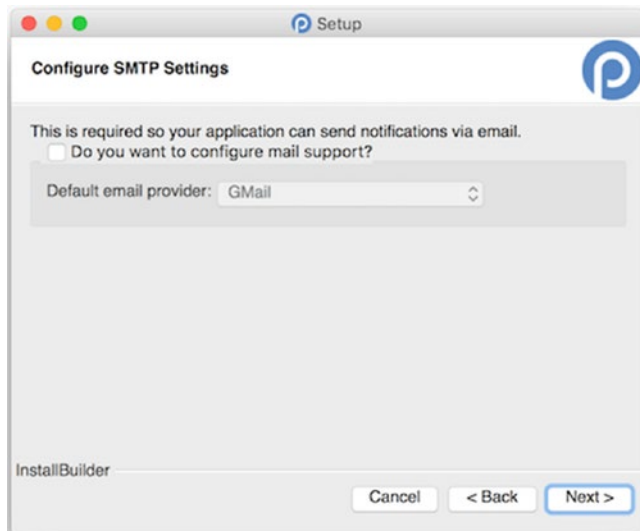
- c) In the Create Admin Account screen, fill in the required details for the admin account. Be sure to remember the password entered here, as it is the password for the admin account and the database root account. (In MySQL, the root account is the super user for the database.)



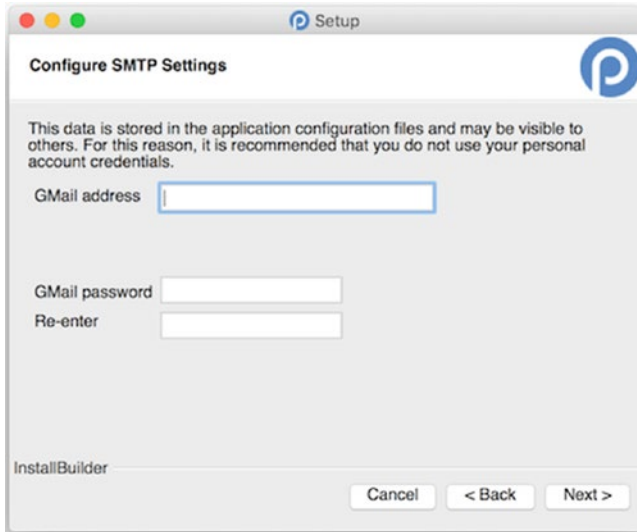
- d) In the next screen, Web Server Port, enter a port number to use. You can leave the default port selected by the installer. The next screen after that prompts for the SSL port. Leave the default selected and click Next.



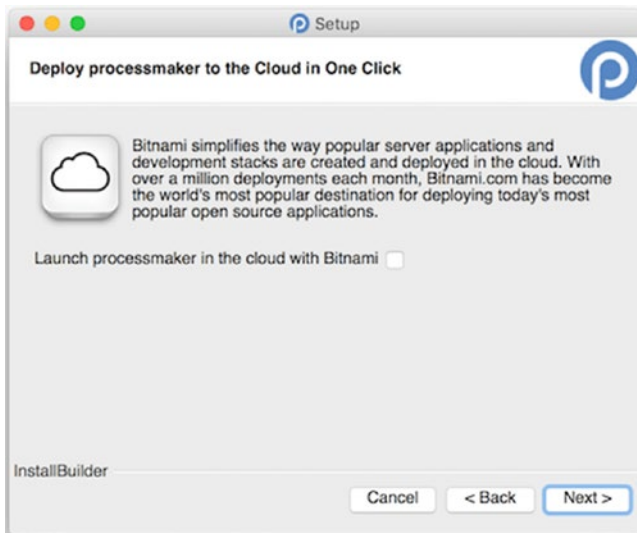
- e) In the next screen, Configure SMTP Settings, you can choose to allow ProcessMaker to send emails. Though this setting is optional, I recommend checking the box “Do you want to configure mail support?” With this box checked, select Gmail as the default email provider if you have a Gmail account. This is the easiest option, and you can set up a generic Gmail account for the purpose of this guide. Alternatively, if you do not have a Gmail account, select Custom and click Next.



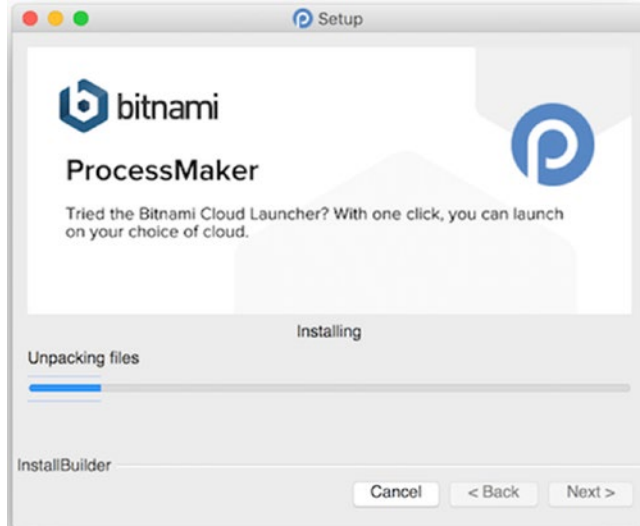
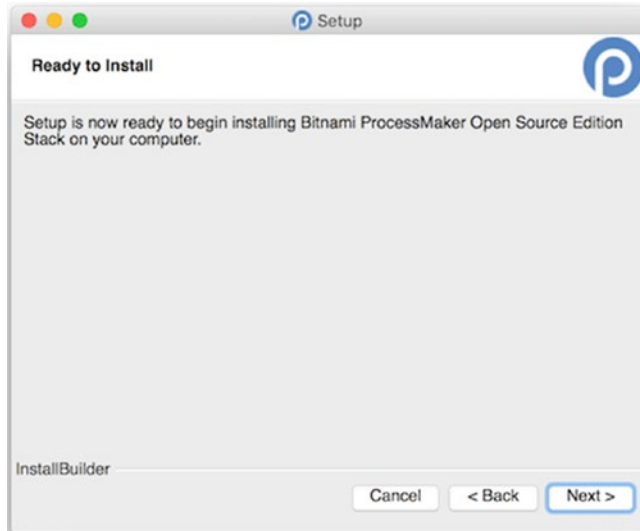
- f) If you selected Gmail, the next screen prompts for your Gmail address and Password. If you selected Custom, the next screen prompts you to enter your SMTP configuration details. You can get this information from your email service provider. When done, click Next.

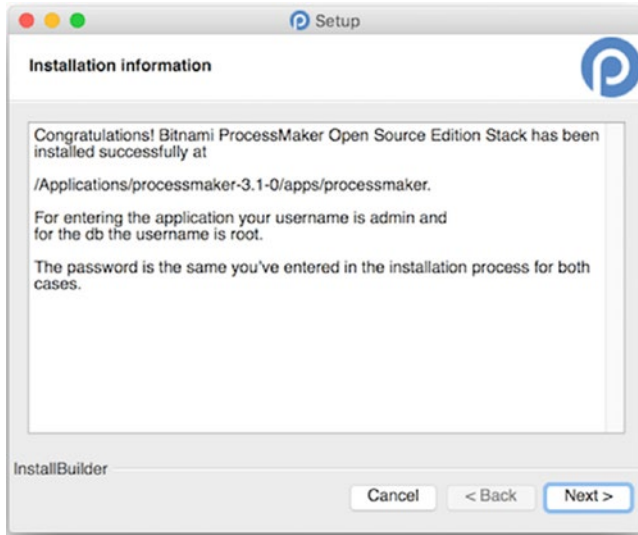


- g) In the next screen, Deploy ProcessMaker to the Cloud in One Click, uncheck "Launch ProcessMaker in the Cloud with Bitnami" as we will be working with a local installation. Click Next.

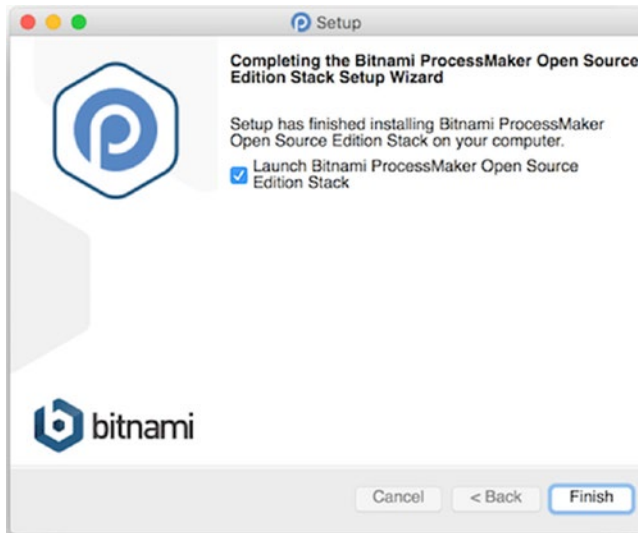


- h) In the next screen, Ready to Install, you are all set for the automated installation. Click Next and wait for the installer to complete.

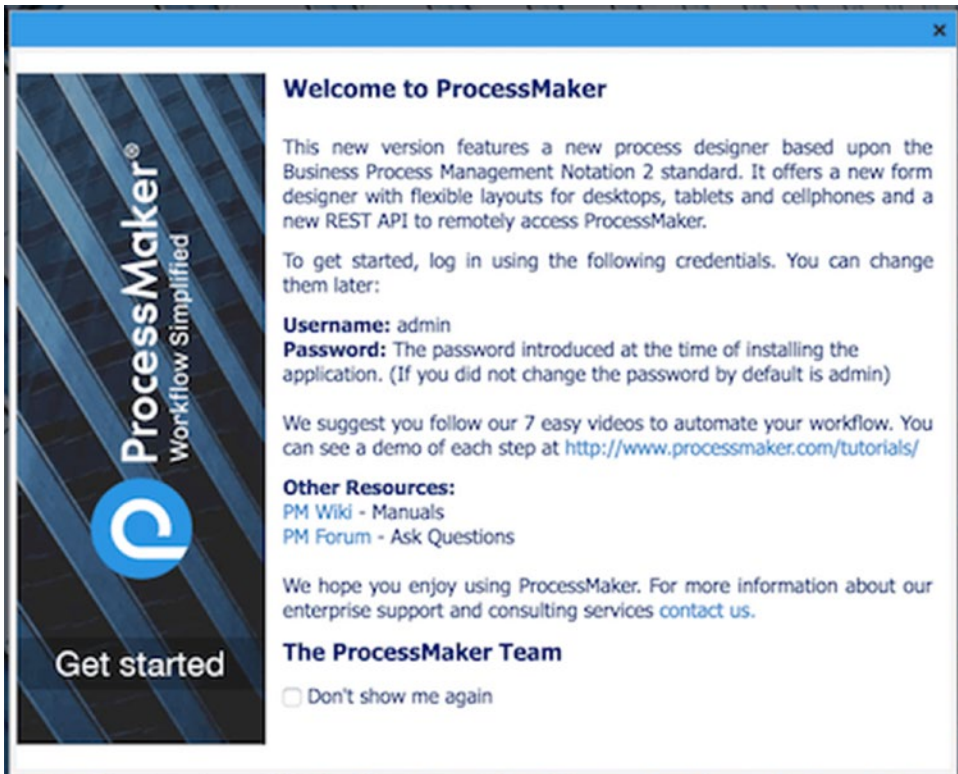





5. Once the installation is completed, the wizard shows you the option to “Launch Bitnami ProcessMaker Open Source Edition Stack.” Leave the option checked and click the Finish button.



The installer should launch your default browser and display the following screen. You have now successfully set up ProcessMaker on your Mac. Check “Don’t show me again” at the bottom of the Welcome to ProcessMaker pop-up and close it to display the login form.



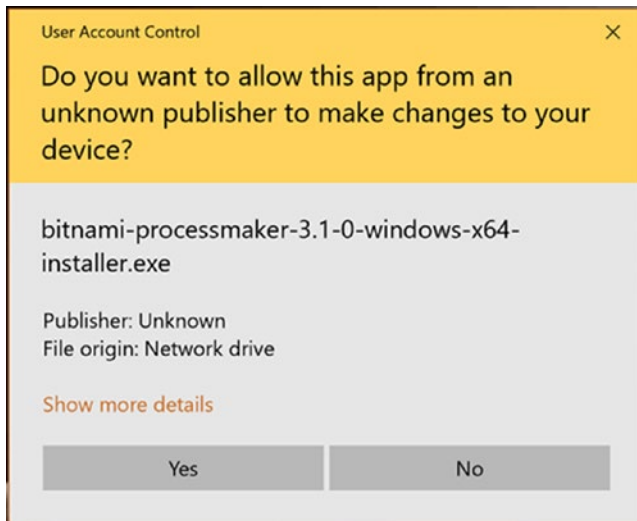
Also, you will notice in the Docker a new icon  which opens the Bitnami Application Manager, discussed later in this chapter. The Application Manager is where you can launch the application, PhpMyAdmin, configure and manage the bundled MySQL and Apache Servers, and view the server events, which can be useful when troubleshooting issues with starting the servers.

ProcessMaker is now installed on your Mac. You can skip the instructions for the Windows installation and move to the next section.

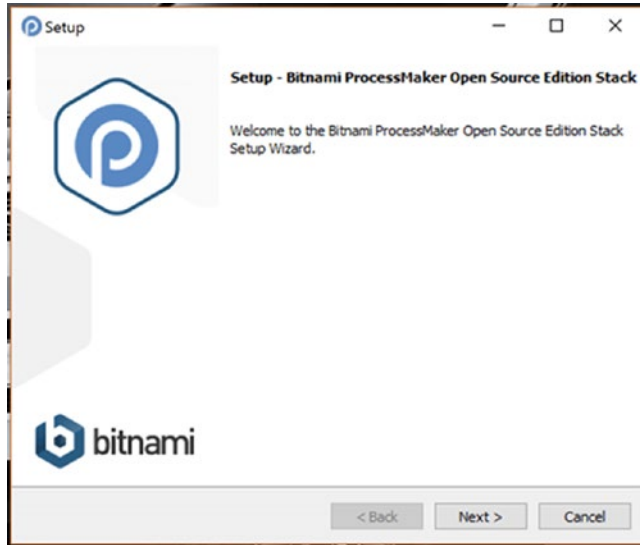
For Windows Users

Windows users should take the following steps to install:

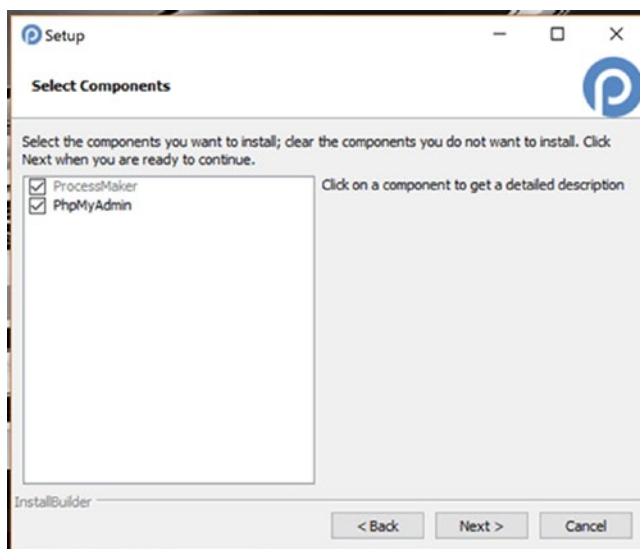
1. Double-click the downloaded installer (bitnami-processmaker-3.x-x-windows-x64-installer for 64-bit) file to launch the installation wizard.
2. If you see a warning from the User Account Control dialog like the following, select Yes.



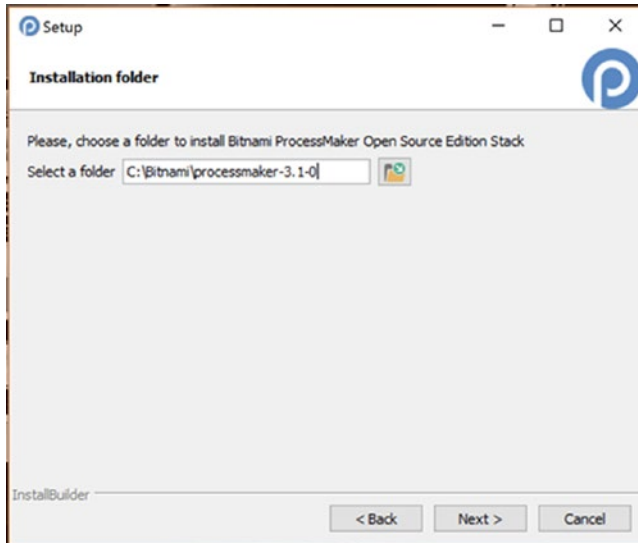
3. You should now see the ProcessMaker setup wizard, shown following. Click the Next button to walk through the wizard and complete the installation.



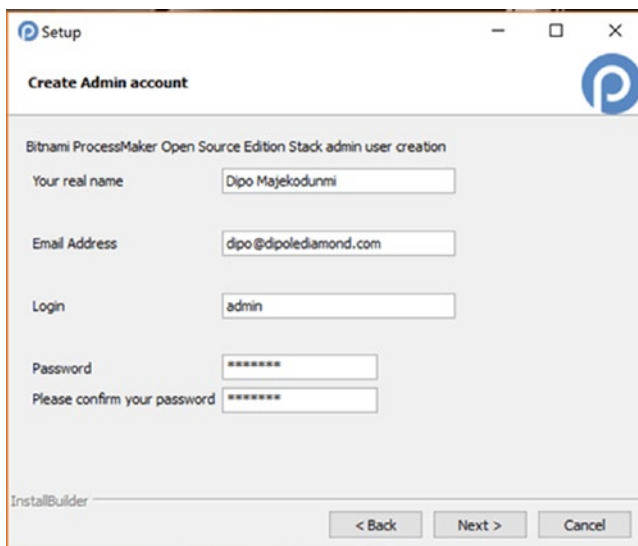
- a) In the next screen, Select Components, leave the ProcessMaker and PhpMyAdmin options checked. PhpMyAdmin is a web application that provides you with a graphical user interface (GUI), with which you manage the MySQL database that will be installed by the installer.



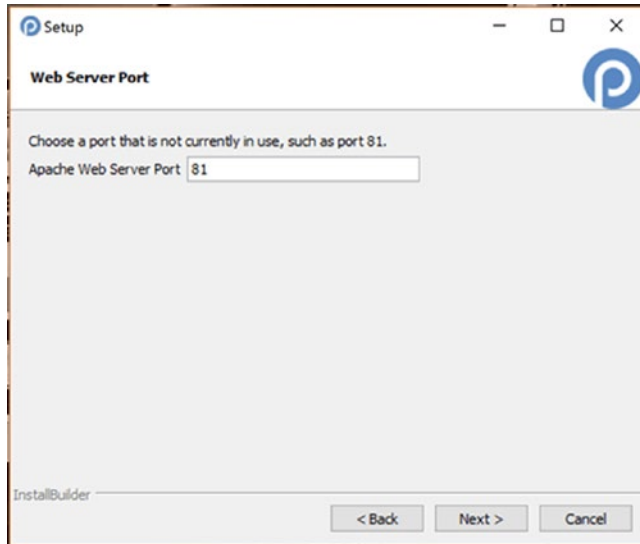
- b) In the next screen, Installation Folder, you can choose a location to install the application or leave the default selection.



- c) In the next screen, Create Admin Account, fill in the required details for the admin account. Be sure to remember the password entered here, as it is the password for the admin account and the database root account. (In MySQL, the root account is the super user for the database.)

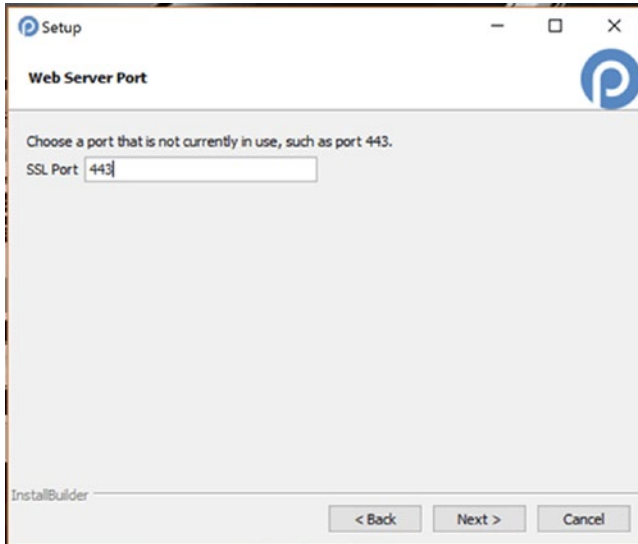


- d) In the next screen, Web Server Port, enter a port for the web server to listen on. This is port 80 by default. If you already have an application on your system listening on that port, you will be shown a different port number. The important thing is to select a free port number.

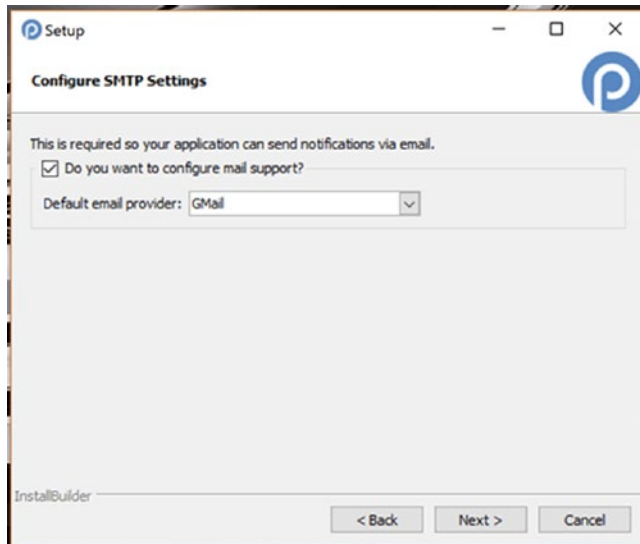


Using the default port allows you access the web server without having to specify a port number (`http://localhost/...`) but with a non-default port number, the URL will have to be suffixed with the port number when accessing the web server from the browser (for example, `http://localhost:81/...` if using port 81 as in the previous screenshot)

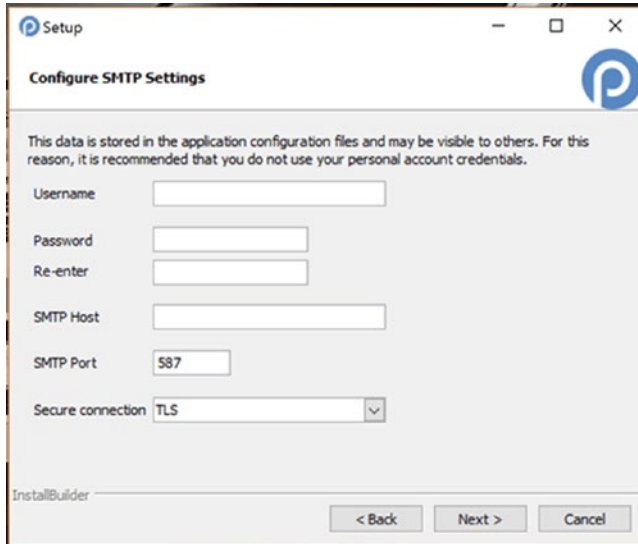
- e) In the next screen, Web Server Port, you select the SSL port. You can leave the default 443, unless you already have an application listening on that port; if so, you can change it to a free port as we did in the previous step. An alternative value is 8443.



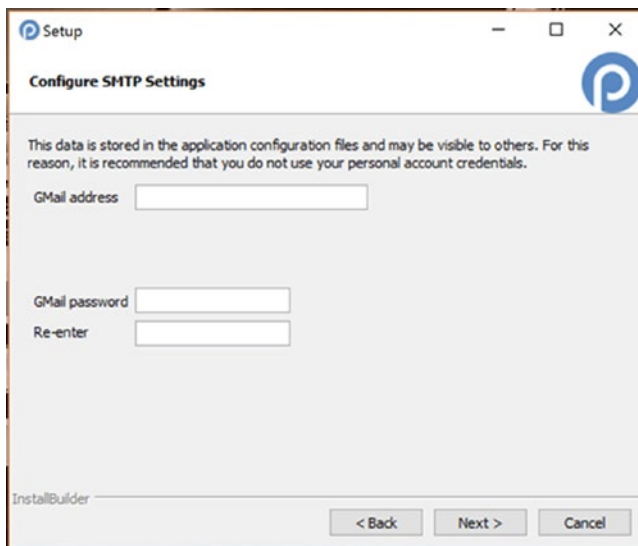
- f) In the next screen, Configure SMTP Settings, you can choose to allow ProcessMaker to send emails. Although this setting is optional, I recommend checking the box “Do you want to configure mail support?” With this box checked, select Gmail as the default email provider if you have a Gmail account. This is the easiest option, and you can set up a generic Gmail account for the purpose of this guide. Alternatively, if you do not have a Gmail account, select Custom and click Next.



- g) If you selected Gmail, the next screen prompts for your Gmail address and Password. If you selected Custom, the next screen prompts you to enter your SMTP configuration details. You can get this information from your email service provider. When done, click Next.

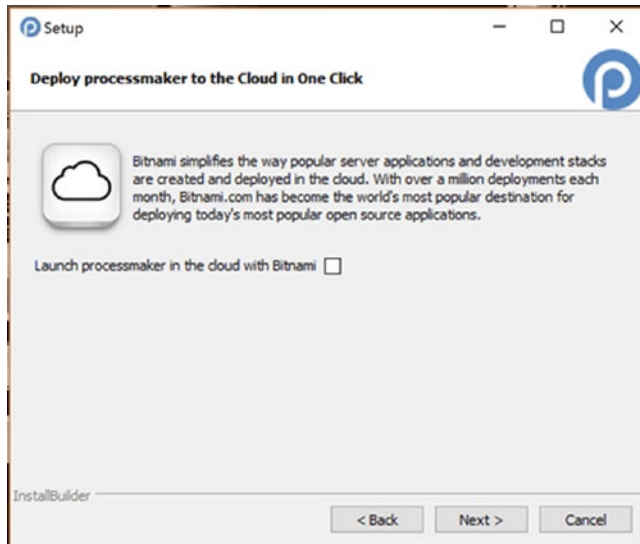


Custom SMTP configuration

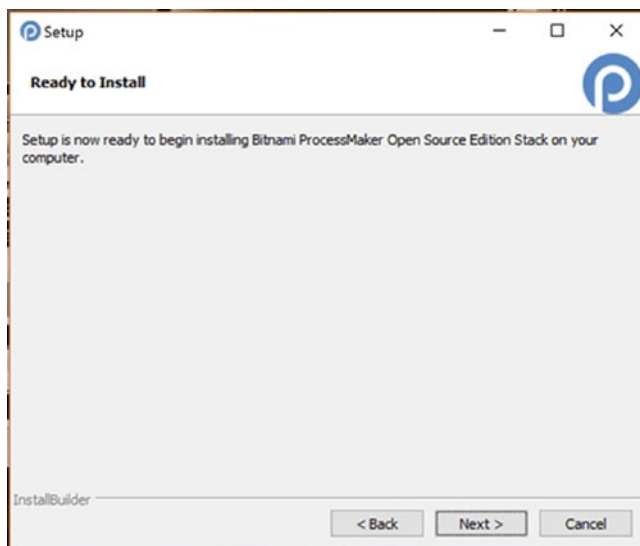


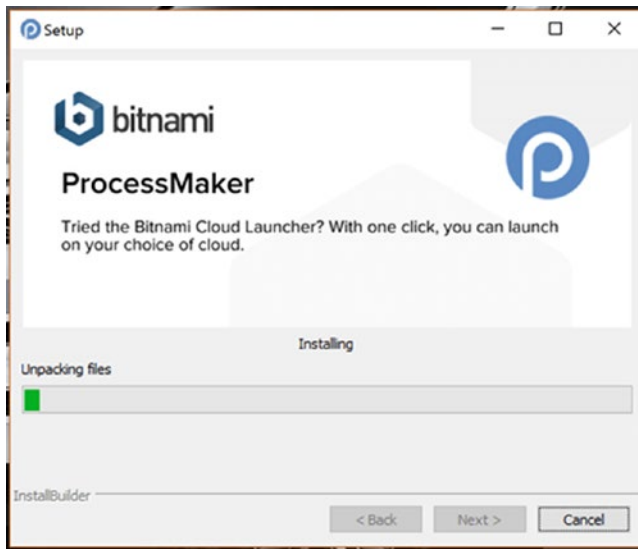
Using Gmail configuration

- h) In the next screen, Deploy ProcessMaker to the Cloud in One Click, uncheck the option to Launch ProcessMaker in the cloud with Bitnami, as we will be installing and learning locally. Click Next.

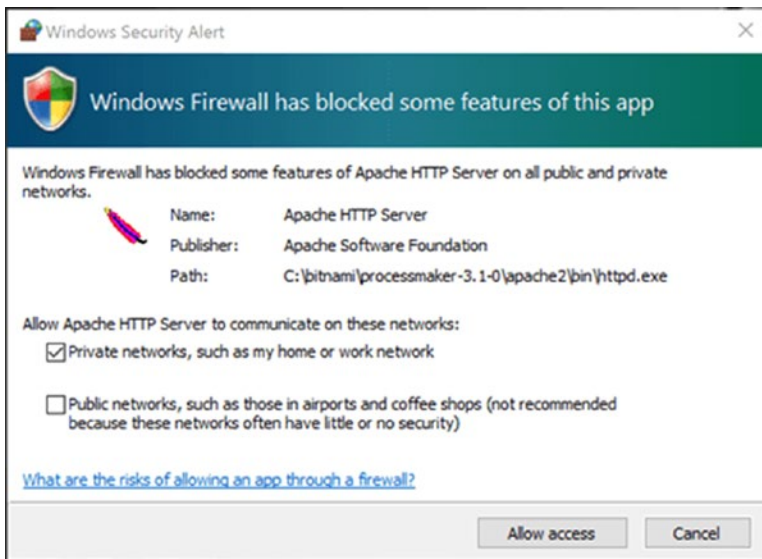


- i) In the next screen, Ready to Install, you are all set for the automated installation. Click Next and wait for the installer to complete.

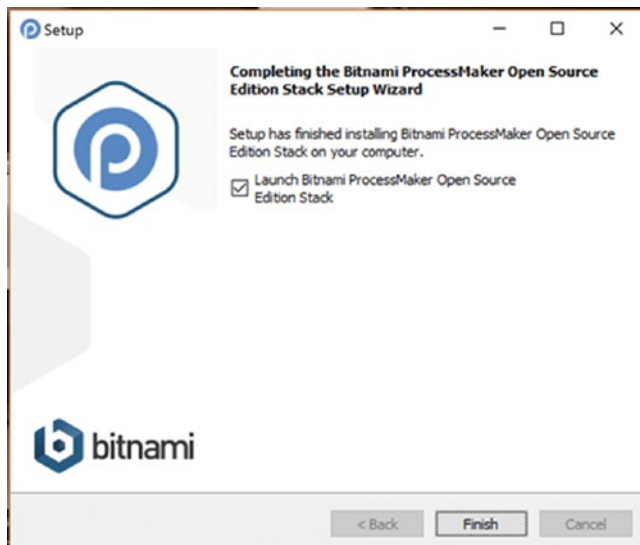
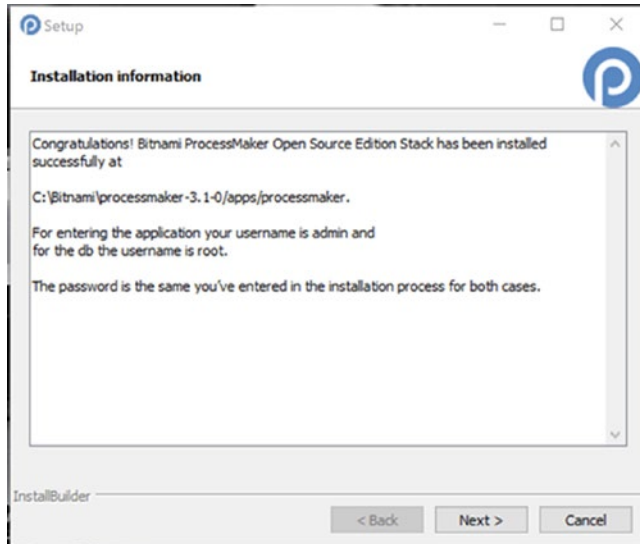




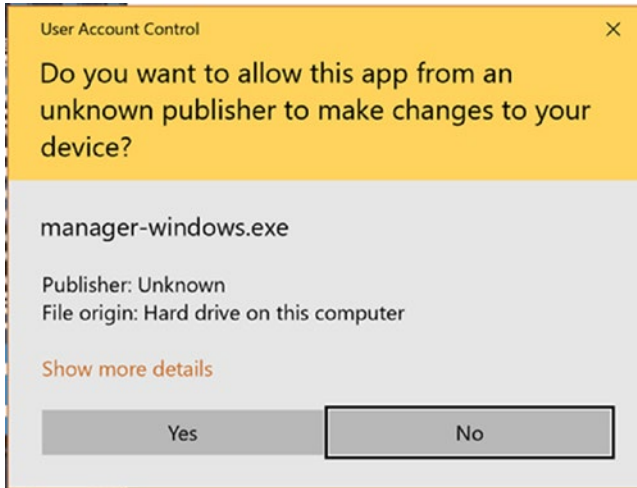
- j) If you are prompted with a firewall warning like the following, click Allow Access to grant ProcessMaker permission through the firewall.



4. Once the installation is completed, you will see the Installation Information screen showing you information about the installation. Click Next to proceed to the last screen. Leave the checkbox for “Launch Bitnami Open Source ProcessMaker Stack” checked and click the Finish button to complete the installation.




5. If prompted, select a browser, preferably Chrome or Firefox. You might also see a User Account Control dialog warning asking you to allow manager-windows.exe to make changes to your computer. Select Yes.



6. You should now see a Welcome to ProcessMaker modal displayed in your browser. Check the “Don’t show me again” box at the bottom of the modal and close it to display the login page.



Also, you will notice a new icon  in the task bar, which opens the Bitnami Application Manager. The application manager is where you can launch the ProcessMaker application, PhpMyAdmin, configure and manage the bundled MySQL and Apache Servers and view the server events, which can be useful when troubleshooting issues with starting the servers.

The Bitnami Application Manager

The ProcessMaker installation process just described also installs the Bitnami Application Manager, which you can use to launch ProcessMaker should you restart your PC.

To launch the Application Manager on Mac OS X, simply type **manager-osx** into Spotlight to find it and click on it. If you have more than one Bitnami installation on your Mac, look for the one with a description matching the ProcessMaker version installed. You can also go to the folder you selected during the install process above or search Applications to launch it.

To launch the application manager in Windows, simply look for the Bitnami ProcessMaker folder in the Windows Start menu. Expand it and select the Bitnami ProcessMaker Open Source Edition Stack Manager Tool.

The Application Manager has three top menu buttons : Welcome, Manage Servers, and Server Events.

The Welcome Screen

The components of the Welcome screen are as follows.



Welcome screen of the Application Manager

Go to Application: This is used to launch the default browser to open the ProcessMaker login page. To launch ProcessMaker, ensure that the web server and database services are up and running. You can click the Manage Servers button on top of the Application Manager interface to check the status of the services.

Open phpMyAdmin: This is used to launch the phpMyAdmin application in the default browser. phpMyAdmin is a web GUI for administering MySQL databases.

Open Application Folder: This opens the folder where ProcessMaker is installed on your system.

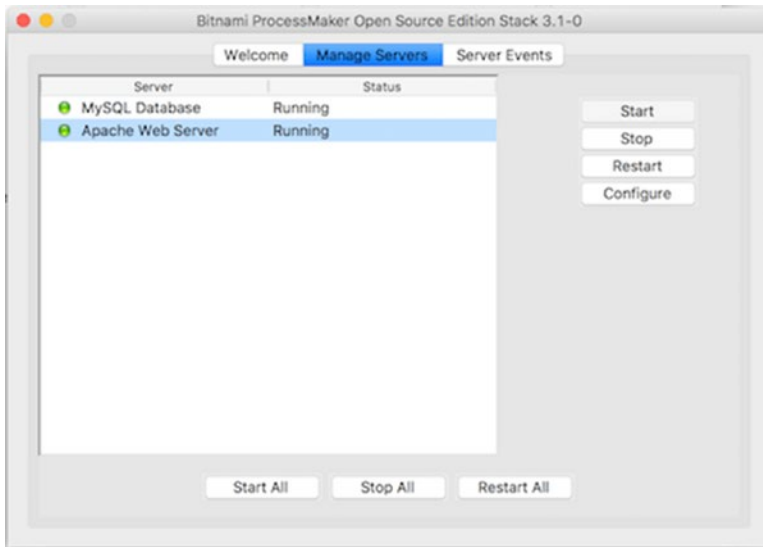
Visit Bitnami: This opens the Bitnami website in your browser. You can use this to learn more about Bitnami and their offerings.

Get Support: This opens the Bitnami documentation (<https://wiki.bitnami.com>) in your browser. You can find detailed information about the different Bitnami stacks. For example, details of the Bitnami ProcessMaker stack information can be found here ([https://wiki.bitnami.com/Applications/Bitnami_ - ProcessMaker](https://wiki.bitnami.com/Applications/Bitnami_-_ProcessMaker)).

The Manage Servers Screen

The Manage Servers screen provides an interface for configuring, starting, stopping and restarting the MySQL database and Apache web server on which ProcessMaker runs. You can select a server by clicking on it and then use the buttons on the right to start, stop, restart or configure it.

The Configure button allows you to change the ports of the servers and edit their config files. At the bottom of the screen are the Start All, Stop All, and Restart All buttons, which allow you to start, stop, or restart both servers at once. The indicators are green when the servers are running and red when they are stopped.

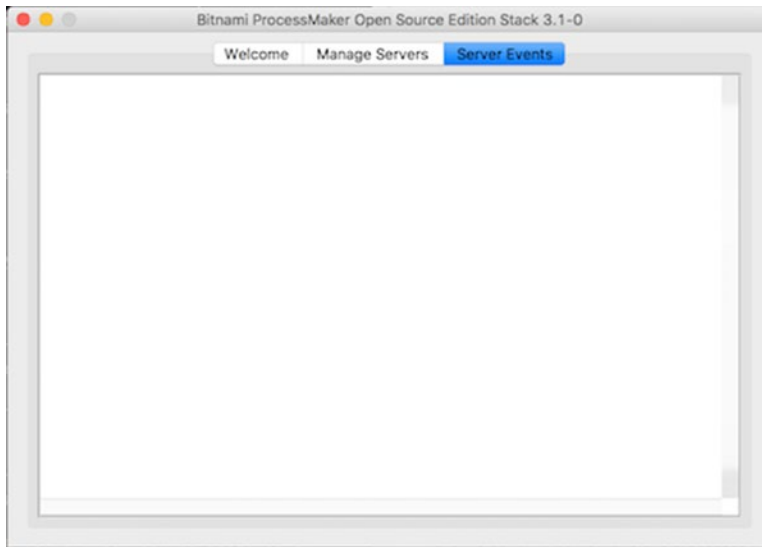


Manage Servers screen of the Application Manager

The Server Events Screen

The Server Events screen logs the status messages of the MySQL database and Apache web server as they are stopped and started. The messages logged can provide useful information for troubleshooting issues when the web server or database fails to start or stop.

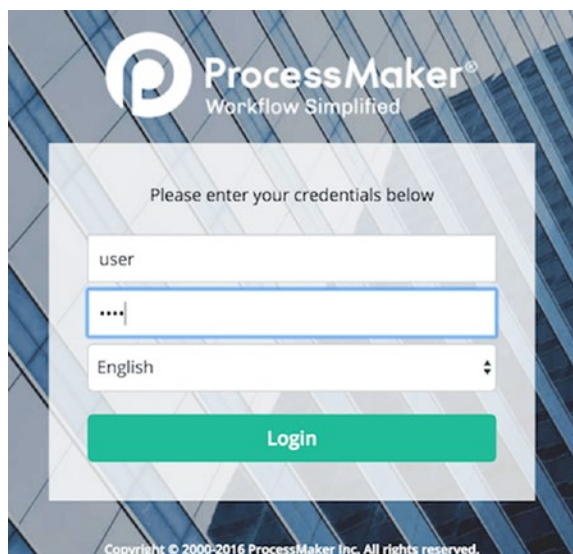
To prevent the ProcessMaker installation from using resources on your system when it's not in use, it is highly recommended that you stop the MySQL Database and Apache Web Server services from the Bitnami Application Manager.



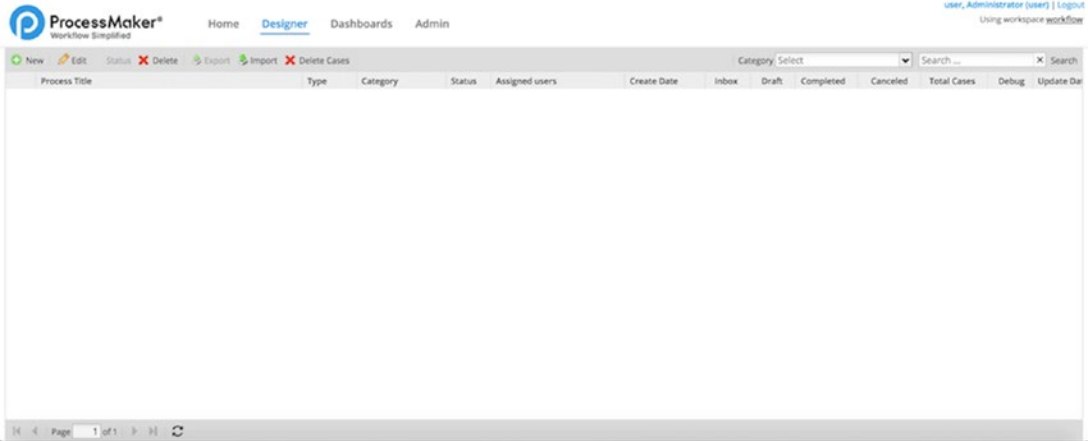
Server Events screen of the Application Manager

Exploring the ProcessMaker Interface

Now that you have ProcessMaker installed, let us dive in to see how it looks and how it works. In the login screen displayed (you might need to close the Welcome pop-up screen to see the form), enter the username and password you used when installing the application and click the Login button. Remember, you can always use the Bitnami Application Manager to launch ProcessMaker later if you close your browser or restart your system.



On successful login, the following screen is displayed. You can see that the Designer main menu is selected, and the Process List is empty; we currently do not have any processes in our installation.



We will begin our exploration of ProcessMaker from the Designer, which is where you will model, build, and configure your first ProcessMaker business process. It is highly recommended to use a modern browser for the following chapters. ProcessMaker currently supports the following browsers as of versions 3.1, 3.1.1 and 3.1.2:

Internet Explorer	Google Chrome	Mozilla Firefox	Microsoft Edge
IE 11	Two latest stable releases	Two latest stable releases	Two latest stable releases

ProcessMaker Concepts

In our exploration of the ProcessMaker interface over the course of this book, you will encounter the following terms used to explain the features. I will end this chapter with a set of working definitions you can use to understand what the terms mean as we progress.

Process: This is a representation of a business process, such as “Employee Leave Request.” It consists of tasks, which accept input and produce an output.

Case: A *case* is an instance of a process. For example, a leave request for John Doe for 5 days is a case. It is an instance of the Employee Leave Request *Process*.

Task: A *task* is a sequence of logically related steps carried out in a process. In ProcessMaker, a task is made up of steps, conditions, input or output documents and triggers. A sample task in the Employee Leave Request process will be “Apply for leave.”

Steps: A *step* is a piece of work that forms a clearly defined action. It could be filling a Dynaform or uploading a document.

Triggers: Custom code to perform specific business logic and add additional functionality to processes.

Conditions: *Conditions* can be defined to skip specific steps or triggers. The conditions are evaluated on a case by case basis.

Dynaforms: *Dynamic Forms* are the custom forms which can be designed in ProcessMaker to capture data from the user while running a case.

Input and output documents: When executing a case, users can upload attachments (*input documents*) and ProcessMaker can generate formatted .pdf or .doc documents (*output documents*) using values captured or computed during a case.

Routing: This determines which task (or tasks in parallel) should be done next by evaluating the set of defined conditions.

Assignment: This determines which user or group of users should carry out a task after it has been routed. This is determined by assignment rules and users/groups assigned to a task.

Process map: A *process map* is a dynamic visual representation of the tasks and derivation rules associated with a business process. It is the workflow diagram.

Plugin: This refers to modules that extend ProcessMaker features.

Role: A *role* is a set of permissions to access specified functionalities and resources in ProcessMaker.

Group: *Groups* are a way to organize users and to simplify the assignment of tasks to multiple users. Groups can also be used to assign process permissions.

CHAPTER 2 GETTING STARTED WITH PROCESSMAKER

In this chapter, we set up a local installation of ProcessMaker using Bitnami Stack installers and learned about the Bitnami Application Manager for managing the installation. We also introduced ProcessMaker concepts that will aid us in our understanding of the topics we will be covering in the following chapters. In the next chapter we explore the ProcessMaker Workflow Designer and learn about the tools we can use in modeling a business process.

CHAPTER 3

The ProcessMaker Workflow Designer

ProcessMaker Designer is a BPMN 2.0-compliant process designer with drag-and-drop functionality that lets you easily model your business processes, create forms for capturing data when running the process, set up business rules or conditions, and assign users to tasks making up the process, among other things.

If you are wondering what BPMN 2.0 compliance means, BPMN (Business Process Model and Notation) is a global standard for business process modelling, which provides a set of graphical notations for the specification of a business process in a Business Process Diagram. The current version of the standard is 2.0, and it is maintained by the Object Management Group (OMG). More information about BPMN 2.0 can be found at <http://www.bpmn.org/>.

BPMN 2.0 provides a uniform means for business owners to describe their business processes clearly, in a standard manner that can readily be understood without any familiarity with the nuances of the business or organization. By being BPMN 2.0 compliant, the ProcessMaker designer allows us to model a process in any BPMN 2.0-compliant application and import it into ProcessMaker, and vice versa.



Please note that the import and export between applications refers to the process diagram only and not the entire process itself.

When you click the Designer tab in the main menu, the Process List screen is displayed as shown in the final illustration of Chapter 2. This screen displays a paged list of all the processes currently in the system. At the moment we have none. The menu bar on the page gives us options for creating, editing, enabling or disabling, importing, exporting, deleting and searching for processes.



Process List menu bar

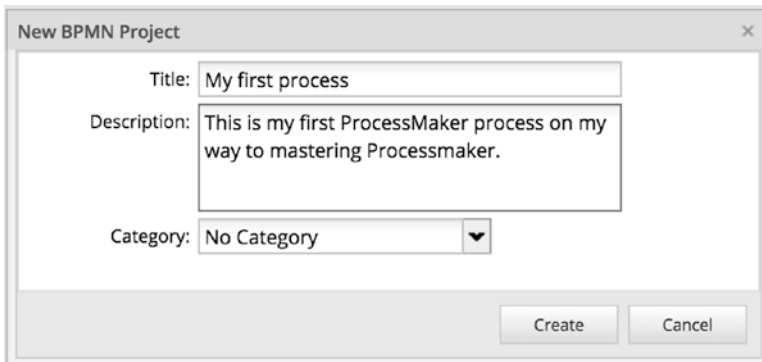
Process List Actions

We will briefly explore the available actions in the Process List as we proceed to create our first ProcessMaker process.

New

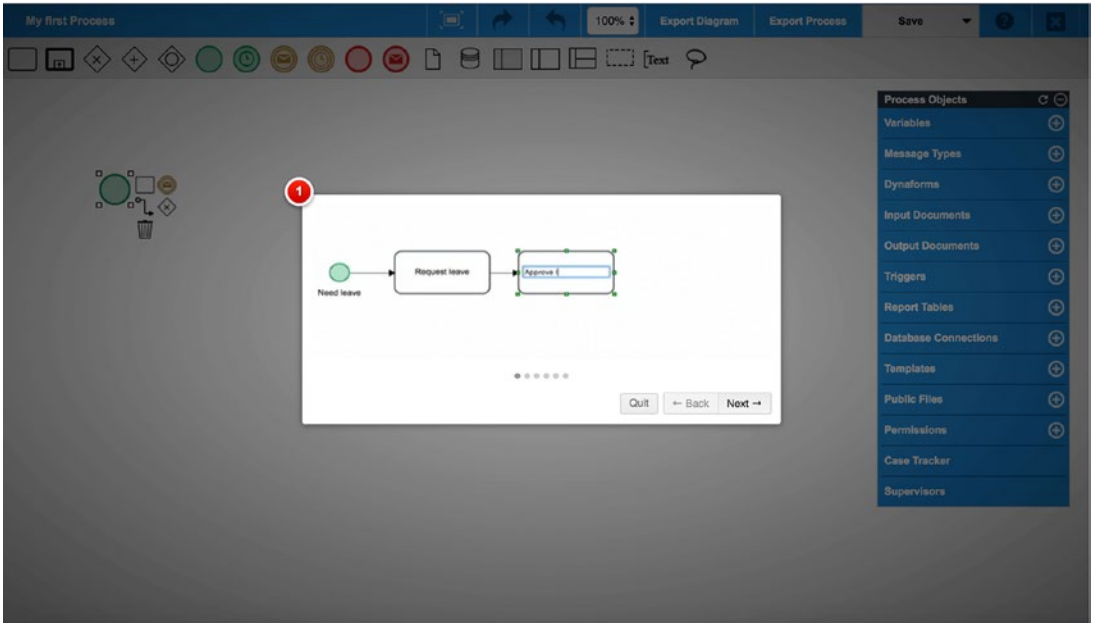
This option allows us to create a new project in ProcessMaker. (The term *project* is used, even though at this point we are creating only a single process, because a project can contain more than one process, such as a main process with sub-processes.) Let us see how this works:

1. Click the New button in the menu bar and the following modal pop-up is displayed.
2. Enter a name for the project and a description. Leave the category as No Category for now. You will see how to create categories when we explore the Admin features.



3. Click the Create button.

- The new project is created, and the Process Designer screen is displayed. The screen is overlaid with an introductory walkthrough that shows you the key features of the designer and what they do. Click the Next button on the walkthrough to quickly explore the designer, and click the Done button at the end.



- Click the Close button in the top-right corner to close the project and return to the list of processes. Your newly created process should now be displayed in the list.

ProcessMaker® Workflow Simplified

Home Designer Dashboards Admin

user, Administrator (user) | Logout
Using workspace workflow

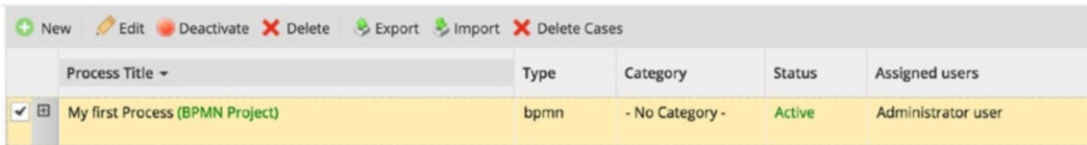
New Edit Status Delete Export Import Delete Cases

Category Select Search ... Search

Process Title	Type	Category	Status	Assigned users	Create Date	Inbox	Draft	Comple...	Canceled	Total Cases	Debug
<input type="checkbox"/> My first Process (BPMN Project)	bpmm	- No Category -	Active	Administrator user	2016-08-15 09:53:30	0	0	0	0	0	Off

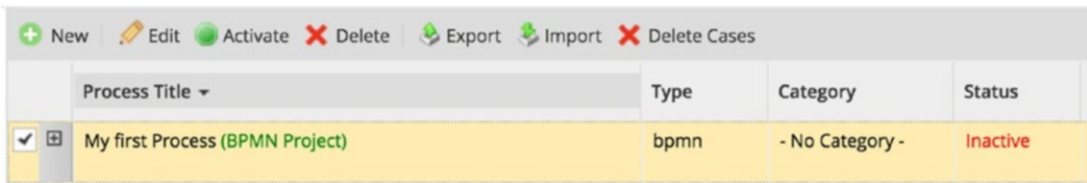
Edit

The Edit option allows you to edit the processes you create in ProcessMaker. To edit a process, you select it from the list and click the Edit button from the menu bar. When a process is selected from the list, it is highlighted in yellow. Clicking the Edit button launches the Process Designer, allowing you to make the required changes to the process.



Status

The Status option displays the status of a process and toggles between Deactivate (for Active processes) and Activate (for Inactive processes) when a process is selected from the list. Newly created processes are active by default, as shown in the previous image. When a process is deactivated, users will no longer be able to work on the process. To deactivate a process, select it and click the Deactivate button.



The process status changes to Inactive as shown here, and the Status button now shows the option to Activate. Select the process and click Activate.

Export

The Export option allows you to export your projects outside of ProcessMaker. This is useful for copying a project from one ProcessMaker instance to another. Let us see how it works. Select your project and click the Export button. The Save dialog is displayed, choose a location on your system and save the project. The project will be saved with a .pmx extension.

Delete and Delete Cases

The Delete option allows you to delete a project. A project can, however, only be deleted if it has no cases. The Delete Cases option is a feature introduced in version 3.0.18 that allows you to delete all the cases of a process. You will learn more about what cases are in a bit, but basically, a case is an instance of a process.

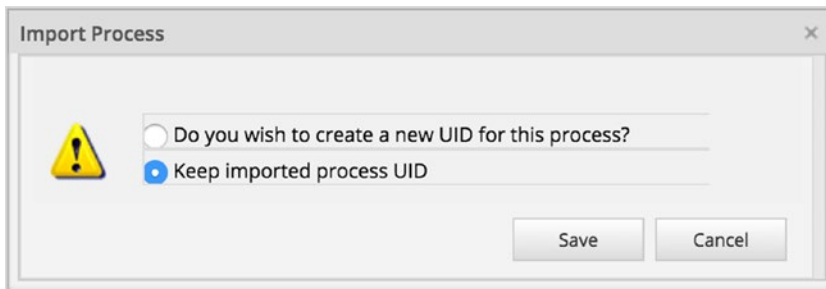
Right now, your process does not have any case, so go ahead and select it and click the Delete button. You should see a confirmation dialog asking if you are sure you want to delete the selected process; click Yes. You should now have a blank process list.

I intentionally put the Export operation before the Delete, so that we can continue with the exported project after deleting it by importing it again as described next.

Import

The Import option allows us to import projects into a ProcessMaker instance. To see how this works, let's import the project we exported earlier. Click the Import button and, in the modal that appears, click the folder icon to browse to the location on your system where you saved the exported project. Select it and click the Upload button.

A modal is displayed asking if you want to keep the imported process UID or create a new one. Leave the option as "Keep imported process UID" and click the Save button.



The Process Designer is opened, with your process successfully imported. Close the designer by clicking the close button in the top-right corner. The imported process is now displayed in the process list. Select the process and activate it if it is inactive, as described in the Status section earlier.

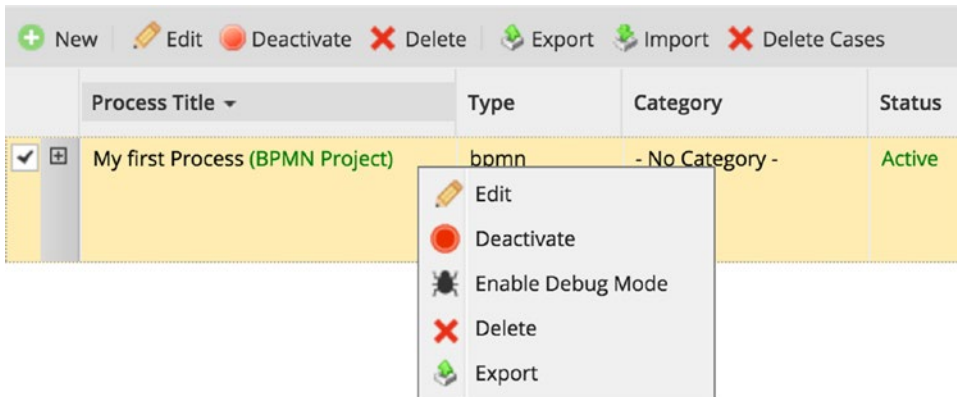
i In case you are wondering what UID means, it refers to the Unique Identifier for the process, which is a string of 32 hexadecimal numbers generated by ProcessMaker. If you are familiar with database terminology, it is similar to a primary key. Objects in ProcessMaker such as users, groups, processes, tasks, cases, Dynaforms, database connections, and so on are identified by their UID.

Category Filter and Search

On the right side of the menu bar on the listing page are the Category filter and Search box. The Category filter allows us to filter the list of processes by category. This is useful when you have a lot of processes. Similarly, the Search box allows us to search for a process by entering the name. Try entering a phrase in the Search box that is not included in the name of the process you just created and click the Search button.

Debug

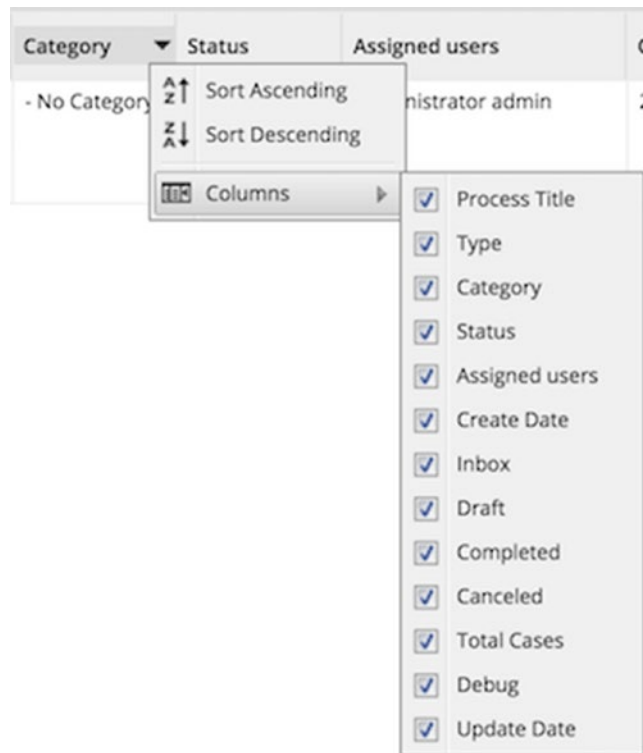
The actions just described, with the exception of New, Import and Delete Cases, can also be accessed by right-clicking on a process from the list.



Right-click the process and you will notice an option to Enable Debug Mode. This option allows us to debug (troubleshoot) the processes we build when they do not run as expected. When the debug mode is enabled, you can see the values stored in system and case variables and how they are being modified by triggers, which are basically custom code written in PHP. You are also able to see any errors that might have occurred when executing triggers.

Process List Columns

The list of processes is displayed in a tabular form with the columns described in the following list. The columns displayed in the Process List can be toggled by mousing over any column title, clicking the down arrow on the right to display a context menu and selecting Columns. This displays the list of columns with checkboxes that can be checked or unchecked to toggle the visibility of the columns as shown here.



Process Title: The title of your BPMN project or process with the project type in brackets and colored green.

Type: The type of process, which can be either BPMN, meaning that the BPM project uses the BPMN 2.0 notation, or Classic, for imported processes created in previous versions of ProcessMaker. From ProcessMaker 3, it is only possible to create BPMN projects.

Category: This refers to the category of the process. Categories are useful for organizing processes and can extend across business lines, for example HR Processes, Finance Processes and so on. The default is-No Category if no category is selected when creating the process. The category of a process can always be changed from the process properties, as we shall see later.

Status: The status of the process, which is either Active or Inactive. No cases can be executed in a process that has Inactive status.

User owner: The user who created the process.

Create date: The date when the process was created.

Inbox: The number of cases of the process that are currently pending and appear in users' inboxes to be worked on.

Draft: The number of cases in the process that are in draft mode. A case is in draft mode when the currently assigned user has initiated a new case, but is yet to complete the first task in the process and route it to the next task.

Completed: The number of completed cases in the process. A case is completed when the final task in the process has been reached and the case can no longer be worked on.

Canceled: The number of canceled cases in the process.

Total Cases: The total number of cases that have been created in the process (including any canceled cases).

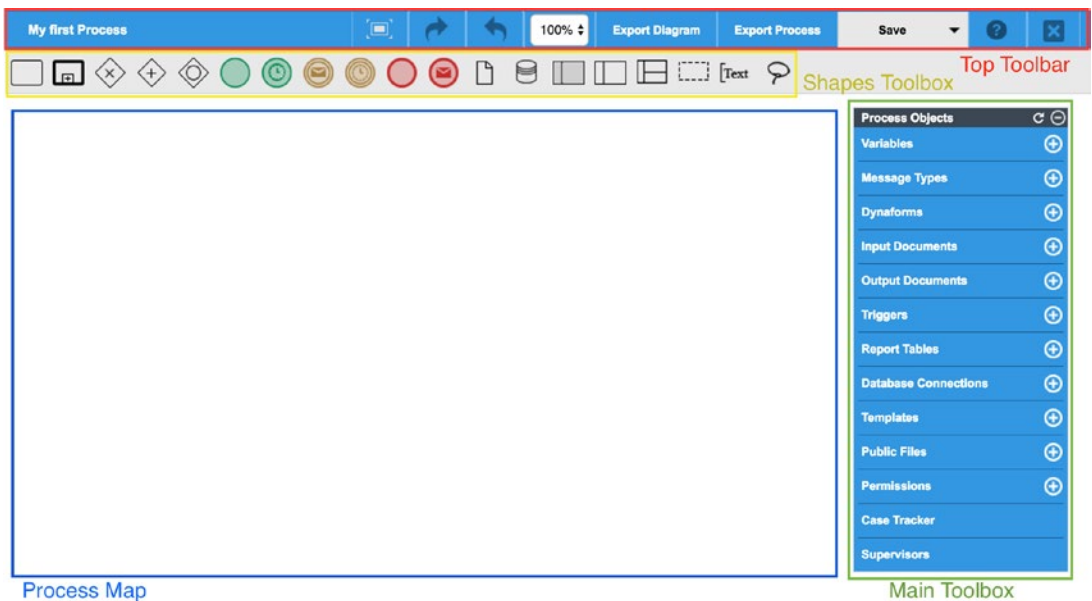
Debug: This shows the debug status off the process and can be either On or Off.

Update Date: This shows the date and time the process was last updated.

Process Designer

The Process Designer is where you model your process, build the forms for capturing data, configure it with the necessary business rules and conditions, assign users and grant them permissions on the process. The designer is where the action happens. In the following sections you will explore the designer and start building your first process.

On the following screen, the four major components of ProcessMaker Designer are highlighted: the Top Toolbar, Shapes Toolbox, Main Toolbox and Process Map area. To access the designer, select your process and click the Edit button. You can also access the designer by double-clicking the process.



Top Toolbar

The Top Toolbar is the blue horizontal toolbar located in the top section of the designer. The toolbar shows the title of the process being worked on at the moment (in the following image, the title of the process is “My first process”) on the left followed by a series of buttons described next.



Full Screen



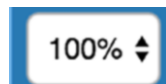
The Full Screen option allows you display the designer in full screen mode. This can be useful when trying to work free of distractions or when presenting to a screen. To leave the full screen mode, press the Esc key on your keyboard.

Undo and Redo



The Undo and Redo options, as the names imply, allow you to undo or redo an action made on the Process Map. This is useful for correcting mistakes while building your processes. Only the most recent twenty (20) actions can be undone or redone. The arrow pointing to the right is the redo action, while the arrow pointing left is the undo action.

Zoom



The Zoom option allows you to scale the size of the Process Map using predefined values of 50%, 75%, 100%, 125% and 150%. This can be very useful for very complex processes with lots of intersecting lines.

Export Diagram

The Export Diagram option allows you export your designed process as a .bpmn file that can be imported into any BPMN 2.0 compliant tool. Simply click the Export Diagram option and, in the Save file dialog that appears, select a location to save the file to.

Export Process

This works the same as the Export option described earlier under the Process list section. It allows you to save a copy of the process to your system. The exported file will be saved as a .pmx file, which can be imported back into ProcessMaker.

Save and Save As



The Save option allows you save all changes to your process as you work on it. ProcessMaker will automatically save the changes you make every 15 seconds to help prevent losing valuable work. When you have unsaved changes, the Save option changes to green.

Clicking the down arrow beside the Save option displays the Save As option, which allows you to save a copy of the process open in ProcessMaker. This can be useful when creating a new process that is very similar to an existing process.

Click the Save As option to create a copy of the process. In the modal window that appears, enter a new title for the process and click Save.

The process title in the Top Toolbar should now show the title of the newly copied process, which means you are now editing a copy of the process. To close the Save As modal without creating a copy, simply click the Cancel button.

Help

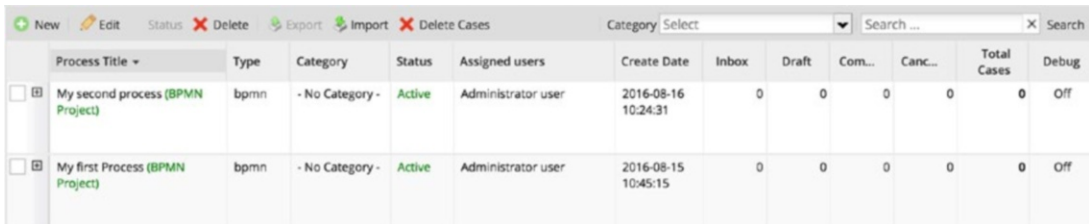


Clicking the Help option displays the walkthrough you saw when you first created the process. This is useful for reminding yourself of the key components of the designer when getting started.

Close



The Close option closes the designer and returns you to the process list. Go ahead and click the Close option, and if you created a copy of your process as described in the Save and Save As sections, you should now see two processes in the list as shown here.



Process Title	Type	Category	Status	Assigned users	Create Date	Inbox	Draft	Com...	Canc...	Total Cases	Debug
My second process (BPMN Project)	bpmn	- No Category -	Active	Administrator user	2016-08-16 10:24:31	0	0	0	0	0	Off
My first Process (BPMN Project)	bpmn	- No Category -	Active	Administrator user	2016-08-15 10:45:15	0	0	0	0	0	Off

Process Map Area



In the Process Designer shown earlier, the Process Map is the large area in the lower left. Think of it as the canvas on which you draw out your process model. You can edit the properties of your process by right-clicking on any free area of the Process Map to display a context menu (right now, the map is blank, but once you start building your process, be sure to click outside the design elements).

Clicking Enable Grid Lines displays grid lines across the Process Map, which can serve as a guide when placing elements of your process on the map. This also toggles the menu item name to Disable Grid Lines, which you can click to clear the grid lines.

To update the properties of your process, click Edit Process from the context menu. This shows the Process Information modal containing properties of the process which you can modify as desired.

Shapes Toolbox

If the Process Map is the canvas, the elements in the Shapes Toolbox are the brushes and paints with which you draw on the canvas.



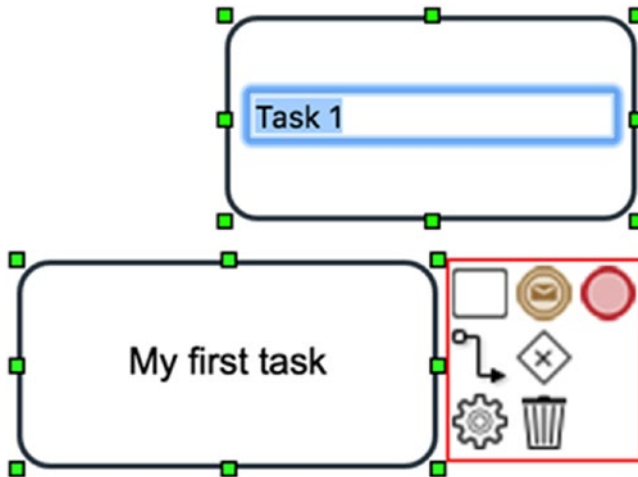
The Shapes Toolbox consists of the BPMN elements that you can use to model your process in ProcessMaker. Some of the elements contained in the Shapes Toolbox are currently for design purposes only and not yet implemented in the BPM engine of ProcessMaker at the time of this writing. I will point out these shapes as we look at each of them in detail in the following sections.

If you are already wondering when you will get to start building something and testing it, we are almost there, and the shapes we are about to look at form the foundation for building your process. Once we walk through all the elements in the Shapes Toolbox, we will proceed to model our Cash Advance and Expense Retirement process described earlier in the guide. For a quick reference on all BPMN elements, see <http://www.bpmnquickguide.com/view-bpmn-quick-guide/>.

Task



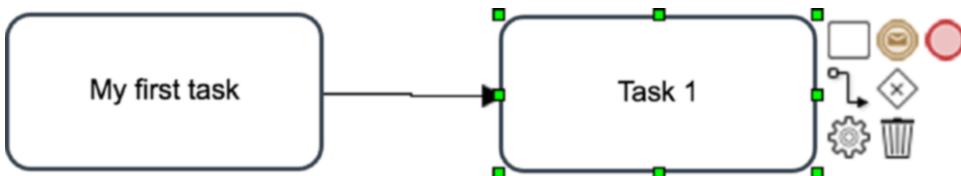
The first shape in our toolbox is the Task element. A task is an activity that cannot be divided into other activities. A task typically consists of one or more sequential steps working toward a common goal. Using our Cash Advance process as an example, a task will be initiating a cash advance request or approving a request. A task can be assigned to a user or can be a script run by the system.



To add a task to your Process Map, drag and drop the task icon from the Shapes Toolbox to a free area on the Process Map. A task will be created on the Process Map with the name of the task highlighted in blue and editable. Enter a new name for your task and click outside the task on the Process Map to apply the new name.

Next, click the newly created task. This will display a quick toolbar (in the lower right of the preceding image) that shows the available elements that can be added to the Process Map from the task. You can hover over each element in the quick toolbar to display its name. To use an element from the quick toolbar, click it to select it, and then click on a location on the Process Map to place it. The delete icon in the quick toolbar will delete the task, while the properties icon will display its properties.

Let's go ahead and add a new task to our Process Map using the quick toolbar. Click the task if you do not have it selected, to display the quick toolbar. Click the task icon in the quick toolbar and then click a location on the Process Map to place the new task. You should now have two tasks on your Process Map as shown next. Double-click the task to make the name editable and enter a new name for the task.



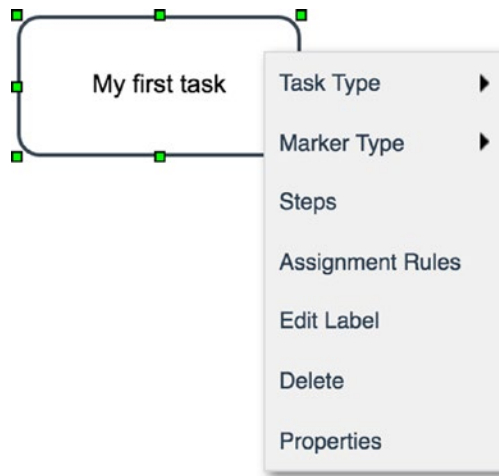
You can reposition your task on the Process Map by clicking and dragging it to the desired position. For finer control over the position, you can also use the arrow keys on your keyboard to move the task when it is selected (a selected task will have the small green boxes on the edges as in Task 1 in the preceding image). You can also resize the task by dragging the green boxes in any direction.

Right-clicking the task displays a context menu with the following items:

- **Task Type:** This allows you set the type of task and by default a newly created task is set to Empty task type. The available options are:
 - **Empty Task:** The default, represents an action in the process.
 - **Send Task:** A task that sends a message.
 - **Receive Task:** A task that receives a message.
 - **User Task:** A task to be performed by a user.
 - **Service Task:** A task performed by a web service or application.
 - **Script Task:** A task that executes a trigger (custom code).
 - **Manual Task:** A task that is to be executed manually outside the application.
 - **Business Rule Task:** A task representing the implementation of a business rule.

Apart from the Script Task type, all other task types are for design purposes and treated as an Empty Task by the BPM engine when running the process. A task with Script Task type executes a trigger (custom code) when a case in the process reaches that task.

You can see how the task types are applied to the task on the Process Map by clicking on any of them. For example, changing our first task to a User Task type by right-clicking it and selecting User Task from Task Type in the context menu should make our task similar to the following image.



- **Marker Type:** This is used to provide a visual indication of how the task will be executed. By default, newly created tasks have no marker (None). The available options are:
 - **Loop:** This indicates that the task will be run repeatedly until a certain condition is met. If you have a programming background, you can think of this as a DO-WHILE loop. This marker is purely for design purposes and is not yet implemented in the ProcessMaker BPM engine at the time of this writing.
 - **Parallel:** This indicates that parallel or multiple instances of the task will be executed, and the task is only completed after all the instances of the tasks have been completed. This is useful for cases where multiple users need to sign off on a document. This is implemented in the ProcessMaker BPM engine and when a task is marked as Parallel, you are able to assign multiple users to the same task, and the task is not completed until all the assigned users have acted on the task.
 - **Sequential:** This indicates that the task will execute a series of scripts sequentially. This marker is purely for design purposes and is not yet implemented in the ProcessMaker BPM engine at the time of this writing.

- **Steps:** This is used to define the specific actions or work that must be done to complete a task. A step can be filling a form, uploading a document, generating a document, or executing some custom code or an external functionality added to ProcessMaker by a plugin.



- **Assignment Rules:** This is used to define how the task will be assigned to users or groups when running the process.
- **Edit Label:** This is used to edit the name of the task and has the same effect as double-clicking the task.
- **Delete:** This is used to delete the task.
- **Properties:** This is used to view and edit properties of the task.

Sub-Process



The next element in our Shapes Toolbox is the Sub-process element. This is used to embed a process within a process. The embedded process is the *sub-process* and the containing process is referred to as the *master process*. A sub-process is basically another process within the organization and is created just as you would any other process.

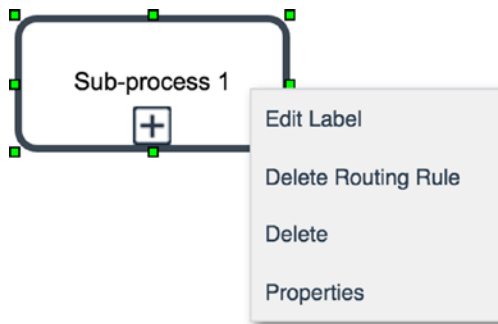
Sub-processes are useful for simplifying process diagrams so that you don't have too many tasks in a process model. Using them makes the process diagram much easier to read and understand. Another benefit of sub-processes is that they allow you to re-use existing processes within other processes that encompass them.

For example, let us assume that the IT department in MSB Corporation has a process for assigning PCs to users, titled “IT System Request.” The HR department also has a process, titled “Employee On-boarding” for on-boarding new employees into the organization, and this process involves IT assigning PCs to the new employees.

The Sub-Process element allows us to embed the IT System Request process within the Employee On-boarding process (the master process) without having to rebuild all the logic of the sub-process in the master process. In the future, if IT makes a change to its process of assigning PCs to employees, we will not need to modify HR’s Employee On-boarding process.

ProcessMaker supports two types of sub-processes, asynchronous and synchronous, and they differ primarily in how they are executed in context of the master process. When a case is run in a process that contains a sub-process, when the case reaches the sub-process task, a new case is created for the sub-process. If the sub-process is *asynchronous*, the master-process continues to the next task in the process without waiting for the sub-process to complete. On the other hand, if the sub-process is *synchronous*, the execution of the master process is paused and is resumed when the execution of the sub-process is completed.

Let us add a sub-process to our Process Map. Drag and drop the Sub-process element from the Shapes Toolbox onto the Process Map. Just as with the Task element, you can double-click it to edit the name, or click it once to display the quick toolbox. Right-click the sub-process to display the context menu. You’ll see the following options:



Edit Label: This has the same effect as double-clicking the sub-process.

Delete Routing Rules: This is used to delete the rules defined for routing the case from the sub-process. We will learn more about routing later.

Delete: This is used to delete the sub-process from the Process Map.

Properties: This is used to configure the properties such as selecting the linked sub-process and its starting task, the type of sub-process, Asynchronous or Synchronous, and defining which variables (data) will be exchanged between the master process and the sub-process.

Click the Properties option in the context menu displayed to explore further. In the properties modal that is displayed, you can edit the name of the sub-process and select the process that should be run as the sub-process.

This is a drop-down list showing the processes defined in ProcessMaker. If you created a copy of the process earlier, you should see the process displayed in the list. Next is the starting task in that process. We currently do not have any starting task (or any task for that matter) defined in the second process. If we did, it would show in the drop-down list.

The Type is currently set to Asynchronous; beneath it is Variables Out, which allows us to specify which variables (data) will be passed to the sub-process when it is started. Change the Type to Synchronous, and you should now see a Variables In list beneath the Variables Out. This is used to define the variables (data) that will be passed back to the master process from the sub-process when it is completed. The Variables In option is displayed only for Synchronous sub-processes, as the master-process does not wait for Asynchronous processes and thus cannot receive data back from it on completion.

Click the Cancel button to close the properties modal, and proceed to delete the sub-process from the Process Map. In the example we gave earlier of making IT System Request process a sub-process of HR's Employee on-boarding process, should it be configured as an Asynchronous or Synchronous sub-process, and why?

Gateways

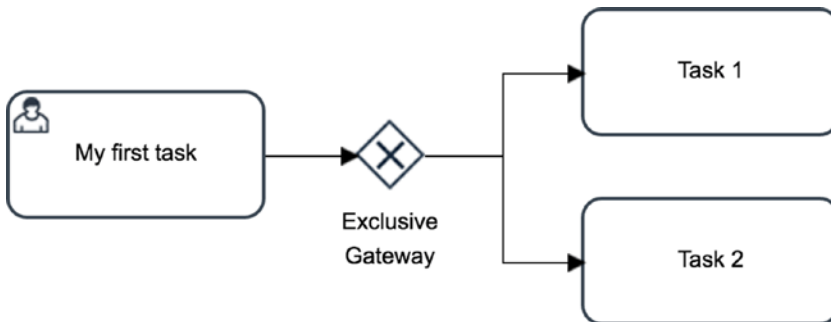


The next three elements in the Shapes Toolbox are Gateway elements, and they are used to determine the flow of a process. They are used to fork (split) and merge (join) paths in a process.

In ProcessMaker, we have three types of gateways; the following sections briefly describe them; we will dive into more elaborate examples later in this book.

Exclusive (XOR) Gateway

This gateway is used to select only one path from two or more paths in the flow of the process. In the following diagram, the process can only be routed to either Task 1 or Task 2.

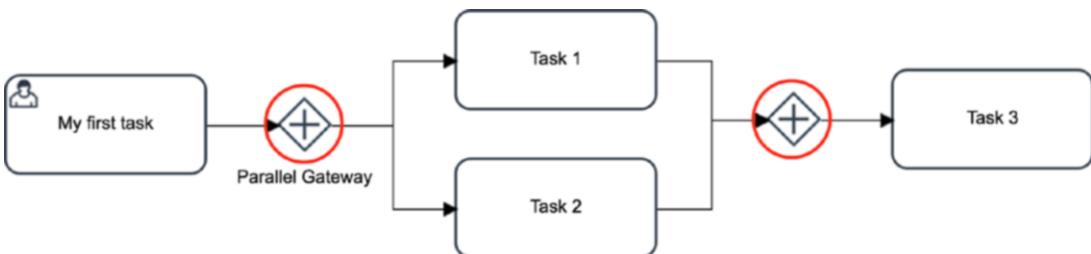


An Exclusive gateway

When using an Exclusive gateway, you add conditions that are evaluated to determine which task to route the case to next. You should ensure that only one of the routing conditions evaluates to True; otherwise, the path to take cannot be determined, causing the case to display an error and stop execution.

Parallel (AND) Gateway

This gateway is used when all paths at a process fork must be taken. The Parallel gateway does not evaluate any condition and creates multiple instances of the case that are routed to all following tasks after the fork. In the following diagram using a Parallel gateway, the case is routed to both Task 1 and Task 2 at the same time.

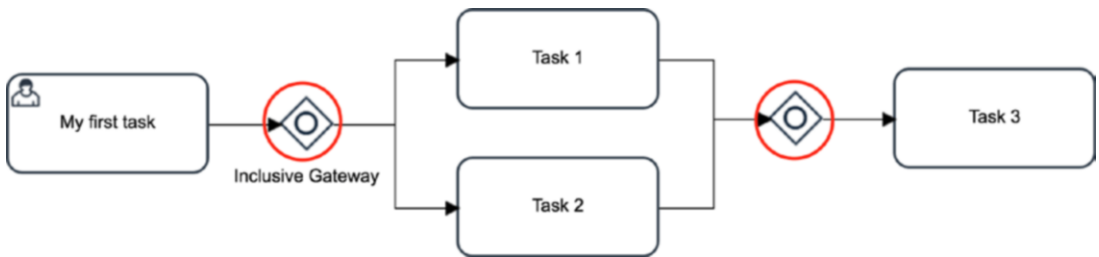


A Parallel gateway

When using a Parallel gateway, the process flow must be merged (joined) back using a converging Parallel Gateway element (the fork and merge gateways are circled in the preceding image). In this previous diagram, Task 3 cannot begin until both Task 1 and Task 2 have been completed.

Inclusive (OR) Gateway

This gateway can be thought of as a hybrid of Exclusive and Parallel gateways; it allows us to create parallel flows, just like the Parallel gateway, and it does so by evaluating conditions, like the Exclusive gateway. The Inclusive gateway can be used to route the flow to one or more following tasks after the fork. The process will flow through all tasks whose routing conditions evaluate to True at the same time. Just as with the Parallel gateway, the fork and merge gateways (circled in red in the following image) must be of the same type; that is, an Inclusive gateway.



An Inclusive gateway

In this diagram using an Inclusive gateway, the process can flow through Task 1 only, or Task 2 only, or both Tasks 1 and 2. However, Task 3 cannot begin until all paths taken are completed. If the case is routed to both Tasks 1 and 2, they must both be completed before Task 3 can begin. If it is routed to only one of them, then Task 3 begins when that task is completed.

Events

Next in the Shapes Toolbox are the Events elements. These fall into three broad categories: Start events (green), Intermediate events (dark yellow) and End events (red). An event is used at a given point to indicate that something occurs at that point in the flow of a process.



Start events, as the name implies, are used to start a process. Intermediate events are used to indicate something that occurs between the start and the end of a process, and End events are used to terminate a process.

In the Shapes Toolbox, you will notice that we have two of each category of events, which might be a bit confusing at first. Right now, just think of them as the same thing, and the icons (or lack of icons) in the elements are just indicators of the specific type of events. To clarify, we have three categories of events—Start, Intermediate and End—which have different event types, which we shall explore in the following sections.

As you can do with the other elements we have seen, you can add an event to the Process Map by dragging and dropping it. To connect the event to other elements on the Process Map, click it to display the quick toolbar, and click the connect icon in the quick toolbar. Your mouse cursor should change to an arrow line, which you can drag and click on the element you want to connect the event to.

Start Events

As mentioned earlier, Start events are used to indicate the beginning of a process. A process can have more than one Start event. Using our Cash Advance and Expense Retirement process as an example, the process can start either with the employee requesting an advance, or with the employee retiring the expense, which might be the case if no advance was received before the expense was incurred.

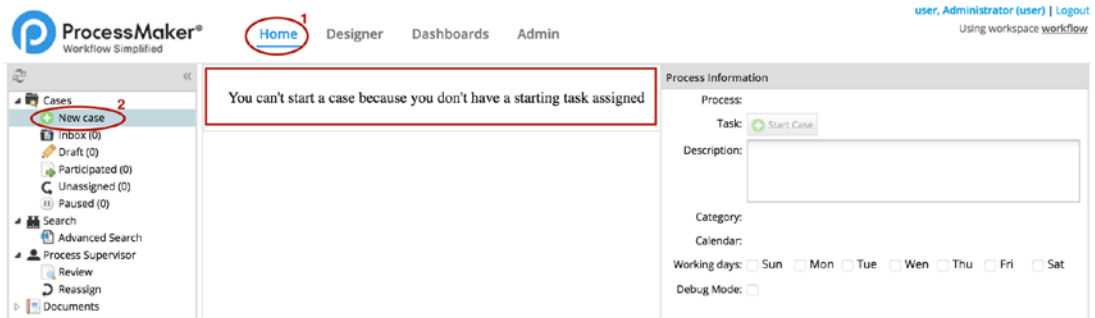
If you have been following along, your Process Map should contain a single task that we created when we looked at the Task element earlier. If your Process Map is blank, no worries. Just drag and drop the Task element from the Shapes Toolbox unto the Process Map and give it a name. If you have been exploring on your own and have added other elements to the Process Map, that's fine.

Let's now explore the properties and different types of Start events in ProcessMaker. Drag and drop the Start Event element from the Shapes Toolbox to the Process Map on the left side of the task we created earlier. Click the Start event you just added to display the quick toolbar. Click the Connect icon in the quick toolbar and drag the line until you are hovering over the Task element ("My first task" in the diagram), and click the Task element to connect it with the Start event.



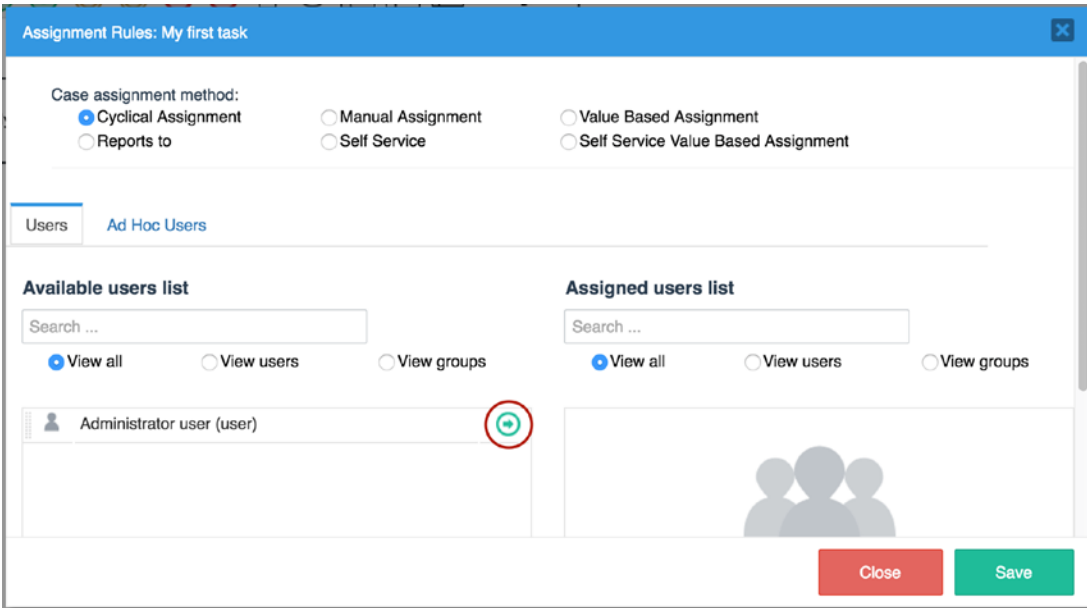
By connecting the Start event to "My first task," we have made it a starting task. To better understand what this means, let us explore the Home menu. This is where the users of our ProcessMaker instance will be able to start cases and interact with them.

Click on Home in the ProcessMaker main menu at the top of the page. In the left menu, click New Case. You are shown the message "You can't start a case because you don't have a starting task assigned."

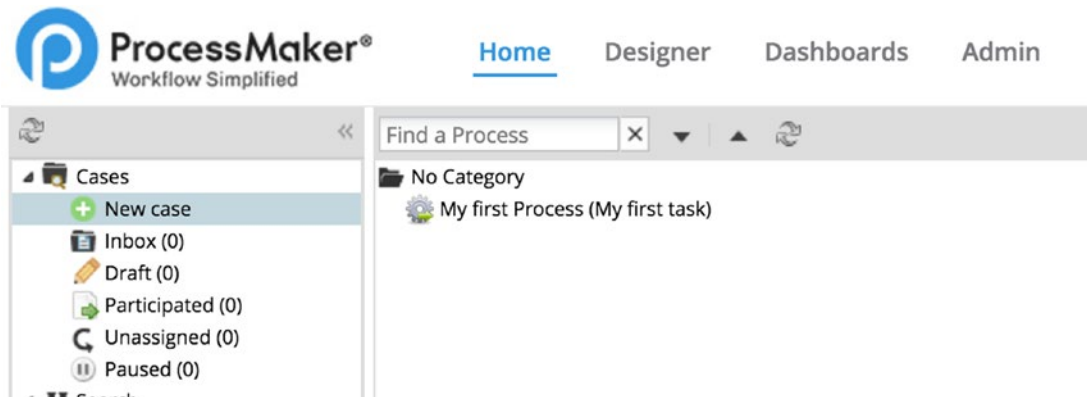


Go back to the Process Designer to continue editing the process. Even though we have created a starting task, it is not yet available to users, because we have not assigned the task to any user or group. You may recall the Assignment Rules menu option in the context menu of the Task element; we will explore the intricacies of configuring assignment rules later, but for the purpose of understanding Start events, we will simply assign "My first task" to ourselves.

Right-click “My first task” to display the context menu and select Assignment Rules. In the Assignment modal dialog that appears, you will see a list of available users. Click the green arrow icon beside the name of the user you are currently logged on as, which should be Administrator. This moves the user to the Assigned Users list. Click the Save button.



Now, go back to Home ➤ Cases ➤ New Case as we did earlier, and you’ll see that “My first task” is now displayed as a starting task for the process titled “My first process.”



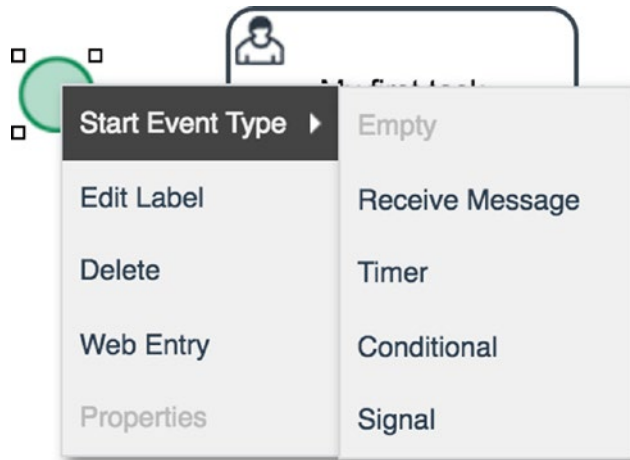
I mentioned earlier that a process can have more than one Start event and used the example of the Cash Advance and Expense Retirement Process. To demonstrate this, rename “My first task” to “Request Cash Advance” (You can edit a task label by double-clicking it or right-clicking and selecting Edit Label).

Next, add a new Start event on the Process Map. Click it to display the quick toolbar and click the Task icon. Drag the mouse cursor to a location on the Process Map and click to place the new task. Label the newly created task Report Expense. Right-click the task and select Assignment Rules to assign it to yourself as we did for the first task. Your Process Map should look like the following image.



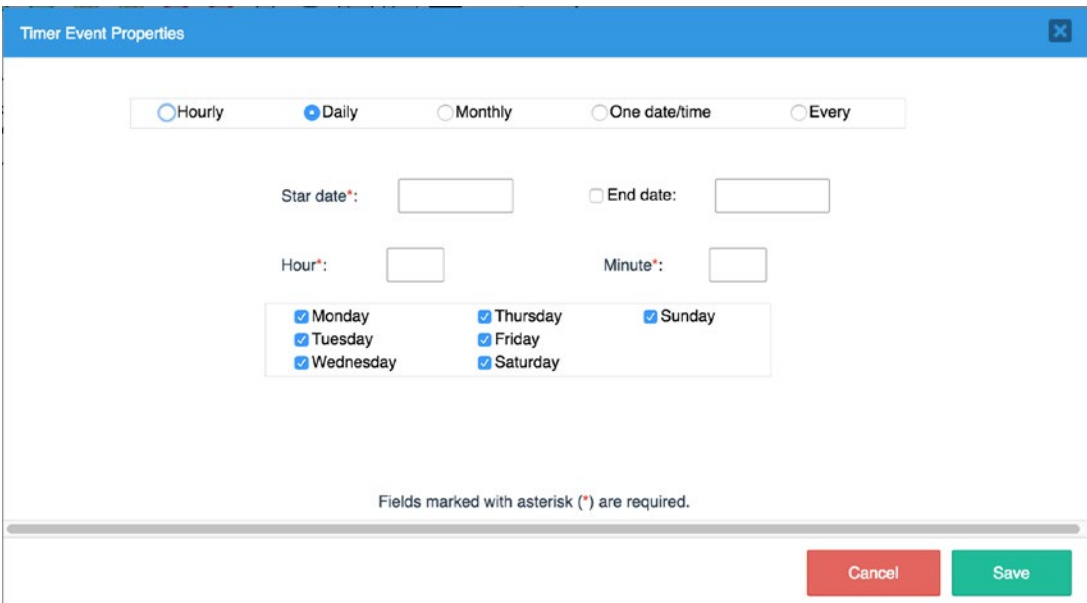
Now go to Home ► Cases ► New Case, and you should see that the process now has two starting tasks. Right now our process does not do anything, and we are still learning to model the process. With an understanding of the effect of Start events, let us head back to the designer to explore the properties and the different types.

With your process opened in the designer, right-click any of the Start event elements to display the context menu with the following options:



Start Event Type: This is used to select the type of Start event, which could be:

- **Empty:** This is the default Start event type and is supported by the ProcessMaker BPM engine. This type of Start event simply starts the process. The Web Entry feature is available only to this Start event type.
- **Receive Message:** This Start event type is used to start a process from a message received from another process. The messages are sent from either intermediate or end events in the sending process.
- **Timer:** The Timer Start event type is used to automatically start a process at a specified time or interval. Right-click the Start event and set the Start Event Type to Timer. Right-click the Start event again and select Properties. You should see a modal that allows you to specify the time interval or specific date and time to start the process. Click the Cancel button to return to the Process Map.



- **Conditional:** The Conditional Start event type is purely for design purposes and is used to indicate that the process should only start once certain conditions are satisfied. These conditions ideally are external to the process.

- **Signal:** The Signal Start event type is also purely for design purposes. It is used to indicate that the process is started by a signal from another process.

Edit Label: This is used to add a label to the Start Event type and can be useful when a process has multiple start events.

Delete: This is used to delete the Start Event element from the Process Map.

Web Entry: This is used to set up the ProcessMaker Web Entry feature for the process. The Web Entry feature allows you to set up a form that can be embedded on an external web site or web page that anonymous users can fill in and submit to start a case in ProcessMaker. This can be useful for processes that are triggered by information received from users outside your organization such as customers.

Properties: This is used to configure additional properties of the Start event as we saw in the Timer Start event type above. This option is available for only the Receive Message and Timer Start event types.

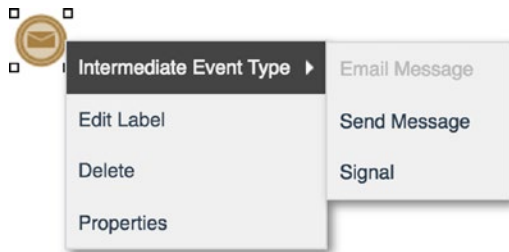
Intermediate Events



Events in this category are used to indicate something happening in the flow of the process, between the start and end. Intermediate events in ProcessMaker have been implemented as two broad types: Intermediate Throwing events and Intermediate Catching events.

Intermediate Throwing Events

You can think of Intermediate Throwing events as intermediate events in which the process initiates an action. To add an Intermediate Throwing event to your Process Map, drag and drop the Intermediate Email Event element from the Shapes Toolbox into the Process Map. The types Of Intermediate Throwing events are the following:



Email Message Intermediate Event Type: This event type is used to send an email message when a process is being run. To use this event type, drag and drop the element to the Process Map, and set the Intermediate Event Type to Email Message. Then right-click the event and select the Properties option. This displays the Email Event dialog box as shown next, where you can select a sender (this is available only if you have configured SMTP settings during the installation), and specify the recipient, message, and content of the email message to be sent.

Send Message Intermediate Event Type: This event type is used to send a message to another process that requires the message. The sent message will be caught or received by the other process using a Receive Message Start event or Receive Message Intermediate event. To see how this is configured, right-click the Email Message Intermediate Event element just created and change the Intermediate Event Type to Send Message. Right-click it again and select the Properties option to display the Intermediate Send Message Event dialog shown next. The dialog allows you to select the Message Type and define the data that will be sent by the message to the receiving process. Click Cancel to return to the Process Map.

Intermediate Send Message Event
✕

Message Type*:

Name	Get value from	
No records found		

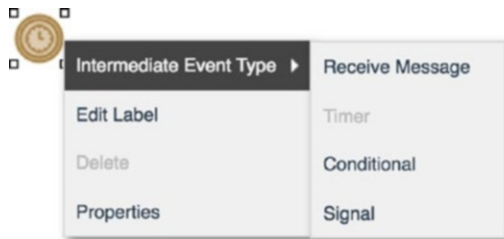
Correlation Value: @@

Cancel
Save

Signal Intermediate Event Type: This event type is purely for design purposes and is used to indicate that the process is sending a signal to another process.

Intermediate Catching Events

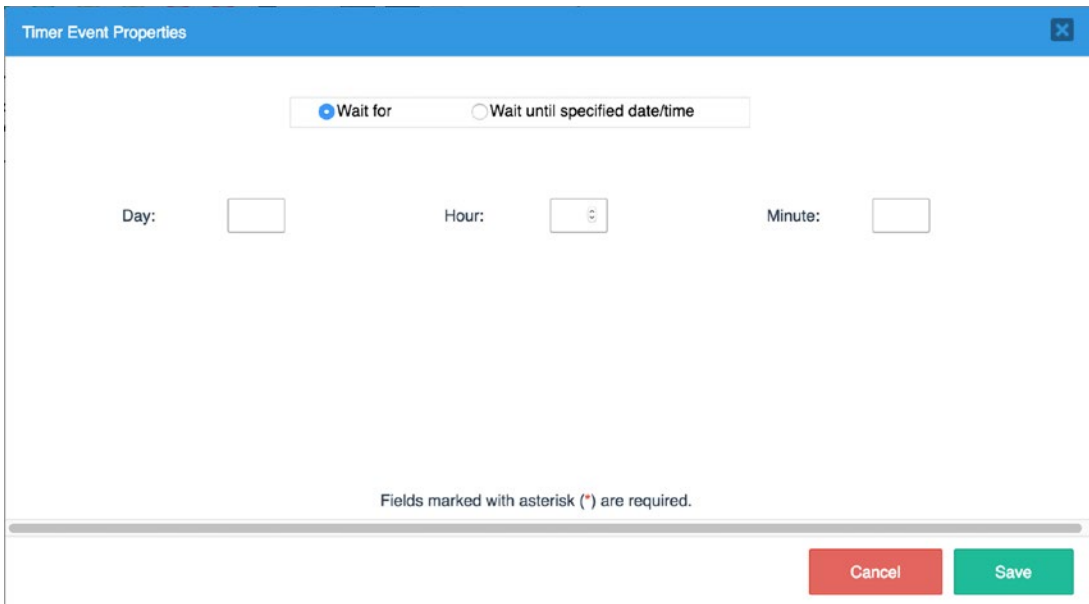
These are intermediate events that are used to represent events that occur in response to other events. For example, when the Send Message Intermediate event type just described is thrown, it can be responded to with the Receive Message Intermediate event type. The available Intermediate Catching events in ProcessMaker are:



Receive Message Intermediate Event Type: This event type is used to receive a message sent from another process. To add a Receive Message Intermediate event to the Process Map, drag and drop the Intermediate Timer Event element from the Shapes Toolbox to the Process Map. Right-click it and change the Intermediate Event Type to Receive Message. Right-click the event again and select the Properties option to display the Intermediate Receive Message Event dialog, which can be used to configure how the received message will be processed. Click the Cancel button on the dialog to return to the Process Map.

Timer Intermediate Event Type: This event type is used to represent a delay or pause in the process for a specified period of time. This can be useful in a process where you have to wait for a period to elapse before continuing the process. For example, let us assume MSB Corporation has a vendor onboarding process which involves publishing the name of the new vendor on the intranet, where employees have 7 days to submit any objections before the vendor is confirmed. The Timer Intermediate event type can be used to pause the process for this period before continuing.

To see how the event is set up, right-click the Receive Message event element and change the type to Timer. Right-click it again and select the Properties option to display the Timer Event Properties dialog, shown next, which can be used to specify how long to pause the process.

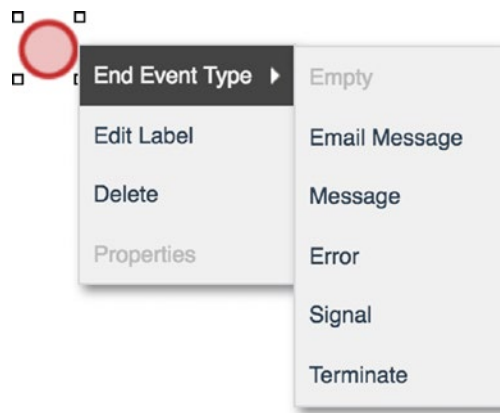


Conditional Intermediate Event Type: This event type is purely for design purposes and is used to indicate that the process flow can continue once a condition is satisfied; otherwise, the process should be stopped.

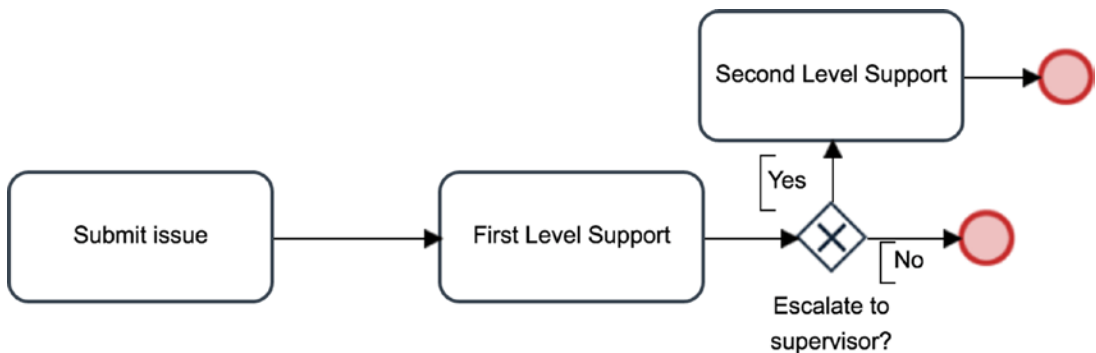
Signal Intermediate Event Type: This event type is purely for design purposes and is used to indicate that the process is sending a signal to another process.

End Events

This category of events is used to indicate the end of the process and show that the process has been completed. Let us look at the different types of End events available in ProcessMaker. Drag and drop the End Event element onto the Process Map from the Shapes Toolbox. Right-click the element to display the context menu with the following options: End Event Type, Edit label, Delete and Properties. Just as in the other event elements we have explored, you can use the End Event Type option to specify the type of End event you want to use in the process.



Empty End Event Type: This is the default end event and is what you will use most often to indicate the end of the process. This End event type is fully supported by the ProcessMaker engine and indicates that the process is ended. You can have more than one end event in a process if there are different flows a process can take to completion. For example, if you have an IT helpdesk process, the process can either end if the reported issue is resolved at the first-level support task, or it can be escalated to a second-level support officer and end when it is resolved.



Email Message End Event Type: This End event type is used to send an email message once the process ends. The sender, recipient, subject and message to be sent are defined by clicking the Properties option from the element's context menu just like we saw in the Email Message Intermediate event type earlier.

Message End Event Type: This End event type is used to send a message to another process at the end of the process. The message sent by this event type must be caught by either a Receive Message Start event or Receive Message Intermediate event in the other process. It is important to keep in mind that the term *message* in this context refers to the inbuilt ProcessMaker Message Types (a type of data used for communication between processes), which you will learn about later.

Error End Event Type: This End event type is purely for design purposes and is not supported by the BPM engine. It is used to indicate that the process flow might have encountered an exception or error causing it to come to an end.

Signal End Event Type: This End event type is purely for design purposes and indicates that a signal will be sent out to other processes at the end of the process.

Terminate End Event Type: This End event type is also for design purposes only and is used to illustrate a situation in which all flows in a process having parallel flows should be terminated once the flow/path that ends at this endpoint is reached irrespective of whether the other parallel flows have been completed or not.

Before we continue with the other elements in the Shapes Toolbox in later chapters, let us take a moment in Chapter 4 to try to model a process to see how these elements all fit together.

In this chapter, we learned about the Process List, how to manage the columns displayed, and what each column does. We also began our exploration of the ProcessMaker Workflow Designer, creating our first process and learning about the BPMN shapes in the toolbox. In the next chapter, we will put the knowledge we have acquired into practice and begin modeling the Cash Advance and Expense Retirement Process, which we will use in examples throughout this book.

CHAPTER 4

Modeling a Process

So far you have seen a lot of theoretical discussion of some of the BPMN elements in the Designer Shapes Toolbox. Before we continue to the remaining elements, let us use what we have learned so far to model our Cash Advance and Retirement process introduced as an example at the beginning of the book.

When modeling a process, we usually begin by identifying the different activities to be carried out in order to accomplish the objectives of the process. These activities are then represented as tasks in the Process Map. Our earlier example listed the different tasks that make up the process, and I'll summarize them again here:

1. An employee makes a request: the starting task.
2. Her supervisor approves the request.
3. Finance gives her an advance for the amount requested.
4. The employee makes the purchase or expense (a manual task, done outside the system).
5. The employee reports the expense and attaches receipts to the report.
6. The supervisor approves the report.
7. Finance reimburses the employee or receives the balance of the advance.
8. Finance updates the accounting system with appropriate accounting entries (a manual task, done outside the system).
9. The expense report is signed as treated by Finance and filed: the ending task.

Our process description here makes a general assumption that the supervisor will always approve the request or report. My experience with supervisors has shown this is not always the case. Our description of the process has not accounted for what should happen if a request is rejected by the supervisor. There are other improvements we could also consider at this point, but for the sake of simplicity, we will leave it at that for now.

The way to account for this possibility is to introduce an alternative flow or path into the process, for requests that are rejected by the supervisor. We will do that with the use of gateways, as you shall see shortly. Let us begin by modeling the Cash Advance Request part of our process. We will add the Expense Retirement part later.

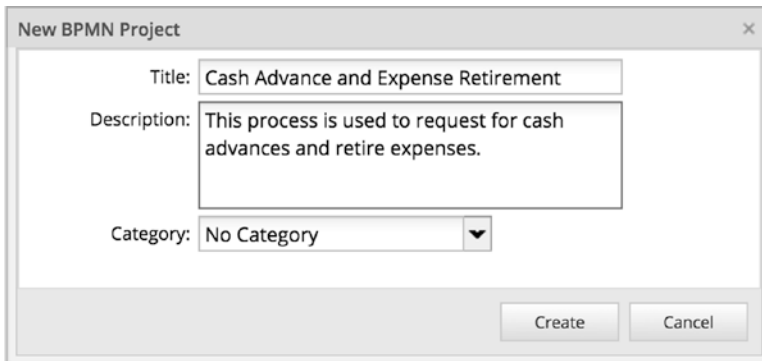
Create a New Process

Click on Design in the ProcessMaker main menu to display the Process List. Click the New button. In the dialog that appears, enter the following values:

Title: Cash Advance and Expense Retirement

Description: This process is used to request cash advances and retire expenses.

Leave the default selection of No Category for the Category field for now. Click the Create button.



Our new process is opened in the designer with the introductory walkthrough overlaid. Click the Quit button on the walkthrough to reveal the Start event on the Process Map. The Start event should be selected and displaying the quick toolbar on its right, as shown in the image here. If the Start event on your Process Map is not showing the quick toolbar, you can display it by clicking once on the Start event.



Add Tasks to the Process

Add the first task to the process by clicking the task icon in the quick toolbar (your cursor should change to a pointer with the task icon attached) and then click to the right of the Start Event element on the Process Map to place the task. Rename the task by editing its label to “Request Advance.”



Add the second task by dragging and dropping a task element from the Shapes Toolbox to the right of the first task. (We can add tasks to the Process Map from the quick toolbar of an element already on the map or by dragging and dropping it from the Shapes Toolbox). Label this new task as “Approve Advance.”



Now add a third task to the map. You can either drag and drop the task from the Shapes Toolbox or use the quick toolbar of the Approve Advance task to add it. If you use the quick toolbar, the newly created task is automatically connected to the preceding task sequentially as we saw with the Request Advance task, which is connected to the

Start event. If you drag and drop the task from the Shapes toolbox, however, the new task is not connected to any other element on the Process Map and we can define the connection later as we please.

Name the third task Disburse Advance. Click on it to display its quick toolbar and click the End icon. Click in an area of the map to the right of the task to place the End event. Your Process Map should now look like the following image.



We can now connect the Request Advance task to the Approve Advance task. We will also connect the Approve Advance task to the Disburse Advance and Request Advance tasks. The first connection will be sequential, and the second will be a decision gateway; that is, the process flow will continue to Disburse Advance if the request is approved or return to the Request Advance task otherwise. From our discussion of the different gateway types earlier, which gateway element would you use for this decision gateway: Exclusive, Parallel or Inclusive?

Connecting Tasks in the Process

A sequential connection between two elements on the Process Map is created by clicking the FROM element to display its quick toolbar, and then clicking the Connect icon in the quick toolbar and dragging it onto the TO element. The TO element is highlighted in green as shown next. With the TO element highlighted, click on it to create the connection. Go ahead and connect the Request Advance task to the Approve Advance task.



If your answer to the earlier question about which gateway to use for the second connection was Exclusive, you answered correctly. We use an Exclusive gateway because the flow can go in *only one* of the two directions, and the direction to go is decided by

evaluating a condition. The condition in this case is “Request has been approved by supervisor.” If True, we proceed to the Disburse Advance task. If False, we return to the Request Advance task.

To add the Exclusive gateway to your process, click the Approve Advance task to display its quick toolbar, click the gateway icon in the quick toolbar to select it, and click Process Map just to the right of the Approve Advance task to add it to the map. The default Gateway element added using the quick toolbar is the Exclusive Gateway type, so we do not need to change its type. Next, click the newly added Gateway element to display its quick toolbar and select the Connect icon. Drag it over the Disburse Advance task and click the task to connect the Gateway element to the task. Click the Gateway element again to display its quick toolbar. Select the Connect icon again, but this time drag it over the Request Advance task and click it to connect the gateway to the Request Advance task. We have now successfully modelled the first part of our process, and your Process Map should look like the following image.



But wait a minute, our Process Map looks a bit weird. The connecting line from the gateway to the Request Advance task passes through the Approve Advance task. To make the Process Map easier to read, we would prefer that the line goes above the task and not through it. This can easily be remedied.

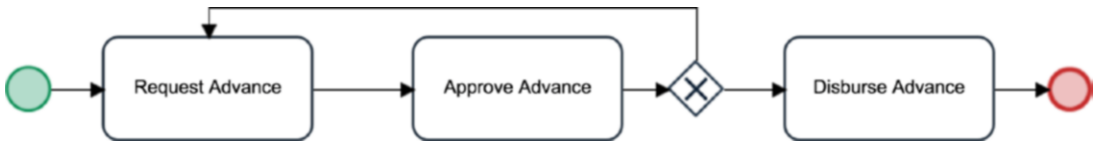
Click the arrow end of the connecting line that terminates at the top of the Request Advance task. This highlights the line and displays a yellow circle on both ends of the line, as shown here.



Next, click and drag the yellow circle on the gateway end of the connecting line. As you do so, the gateway element changes to display all the possible endpoints for the connecting line as black circles, as shown here.



Drag the yellow circle of the connecting line to the top endpoint on the gateway element and drop it. Our Process Map should now be easier to read and look as displayed next.



This chapter, though quite short, has allowed us to practice the basics of modeling a process. In the next chapter, we continue our exploration of the BPMN shapes in the Shapes Toolbox and use the additional knowledge to complete the process model.

CHAPTER 5

Making the Process Comprehensible

Now that we have created the first part of our process, we will proceed to look at the remaining elements in the Shapes Toolbox. These elements are mostly for adding more information to the process model or grouping related tasks and events in order to make the process easier to read and comprehend. We will describe them and where applicable, we will use them to modify the process model we have created so far.

The Shapes Toolbox, Continued

To complete our tour of the Shapes Toolbox, we need to look at the data elements, pools and lanes, artifacts, and the Lasso.

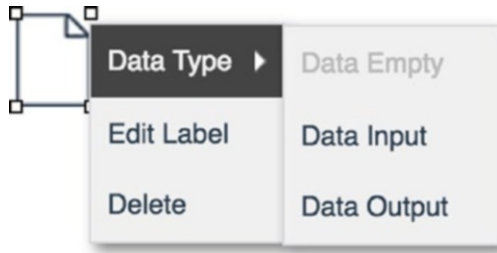
Data Elements

The next elements we will look at in the Shapes Toolbox are the data elements, which are used to show data inputs and outputs in a process.



Data Object

The Data Object element is used to show information flowing through the process. For example, in the Cash Advance and Expense Retirement process, the receipts to be uploaded by the employee when retiring the expenses can be denoted with a data object. This helps to show someone reading the process model that at a particular task, additional information or data is introduced into the process.



You can add a data object to the Process Map by dragging and dropping it from the Shapes Toolbox to the desired location on the Process Map. You can indicate the type of the data object by right-clicking it and selecting Data Type from the context menu. Data objects in ProcessMaker are of three types:

1. **Empty:** This is the default type used when a data element is added to the Process Map.
2. **Input:** This is used to indicate that the data or information is external to the process. For example, the information provided by the receipts uploaded by an employee in the Cash Advance and Expense Retirement process is external to the process.
3. **Output:** This is used to indicate that the data or information is generated by the process. For example, if the Cash Advance and Expense Retirement process generated a confirmation slip for reimbursed or refunded funds after retirement, that will be marked as an Output Data Object.

Data Store

The Data Store element is used to show an external source of data from which a process might read or save data. This can be an external database or physical filing system. Continuing with our Cash Advance and Expense Retirement example, the Accounting system to which the Finance department posts the accounting entries can be indicated as a data store. A data store can be added to the Process Map the same way as every other element in the Shapes Toolbox, by dragging and dropping it on the Process Map.

Pools and Lanes

A pool is used to partition processes, especially processes from different organizations. Our earlier discussions of event types mentioned that certain event types such as the Send Message Intermediate Event and the Receive Message Intermediate Event are used to send messages between processes. This is because objects in different pools cannot be connected directly and can only send messages. For processes in the same organization, these messages are usually represented as sub-processes within the same process and will be placed in the same pool as the other elements of the master process. However, when we are representing a process from another organization, it will be in a separate pool, and any communication between the processes will be done via messages and events.



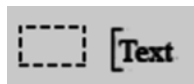
In addition, we use lanes to categorize or group related activities in a pool. The grouping can be along departmental, functional, or geographical lines. Using our Cash Advance example, we could create lanes by functional lines such as employee, supervisor, and finance.

There are two types of pools that can be used in a Process Map. The default is a normal pool, which shows its elements and their connections. We can add elements to this type of pool and connect them as we please. We can also add lanes to this type of pool. This is the type of pool we will be using most of the time.

The second type of pool is a black box pool. As you might have guessed from the name, this type of pool is used to indicate a process for which we have no knowledge of its constituent elements or their connections. The black box pool in ProcessMaker at the time of this writing is purely for design purposes.

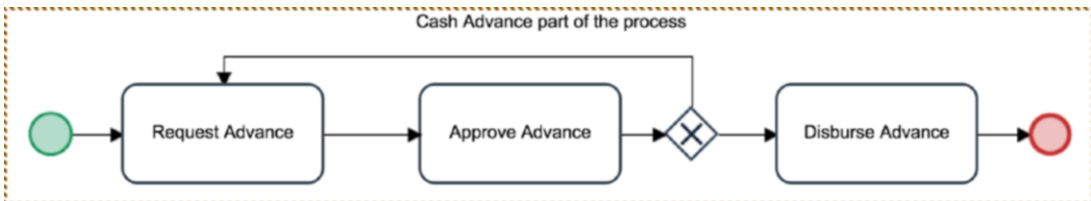
Artifacts

The next elements in the Shapes Toolbox are referred to as *artifacts* and can be thought of as annotation elements. They are used to add descriptive information to the Process Map about the elements making up the map to make it easier to read and understand.



Group

The Group artifact is used to highlight portions of the Process Map and label them. For example, you can group together the tasks that make up the advance part of the Cash Advance and Expense Retirement process as shown here (the group is indicated by the dotted rectangle).



A Group artifact is added to the Process Map by dragging and dropping it onto the map and using the green arrows on the edges to expand the rectangle to cover the elements to be grouped. You can edit the label by double-clicking it and giving the group an appropriate name.

Text Annotation

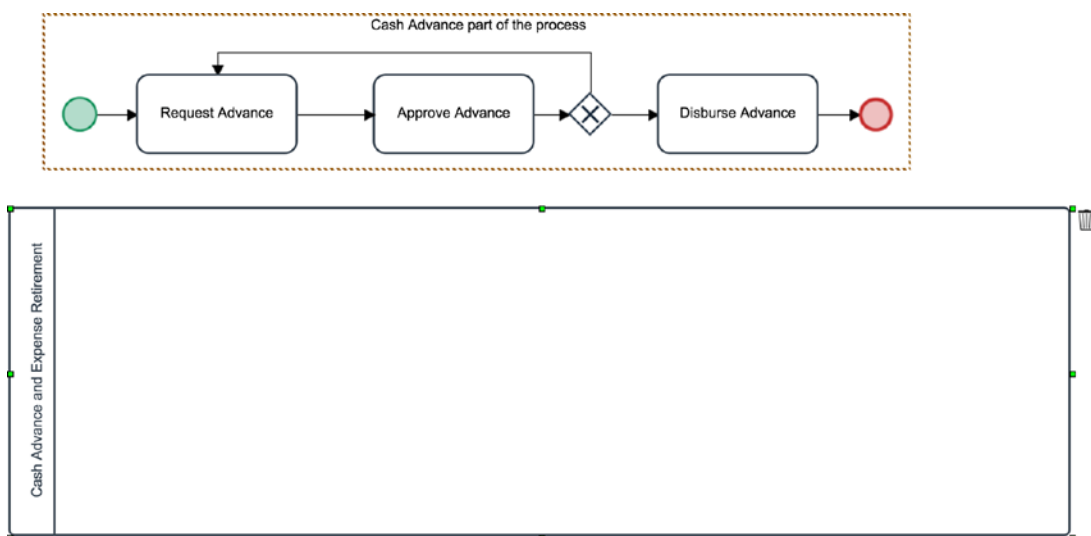
A Text Annotation artifact, as the name implies, can be used to add text descriptions to the Process Map. This is very useful for providing additional information that cannot be readily captured by the elements or their labels. To use the text annotation, simply drag and drop it on the Process Map and type in the text to be added to the map.

Tying It All Together

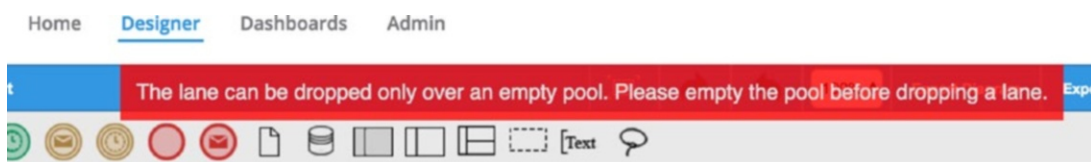
The last item in the Shapes Toolbox is the Lasso. It is not an element but is used for selecting multiple elements for the purpose of moving them together on the Process Map.

Let us proceed to use what we have learned to complete modeling the Cash Advance and Expense Retirement process we began earlier. We will create a pool for our process, add the retirement part of the process, group the activities to lanes, add data elements for receipts and accounting system and text annotations for more details.

Drag and drop a Pool element from the Shapes Toolbox to an empty area of the Process Map below the process model we have built so far. Edit the label of the pool and name it “Cash Advance and Expense Retirement.” Click the pool to show the green boxes on its edges and expand the pool so that it is large enough to contain the process model built earlier as shown next.



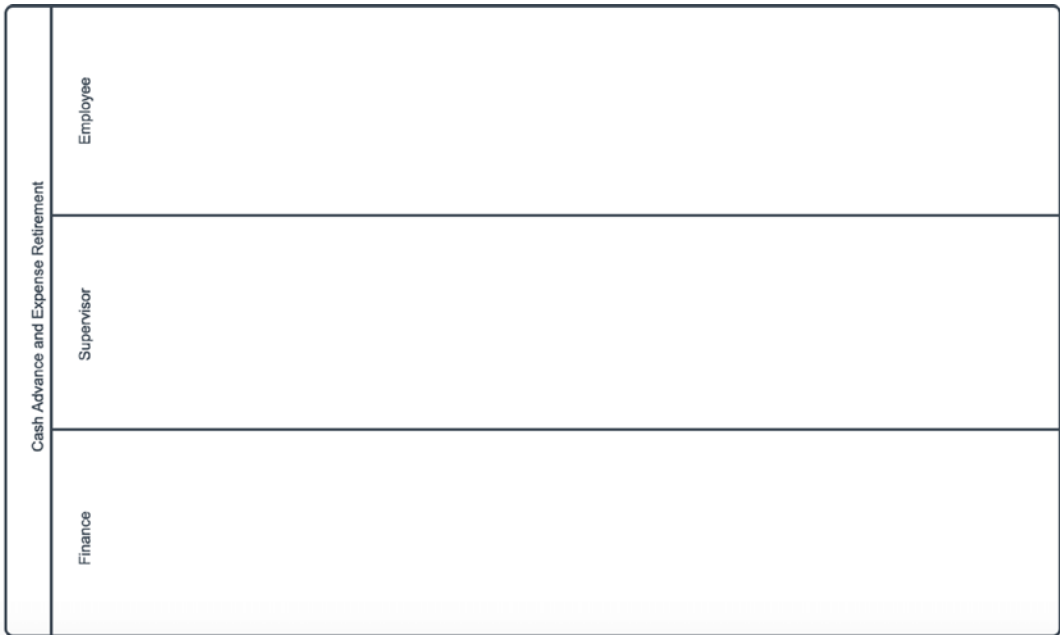
Before we add our existing process model to the pool, we will add the lanes. Currently, ProcessMaker only allows the first lane to be added to an empty pool. That is, you cannot divide your pool into lanes if you have already added any element to the pool. Attempting to do so will display the error shown here.



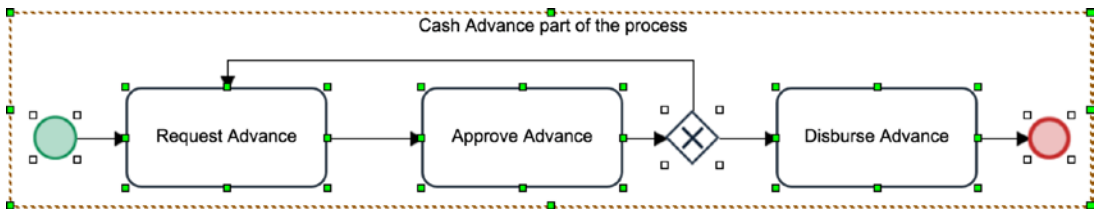
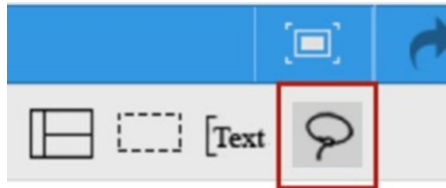
Now we will proceed to add lanes to the pool to group the activities by functionality or responsibility. Remember that we identified three possible lanes for the process earlier, namely Employee, Supervisor, and Finance. Let's go ahead and add the lanes. Ensure the pool is empty and then drag a lane element from the Shapes toolbox onto the pool. The lane label is activated and editable. Rename it to "Employee" and click outside the label to save the changes.



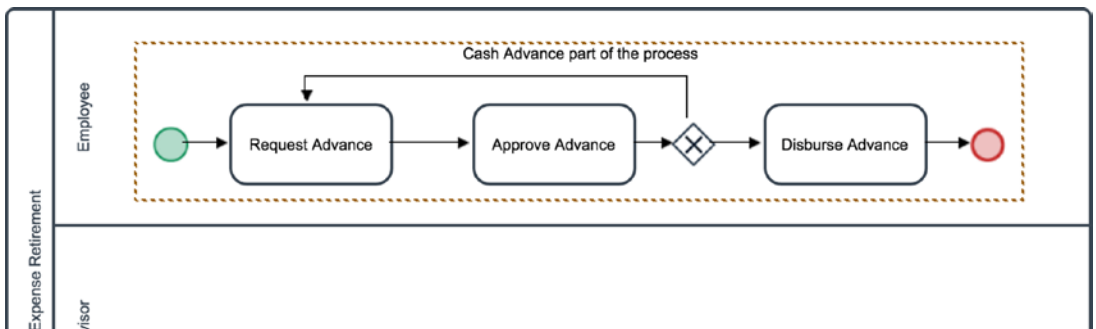
Next drag another lane to the pool and name it "Supervisor." Repeat the process one more time and add a lane labeled "Finance." You should now have three lanes in your pool as shown next. You can use the Zoom dropdown in the Top toolbar to zoom out (set it to 75% or 50%) and see all the elements on the Process Map in one glance.



With the pool divided into lanes, let us move our existing process model into the pool. Click in an empty area of the Process Map to make sure no element is selected. Then click the Lasso tool to select it. The tool background becomes highlighted to a darker shade of gray, as shown in the following image. With the Lasso tool activated, click and drag around the designed process model to select all the elements as shown here (you can see the selection boxes showing on all the elements).



With all the elements selected, drag it into the Employee lane of the pool we just created as shown next.

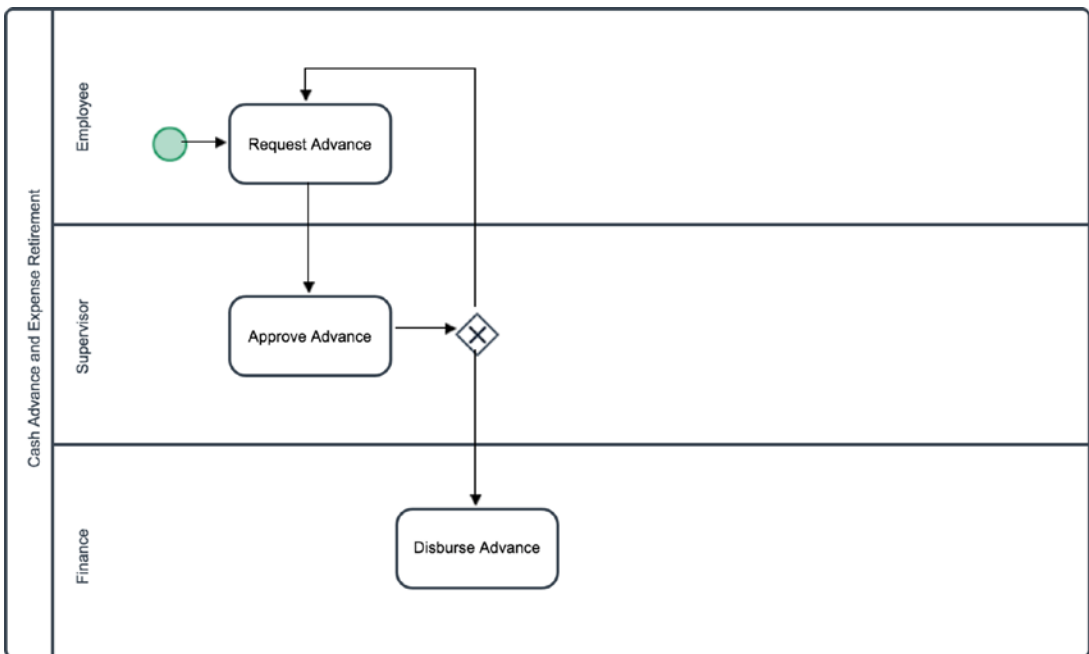


We now have our process in a pool, and next we will move the tasks to their appropriate lanes. First, let us delete the Group artifact (the dotted rectangle with the label “Cash Advance part of the process”) for now. Click any free area within the rectangle and then the Delete icon in its quick toolbar.

The Request Advance task will remain in the Employee lane because the task is performed by the employee. We drag the Approve Advance task and the gateway element to the Supervisor lane and the Disburse Advance task to the Finance lane. To drag an element, click it to select it—selection is indicated by the green boxes on the edges—and then use the mouse cursor to drag it to the desired location.

Delete the End event element by clicking it to display its quick toolbar and then clicking the Delete icon in the toolbar. We are deleting the End event because we will be adding the expense retirement part of our process shortly.

You can click the endpoint of the connecting arrows to reposition them so as to make the Process Map neater as we saw earlier when showing how to connect tasks. Your Process Map should now look like the following image.



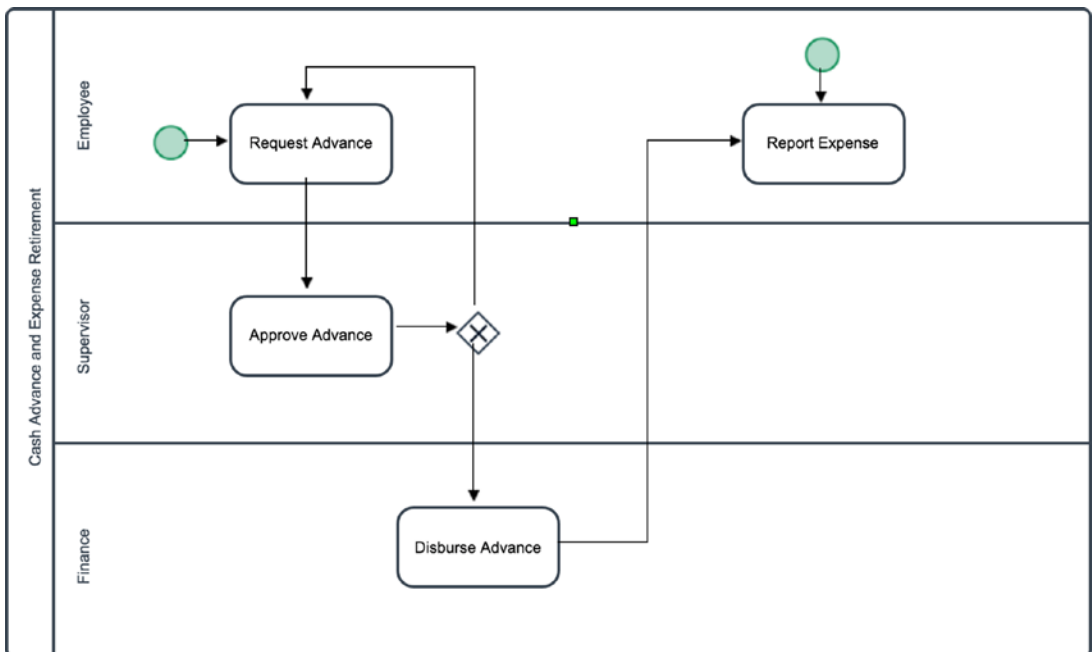
Complete the Process Model

Let us now add the Expense Retirement part of our process to the model. From our description of the process earlier, we can identify three tasks for the expense retirement process: Report Expense, Approve Expense Report, and Process Expense Report. The Report Expense task will be done by the Employee, the Approve Expense Report task by

the Supervisor, and the Process Expense Report task by Finance. This last task involves refunding or reimbursing the employee, posting the transaction on the Accounting system, and sending an acknowledgement document to the employee.

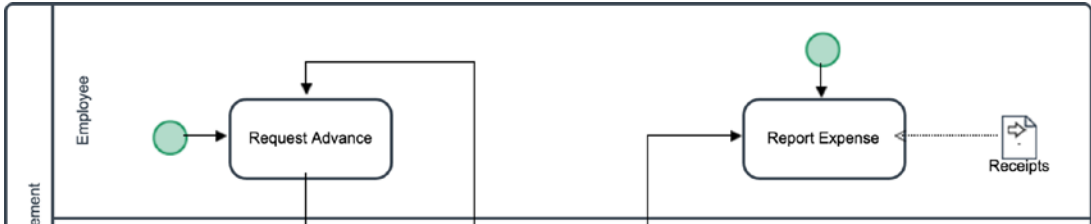
Remember that the process can start either from a request for cash advance or from reporting an expense. This means that the Report Expense task will be a starting task. Can you try to add the tasks to the Process Map on your own?

To continue from where we stopped, click the Disburse Advance task to display its quick toolbar and select the Task icon. Next, click in the Employee lane of the process to add a new task. Name the task “Report Expense.” Now drag and drop a Start Event element from the Shapes toolbox to the area above the newly created task. Click the Start Event element to display its quick toolbar and select the connect icon. Click on the Report Expense task to make it a starting task. Your Process Map should now look like the following image.



One of the requirements of the Report Expense task is that the employee will have to upload receipts as supporting documents for the expense being reported. To indicate this sub-task in the process model, drag and drop a Data Object element from the Shapes Toolbox to the right side of the Report Expense task. Click the data object to display its

quick toolbar, click the Connect icon, and click the Report Expense task to connect the object to the task. Right-click the data object and set the Data Type to Input. Also edit the label and name it Receipts.

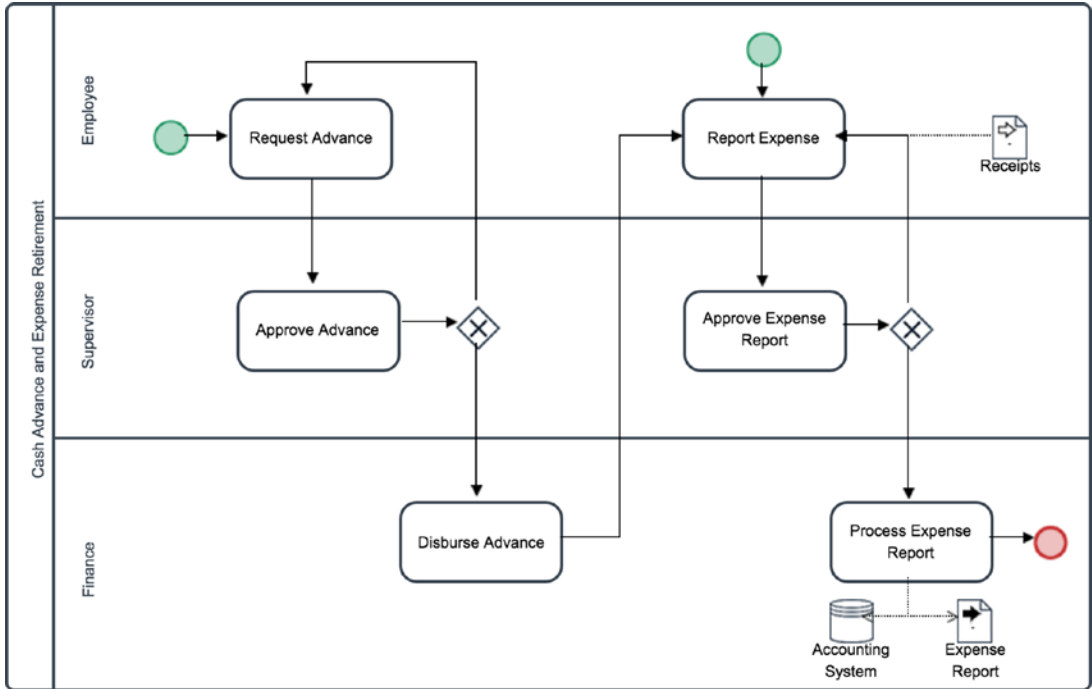


Next, we add the Approve Expense Report task in the Supervisor lane by clicking the Report Expense task to display its quick toolbar and selecting the Task icon. Click in the Supervisor lane to add the task and edit the label to “Approve Expense Report.” Just as we did for the Cash Advance leg of the process, we will add an Exclusive gateway to this task with one route going back to the Report Expense task and the other proceeding to the Process Expense Report task, which we will add to the Finance lane.

Click the Approve Expense Report task and, using the quick toolbar, click the Gateway icon and add an Exclusive gateway to the right of the task. Click on the Gateway element and from its quick toolbar select the Connect icon and click the Report Expense task. Next, click the Gateway element again and this time, select the Task icon from the quick toolbar. Click in the Finance lane to add the new task and label it “Process Expense Report.” Click the Process Expense Report task you just added and add an end event to the map from its quick toolbar on the right side of the task.

This last task will require the Finance officer to store details of the transaction in an Accounting system (external data store) and also generate a soft copy of the approved report. Let us show these in our process model. Drag and drop a Data Object and a Data Store element from the Shapes Toolbox to the bottom of the task. Connect the Process Expense Report task to the Data Object and Data Store elements by using its quick toolbar, selecting the connect icon and clicking on the Data Store and repeating the same steps for the Data Object.

Right-click the Data Object and change its Data Type to Output. Also, edit its label to “Expense Report.” Finally, edit the Data Store label to “Accounting System.” Your Process Map should now look like the following image.

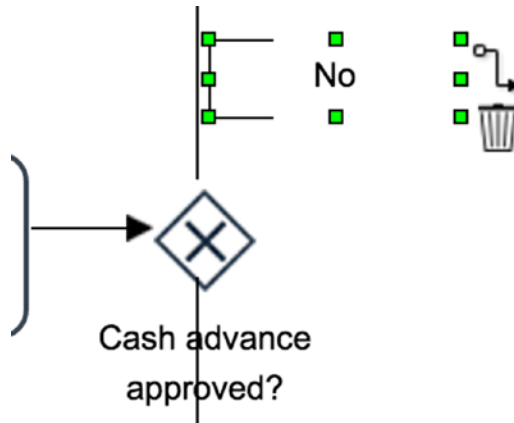


You can click the endpoints of the connecting arrows to reposition them so that the Process Map is easy to read. You can also drag the other elements to reposition them on the Process Map if they are not well aligned when you drop them on the map. As you drag the elements around, you will see dotted blue guidelines appear when the element aligns with other elements on the map.

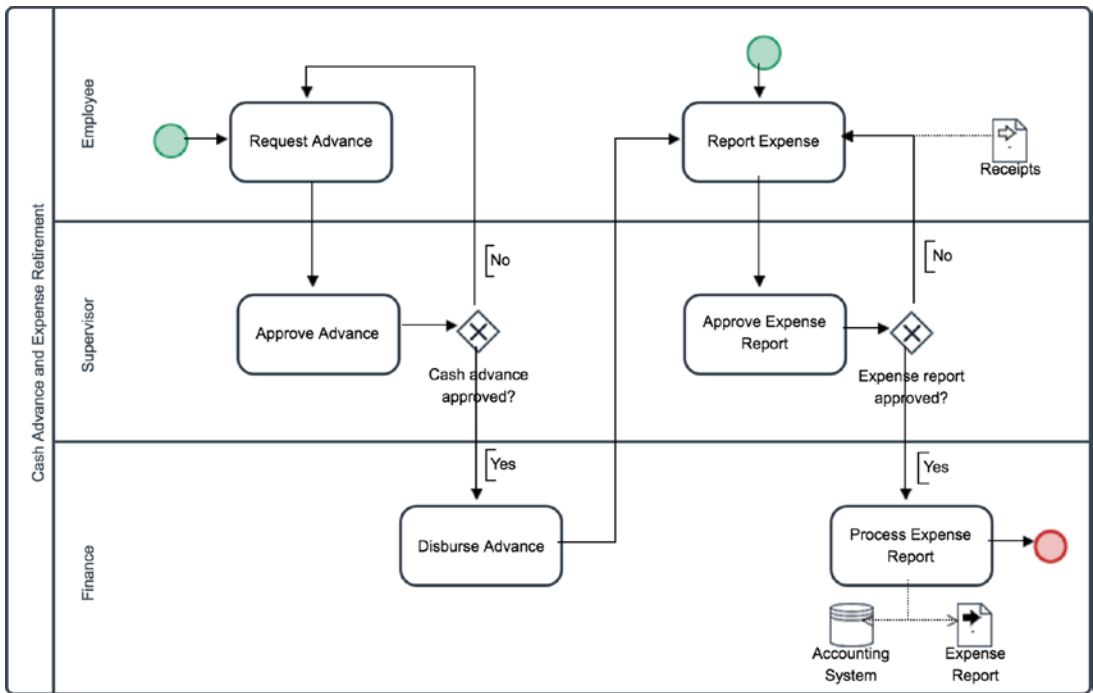
Our process model is now practically complete. We can look at it and have an idea at a glance of how the process works. However, it appears to be missing something. Can you guess what? The decision gateways that diverge the flow do not tell us what informs the decision to take one route or the other. To fix that, let us add some annotation to the Process Map.

Right-click the Gateway element beside the Approve Advance task and edit its label to “Cash advance approved?” Next, edit the label of the second Gateway element beside the Approve Expense Report task and change it to “Expense report approved?”

Now drag and drop a Text Annotation element from the Shapes Toolbox to the side of the line connecting the Approve Advance task to the Request Cash Advance task and set the text to No as shown here.



The Text Annotation element can be resized by clicking and dragging the green boxes on its edges that are displayed when it is selected. Repeat this process to add annotation (set the value to Yes) for the other connecting arrow from the Approve Advance task to the Disburse Advance task. Also add annotations for the Gateway element for the Expense Report leg of the process. Your final process should now look like the following image.

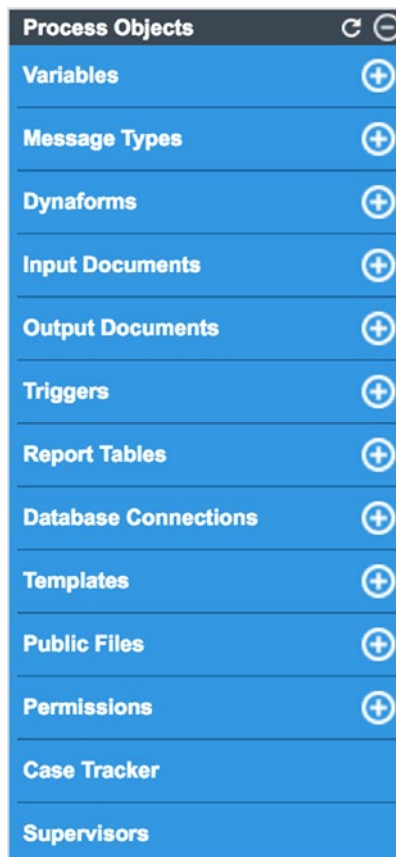


Congratulations! You have successfully created a BPMN 2.0-compliant business process model. Using all we have learned so far, you can now effectively create standard process models for the processes in your business or organization that can be easily read and understood by others. In the next chapter we begin our exploration of the building tools ProcessMaker offers to build the process.

CHAPTER 6

Building the Process

What we have now is a blueprint of our process, which is good for explaining what it does and how it should work. It is like a blueprint of a building prepared by an architect. However, we need to translate that blueprint into a building before it can be used. Similarly, we need to build out our modeled process in order to automate it.



ProcessMaker not only allows you to model your business processes in a standards-compliant way as we have done, it also enables you to automate your forms-based business processes. Automating the process allows the participants/actors in the process to log in to ProcessMaker to fill out the forms, and to assign and route them to other users to work on, instead of printing and filling out paper forms and taking them around to be signed and processed. Automation also allows you to enforce business rules, and so gain insight into how the process works, such as how long particular tasks take on the average.

To automate the process, we have to build the objects, such as forms, that the users of the process will interact with, define conditions for determining how cases in the process are routed, assign users to tasks, set up email notifications, and design templates for output documents to be generated, among other things.

When we looked at the Process Designer earlier, we identified four major components: the Top Toolbar, the Shapes Toolbox, the Main Toolbox, and the Process Map area. So far we have worked with the Top Toolbar, Shapes Toolbox, and Process Map. To bring our process to life, we will now look at the Main Toolbox, which contains the objects we will use in building our process.

Variables

The first type of object in our toolbox is the variable. You can think of a variable as a container for storing the data input by users or generated by the system for every case (a case is an instance of a process) of the process. The Variables button in the Main Toolbox allows us to create and edit the variables we want to use for our process. In addition to the variables we create, ProcessMaker also automatically adds a set of variables to all processes. These are referred to as System Variables and we shall see them later.

Referring back to our Cash Advance and Expense Retirement forms from Chapter 1, let us try to identify the fields to be filled out on the forms. These fields are some of the data that will be input into our process. We will therefore need to create variables to store these data.

Cash Advance Form Fields		Expense Retirement Form Fields	
1.	Request Date	1.	Report Date
2.	Employee Name	2.	Employee Name
3.	Department	3.	Department
4.	Reason for Expenses	4.	Reason for Expenses
5.	Requested Amount	5.	Expense Breakdown
		a)	Description
		b)	Receipt Attached (Y/N)
		c)	Amount
6.	Requestor Name	6.	Total Amount Spent
7.	Requestor Signature	7.	Amount Advanced
8.	Approved By	8.	Amount to Be Reimbursed
9.	Approver Signature	9.	Amount to Be Refunded
10.	Amount Advanced	10.	Prepared By
11.	Date Advanced	11.	Signature (Prepared By)
12.	Disbursed by	12.	Approved By
13.	Finance Officer Signature	13.	Signature (Approved by)
14.	Received by	14.	Amount Refunded
15.	Receiver's Signature	15.	Cash/Check(Amount Refunded)
		16.	Cash/Check (Amount Reimbursed)
		17.	Transaction Reference
		18.	Processed By
		19.	Finance officer signature

In addition to the variables whose values will be supplied in the forms, we can also have variables that we use to hold other types of data, such as which decision to take on a task. For example, we will require a variable where we can store the decision of the supervisor to approve or reject a cash advance request or expense report.

If you are wondering if you have to create all the variables your process will require before you can proceed to building your process, the answer is No. You can also create variables from the Dynaform Designer. This comes in handy when you realize while building a form that you need additional fields.

Hovering over the plus (+) icon in the Process objects toolbox highlights it in green and shows the Create label. When using this toolbox, clicking the (+) icon opens the Create form for that option, and clicking the option name displays the list of the objects. To see how this works, let us create our first variable. Click the plus icon (+) beside the Variables option. This displays the Create Variable form, shown next. The following sections briefly explain each of the form fields.

The screenshot shows the 'Create Variable' dialog box. It includes the following fields and options:

- Variable Name*:** A text input field containing the text 'Name'.
- Variable Type*:** A dropdown menu currently showing 'String'. Below it, a note reads: 'Supported Controls: text, textarea, dropdown, radio, suggest, hidden.'
- Database Connection:** A dropdown menu currently showing 'PM Database'.
- Sql:** A text area with the placeholder text: 'Insert a SQL query like: SELECT [Key field], [Label field] FROM [Table name]'.
- Define accepted variable values:** A checkbox that is currently unchecked.

At the bottom center of the form, a note states: 'Fields marked with asterisk (*) are required.' At the bottom right, there are two buttons: a red 'Cancel' button and a green 'Save' button.

Variable Name

This is a required field, and you use it to give a name to the variables. Variable names follow the standard PHP variable naming rules: “A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores”. The name you give your variable must be unique.

You can use camel case, Pascal case or snake case when naming your variables; however, try to be consistent and not mix it up. This makes it easier for others to work on your processes. The following definitions of the different case types are adapted from Wikipedia (https://en.wikipedia.org/wiki/Camel_case):

Camel Case: The practice of writing compound words or phrases such that each word or abbreviation in the middle of the phrase begins with a capital letter and omits hyphens. Camel case may start with a capital letter or with a lowercase letter. Examples: `EmployeeName`, `requestDate`.

Pascal Case: The practice of writing compound words or phrases such that the first letter of each concatenated word is capitalized. No other characters are used to separate the words, like hyphens or underscores. Examples: `EmployeeName`, `RequestDate`.

Snake Case: The practice of writing compound words or phrases in which the elements are separated with one underscore character (`_`) and no spaces, with each element's initial letter usually lowercased within the compound and the first letter either upper or lower case. Examples: `employee_name`, `Request_date`.

Also ensure that the names of the variables are descriptive enough to let you know what it does. Looking at the following two variables, can you guess what data we intend to store in them?

- `atbr`
- `amount_to_refund`

The second variable is clearly more descriptive, and if you return to this process weeks or months later, you will not have to wonder what data the second variable stores, as you would with the first variable.

Throughout this guide, we will adopt the lower snake case for our variable names.

Variable Type

This is a required field and is used to indicate the type of data that will be stored in the variable. The available options are as follows:

String: This is ideally used for storing text data but can also be used for letters, numbers, symbols and spaces. An example of data we would store as a string variable is a name or address.

Integer: This is used for whole numbers between -2147483648 and 2147483647. The number can be either positive or negative. An example of data we would store in an integer variable is a quantity of items.

Float: This is used for real numbers with decimal points between -3.402823466E+38 and 3.402823466E+38. If the number has 9 or more decimal points, it will be rounded up to the nearest whole number. An example of data we would store as float is Amount (currency).

Boolean: This is used to store True or False values, which can be used for logic evaluation. The system casts the number zero or empty string as False. An example of data we would store in a Boolean variable is the answer to a YES/NO question.

Datetime: This is used to store date and time values. The date format is YYYY-MM-DD or YYYY-MM-DD HH:MM:SS. An example of data we would store in a Datetime variable is a request date.

Grid: This is used to store tabular data. It is stored in ProcessMaker as an array. An example of data we would store in a grid is the list of items in an expense report.

Array: This is used to store a collection of values. An example of data we would store in an array variable is the payment type, which would be a collection of the different payment options, like cash, check, or bank transfer.

File: This is used to store files that are uploaded into the process. An example of data we would store in a file variable is a receipt. When using a file variable, an additional field for the related input document will be displayed. We will learn more about input documents later.

Multiple File: This is used to store multiple files and can only be used with the new Multiple File control introduced in version 3.1. An example of data we would store in a multiple file variable is a collection of receipts.

Database Connection

This field is used to specify the database connection that will be used when executing SQL queries for variables that have specified an SQL Query. It is set to PM Database by default, which is the ProcessMaker database. If you have created other database connections, as you will learn how to do later, those connections will be available for selection in this field.

SQL

This field is used to specify an SQL query that can be used to fetch data to be stored in the variable. If the variable is used in a text box or text area, the result of the query will be the default value of the field. For checkboxes, radio buttons, or dropdowns, the result of the query will be available as options to select. The database connection selected in the Database Connection field earlier will be used for the query and should not be stated as part of the query.

For portability of processes between Windows and Linux environments, ProcessMaker recommends using UPPERCASE for table and field names in the query. Windows is case-insensitive, while Linux is case-sensitive. This is good advice that should be adhered to. I have spent hours trying to figure out why a process that was working fine on my system stopped showing drop-down options when deployed to the live server only to realize later that the query was written in lowercase.

Define Accepted Variable Values

If you want to provide users with a limited set of options to choose from when using a dropdown box, radio button, suggest box, or checkgroup field in a form, check the box to define these options. This field is available for string, integer, float, Boolean and array variable types. Clicking on the checkbox exposes input fields for you to enter a key and label for each of the options. For example, suppose we want users of an application form to select the type of employment (full time, part time, or contract) from the options in a field instead of filling in whatever they like. Here we would enter a key and label for each option and click the Create button to add it to the list of accepted variable values. The key must be unique and is what will be stored in the database. The label is the text that will be displayed to the user. You can edit and delete the values added to the list.

Create the Variables

Now that we are familiar with the Create Variable form, let us create the variables for the Cash Advance Request form. The following table shows the variable name and variable type for the variables we will be creating and the associated fields from our form.

Two things to note: first, we have replaced the signature fields with Datetime fields. This is because we will not be capturing user signatures on the forms that will be filled on the system. Instead, we will capture the name of the user that is currently logged in when an action was performed (for example, the user who approved a request) and take a timestamp of the time the action was performed and that will serve as “signature.” This timestamp will be displayed in a text field, so we can store them in a variable of type string and not Datetime as we did for the other datetime fields.

In case you are wondering if it is possible to capture signatures electronically, the Enterprise edition of ProcessMaker has plugins that you can use to capture electronic signatures from signature pads or add digital signatures to the forms. It also enables finger signature from the ProcessMaker mobile app.

The second thing you will notice is that our process model does not include a task for the employee to acknowledge the receipt of the funds as shown on the form with the Received by and Receiver’s Signature fields. We will be leaving the acknowledgement part of the form out of the process as it is not included in the model. As an exercise, can you think of how this can be added to our process model?

S/N	Form Field	Variable Name	Variable Type
1	Request Date	request_date	Datetime
2	Employee Name	employee_name	String
3	Department	department	String
4	Reason for Expenses	expense_reason	String
5	Requested Amount	amount_requested	Float
6	Requestor Name	requestor_name	String
7	Requestor Signature	requestor_datetime	String
8	Approved By	approver_name	String

(continued)

S/N	Form Field	Variable Name	Variable Type
9	Approver Signature	approver_datetime	String
10	Amount Advanced	amount_advanced	Float
11	Date Advanced	date_advanced	Datetime
12	Disbursed by	disbursed_by	String
13	Finance Officer Signature	disbursed_datetime	String
14	Received by	-	
15	Receiver's Signature	-	

Click the (+) icon for the Variables option in our toolbox to display the Create Variable form. Enter **request_date** in the Variable Name field and change the variable type to Datetime. Click the Save button. The newly created variable should be displayed in the Variables modal as shown here.



Click the Create button in the top-right corner and proceed to create the remaining variables according to the table above. Once you are done, the Variables modal should display the list of all the variables we have created. The modal has a Search box on the top left that you can use to find a variable. This comes in handy when we have a process with so many variables. We will now proceed to create our first form and see how the variables we created are used in the form.

Dynaforms

The next tool we will explore in our main toolbox is Dynaforms (dynamic forms). As you may have guessed, this is where we create the forms that users will fill in when using the process. Just as we did with the Variables option, let us begin by clicking the (+) icon

beside the Dynaforms option to create our first dynaform. This displays the Create Blank Dynaform modal with fields for the Title and Description of the form.

Let us create the Cash Advance Request form. In the Title enter **Cash Advance Request Form** and in the Description field, enter **This form will be filled by an employee requesting a Cash Advance**. It is good practice to give your forms meaningful names and provide descriptions, especially when working on a process with others. It is a means of documenting what you are doing to make it easier for others and your future self to understand how the different pieces fit into the overall process.

The image shows a screenshot of a software interface titled "Create Blank Dynaform". The main area is labeled "Dynaform Information" and contains two input fields. The "Title*" field contains the text "Cash Advance Request Form". The "Description:" field contains the text "This form will be filled by an employee requesting a Cash Advance". At the bottom right of the modal, there are three buttons: a red "Cancel" button, a green "Save & Open" button, and a green "Save" button.

With the title and description filled in, click the Save & Open button to create the blank form and open the form in the Dynaform Designer. The Dynaform Designer is where you lay out the design of your form, adding fields and associating them with variables.

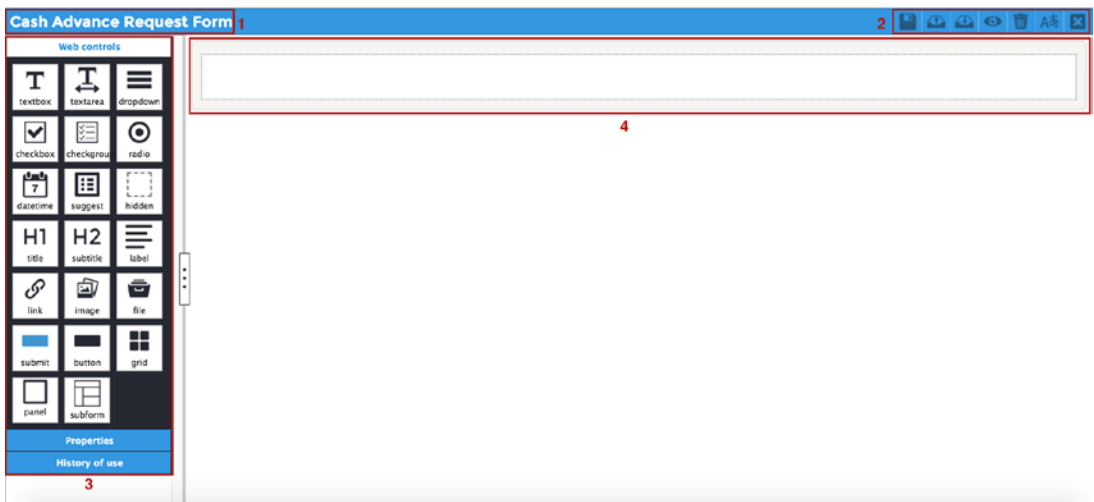
Before we return to add fields to the Cash Advance Request Form in Chapter 9, in the next chapter we'll take a quick detour to explore the Dynaform Designer and then in Chapter 8 explore the different types of form controls available to us when designing our form.

CHAPTER 7

The Responsive Dynaform Designer

Designing forms in ProcessMaker is quite easy with the new Responsive Dynaform Designer, introduced in ProcessMaker version 3. The Designer allows to you drag and drop controls such as text boxes, dropdowns, and grids to build a form without writing any code. You can also add third-party libraries to the forms you build. The forms are responsive by default and use the popular bootstrap framework to divide each row into multiple columns allowing us to create different layouts for our forms. Let's familiarize ourselves with the Designer workspace, as we will be spending a lot of time here when building our process.

The Dynaform Designer is divided into the segments shown here, explained throughout this brief chapter.



The Dynaform Designer

Dynaform Title

The title is displayed in the top-left corner of the Dynaform Designer (labeled 1 in the image). This is the same title we defined when creating the dynaform. The title can be changed by editing the properties of the dynaform.

Dynaform Designer Menu

The Designer menu is located in the top-right corner of the Designer (labeled 2 in the image). The menu lists actions that can be performed on the dynaform. Let us quickly look at what each option does.

Save



Click this icon to save the changes to the dynaform. A green “toast” message is shown when the form is successfully saved.

Export



Use this option to export the dynaform and save it to your system. The dynaform is exported as a .json file that contains the definition of the form, form fields, and their properties. This is very useful for backing up your dynaform before making changes that might break the form or creating another form with very similar fields.

Import



Use this option to import a previously exported dynaform .json file to overwrite the current dynaform. If the dynaform already has fields in it, you will be prompted to confirm that you want to overwrite the form with the imported form. We will see examples of import and export later in this book.

Preview



This is used to see how the dynaform will render on a user's screen when running a case. The preview has three modes: Desktop, Tablet, and Mobile. We will illustrate this when we add a few controls to our form. Also, the Preview mode allows us to test form validations, such as required fields and fields such as dropdowns that use SQL queries to populate their options.

Clear



This is used to remove all controls from the dynaform and reset it to a blank form. A prompt is displayed, requiring you to confirm the action before the form is cleared.

Language



The Language option allows you to define translation files for the dynaform, so you can change the labels displayed on the dynaform based on the selected language. The default language is English.

Close



Clicking this closes the Dynaform Designer and takes us back to the Process Designer. If there are unsaved changes in the form, you will be prompted to either save or discard the changes. To return to the Dynaform Designer, click Dynaforms (click the label and not the plus icon) in the Main Toolbox to display the list of dynaforms in the process. Click the Edit button for the dynaform to open the Dynaform Designer.

Dynaform Control and Properties Panel

On the left side of the Dynaform Designer is the control and properties panel (labeled 3 in the image earlier). The panel is divided into three panels, Web Controls, Properties and History of Use, in the open source edition of ProcessMaker. The Enterprise edition includes a fourth section, Mobile Controls, which includes additional controls for the mobile app, such as the signature, image, audio and video controls.

Web Controls

This displays the different controls that we can use to build our form. The panel contains the following controls that can be added to the form:

textbox

textarea

dropdown

checkbox

checkgroup

radio

datetime

suggest

hidden

title
subtitle
label
link
image
file
fileupload
submit
button
grid
panel
subform

We will explore the web controls in detail in the next chapter.

Properties

This option displays the properties of the selected control in the form. We can use it to configure and change the properties of the control as required.

History of Use

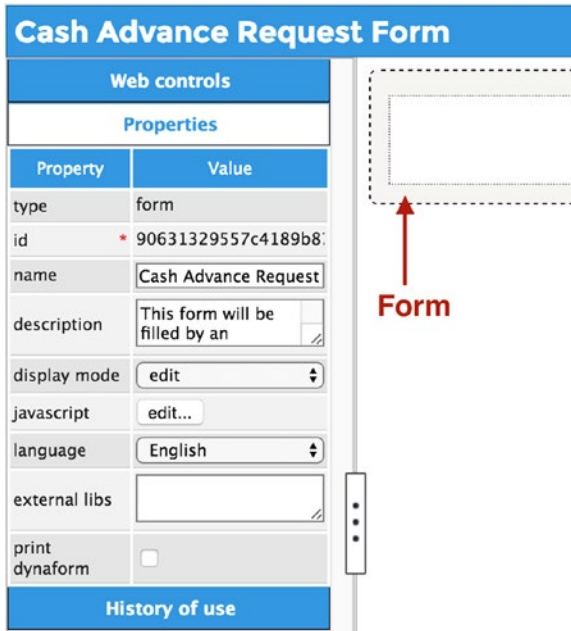
This appears to be a placeholder for a yet-to-be-implemented functionality. It is likely to show the different versions of the dynaform with an option to revert to a specific version.

Dynaform Container

This is the container for the different controls that will be added to the form (labeled 4 in the image of the Dynaform Designer earlier). The container is made up of the *form control* and one *row control* when the form is created. Additional rows are added to the form by dragging web controls onto the empty row control.

Form Control Properties

Clicking the gray portion of the container selects the form and displays its properties in the properties window in the left panel of the Designer as shown in the following image.



The properties of the form are explained next:

Type: The type of control. This property is displayed for every control.

Id: The unique identifier of the form.

Name: The title of the dynaform that was set when the form was created. The title of the form can be changed by editing this property.

Description: The description of the dynaform that was provided when the form was created. The form description can be edited by changing this property.

Display mode: This is used to indicate the default display mode for controls on the form. The options are Edit, View, and Disabled. The Edit mode allows controls on the form to be editable; users will be

able to enter and change their values. With the View and Disabled modes the controls will be read-only. The View mode replaces the input control with a plain white space showing only the label and entered value, while the disable mode still shows the input control, but greyed-out. By default all controls on the form inherit the display mode of the form, but this can be overridden for each control.

Javascript: This is used to add custom JavaScript code to the dynaform to enhance the form and add custom logic to the form such as hiding and showing fields or performing a calculation based on values entered by a user.

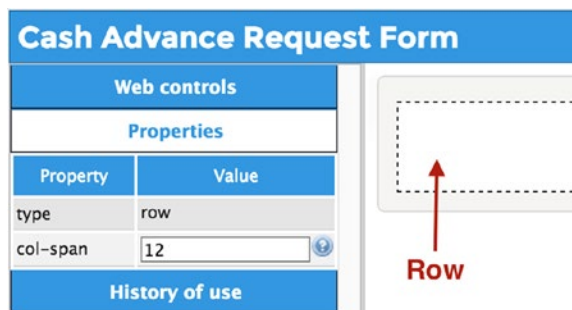
Language: This is used to select the display language for the form. The options available depend on whether other language translation files have been added for the form.

External libs: This is used to add third-party JavaScript and custom style sheets to your form. This extends the possibilities of the functionality that we can add to a dynaform and allows us to also change the look and feel as we desire.

Print dynaform: This option adds a print icon to the dynaform that can be used by users to print the form.

Row Control Properties

The row control is key in laying out the fields in our form. Clicking on the white part of the dynaform container selects the row and displays its properties in the properties panel on the left. The control has only one editable property, col-span, which allows us to divide the row into columns and set the size of each column.

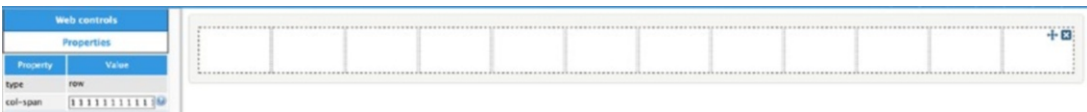


Each row consists of 12 columns and the col-span setting is used to group them together to divide the row. It is recommended that each column have a col-span value of at least 3 to render properly on all devices. The concept is very similar to Bootstrap’s grid system.

i Bootstrap is one of the most popular front-end/UI frameworks for building mobile-first, responsive web applications. You can learn more about the grid system here: http://www.w3schools.com/bootstrap/bootstrap_grid_system.asp.

To help you understand the system, the following examples show different values set for the col-span property of a row and the resulting effect in dividing the row into columns.

Setting col-span to 1 1 1 1 1 1 1 1 1 1 1 1 we have 12 columns of 1 span each.



Setting col-span to 3 3 3 3 we have 4 columns of 3 spans each.



Setting col-span to 2 3 6 1 we have 4 columns, the first with 2 spans, the next with 3 spans, followed by a 6 span column and a single span column.



The value of the col-span property must add up to 12. If it is more than 12, the last column is automatically reduced to make the value 12, and if less than 12, an additional column is added to make up the difference.

Go ahead and experiment trying different values to get comfortable with dividing the rows into columns.

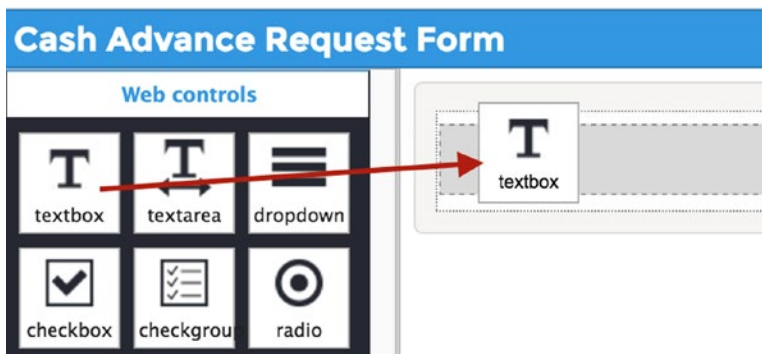
In the next chapter, we explore the web controls, which serve as the building blocks for creating our dynaforms.

CHAPTER 8

Dynaform Web Controls

The dynaform web controls allow users of our process to interact with the form. Before we start building out our form, let us quickly learn more about the different web controls available to us. Our approach will be to add one of each type of control available to the form and then proceed to explore them one after the other. To add a control to the form, simply drag and drop it in an empty row on the form.

You will be prompted to Create/Select a variable to associate with the control.



Creating Variables from the Dynaform Designer

Our lesson on variables mentioned that you can also create variables from the Dynaform Designer. In the prompt displayed when adding a new control to the form (see the following image), we can see two options:

The image shows a dialog box titled "Create/Select Variable". It has two radio buttons: "Create variable" (selected) and "Select variable". To the right is a green button with a plus sign and the text "Variables". Below the radio buttons is a "Variable Name" field containing "textVar". Underneath is a "Settings" link. The settings are: "Type" is a dropdown menu set to "String"; "Database Connection" is "PM Database"; "Sql" is "..."; and "Options" is "[]". At the bottom right are "Cancel" and "Save" buttons.

Create Variable: This allows you to create a new variable by entering the variable name. Below the Variable Name field is a Settings link, which you can click to edit the settings for the variable, such as the variable type or associated input document. The fields available in Settings depend on the type of control you are adding to the form. For example, if adding a textbox, you will see options to define whether the variable is a string, integer or float. If adding a file control, however, the options displayed in the settings will be what input document to associate with the variable.

Select Variable: This allows you to choose from the variables created earlier. The only variables that will be listed will however be those that match the control being added to the form. For example, only string, integer, or float variables will be available for selection when selecting a variable for a textbox control.

If you have the Create/Select Variable dialog open, click the Cancel button. This will still add the control to the form but not create a new variable. We don't want to pollute our workspace with needless variables in the Cash Advance process, so for the examples in this chapter we will use the first process we created earlier as a playground to explore the web controls. Click the Clear button in the Dynaform menu to clear the form, save it, and then close the dynaform. Close the Workflow Designer to return to the list of processes.

Data Type	Variable Name
[float]	amount_advanced
[float]	amount_requested
[string]	approver_name
[string]	department
[string]	disbursed_by
[string]	employee_name
[string]	expense_reason
[string]	requestor_name

Open the process titled “My first process” (the name could be different if you used a different name when creating it earlier). Click the Create icon for Dynaforms in the Main Toolbox to create a new dynaform. Title it **My first form** and give it the description

A playground to learn about form web controls. Click the Save & Open button to open the new form in the Dynaform Designer. With our form all set up, we can start adding controls to it and learning about them.

Textbox



We begin with the textbox. This is a control you will use a lot in many of the forms you create. Drag and drop the textbox control from the web controls panel on the left onto the empty row in the dynaform container on the right. The Create/Select variable dialog is displayed. Change the variable name to **my_textbox** and click the Save button to add the textbox to the form. This creates a new variable for us and adds a new empty row control to the dynaform container below the row where we placed the textbox control.



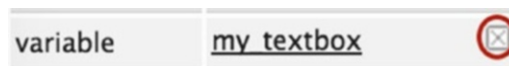
The textbox control has a label, `text_1`, as we can see in the image. Click the textbox control to select it. This displays its properties in the Properties panel on the left. Let us look at the properties of the textbox.

type: This is set to text by default and cannot be changed. It’s a “text” box, after all.

variable: This is the variable that is associated with the control; in other words, the value entered in this textbox will be stored in this variable. Remember from the discussion of variables that you can think of them as a container. Whatever the end user of this form enters in this control will be stored in that container.

To change the variable associated with a control, click the variable name. If there is no variable associated, there will be an ellipsis (...) where the name should be. Clicking the ellipsis or variable name displays the Select/Create Variable dialog, which allows you to select a variable or create a new one as we have seen. You can also clear the variable associated with the control by clicking the Close button beside the variable name.

Go ahead and try it out. Click the Close button beside the variable name to clear the variable. In the confirmation dialog that appears, click Yes. Next, click the ellipsis beside the variable name, and choose the Select Variable radio option in the Create/Select Variable Name modal that appears. You should see the variable (`my_textbox`) we created earlier. Click it to select it.





variable data type: This shows the data type of the variable associated with the control. Recall that the data type of a variable can be a string, integer, float, Boolean, datetime, grid, array, or file.

protected value: This is a new property introduced in version 3.0.18 of ProcessMaker; it simply allows us to make the value of the control immutable. That is, once the value of the variable is saved, it cannot be changed.

id: This is the unique identifier of the control on the form. It is set to the name of the associated variable by default if a variable is associated with the control. The id of every control on the form must be unique.

label: This is the name of the control that is displayed to the end users of the form. It should be a meaningful name that helps the users know what information should be input or displayed in the control. Our textbox control's label `text_1` doesn't convey any meaningful information. Change it to **What's your work email?** You will notice that the label on the control also changes.

Property	Value
type	text
variable	<u>my_textbox</u> 
variable data type	string
protected value	<input type="checkbox"/>
id *	<input type="text" value="my_textbox"/>
label	<input type="text" value="text_1"/>
default value	<input type="text"/>
placeholder	<input type="text"/>
hint	<input type="text"/>
required	<input type="checkbox"/>
text transform to	<input type="text" value="none"/>
validate	<div style="border: 1px solid #ccc; padding: 5px;"> Use a pattern (to be used in a search). </div> <input type="button" value="help"/>
validation error message	<input type="text" value="Error message"/>
max length	<input type="text" value="1000"/>
formula	<input type="button" value="edit..."/>
display mode	<input type="text" value="parent"/> 
datasource	<input type="text" value="database"/>
DB Connection	<u>PM Database</u>
sql	<input type="text" value="..."/>

Default value: As the name implies, you can use this property to define the default value for the textbox. If the user of the form does not change it, this is the value that will be stored in the associated variable of the control. Set the default value to **no-email@msbcorp.com** or any text you please.

Placeholder: This is used to set the placeholder text for the control. A placeholder is text that is displayed in the control when it is empty. The value of the control is not set to the placeholder if no text is entered into the control. This property is useful for providing information to the user on how to fill in the textbox; for example, we could set our placeholder to **first-name.lastname@msbcorp.com**.

Hint: This is used to provide hints to the user on filling in the form. Setting the hint property adds an info icon beside the control, which when hovered over displays the hint. Set the hint to **Enter your official email address**.

Required: This is used to indicate that the field is required and must be filled. A red asterisk is added to the label of the control to indicate that it is required if this property is checked. Go ahead and check the box.

Text transform to: This is used to define what case to use for the text entered into the control. The available options are None, lowercase, UPPERCASE, Title Case and Capitalize phrase.

Validate: This is used to define regular expressions that can be used to validate the value entered into the control. Also referred to as *regex*, regular expressions are patterns used to match character combinations in strings. Regular expressions are beyond the scope of this book, but you can learn more about them here: https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Regular_Expressions. A common pattern that is validated in a textbox control is email address, and this is provided for us in ProcessMaker out of the box. Clicking the Help button displays the regex for an email address. Click the regex to select it and add email validation to the textbox.

Validation error message: This is the error message that will be displayed to the user of the form if the value entered in the textbox does not pass the validation. Let's set the error message to **Please enter a valid email address**.

Before continuing to the remaining properties, let us see the effect of our changes on the control. To preview what our control will look like to end users, click the Preview icon in the dynaform menu in the top right of the screen. If the session has timed out, you will be redirected to the login screen. There is no need to panic; just enter your username and password and you will be redirected back to the preview of the form. In the preview screen, we should see a textbox like the one shown next.



Proceed to delete the default value, **no-email@msbcorp.com**, from the textbox. This will display the placeholder we set as shown here.



Now type an invalid text like **horo** into the textbox and click outside it to deselect it. You will see the validation error message we set to check that the value entered by the user is a valid email address.



Delete the text entered into the textbox, leaving it blank, and click outside the field. This will show us another error message, telling us that this field is a required field.



Finally enter a valid email address in the field, and the error messages should all clear up, as shown here.



Let us return to the remaining properties of the textbox control. Click the Close button in the preview window to return to the Dynaform Designer.

Max length: This property is used to define the maximum number of characters that can be entered into the control.

Formula: This is used to define a formula that will determine the value of the control. For example, let us assume we wanted to reimburse expenses based on distance traveled and have a set rate per kilometer; we can determine the amount to reimburse by setting the formula property on the `amount_to_reimburse` field. You would have a control named `cost_per_km` where the user enters an amount for the cost per kilometer traveled and another control named `total_distance` for entering the distance covered. You could then set the formula property on the `amount_to_reimburse` field to **`cost_per_km * total_distance`** to get the value automatically.

Display mode: With this property you can define whether the control should be editable or not. You can also inherit the display mode of the parent (the containing form or grid). In our formula example, we would want to make the `amount_to_reimburse` field read-only by setting its display mode to View.

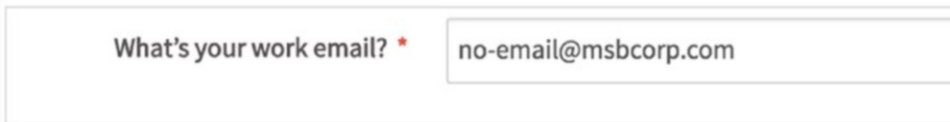
To see how the different display modes are rendered on the form, let us change this property on the textbox. We have already seen what the default parent display mode looks like when we previewed the form. Change the display mode to View and click the Preview icon. It should look like the following image.

A screenshot of a form field. The label is "What's your work email? *" and the value is "no-email@msbcorp.com". The field is active, with a cursor visible at the end of the text.

Close the preview window, select the textbox control, change the display mode property to Disabled, and preview it again. It should look like this.

A screenshot of a form field. The label is "What's your work email? *" and the value is "no-email@msbcorp.com". The field is disabled, indicated by a grey background and a greyed-out text color.

Close the preview window, select the textbox control, change the display mode property to Edit, and preview it again. It should look like the following. This is just like the Parent display mode, which tells us that the display mode of the parent of this control is also set to Edit.



The image shows a web form with a label "What's your work email? *" and a text input field containing the value "no-email@msbcorp.com". The form is styled with a light gray border and a white background.

Datasource: This is used to define the source for possible values of the control. It can be either a database or an array variable we created. This property is set to database by default and the value will only be sourced from the database if an SQL query is defined in the Sql property of the control.

If the datasource property is changed to array variable, a data variable property is added to the control's properties, from which you can select an array variable containing the list of options.

DB connection: This is used to specify which database connection to use for the SQL query if any is defined for the control.

Sql: This property is used to set the SQL query that will be used to fetch possible values for the control from a database. Because this is a textbox control, only the first returned value will be used; so when using SQL queries to define the value of the control, make sure the result of the query is a scalar value.

Wow! That was a lot to take in. Before we continue to look at the properties of the other controls, I would like to point out that they all have a number of properties in common with the textbox control, and for those I will simply refer back to this description of the equivalent textbox property. We will look only at those properties that are unique to the remaining controls.

Textarea



The textarea control is basically a variant of the textbox control. It allows us provide an input where users can enter unlimited text. Let us add a textarea to our form to explore its properties. Drag and drop the textarea control to the empty row below our textbox. On the Create/Select Variable dialog that appears, set the variable name to **my_textarea** and click the Save button. Select the textarea in the dynaform container to display its properties in the Properties panel on the left.

A quick comparison of the textarea properties with the textbox properties reveals that the following properties are not available for the textarea:

Text transform to: Because we are entering large text, it is highly unlikely we want to capitalize everything the user enters.

Max length: There's no maximum, because using a textarea implies we want the user to be able to enter an unlimited amount of text.

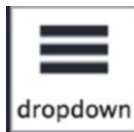
Formula: The result of a calculation is most likely numeric, and this control is for large amount of text.

However, there is a new property not included in the textbox properties—*rows*. This property allows you to set how many rows should be displayed when the textarea is rendered on the form. Let's see it in action. Change the label property of the textarea to **Tell us something** and then click the Preview icon from the dynaform menu to see how it is rendered on the form. If your session has timed out, you will be redirected to the login page. Just log in with your username and password, and you will be redirected back to the preview.

You should see the textarea rendered as shown in the following image.

To count the number of rows, click in the textarea and press Enter on your keyboard five times. The fifth Enter keystroke takes you to the sixth row, and a scroll bar should appear on the side of the textarea. Close the preview window to return to the Dynaform Designer. Select the textarea again to display its properties and change the row property to 10. Preview your form again, and you should see that the displayed textarea control is now bigger. Note that setting a value higher than 10 does not make the textarea rendered any larger; it instead adds a scrollbar. Change it back to 3 so it does not fill up our playground form.

Dropdown



The dropdown control is used to provide a list of options from which the user can select a value. The options in a dropdown can be populated from a database or predefined in the variable associated with the dropdown control. We will explore using database connections later in the guide, but for now we will stick to defining the options in the variables. Let us add a dropdown control to our form to explore its properties.

Drag and drop the control to the empty row beneath the textarea. In the Create/Select Variable modal that is displayed, change the variable name to **my_dropdown**. Then click Settings to display the settings of the variable being created. Since we will be using options and not an SQL query, click the square brackets to display the options modal.

In the options modal displayed, you will see an empty table with the headings Key and Label, with a Create button on top. Click this button to define the options that will be available for users of our form to select. The key is the value that will be stored in the variable, and the label is what will be shown to the user. The label is usually more descriptive, with the key being a code. For example, if we wanted users to select their language, we would put the ISO code of the language as the key and the name of the language as the label.

Create/Select Variable
✕

Create variable
 Select variable

+ Variables

Variable Name:

Settings

Type:

Database Connection: PM Database

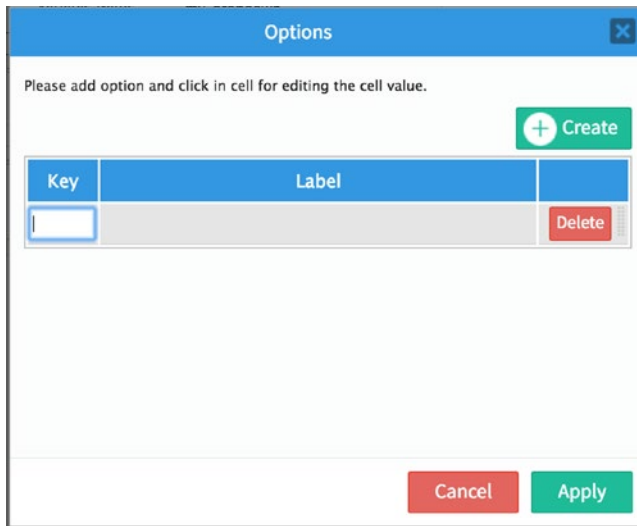
Sql:
← click to define sql query

Options:
← click to define options

Cancel
Save

Let us add some languages to our options. Type **de** in the Key column, and then click in the Label column and enter **German**. Click the Create button again to add another option and repeat for all the following values.

Key	Label
de	German
en	English
es	Spanish
fr	French
pt	Portuguese
zh	Chinese

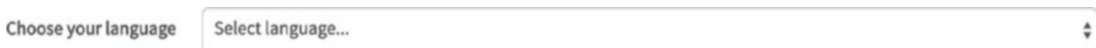


Once you are done, click the Apply button to save the options. Then click the Save button on the Create/Select Variable screen to create the variable.

Click the control in the Dynaform container to display its properties. The properties as we can see are similar to those we have seen in the two previous controls. Change the label property to **Choose your language** and proceed to preview it. We can see the dropdown control rendered as displayed here.



Something seems odd about our dropdown control. Can you guess what it is? The option German is preselected for us by default. If the user does not pay attention, they can forget to choose an option and use the default. To correct this, we would rather show a message indicating that the user has not chosen an option yet. Close the preview, select the control from the dynaform container and set the placeholder property to **Select language....** Preview the form again, and the dropdown control should now render as shown next.



As you can see, this is much better and clearly indicates to the user that this field has not been filled and they need to choose an option from the dropdown. Remember that we can also populate our options from a database query as we shall see later on. Close the preview to return to the Dynaform Designer.

Checkbox



The checkbox control is one you will most likely be familiar with from different forms you might have filled online, like login forms asking “remember me on this computer?” or sign-up forms asking to check the box to indicate “you have read and understand the terms.” A checkbox control allows you offer users of your form a binary choice like yes/no, on/off, agree/disagree, true/false, and so on.

Let us add a checkbox control to our form and explore its properties. Drag and drop the checkbox control from the web controls panel to the empty row below the dropdown control. In the Create/Select Variable modal dialog that opens, change the name of the variable to **my_checkbox**. Click Settings to display the settings of the variable. You will see that the only available type is Boolean. This is because a checkbox can only be associated with a Boolean variable.

Create/Select Variable
✕

Create variable
 Select variable

+ Variables

Variable Name:

Settings

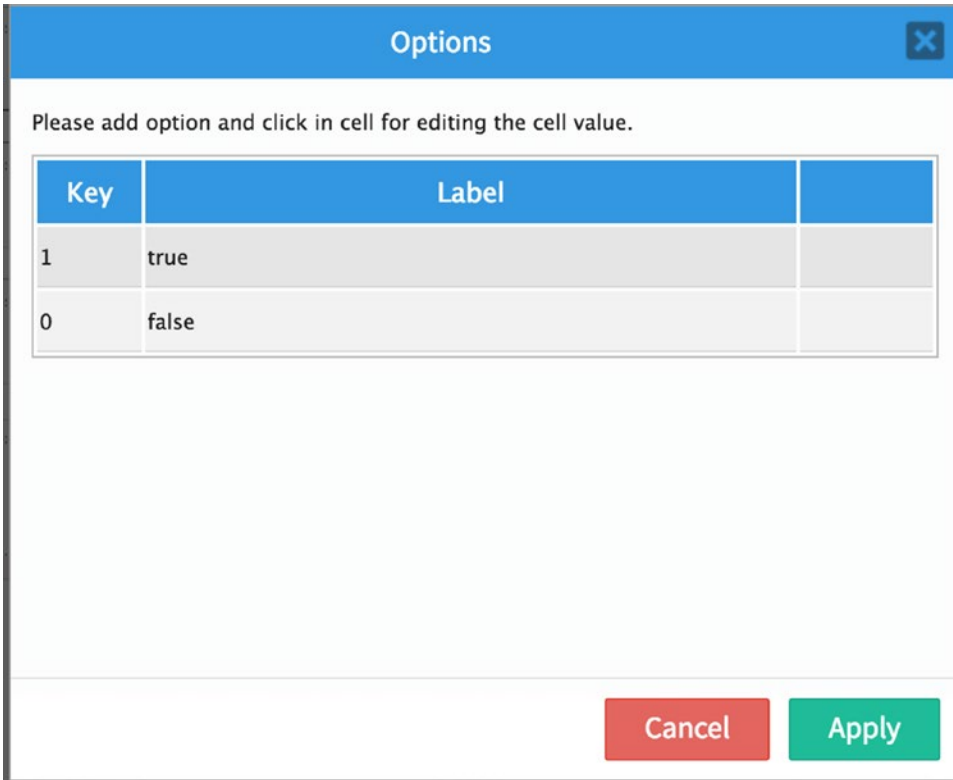
Type: Boolean

Options: [{"value": "1", "label": "true"}, {"value": "0", "label": "false"}]

Cancel
Save

If you click on the options, you will see a Key/Label table similar to the one we saw for the dropdown control earlier. However, in this options table, there is no Create button to add options, and we already have two options prefilled for us. Also, we cannot change

the values in the Key column, which are 1 and 0 (remember, it gives us binary options). All we can do is change the labels, which are set to true and false, respectively.



When the checkbox is checked, the variable will be set to 1 and if unchecked, it will be set to 0. Let us change the labels to something else. Change True to Yes and False to No. Click the Apply button and save the variable.

With the checkbox added to the form, click it in the dynaform container to select it and display its properties.

In the properties panel, we can see that all the properties available are the same as we have seen with other controls. The options property can be edited to change the labels of the options as we did when creating the variable to associate with this control. Change the label property to **I accept the terms and conditions**. If you want a user to always check the box, you can check the required property. Go ahead and preview the form to see how the checkbox looks.

Checkgroup



The checkgroup control is used to allow users select multiple options. Each option is displayed with its own checkbox, and the options that are checked are stored into the variable associated with the control. To be able to store multiple options, the associated variable will be an array data type. Let us add a checkgroup to our form and explore its properties.

Drag and drop the checkgroup control to the row beneath the checkbox on the form and, on the Create/Select Variable dialog screen, change the variable name to **my_checkgroup**. Click Settings to display the settings of the variable. The only variable type available to select is Array, as just explained. Also, just as we did for the dropdown control, we will be using the options array to populate available options for the user to select, even though we can also use database query to select possible options.

Click the square brackets in front of the options settings to display a Key/Label table, which we will use to define the options. Remember that the key is the value that will be stored in the variable, while the label is what will be shown to the user. Create the options as shown in the following table. Click the Apply button and save the variable created.

Key	Label
1	Tennis
2	Swimming
3	Soccer
4	Hockey
5	Other

Select the checkgroup control from the dynaform container to display its properties. Change the label property to **Which sports do you watch?** You will also observe that the properties are similar to the checkbox properties. If you want the user to select at least one of the options, check the Required property. Click the preview icon to see how the

control is rendered on the form. The labels are displayed beside the checkbox and listed vertically. One of the properties I would love to see is an option to choose how to display the options, vertically or horizontally. There is a workaround, however, which we will see when we look at ways you can enhance the look and feel of your forms later in this book.

Which sports do you watch?

- Tennis
- Swimming
- Soccer
- Hockey
- Other

Radio



The radio group control is similar to the checkgroup, with the difference that while the checkgroup allows you to select multiple options, the radio group allows you to select only one option from those available. As we have done for the other controls we have seen so far, let us add a radio control to our form.

Drag and drop the radio control from the web controls panel to the row beneath the checkgroup. In the Create/Select Variable modal dialog, change the name of the variable to **my_radio**. Click Settings to display the settings of the variable. You will see that we have four different types for our variable, with string selected by default. When using the radio control for a binary option as we did for the checkbox, the variable type should be set to Boolean. If you are offering multiple options with custom keys (remember that the key is the value that will be stored in the variable), however, use a variable type that matches the data type of the key.

To better see how this works, change the Type under settings to Boolean. You see that the options setting changes to what we had when exploring the checkboxes. If you click on the options, you will only be able to edit the labels and not add new options in the Key/Label table displayed. Click Cancel to close the Options modal. Change the Type

setting back to String and then click Options. In the Key/Label table displayed, create the options as shown here.

Key	Label
CSH	Cash
CHQ	Check
CRD	Credit Card
TRF	Wire Transfer

With the options created, click the Apply button and save the variable. Click the radio control in the dynaform container to select it and display its properties. We can see that the properties are the same as we have seen in the other controls explored so far. Change the label to **How would you like to be paid?** and proceed to preview it. You will observe that while the radio control is similar to the checkgroup control above it, we can only select one option, instead of the multiple options we can select in the checkgroup.

How would you like to be paid?

Cash
 Cheque
 Credit Card
 Wire Transfer

Keep in mind that we can also use an SQL query to a database to fetch the options that can be displayed in a radio control. Close the preview and let us move on to the next control.

Datetime

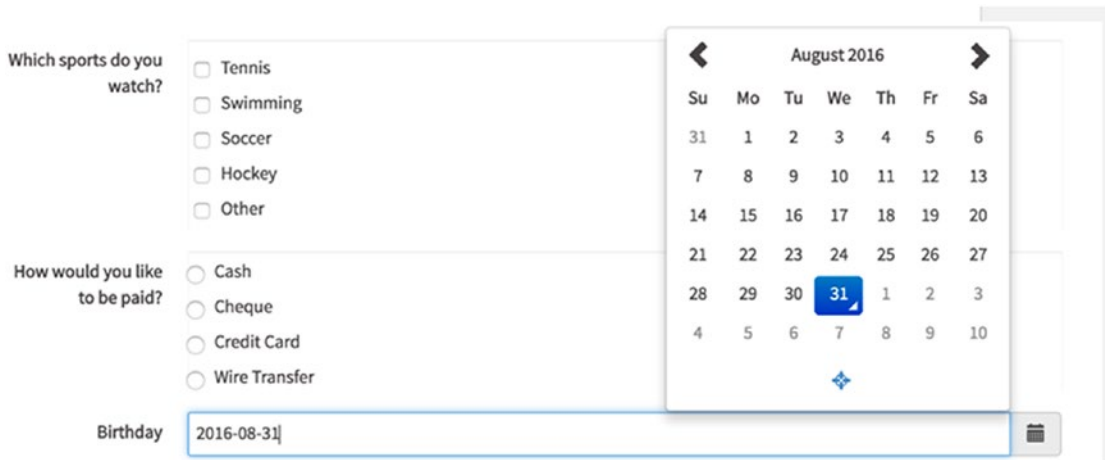


The datetime control is used for providing a calendar that users can use to select dates when filling in the form. Using a datetime control is the preferred way of capturing dates

in forms, instead of using a textbox that allows the user to type freely. The datetime control helps us ensure that the user has entered a valid date. It also allows us to perform calculations with the date or check whether the date is in the past or future. The datetime control in ProcessMaker uses the `moment.js` library, which is a “full featured date library for parsing, validating, manipulating, and formatting dates.” You can learn more about `moment.js` at <http://momentjs.com/>.

Let us add a datetime control to our form and check out its properties. As usual, drag and drop the datetime control from the web controls panel to the empty row on the form. In the Create/Select Variable modal dialog, change the variable name to **my_datetime**. If you click the settings, you will see that the type is a datetime. Save the variable and select it in the dynaform container to show its properties. A cursory glance reveals properties we have not seen in the other controls explored so far, which we shall now learn more about.

To better understand the effect of the different properties, let us change the label property to **Birthday** and preview the form to see how the datetime control looks and works with the default properties (see the following image). The control has a calendar icon on the right that when clicked displays the calendar with today’s date selected (I am writing this draft on August 31). You can click the arrows in the calendar displayed to change the month and click the month to change the year of the calendar.



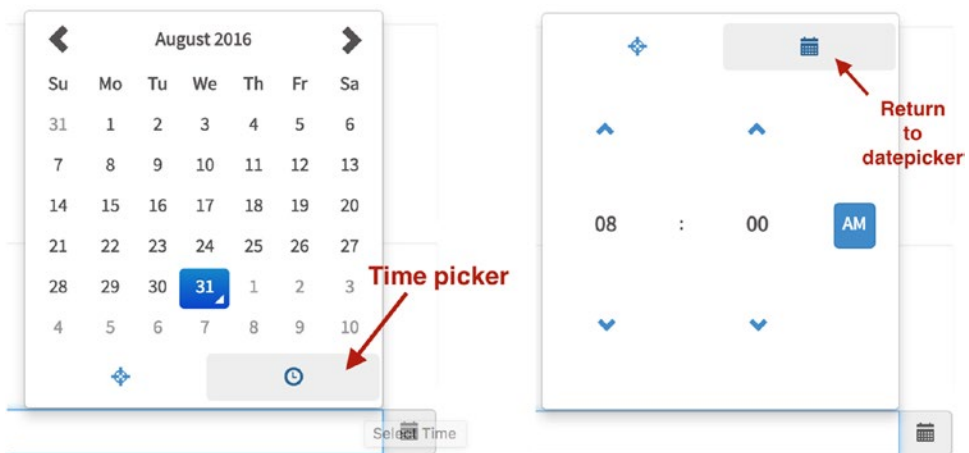
Let us now look at the properties and change their values to see the effect they have on the control. Close the preview and select the control from the form to display the properties again.

Format: This property allows you to change the way the datetime value is formatted. The default is YYYY-MM-DD, which as we can see from the image translates August 31st 2016 to 2016-08-31. To learn more about the different possible formats, visit <http://momentjs.com/docs/#/displaying/format/>. If you also hover over the hint icon beside the property, you will see a summary of available format options. Let's change our controls format to see how it works. Change the property to **MMMM Do, YYYY** and preview the control in the form. Pick a date, and it should be formatted as shown next.

Birthday 

This property really comes handy for making sure users select the right date without having to wonder whether month comes before day or vice versa.

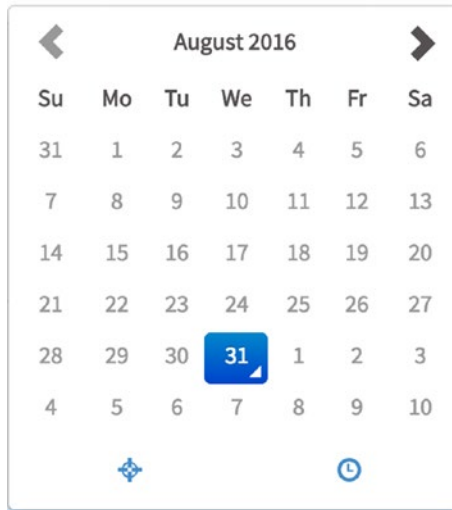
This property is also used to add a time selection to the date. If we wanted users to select a time we simply include it in the format. For example, change the format property to **HH mm A MMMM Do, YYYY**. Preview the form and you will see that a time picker is added to the calendar as shown here. Click it to select the time and then use the calendar icon to return to the datepicker.



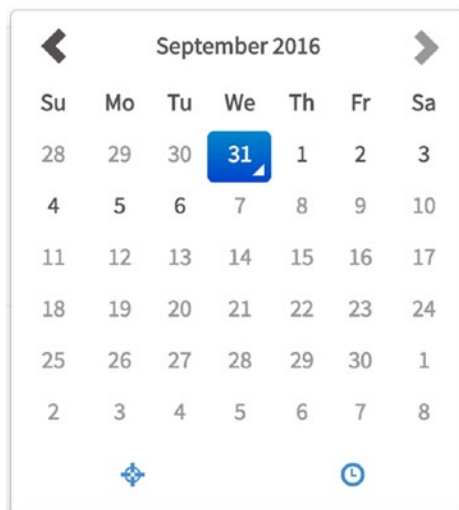
Close the preview and let us move on to the next property.

Min date: This property is used to define the minimum date that can be selected from the calendar. The value can be set by using the calendar icon beside it, which makes it fixed, or by using a variable like @@today or @@min_date, which can be set every time a case is run, making it more flexible. If you are wondering about the @@ prefix in front of the variable name, I will explain it when we learn how to work with variables.

To see it at work, use the calendar icon and set the value to today's date (I am using August 31, 2016). Preview the form and click the calendar icon on the datetime control to display the datepicker. You will observe that you are unable to select any date before the date you set as the min date as they will be disabled.



Max date: This property is the opposite of the min date, allowing you to specify the maximum date a user can select. Just like min date, it can be set using the calendar beside it or using a variable. Set the max date property to a week from today (September 6, in this example) and preview the form. You will observe that you are unable to select any date after the date set as the max date.



Close the preview and let us look at the next property.

Initial selection date: This property is used to define what date should be selected by default on the datepicker. The options for this property are:

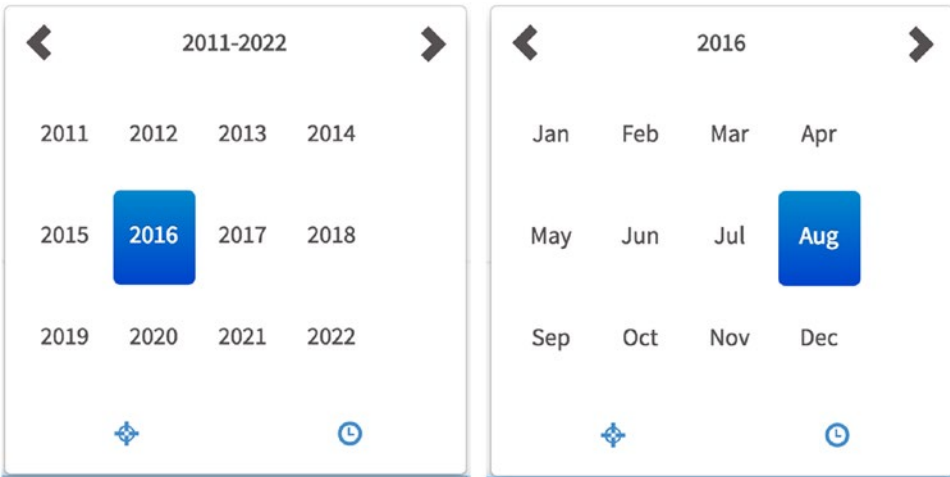
- **True:** This is the default and indicates that the initial date be set to the current date.
- **Year:** Sets the initial date to the first day of the first month of the current year.
- **Month:** Sets the initial date to the first day of the current month.
- **Day:** Sets the initial date to the current date.
- **Hour:** Sets the date to the current hour (the format property must include hour).
- **Minute:** Sets the date to the current minute (the format property must include minute).

To try out the effect of this property, you might have to clear the min date and max date properties (use the Clear icon beside them) if you set them earlier, as the value set by this property may be outside the range defined by the min and max date properties.

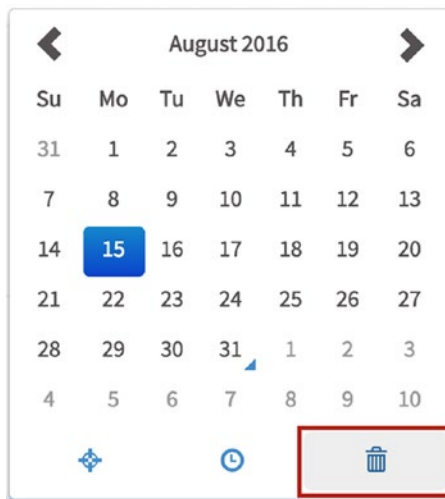
Default date: This property sets the default date to be shown on the datepicker. You can think of it as an extension of the initial selection date property discussed earlier, that allows you to specify a date not covered by the options above. It can be set using the calendar beside it or with the use of a variable just like the min date and max date properties. Set it to a value using the calendar and preview the form to try it out.

Datepicker view mode: This property sets how the datepicker should be displayed, and the options are

- **Days:** This is the default as we have seen in the examples earlier.
- **Month:** This changes the datepicker to allow users to begin selection from the month.
- **Year:** This configures the datepicker to allow users to begin selection from the year.



Show Clear button: This is used to indicate that the Clear button icon should be displayed in the datepicker. Users can click the Clear button in the datepicker to clear the value selected. This is set to Hide by default. Change the property to Show and preview the form. It should display a delete icon on the bottom of the date picker as shown in the image on the right. Close the preview window to return to the Dynaform Designer.



We have now explored all of the properties particular to the datetime control. Before we move on to learn about the next control in the web controls panel, Suggest, let's take the opportunity to try repositioning a row.

Repositioning a Row

Our controls are beginning to reach the end of the form, meaning we might have to scroll down to locate the next empty row. Let us quickly see how we can reposition rows before moving on. Simply click a free portion of the row (our last row is empty at the moment, so we can click anywhere in it). You'll see the handle and delete icons, as shown next.



Click and drag the handle to the top of the form as shown in the following image, and drop it above the row containing the textbox control we created earlier.

The screenshot shows a web form with several rows. The top-left corner of the form area is circled in red, highlighting a handle icon consisting of a blue square with a white plus sign and a blue square with a white minus sign. The form contains the following controls:

- What's your work:
- Tell us something:
- Choose your:
- I accept the terms:
- Which sports do: Tennis Swimming Soccer Hockey Other
- How would you like to: Cash Cheque Credit Card Wire Transfer
- Birthday:

Suggest



The suggest control is similar to a textbox and is used to offer suggestions to the user about the possible values that can be entered into the field. As the user types into the control, a list of suggestions is displayed and filtered based on the user's entry. The user can select one of the options or type a completely new value, as these are just suggestions. The list of suggestions can come either from a database query or a list of options, just as in the dropdown control. Let us see how it works.

Drag and drop a suggest control from the web controls panel to the empty row we just dragged to the top of our form. In the Select/Create Variable dialog that appears, change the variable name to **my_suggest**. Expand the settings and click the square brackets in front of options setting to define the options we will use. As we did with the dropdown, checkgroup, and radio controls, add the options in the following table below, click Apply, and save the variable. You might observe that we are using the same values for the key and label this time, which makes sense as we are suggesting what the user should enter, and this is more likely to be a phrase as opposed to a code.

Key	Label
Computer scientist	Computer scientist
Database administrator	Database administrator
Data analyst	Data analyst
Data scientist	Data scientist
Network analyst	Network analyst
Network administrator	Network administrator
Software designer	Software designer
Software analyst	Software analyst
Software quality analyst	Software quality analyst
System administrator	System administrator
Web developer	Web developer

Select the control from the dynaform container to display its properties. Change the label property to **What's your occupation?** and proceed to preview the form. Start typing one of the values from the table, like **data**, and the matching values will be displayed as suggestions which you can click to select as shown in the following image.

The image shows a preview of a form with three rows. The first row is labeled 'What's your occupation?' and contains a text input field with the value 'dat'. The second row is labeled 'What's your work email?' and contains a text input field with a red asterisk. The third row is labeled 'Tell us something' and contains a text input field with the value 'Data scientist'. A dropdown menu is open over the first row, displaying a list of suggestions: 'Database administrator', 'Data analyst' (highlighted in blue), and 'Data scientist'.

We can close the preview and return to the dynaform to learn about the next control.

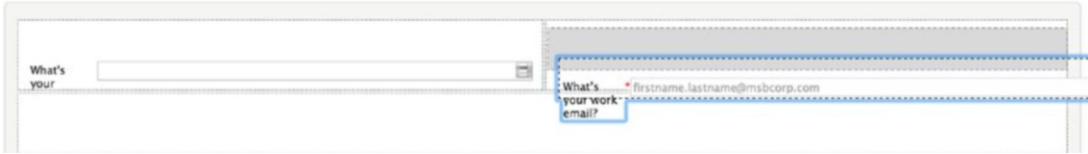
Dividing a Row

Before we continue, let us rearrange our form to maximize the space. Right now, we are putting one control in a row, which is taking up too much screen real estate and forcing us to scroll down. Remember that earlier we discussed dividing a row into columns; we will now use that knowledge to improve our form. We will split the rows into two columns and have the controls render side by side.

Click the first row in the dynaform container (click on a free space in the row and not the control in the row). With the row selected, change the col-span property from **12** to **6** as shown here. This splits the row into two columns.

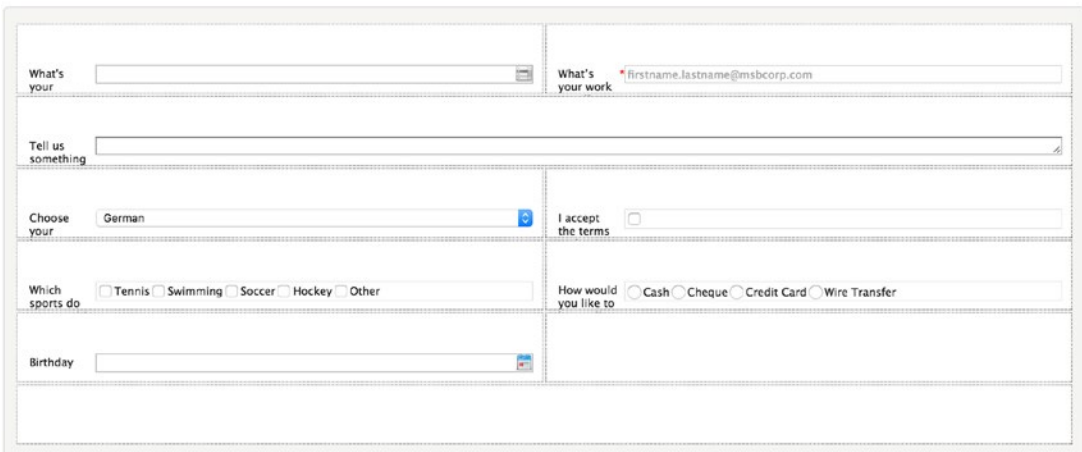
The image shows the Dynaform editor interface. On the left is the 'Web controls' panel with a 'Properties' section. The 'Properties' section has a table with two columns: 'Property' and 'Value'. The 'type' property is set to 'row'. The 'col-span' property is set to '6'. Below the 'Properties' section is the 'History of use' section. On the right is the preview of the form, showing two rows. The first row is labeled 'What's your' and contains a text input field. The second row is labeled 'What's your work' and contains a text input field with the value 'firstname.lastname@msbcorp.com'.

Next, select the textbox control in the second row and drag it into the right column of the first row. When dragging the control, try selecting it using the label part of the control.



The next control is a textarea and we will probably want it to span the width of the form, so we will leave it as it is. You can drag it into the empty row above it (the row we moved the textbox from). Divide the third row (where we just moved the textarea control from) into two columns by selecting it and changing its col-span property to **6 6**. With the row divided, drag the dropdown control to the left column and the checkbox control to the right column.

Repeat the same process to divide the fourth row into two columns and drag the checkgroup control to the left columns and the radio control to the right column. Divide the fifth columns also into two and drag the datetime control to the right column. The dynaform container should now look like the following image, taking up less screen space.



Preview the form. It should look like the next image.

The image shows a preview of a web form with the following elements:

- Top row: "What's your occupation?" (text input), "What's your work email? *" (text input with value "no-email@msbcorp.com").
- Second row: "Tell us something" (large text area).
- Third row: "Choose your language" (dropdown menu with "Select language..." selected), "I accept the terms and conditions" (checkbox).
- Fourth row: "Which sports do you watch?" (checkboxes for Tennis, Swimming, Soccer, Hockey, Other), "How would you like to be paid?" (radio buttons for Cash, Cheque, Credit Card, Wire Transfer).
- Fifth row: "Birthday" (calendar input with value "August 15th, 2016 00 00 AM").

Close the preview and let us move on to the next control in the web controls panel.

Hidden



The hidden control, as you might have guessed, is hidden; that is, it is not visible on the form. It is useful for storing information we do not have to display to the user but require for evaluating logical conditions or performing calculations. For example, we can use the hidden control to store the value of a variable indicating whether the form was approved or rejected by the previous user, and then hide or show certain fields based on the value.

Drag and drop a hidden control to the last row and in the Create/Select Variable modal dialog, change the variable name to **my_hidden**. Expand the settings to see the available settings. You will see that the supported data types for the variable include string, integer, float, Boolean, datetime. Leave the type as string and save the variable. Select the control in the dynaform container to display its properties. All the properties are familiar from other controls we have seen. Preview the form. You will notice that the control is not displayed on the form. We will see how we can use values stored in a hidden field later in the book. Close the preview.

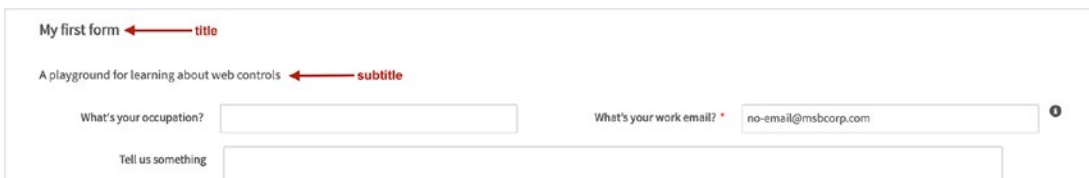
Title and Subtitle



The next two controls are used to provide headings that can be used to section our form. The title control adds a Heading 1 title (`<h1>Title text</h1>`) to the form, while the subtitle tag adds a Heading 2 title (`<h2>Subtitle text</h2>`). Let us see how these controls look on the form. Ideally, the title will be at the top of the form, so let us drag the empty row from the bottom to the top of the dynaform container as we learnt in the lesson on repositioning a row. With the row positioned at the top, drag a title control from the web controls panel to the row. Click the title to select it and display its properties.

The controls has just three properties, of which only two are editable. We are however only interested in the label property, which is the text displayed as the title. The id property can be left as the default ID generated by ProcessMaker. Change the label property to **My first form**. Go ahead and preview the form. The title should be displayed at the top of the form.

The subtitle control is practically the same as the title control with the only difference being that it renders its label in a H2 tag while the title control uses a H1 tag. Drag the empty row from the bottom of the container and place it below the first row. Drag and drop a subtitle control into the second row and select it to display its properties. Change the label to **A playground for learning about web controls**. Preview the form.



Our newly added title and subtitle might not look fancy at the moment, and it might be hard to tell them apart from the labels on other controls on the field. You will learn how to style them to make them visually distinct when we enhance the look and feel of our forms later in the book.

Label



This control is used for adding text to our forms. For example, we can use it to add the terms and conditions to a form. Let us rearrange our form. Drag the checkbox control (the control with the “I accept the terms and conditions” label) to the empty right column beside the datetime control (the control with the label “Birthday”). Then drag the datetime control to the column (beside the “Choose your language” dropdown control) we moved the checkbox from.

Now add a label to the form by dragging a label control from the web controls panel to the empty column beside the checkbox control. This should be the left column of the seventh row on the form. Click the label control to select it and display its properties. Change the text property to **Your use of this form is subject to acceptance of the terms and conditions of MSB Corp.** Preview the form to see how the label is displayed.

Your use of this form is subject to acceptance of the terms and conditions of MSB Corp.

I accept the terms and conditions

Link



The link control is used to add a hyperlink to the form. Continuing with the terms and conditions example, we can use the link control to add a hyperlink that directs user of the form to the terms and conditions page on a website or intranet. Let us add one to the form. Divide the last row in the dynaform container into three columns by selecting it and setting its col-span property to **4 4 4**. Drag and drop a link control from the web controls panel to the first column. Click the control to select it and display its properties.

The label property is the label that will be shown beside the link. Set it to **T&C link**. The display text property is the text that will be displayed and the users can click. Set it to **Click to read our terms and conditions**. The href property is the URL that the link

will redirect to when it is clicked. Set it to <http://www.processmaker.com/terms-of-service>. The hint property is used to provide more details on the link.

Preview the form. Click the link and it should open a new tab or window in your browser to display the ProcessMaker Terms of Service page. Close the preview to return to the Dynaform Designer.

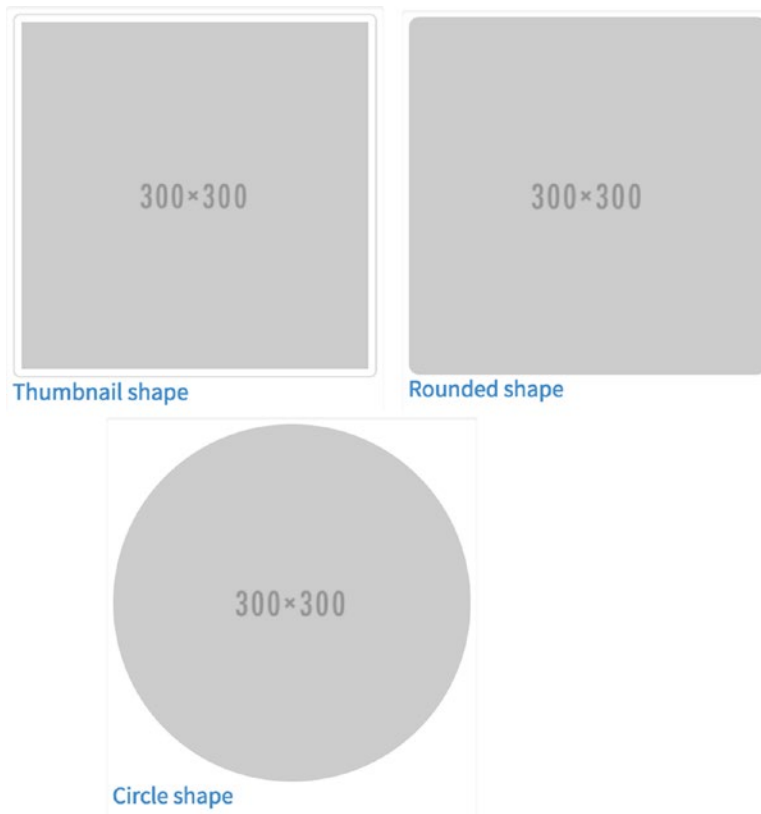
Image



The image control is used to display an image on the form. The image can be loaded from the web or any external source by providing a URL to the image in the src property of the image control. To see it in action, drag an image control to the middle column of the eighth row, beside the link control. Click it to select it and display its properties.

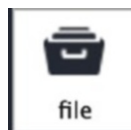
The label property can be used to provide information about the image. Set it to **Placeholder**. The hint property, as you've seen, is used to provide a hint on the control. The src field is used to set the URL of the image we want to display. Set it to <https://placeholder.it/300x300>. This will generate a placeholder image for us from Placeholder.it, a free placeholder image generation site.

The shape property is used to specify how the image should be displayed. There are three options: thumbnail (this is default) , rounded (adds rounded corners to the image) and circle. You can try changing it to the different options and previewing the form to see how it renders (see the following images).



The alternate text property is used to define the text that will be displayed if the image cannot be loaded. Set it to **placeholder image**. The comment property is used to add text that can be displayed at the bottom of the image (for example, the text “Thumbnail shape” in the first image here). The title (mouseover) property is used to specify text that will be displayed when the user’s mouse is hovered over the image. Set it to **Demonstrating image control**. Go ahead and preview the form.

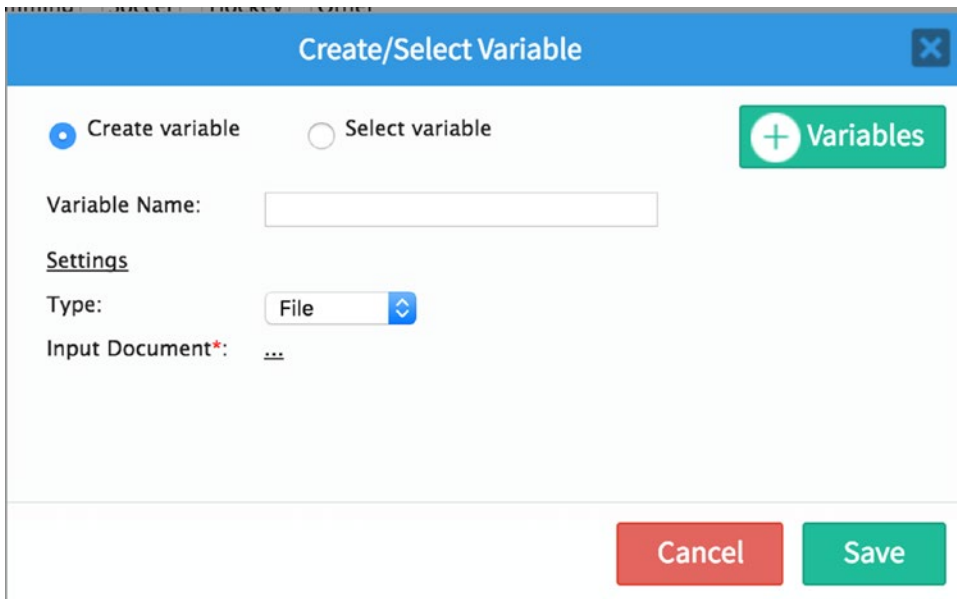
File



Next in the panel is the file control, which as you might have guessed is used for uploading a file into our form. This is useful for attachments. Documents can also be

added to a case (remember that a case is an instance of a process) by adding an input documents step to the task in the process. We will explore this alternative approach later on. For now, let us add a file control to the form to see how it works.

Drag and drop the file control in the last column on the eighth row. Click it to select it and display its properties. In the Create/Select Variable modal that appears, we will not be creating or selecting a variable this time around. The file control can only be associated with a variable with a file data type, and this requires us to select an input document which we do not yet have. We will learn about input documents later on.

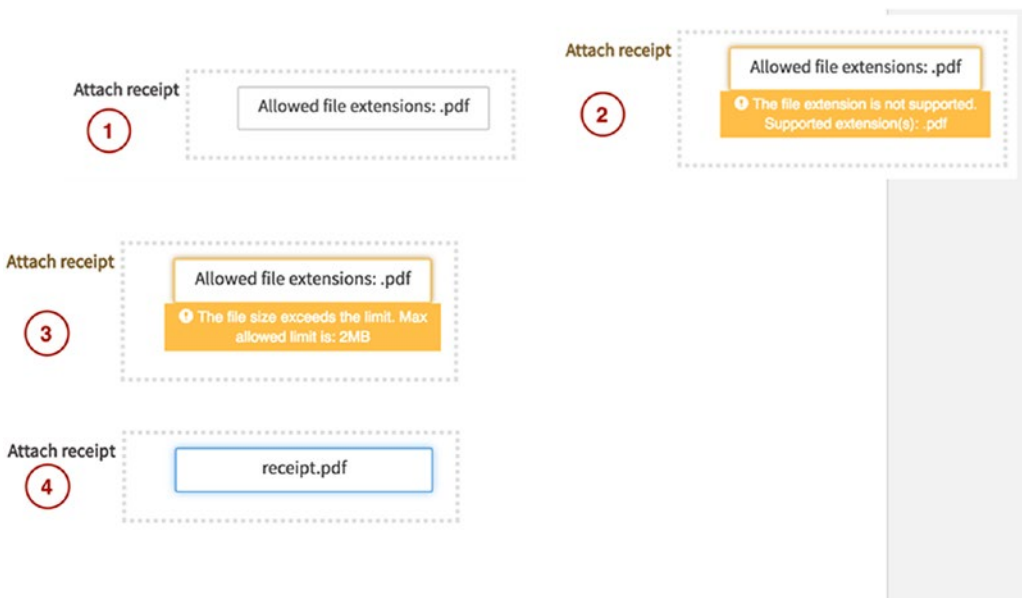


Expand the settings to show the available settings for the variable as shown above. You will see that the input document setting is required (marked with a red asterisk). Click the ellipsis in front of it to display a modal from which you can select an input document. We don't have any at the moment, so close the modal by clicking the Close (x) button in the top right corner. If you click the Save button, you will get an error message: "The input document is required, please select the value." as expected since we have not provided one. Click the Cancel button. This will still add the control to our form, but the control will not be associated with a variable. That means we will not be able to store the file provided by the user. This is fine for now, as we are just playing around and exploring the control. Click on the control to select it and display its properties.

The label property adds a label to the field. Set it to **Attach receipt**. The file extensions property is used to indicate which type of files the user can upload. This is set to * by default, which is a wildcard implying that the user can upload any type of file. Ideally, we would want to restrict this to the extensions of the type of files we expect. For example, if expecting images, we would set it to **.jpg, .png, .gif, .bmp** which are the common image file extensions. Note that the extensions are separated by comma. Set it to **.pdf** so that the user can only upload .pdf files.

The max file size property, as the name implies, is used to set a limit on the size of the file that can be uploaded. This is set to 1MB (1024KB) by default. The next property size unit is related to the max file size property and specifies the unit of the max file size specified earlier. The options are KB (kilobytes) and MB (megabytes). Let's change the max file size to 2MB. Set the max file size property to 2 and change the unit type property to MB.

Preview the form. The control displays as shown next. Click the control to display the file selection dialog on your system. Choose a file that is not a pdf file, and it should display an error message as shown in the second image. If you choose a pdf file larger than 2MB, it should display an error message as shown in the third image. Choosing a pdf file that is less than 2MB correctly selects the file, as shown in the fourth image.



Let's close the preview window and look at the next control which, is a new enhancement added in version 3.1.

Multiple File Uploader



The multiple file uploader (fileupload) is an enhancement to the File control we just explored. As you might have guessed, this control allows us upload multiple files to a dynaform unlike the file control which is limited to just one file.

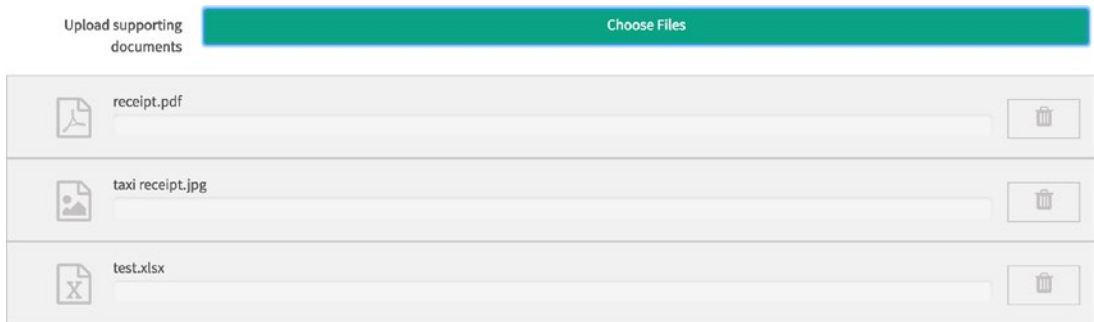
Unlike the file control, we are able to add the multiple file uploader to a dynaform without associating it to an Input Document. The association of the control with an input document can however still be done from the control's properties.

Drag and drop the fileupload control to the next row and in the Create/Select variable screen, change the variable name to **my_fileupload** and save it. Select the control in the dynaform container to display its properties. You will observe that the properties are the same with the file control with the exception of an Input document property which can be used to select the Input Document to link with the control. The common properties, such as file extensions, max file size, and so on have the same effect as in the file control.

Change the label property to **Upload supporting documents** and preview the dynaform. The control is now displayed as shown in the following image.

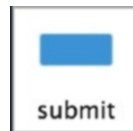


Click the green Choose Files button and upload a few documents. The uploaded documents should be displayed as shown next. When you're running a case, the progress bar below the filename will show the upload progress of the file; however, because we are in the designer preview mode, no progress is shown. The uploaded files can be deleted by clicking the Delete icon beside each file.



Close the preview mode and return to the Dynaform Designer.

Submit and Button



The submit and button controls are buttons that allow the user to trigger an action on the form. As you might have guessed, the action triggered by the Submit button is to submit the form. This action validates the form, and if the form is valid, closes the form, saves the values entered into the associated variables, and moves the case to the next step in the process. If the form is invalid, that is, some of the fields have errors, the form will not be submitted.

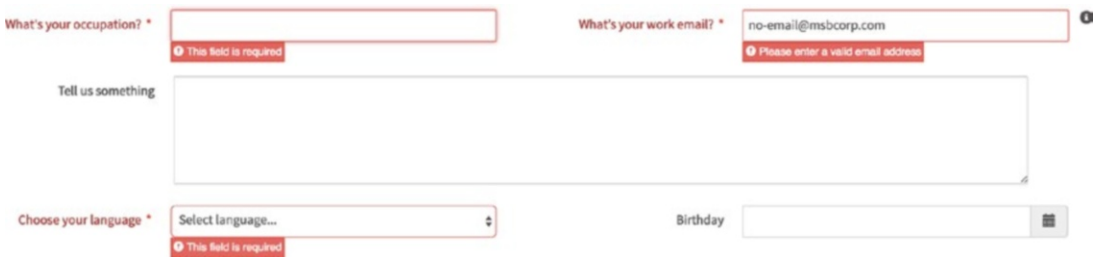
The button control provides us a way to trigger actions that do not submit the form. We can use this trigger to perform a calculation, custom validations, display a message or update a field. We do this by handling the action triggered with JavaScript code. Let us see how they both work. Divide the last row into two columns by selecting it and setting its col-span property to **6 6**.

Drag and drop a submit control to the left column. Click the submit control to select it and display its properties. Change the label to **Submit Form**. Next, drag a button control to the right column beside the submit control. Select it to display its properties and set the label to **Display Alert**. Before we preview the form, let us make some other fields required so we can see the validation at work. Select the suggest control (it should be in the third row) and make it required. Repeat the same for the dropdown control

under it (it should be in the fifth row). Once done, proceed to preview the form. The buttons should be displayed at the bottom of the form as shown here.



Click the Display Alert button. Nothing happens. This is because we have not handled the action it triggers. Click the Submit Form button. This will trigger a validation of our form and the fields we marked as required will be highlighted with validation errors as shown next.



Enter valid values for the fields highlighted and click the Submit button again. The form should now pass validation, and the fields will be displayed as shown next. The form is not closed, because we are previewing it. If we were running a case, the form would be closed on successful validation and proceed to the next step.



Now let us handle the action triggered by the button control. Close the preview and return to the Dynaform Designer. Click the button control to select it and display its properties. We will change the id and name properties to something more meaningful. Set both properties to **alert_button**. Next click the gray part of the dynaform container to select the form and display its properties. Click the Edit button beside the JavaScript

property. This displays the JavaScript editor for the form. We will add some JavaScript code to our form to handle the button click action. Copy and paste the following code into the editor and click the Save button.

```
$("#alert_button").click(function(){
    alert("The display alert button was clicked");
});
```

Do not worry if you are not familiar with JavaScript; the ProcessMaker JavaScript editor supports jQuery, a fast, small, and feature-rich JavaScript library that makes it easy to use JavaScript to make web pages interactive. We only use it to add custom functionality to our form, and you do not have to master it to be able to use it. If you are new to jQuery, Codeschool (<https://www.codeschool.com/courses/try-jquery>) and Codecademy (<https://www.codecademy.com/learn/jquery>) have free jQuery courses that can get you up to speed on the basics in a matter of hours.

Let me quickly explain what we are doing with this code.

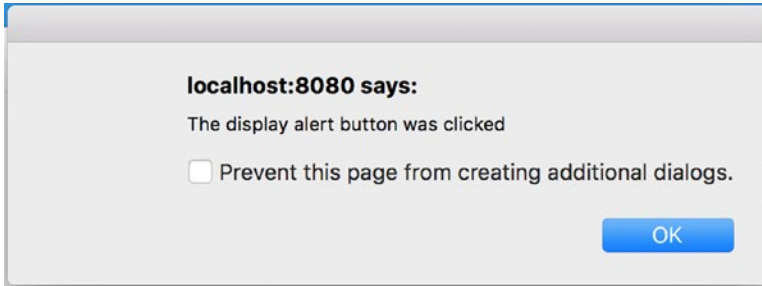
`$("#alert_button")` This is referred to as a jQuery selector and simply instructs the program to select the control on our form with the id `alert_button`.

`.click()` This is called an event handler and tells the browser to execute the function in the bracket when the control we selected is clicked.

`function(){}` This is a function, and it encapsulates a set of statements that perform a task or calculates a value.

`alert("The display alert button was clicked");` This is the only statement in our function and it calls a built-in JavaScript function called `alert`, which is used to display an alert message in the browser. The function expects the message we want to alert, which we provide in quotes, and the semi-colon signifies the end of the statement.

To see the effect of the JavaScript code on the form, preview the form and click the Display Alert button. The browser should display an alert with the message as shown next. Click the OK button to dismiss the alert. Close the preview and return to the Dynaform Designer.



Grid



The next control we will explore is the grid, which allows us to add a tabular list of controls to our form. This is useful for capturing a list of items in the form. For example, if we wanted the users of our form to provide an inventory of items in a store, we would use a grid. Let us add one to the form to see how it works.

To make our form consistent, move the last empty row above the row containing the buttons. Next, drag and drop a grid control from the web controls panel to the row you just placed above the buttons. In the Create/Select Variable modal that appears, change the variable name to **my_grid**. If you expand the settings, you will observe that the Type is grid, which is the only type of variable we can associate with a grid control. Click the Save button to save the variable. Select the grid control from the dynaform container to display its properties.



Adding Controls to the Grid

Before we explore the properties, looking at the grid on the form shows the title of the grid and the different type of controls we can place within our grid. By now, we are already familiar with all the supported controls, so we will add one of each to the grid to see how they are rendered in the grid. If you try adding any of the unsupported controls to the grid, you will be shown an error message. Switch to the web controls panel and drag a textbox on top of the grid control. If you click the textbox control you just added to the grid, you will see its properties displayed in the properties panel on the left.



However, you will notice that there is no variable or variable data type property. This is because the control belongs to the grid, and its value will be stored in the variable associated with the grid. Also, at the bottom of the properties panel, there is a new column width property. This is used to define how wide the control will be rendered in the grid. This property is influenced by the layout property of the containing grid. We will learn more about it shortly. For now, set the id property of the textbox to **my_grid_item** and the label property to **Item**.

Proceed to add the other controls to the grid. Drag a textarea control and place it beside the textbox in the grid. Select it to display its properties and set its id property to **my_grid_description** and its label property to **Description**.

Next, drag a dropdown control and place beside the textarea you just added. Select it to display its properties. Set the id property to **my_grid_category** and the label property to **Category**. We will also need to add the options we want users to be able to choose from for our dropdown. Click the square brackets in front of the options property to define the following options and click the Apply button. To prevent an option from being selected by default we set the placeholder property to **Select...**

Key	Label
Accessories	Accessories
Consumables	Consumables
Electronics	Electronics
Stationeries	Stationeries

The next control we add is the checkbox. Drag and drop it beside the dropdown control. Select it to display its properties. Set the id to **my_grid_invoice_attached** and the label to **Invoice Attached**. Next, drag and drop a datetime control beside the checkbox. Select it to display its properties and set the id to **my_grid_purchased** and the label to **“Date Purchased**.

Next, we add a suggest control to the grid, placing it beside the datetime control. Select it to display its properties. Set the id to **my_grid_supplier** and the label to **Supplier**. Just like we did with the dropdown control, we need to define options for the suggest control. Click the square brackets for the options and define the options shown in the following table. Click the Apply button when done.

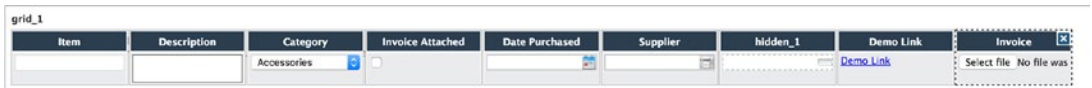
Key	Label
ACME Corp	ACME Corp
Office Supply Inc	Office Supply Inc
The Electronics Place	The Electronics Place
XYZ International	XYZ International

The next supported control is hidden. Drag and drop the hidden control beside the suggest control and select it to display its properties. Set the id property to **my_grid_hidden**. You will observe that the hidden control does not have a column-width property like the others. This makes sense, as it will not be displayed on the form. Next, drag a link control and place it beside the hidden control.

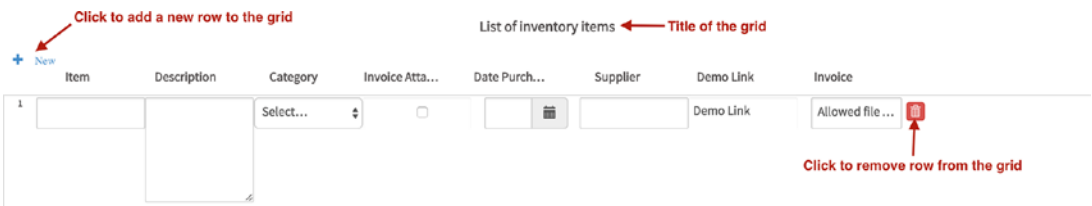


You might have to scroll to the right depending on your screen size. Select the link control and set the id and name properties to **my_grid_link** and the label and display text properties to “**Demo Link**”.

Finally, drag and drop a file control after the link control. Select it to display its properties. Set the id and name properties to **my_grid_invoice** and the label to **Invoice**. Our grid should now look like the following image. To see the grid at once, you might have to collapse the left panel using the close bar between the dynaform container and the properties panel (see image on the right). You can click it again to expand the panel.



Before we preview the grid, select it and change its title property to **List of inventory items**. Click the Preview button from the dynaform menu and scroll to the end of the form. You should see the grid displayed, similar to the image shown here.



Adding and Deleting Rows

In the grid displayed, you will observe that there is a New link at the top of the grid and a delete icon added to the end of the row of controls. The New icon is used to add a new row to the grid. Click it and you should see a new row added to the grid. As you might have guessed, the Delete icon beside each row removes that row from the grid. click the Delete icon beside the two rows displayed on the form. The grid should display as shown here, with no records.

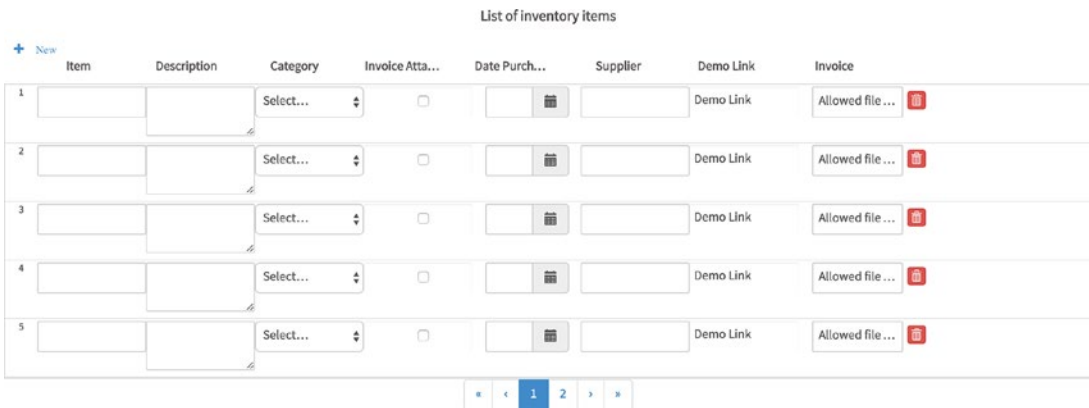


We can modify the properties of our grid to disable the New and Delete options on the grid. This can be useful when we are displaying grids in our forms that have been prepopulated from a database and do not want the users to add new records or delete records from the grid. To see this at work, close the preview to return to the Dynaform Designer. Select the grid and in its properties, uncheck the add row and delete row properties. Go ahead and preview the form again. You will see that the New link and Delete icon are no longer available.

Paging Records in the Grid

The next property on the grid that we will look at is the page size property. It is used to define if we want the records in the grid to be paged. This can be useful when expecting very long grids. The property is set to none by default which means no paging and all rows added to the grid will be displayed at once. Let us change it to 5. Then enable the add row and delete row properties by checking them. Also select the textarea control in the grid and change its rows property to 2 so that it takes up less space. With that done, go ahead and preview the form.

The grid is now displayed with a pager at the bottom. Click the New link five or more times to add enough records that exceed the page limit of 5 we defined in the page size property. The pager at the bottom should now have a second page. Click on page 2 to display the additional rows in the grid.



Modifying the Grid Layout

Looking at our grid, we can see some space on the right side of each row, while our Date Purchased column appears squeezed. This is because we are using the responsive layout and set each control to take 10% of the size of the grid. The layout property of the grid has two options, **responsive** and **static**. The responsive layout is the default and sets the grid size to 100% of the form width; the controls inside it are sized based on the percentage assigned to their column width property.

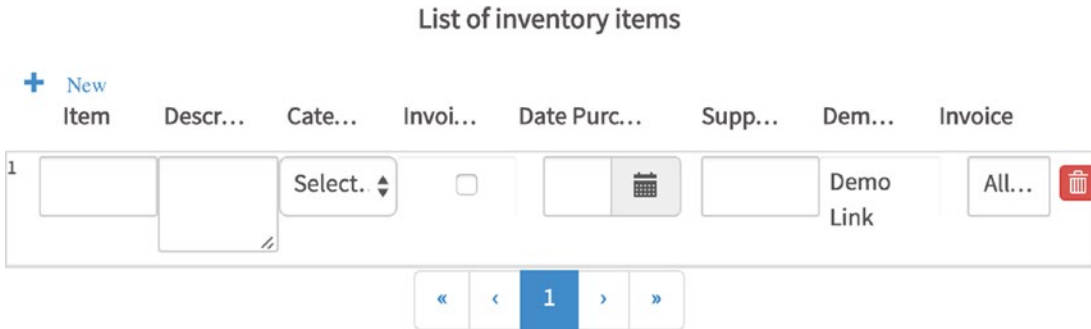
For example, if we wanted all controls in the grid to have the same width, we would divide the width of the grid (94%, because 3% is reserved for the serial number of the rows and another 3% for the Delete icon) by the number of controls (8; because we do not count the hidden control) to determine the value to set for each controls column width. Dividing 94% by 8 gives us 11.75%. We will, however, have to round this down to a whole number (rounding it up will cause us to exceed 94%) to give us 11% per column. We use whole numbers because ProcessMaker disregards the decimal and rounds the values to the nearest lower whole number.

If we set each control's column width property to 11% ,we will be left with an extra 6%, which we can then add to the date purchased control to make it wider. Change the column width property of the controls in the grid to 11 with the exception of the date purchased datetime control. Set that to 17 and preview the form. The rows of the grid should now fill out the grid as shown in the following image.

The screenshot shows a web form titled "List of Inventory Items". At the top left, there is a "+ New" button. Below it is a table with 8 columns: "Item", "Description", "Category", "Invoice Attac...", "Date Purchased", "Supplier", "Demo Link", and "Invoice". The first row contains the following data: "1", an empty text box, "Select..." (with a dropdown arrow), an empty checkbox, an empty text box with a calendar icon, an empty text box, "Demo Link", and "Allowed file ex..." (with a red icon). Below the table is a pagination bar with a blue button labeled "1" and navigation arrows.


When you have a sizable number of columns in your grid, the responsive layout might not be the best option. Imagine we had a grid with 15 columns and we used the responsive layout. If we made the columns equal in size, we would have to set each

column to 6% with a spare 4% to add to other columns. Such a form would appear squashed. To illustrate this, preview the form and resize your browser window to half the size. The grid will appear as displayed next.



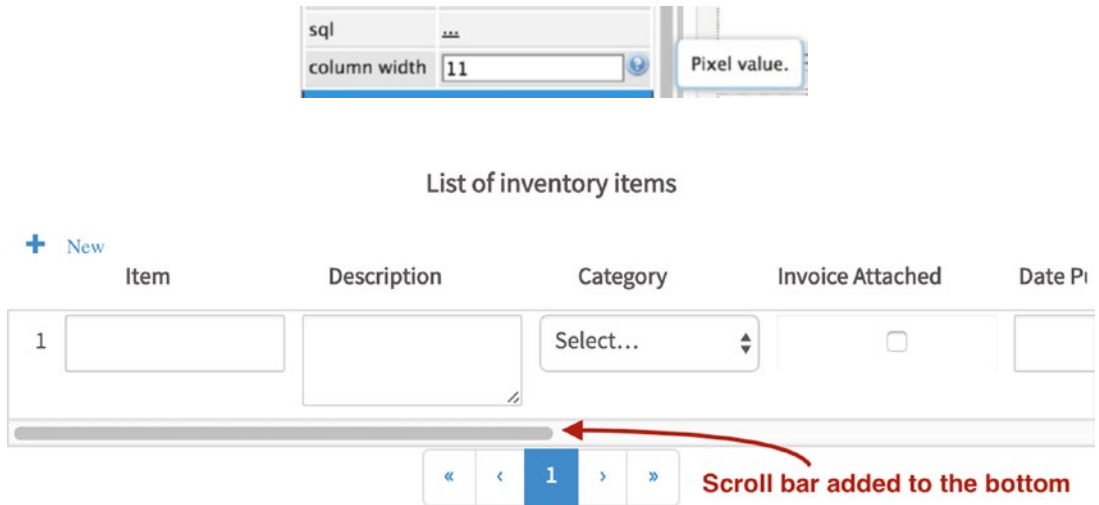
To work around this, we can change the layout property of the grid to Static. The static layout allows us to set the pixel size for each column on the grid and if the size exceeds the browser screen, a scrollbar is added to the grid allowing users to scroll to view the remaining columns in the grid. Let us change our grid’s layout property to Static to see how it works.

Close the preview, select the grid, and change its layout property to Static. Now select the controls in our grid and change their column width property to 140. Since we are now using the static layout, the column width is now measured in pixels and not percentages as with the responsive layout. If you hover over the hint icon beside the property, it will display “Pixel value,” as shown in the image following (if we were using the responsive layout, it would show “Percentage value”). This can be useful for verifying what unit of measurement is being used.

 According to WhatIs.com (<http://whatIs.techtarget.com/definition/pixel>), “The pixel (a word invented from “picture element”) is the basic unit of programmable color on a computer display or in a computer image. Think of it as a logical—rather than a physical—unit. The physical size of a pixel depends on how you’ve set the resolution for the display screen.”

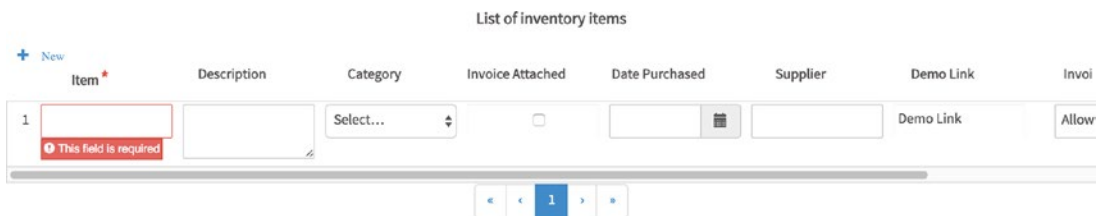
With the column width property all set, proceed to preview the form. Resize your browser to half the width and you will observe that instead of squeezing the columns

together, a scrollbar is added to the bottom (as shown in the image), allowing us to scroll and display the controls in the grid.



Validating Required Fields

To end our discussion of the grid control, I would like to point out that you can mark the controls in a grid as required to make sure that the user fills them before submitting the form. To try it out, select the textbox (item) and check its required property. Preview the form and click the Submit button. A validation error is displayed on the field in the grid as shown here.

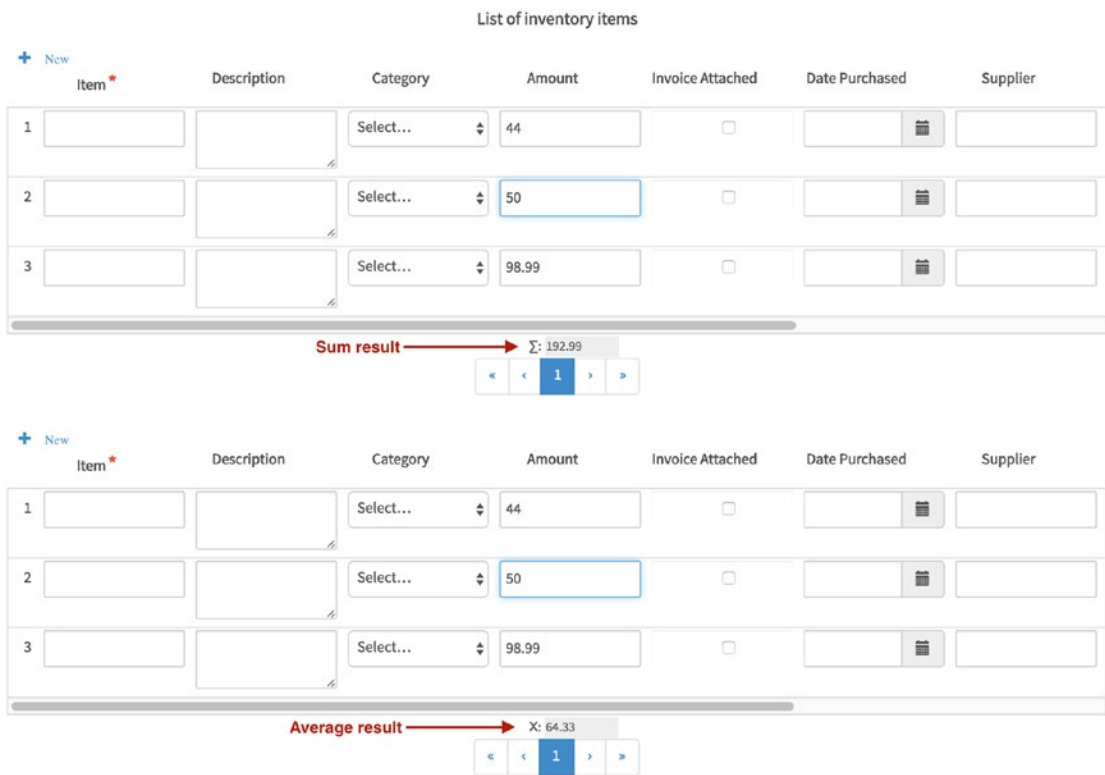


Mathematical Functions in Grids

Another cool feature of the grid control is the function property it adds to textboxes in the grid. Let us add another textbox control to our grid. You can place it after the dropdown control. You might have to click and drag it to the position after adding it to the grid. Set its id property to **my_grid_amount** and the label to **Amount**. Scroll down

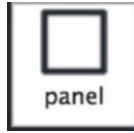
the properties panel and you will see a function property. The default value is None. Change it to Sum and change the column width property to 140. Proceed to preview the form.

You will observe a sigma sign (Σ) at the bottom of the column in the grid. Click the new link to add a couple of rows to the grid, and then enter a numeric value in the Amount field in the rows added. You will see that the total sum value is automatically displayed beside the sigma sign as shown in the following image. Close the preview, select the amount field textbox control, and change the function property to Average. Preview the form again and add a couple of rows as we did with the sum. The sigma sign is now replaced with X. Enter numeric values in the amount field, and you will notice the average is automatically displayed at the bottom as we saw with the sum control.



There is still a lot more we can do with grids, such as prepopulating them from a database or using JavaScript to perform calculations. Let us now move on to the next control in the web controls panel.

Panel



The panel control allows us to extend the functionality of ProcessMaker dynaforms by providing us with a blank canvas to add HTML or HTML5 code to the dynaform. The control is linked to the form's external libs property we saw earlier, which allows us add custom libraries or stylesheets to the form. The code in the panel is also accessible via the JavaScript editor of the form. This combination allows us to add new and powerful functionality to dynaforms beyond what is available with the web controls.

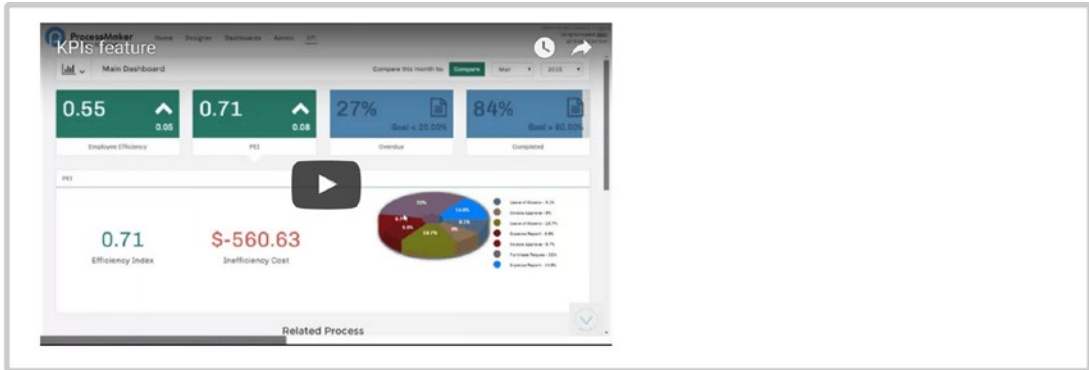
Let us add a panel to our form to see how it works and explore some of the things we can do with it. Drag the empty row at the bottom of the form above the row containing the buttons to make our form consistent. If you cannot see the handle to drag the row after selecting it, you might need to scroll to the right of your browser window. With the row in place, drag a panel control into the row. Select the panel to display its properties. The content property is where we can put our HTML code, and the border property allows us to specify the size in pixels of the border that will be drawn around the panel. Preview the form. You should see a rectangle with rounded corners just above the buttons and below the grid. That is our panel. It does not look like much at the moment, but it adds a new set of functionality to our form as we shall soon see.

First, let us see how the border property affects the panel. Close the preview and change the border property of the panel to **5px**. Preview the form again and you should see that the borders of the panel is thicker making it more pronounced on the form. Now let us explore some of the types of content we can add to our panel.

Return to the Dynaform Designer, select the panel, and click the Edit button beside the content property. This displays the content editor, where we can type in our HTML code. Let us start by embedding a video into our form. Paste the following code into the editor. It is an embed code for a video from YouTube (You can check this link to learn how to get the embed code for a YouTube video: <https://support.google.com/youtube/answer/171780?hl=en>).

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/dyT9XTPfo8s" frameborder="0"
allowfullscreen></iframe>
```

Click the Save button and preview the form. The video should display in the panel as shown here.



Submit Form

Display Alert

Let us get more fancy. The Dynaform designer in ProcessMaker 3.0 uses the Bootstrap UI framework (<http://getbootstrap.com/>), which allows us to use Bootstrap components in our form. Remember the alert we displayed with the button control earlier? Let us add a fancier version using Bootstrap’s modal component. Click the Edit button beside the content property of the panel and replace the iframe code we added earlier with the following code. Do not worry if you have no idea what this does. I just copied the boilerplate code for the modal component from <http://getbootstrap.com/javascript/#modals> and changed the text of the button and modal body.

```
<!-- Button trigger modal -->
<button type="button" class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">
```

Launch modal from panel

```
</button>
```

```
<!-- Modal -->
```

```
<div class="modal fade" id="myModal" tabindex="-1" role="dialog"
aria-labelledby="myModalLabel" aria-hidden="true">
<div class="modal-dialog" role="document">
<div class="modal-content">
<div class="modal-header">
<button type="button" class="close" data-dismiss="modal"
aria-label="Close">
```

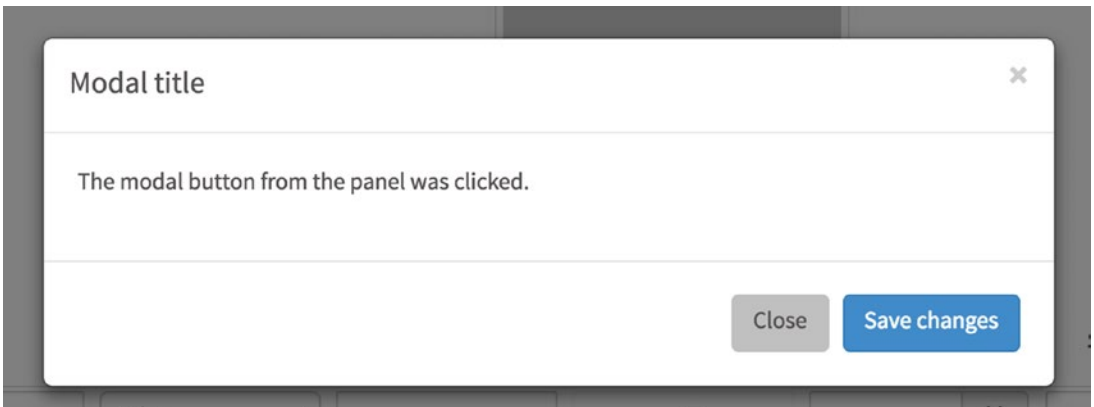
```

<span aria-hidden="true">×</span>
</button>
<h4 class="modal-title" id="myModalLabel">Modal title</h4>
</div>
<div class="modal-body">
    The modal button from the panel was clicked.
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-
    dismiss="modal"> Close
    </button>
    <button type="button" class="btn btn-primary">Save changes
    </button>
</div>
</div>
</div>
</div>

```



Click the Save button and preview the form. A button is now displayed in our panel. Click the button and a modal should be displayed on the screen as shown next.



Dismiss the modal using the Close button. Close the preview to return to the Dynaform Designer. Before we move on to the next and last control, let us see how we can connect other controls on the form to the content of the panel control. Select the form by clicking the gray part of the dynaform container to display its properties. Click the Edit button beside the JavaScript property to display the JavaScript editor. Paste the following code beneath the code already in the editor.

```
$("#my_textarea").setOnChange(function(newVal, oldVal){
    $(".modal-body").html(newVal);
});
```

This code is similar to the lines we saw earlier:

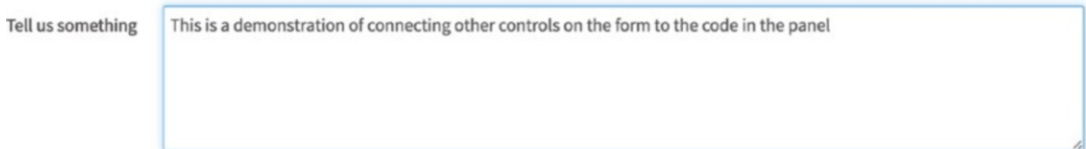
`$("#my_textarea")` This is a selector that selects the `my_textarea` control on the form.

`.setOnChange()` This adds an event listener to the control and defines the function that will handle the event. This time around we are listening for the change event; we want to know when the text in the textarea changes and once it does we will execute the function in the brackets.

`function(newVal, oldVal){}` This is the function that will be executed once the change event is triggered. You will notice that this function is different from the one we saw earlier as it has two parameters (`newVal` and `oldVal`). When the change event is triggered, it sends us the new and old values of the control, and the function we are using to handle the event can take these as parameters to be passed to the statements it will execute. The new value of the textarea is stored in the `newVal` parameter and the old value in the `oldVal` parameter.

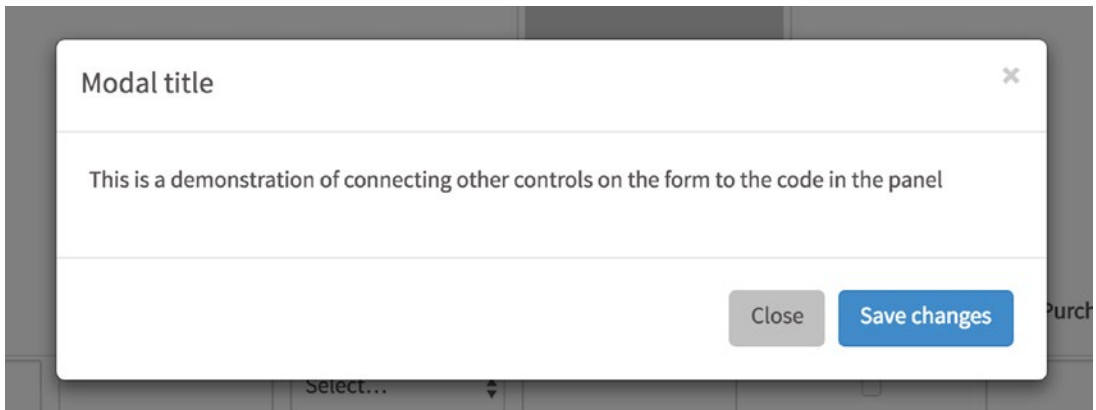
`$(".modal-body").html(newVal);` This is the statement we want to execute in our function. We use the selector again to select the div (HTML element) with the class `modal-body` from the code we placed in our panel. We then use jQuery's `html` function to add the new value of the textarea control inside the div.

Do not worry if you do not understand much of this. Going through one of the free jQuery courses suggested earlier will help clarify things. Now let us see the effect of this on our form. Click the Save button and preview the form. Click the “Launch demo modal” button in the panel control, and it should display the message we saw before. Close the modal and proceed to type some text in the textarea (the “Tell us something” field) as shown here.



Tell us something

Now click the “Launch demo modal” button in the panel control again and it should now display the message typed in the textarea as shown next.



Modal title ×

This is a demonstration of connecting other controls on the form to the code in the panel

Close Save changes

Close the modal and the preview window. That concludes our discussion of the panel control, but there is much more we can do with it than shown. Feel free to try different ways you can use the panel control to enhance your dynaforms.

Subform

The last control in the web controls panel is the subform, which allows us to embed a dynaform in another dynaform (the master dynaform). Before we explore the control, there are a few things to note. A form can only be added once to a dynaform, the subform

cannot be edited within the master form, and a subform cannot contain another subform. You can, however, have more than one subform in a master form.

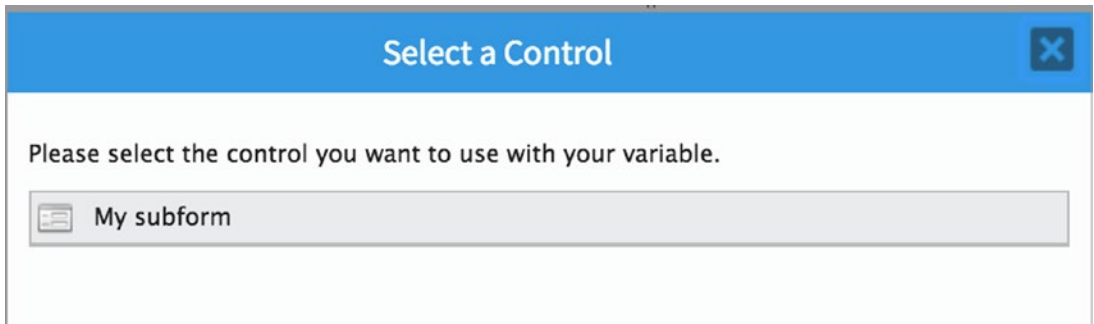
To use the subform control, we will need an additional form, so let us create one. Close the Dynaform Designer to return to the process designer. In the main toolbox on the right, click the Create icon (+) beside the Dynaforms option. In the Create Blank Dynaform modal that appears, set the title to **My subform** and the description to **A form to demonstrate subforms in ProcessMaker**. Click the Save & Open button to open the blank form in the Dynaform Designer.

We will keep this form simple. Add a subtitle control to the first row, select it, and set its id to **subform_subtitle** and its label to **“This is my subform**. We change the id of the subtitle control because we already have a subtitle with the same id in the first form we want to embed this form in. If there are controls with duplicate ids, we will not be able to save the form or preview it after adding the subform.

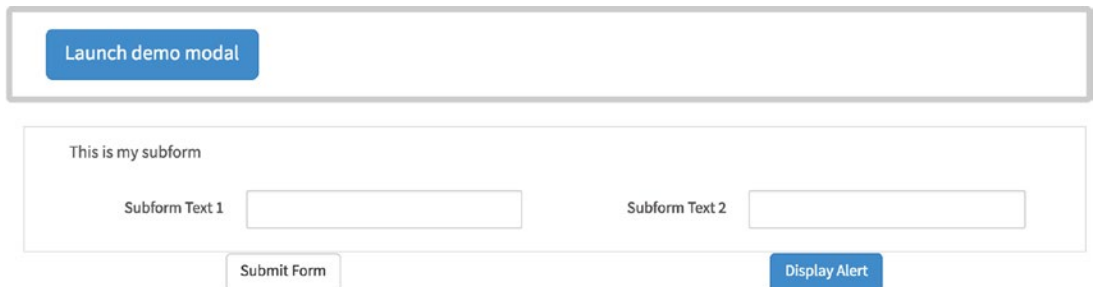
Select the second row and set its col-span property to **6 6** to divide it into two columns. Drag a textbox control to the first column of the second row and in the Create/Select Variable modal, change the variable name to **my_subform_text1** and save it. Select the textbox and change its label property to **Subform Text 1**. Repeat the process to add another textbox to the second column of the second row with variable name set to **my_subform_text2** and the label to **Subform Text 2**. A preview of the form should look like the following image. Save the dynaform and close the Dynaform Designer.



Now, let us add this new form to our first form. Click Dynaforms from the Main Toolbox on the right, which displays the list of dynaforms. Click the Edit button beside the “My first form” dynaform to display it in the Dynaform Designer. Drag the last row above the row with the buttons so that our form is consistent. Next, drag and drop a subform control to the newly placed row. A Select a Control modal is displayed as shown next, with our recently created form available for selection.



Click My Subform to select it and add it to the master form, “My first form.” The subform is now embedded in the master form. Preview the form and the subform should now display in the master form as shown here.



This brings us to the end of our exploration of the controls available to us when designing forms in ProcessMaker Open Source edition. The Enterprise edition includes a set of mobile controls as mentioned earlier, but that is outside the scope of this book. With all the knowledge we have acquired on dynaforms, we are now more than ready to continue building our Cash Advance and Expense Retirement process.

Go ahead and close the preview, the Dynaform Designer, and the Process Designer to return to the list of processes.

In the next chapter, we will build the initial forms to be used in the Cash Advance and Expense Retirement process, leveraging the knowledge we have acquired in this chapter.

CHAPTER 9

Adding Forms to the Process

Having learned a lot about dynaforms and the controls, let us now build the dynaforms we will need for the Cash Advance leg of the process we modeled earlier. We will be building three forms—one form each for the requestor, approver, and Finance officer—which are basically variants of the same form. The forms we will build will look like this.

The screenshot shows a web form titled "Cash Advance Requisition". It is divided into several sections:

- Request Details:** Contains fields for "Request Date" (with a calendar icon), "Employee Name", "Department" (a dropdown menu with "Select your department" selected), and "Requested Amount".
- Reason for Expenses:** A large text area for providing details.
- Disbursement Details (Finance):** Contains fields for "Amount Advanced" and "Date Advanced" (with a calendar icon).
- Signoff/Approval:** Contains three rows of fields for "Requested By", "Approved By", and "Disbursed By", each with a corresponding "Date and Time" field.
- Disclaimer:** A paragraph of text at the bottom stating: "I agree to account for this advance within ten working days of the date advanced as defined in the preceding section, either with adequate receipts, cash or a check for the balance made payable to MSB Corporation. I understand that my failure to account for advanced funds in full within sixty days will result in a Payroll deduction for the balance due. By clicking the submit button below, I agree to allow MSB Corporation to make any such deductions from my pay."
- Submit Request:** A button at the bottom center.

Cash Advance Requisition form

Building the Form

Open the Cash Advance and Expense Retirement process from the process list. Click on Dynaforms in the Main Toolbox (process objects) to display the list of dynaforms. You should have the blank dynaform we created in Chapter 7 before we took a detour to learn about dynaform controls, Cash Advance Request Form. Click the Edit button beside it to open the form in the Dynaform Designer.

Add a title control to the first row and change the label to **Cash Advance**

Requisition. Next add a subtitle control to the second row and change its label to **Request Details**. Select the third row and split it into two by setting the col-span property to **6 6**. Now place a datetime control in the left column of the third row. In the Create/Select Variable modal that appears, choose the Select Variable option, and a list of the datetime variables we created earlier will be displayed. Select the **request_date** variable from the list to associate it with the datetime control. Select the control on the form to display its properties. Change the label to **Request Date** and make it required.

Place a textbox control in the right column and in the Create/Select Variable modal that appears, switch to Select Variable and, from the list of variables shown, select **employee_name** to associate it with the textbox. Select the textbox control in the form and change its label to **Employee Name** and make it required.

Divide the next row into two by selecting it and changing the col-span property to **6 6**. Add a dropdown control to the left column and in the Create/Select Variable modal, switch to Select Variable and select **department** from the list of variables. We are using a dropdown control because departments are usually a finite set in any organization and we can improve the user experience of our form by allowing the user to choose from a list instead of having to type it out. Select the dropdown control from the list and change the following properties.

Set the label to **Department**, set the placeholder to **Select your department** and make it required. We will also need to define the options that can be selected in the dropdown control. Ideally, we would store the list of departments in a PM table in the database so that it can be available to all our processes, and then use an SQL query to add it to our dropdown control. We have yet to learn about PM tables, so we will just manually define the options as we did when learning about the dropdown control. Alternatively, we can also query the list of departments configured in the system. We will refactor the form later to display the options from the departments we will create in ProcessMaker.

Rather than define the list of departments on the Options property of the dropdown control, we will define them on the variable. This way, the list of departments will be readily available everywhere we use the variable, and when we want to refactor the source of the list of departments to a PM table, we only have to make the change on the variable and not have to edit every form where the variable is used.

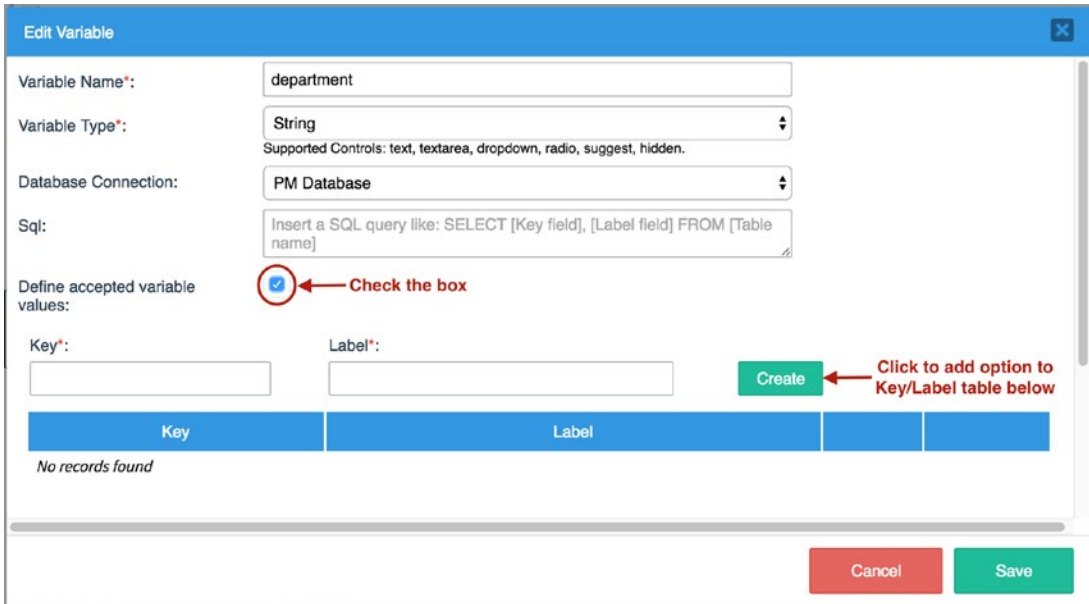
Click on *department* beside the Variable property of the dropdown control. This shows the Create/Select Variable modal. Click the Variables button (see the next image) to display the list of variables for editing.

Click to display list of variables for editing

Next, click the Edit button beside the department variable in the list to edit it.

Name	Type	Edit	Delete
amount_advanced	float	Edit	Delete
amount_requested	float	Edit	Delete
approver_datetime	datetime	Edit	Delete
approver_name	string	Edit	Delete
date_advanced	datetime	Edit	Delete
department	string	Edit	Delete
disbursed_by	string	Edit	Delete
disbursed_datetime	datetime	Edit	Delete
employee_name	string	Edit	Delete
expense_reason	string	Edit	Delete

In the Edit Variable modal, check the “Define accepted variable values” option to display the Key/Label table as shown next. Add the list of departments in the table by entering the key and label values in the textbox and clicking the Create button. Repeat until all departments have been added. Click the Save button to close the modal. Also close the list of variables.



Key	Label
Administration	Administration
Finance	Finance
Human Resources	Human Resources
Information Technology	Information Technology
Sales	Sales

You should now see the departments you just defined in the Options property of the dropdown control. We could also have done this by closing the Dynaform Designer and going to Select Variables from the Main Toolbox, but this saves us time because we don't leave the Dynaform Designer.

The next field we will add to the form is for the requested amount. Drag a textbox control to the right column of the fourth row and in the Create/Select Variable modal,

switch to Select Variable and choose the **amount_requested** variable. Select the control in the form, change the label to **Requested Amount**, and make it required. You may have noticed that we have altered the layout of the form a bit from the printed form layout we saw earlier. What we have done is group the information of the person making the request and the expense details and subtitled everything as request details.

Next add a textarea to the fifth row and in the Create/Select Variable modal, switch to the Select Variable option and choose **expense_reason**. Select the control from the form and change its label to **Reason for Expenses** and also make it required. We are now done with the first section of the form. Let's preview it to see how it looks. Your form should look like the following.

Desktop preview of the form

When we explored the dynaform menu earlier, we learned that the preview feature has three modes, Desktop, Tablet, and Mobile, that give us an idea of how the form will be displayed on the different screen sizes.



The default preview mode is desktop, so the image earlier gives us an idea of how the form would look on a desktop. In the top-right corner of the preview window, click on the icon for the other display modes to see how the form would render on a tablet or mobile (see the images for tablet and mobile preview next).

Cash Advance Requisition

Request Details

Request Date *	Employee Name *
<input type="text"/>	<input type="text"/>
Department *	Requested Amount *
<input type="text" value="Select your department"/>	<input type="text"/>
Reason for Expenses *	
<input type="text"/>	

Tablet preview of the form

Close the preview and let us add the remaining sections of the form. The next section we will add is the Disbursement Details. This section of the form is going to be filled in by the Finance department, and we could choose to leave it out of this form, which will be filled by the person requesting the advance. Alternatively, we can add it to the form and disable the fields so that the person filling the form cannot edit them. We will choose the latter option, as it allows us to use a copy of the form when creating the form for the Finance task.

Cash Advance Requisition

Request Details

Request Date *

Employee Name *

Department *

Select your department

Requested Amount *

Reason for Expenses *

Mobile preview of the form

Drag a subtitle control to the next row and set its label to **Disbursement Details (Finance)**. Divide the next row into two columns by selecting it and setting the col-span property to **6 6**.

Next, drag a textbox to the left column, and in the Create/Select Variable modal, switch to Select Variable and choose **amount_advanced**. Select the control in the form and set its label to **Amount Advanced** and the display mode to Disabled.

Also add a datetime control to the right column, and in the Create/Select Variable modal, switch to Select Variable and choose **date_advanced**. Select the control in the form, set its label to **Date Advanced** and change the display mode to disabled.

The next section of the form is Signoff/Approval, where we keep track of everyone who has signed off on the form. The fields in this section will be disabled, since we will be updating them using triggers when we discuss that tool in Chapter 12.

Define the section by adding a subtitle control to the next row and setting its label to **Signoff/Approval**.

Divide the next row into two columns and place a textbox in the left column. Switch to select variable option in the Create/Select Variable modal that appears and choose **requestor_name** from the list of variables. Select the control in the form, set its label to **Requested By**, text transform to property to UPPERCASE and display mode to Disabled.

Repeat the same process to add a textbox to the right column, and choose **requestor_datetime** as the variable. Select the textbox and change the label to **Date and Time**, and its display mode to disabled.

Let's proceed to add the signoff fields for the approver and Finance officer (disbursed by). Divide the next row and place a textbox control on the left associating it with the **approver_name** variable. Set its label to **Approved By**, its text transform to property to UPPERCASE, and its display mode to Disabled.

In the right column, place another textbox associating it with the **approver_datetime** variable. Set its label to **Date and Time** and its display mode to Disabled.

Finally, we add signoff fields for the Finance officer. Divide the next row into two as before, and place a textbox in the left column associated with the **disbursed_by** variable. Set its label to **Disbursed By**, text transform to property to UPPERCASE, and display mode to Disabled.

In the right column, place another textbox associating it with the **disbursed_datetime** variable. Set its label to **Date and Time** and its display mode to Disabled.

We are now almost done with our form. All that is left is a Submit button that allows the user to submit the form and send it to the next step. We could also add a variation of the acknowledgement part of the printed form to our form. Place a label control in the next row, select it in the form, and set its text property to the following:

I agree to account for this advance within ten working days of the date advanced as defined in the preceding section, either with adequate receipts, cash, or a check for the balance made payable to MSB Corporation. I understand that my failure to account for advanced funds in full within sixty days will result in a Payroll deduction for the balance due. By clicking the Submit button below, I agree to allow MSB Corporation to make any such deductions from my pay.

Finally, to complete the form, add a submit control to the last row and set its label to **Submit Request** and its id and name to **submit_button**. Our form is done, for now at least. Go ahead and preview your work. The form should look like the following.

Cash Advance Requisition

Request Details

Request Date * <input type="text" value=""/>	Employee Name * <input type="text" value=""/>
Department * <input type="text" value="Select your department"/>	Requested Amount * <input type="text" value=""/>
Reason for Expenses * <div style="border: 1px solid #ccc; height: 50px; width: 100%;"></div>	

Disbursement Details (Finance)

Amount Advanced <input type="text" value=""/>	Date Advanced <input type="text" value=""/>
---	---

Signoff/Approval

Requested By <input type="text" value=""/>	Date and Time <input type="text" value=""/>
Approved By <input type="text" value=""/>	Date and Time <input type="text" value=""/>
Disbursed By <input type="text" value=""/>	Date and Time <input type="text" value=""/>

I agree to account for this advance within ten working days of the date advanced as defined in the preceding section, either with adequate receipts, cash or a check for the balance made payable to MSB Corporation. I understand that my failure to account for advanced funds in full within sixty days will result in a Payroll deduction for the balance due. By clicking the submit button below, I agree to allow MSB Corporation to make any such deductions from my pay.

Our process still requires additional forms for the approver and Finance officer. The approver's form will provide functionality for approving or rejecting the request, and in the Finance officer's form, the disbursement details fields we disabled in the request form will have to be enabled and made required for the Finance officer to input details of the disbursement.

Adding Comments to the Form

Considering that the approver might reject the request, we also want a way for the approver to provide feedback to the requestor on why the request was rejected.

There are two ways we can go about this: adding a comments field to the form or using case notes. We will learn more about case notes later when we run our process. For now, let us modify our form to accommodate a use-case where we will want the comments displayed on the form.

A simple way to do this will be to just add a textarea where users can type their comments, but the drawback here is that we are unable to know who entered the comment, and a user could overwrite the comments entered by another user. To safeguard against these shortcomings, we will create a comment section with two textarea fields, a hidden field, and a button.

The first textarea will contain all the comments entered so far. The second will be for the current user to add his/her own comment, the hidden field will store the name of the current user, and the button will trigger the action that will add the current user's comment to the comment log. We will use a little JavaScript to handle the click action and perform the merging of the comments.

Begin by dragging two empty rows above the Approval/Signoff subtitle. Drag the rows by selecting the last empty row on the form and dragging it using the handle in the top right corner of the row. In the first of the two rows, place a subtitle control and change its label to **Comments**. Divide the next row into three columns, setting its colspan property to **3 3 6**. Add a textarea control to the first column. In the Create/Select Variable modal, change the variable name to **comment** and click the Save button to create a new variable and associate it with the control.

Select the textarea and set its label to **Enter your comment** and the rows property to 2. In the middle column, place a button control (not submit). Change its id and name to **comment_button** and its label to **Add your comment**. In the last column, place another textarea and in the Create/Select Variable modal, change the variable name to **comments** and save it. Select the textarea and change its label to **Comments** and the display mode to View.

Next, place a hidden control in the last row on the form and save the variable name when prompted as **current_user**. With the controls in place, we complete the setup by adding a little JavaScript to the form. Select the form by clicking the gray area of the dynaform container and click the Edit button beside the JavaScript property to launch the editor and paste the following code into it.

```

//Comments handler
function addComments() {

    var currentComment = $('#comment').getValue();
    var currentUser = $('#current_user').getValue();
    var comments = $('#comments').getValue();

    if(currentComment.trim() !== '') {
        comments = currentUser + ': ' + currentComment + '\n' + comments;
        $('#comments').setValue(comments);
        $('#comment').setValue('');
    }
}

//Register comments handler to button events
$("#comment_button").find("button").on("click", addComments);
$("#form").submit( addComments );

```

A quick explanation of what this code does. Lines beginning with `//` indicate a comment. Adding comments to the code can be a useful reminder to your future self and others working with you on what you were thinking when you wrote the code. First we create a function called `addComments`. We are taking a slightly different approach this time in handling the action triggered by clicking the Add Your Comment button. Before, we used an anonymous function (we did not give it a name) that we defined inside the brackets of the `click` function as shown here:

```

$('#id_of_clicked_button').click(function(){ ... do some
stuff... });

```

This time we are creating our function first, and we call it `addComments`.

```

function addComments() { ... }

```

This approach allows us to reuse the function in more than one place in the form. For example, if we wanted to make sure that the comment was saved when the form was submitted, we would not need to rewrite the statements for saving the comment; instead we just execute the `addComments` function again.

So what does the `addComments` function do?

```
var currentComment = $('#comment').getValue();
```

This creates a variable called `currentComment`. We use the jQuery selector to select the comment textarea and ProcessMaker's `getValue()` helper function to get the text in the control and store the value in the `currentComment` variable we just created. Note that this variable is a JavaScript variable and it is different from the process variables we created earlier. The values of the JavaScript variables are temporary and available only when the form is being used.

```
var currentUser = $('#current_user').getValue();
var comments = $('#comments').getValue();
```

We are creating two additional variables and storing the value of the hidden `current_user` control and the comments textarea control. The `current_user` value will be populated by a trigger when we run the case, but for now we will put a default value to test that it works.

```
if(currentComment.trim() !== '') { ... }
```

We check that the user has entered a comment using the `.trim()` function to remove any leading or trailing spaces from the text in the comment textarea and then checking that it is not equal to an empty string (`!==`).

```
comments = currentUser + ': ' + currentComment + '\n' + comments;
```

If there is a comment, we concatenate (the `+` operator is used to join strings in JavaScript) the value of the `current_user` hidden control with a colon, then the comment entered in the comment textarea, then a newline character (`\n`), which is similar to pressing Enter on your keyboard, and finally whatever text was already in the comments textarea. We save it to the `comments` variable, overwriting it.

```
$('#comments').setValue(comments);
```

We use ProcessMaker's `setValue()` function to update the value of the comments textarea to the concatenated comment:

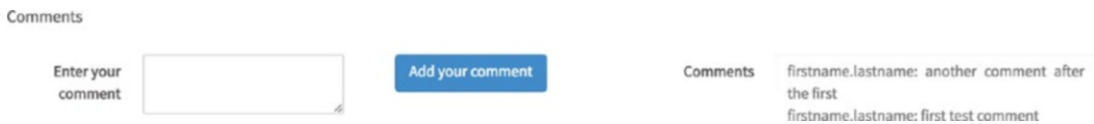
```
$('#comment').setValue('');
```

We use ProcessMaker's `.setValue()` function again to clear the value of the comment textarea by setting it to an empty string:


```
$("#comment_button").find("button").on("click", addComments);
$("#form").submit( addComments );
```

Finally, we use the jQuery selector to select the `comment_button` control and the jQuery `.find()` function to get the actual button element and register the `addComments` function as an event handler for its click event. This way, we can register the function to the submit event of the form without having to rewrite the code for the process of adding a comment. We use the jQuery selector to select the form and pass the `addComments` function as the handler for its submit event.

Click the Save button to close the JavaScript editor. Before testing the comment feature we just added, select the hidden control and set the default value property to **firstname.lastname**. Now preview the form. There should be a comment section on the form as shown next. Type in some text in the Enter Your Comment field and click the Add Your Comment button. This should move the text from the first textarea to the read-only Comments textarea and append the default value we set for `current_user` to the comment.



We can make our comments fancier by using a panel with custom HTML and styling like a chat conversation if we want or use a grid to tabulate the comments. The key takeaway, however, is that we can add comments and display them on the form. Close the preview and delete the default value we set in the `current_user` hidden control. Next, we will create a copy of the form and modify it for the approval form.

 **Notes from Amos** I personally don't like using the `find()` function because you have to know what HTML element you are searching for ("button", "div", "input", "select", "textarea", and so on) with each type of control, so I directly select the element through its ID `form[id]`, which requires escaping the square brackets [and] with `\`: `$("#form\[comment_button\]").on("click", addComments);`

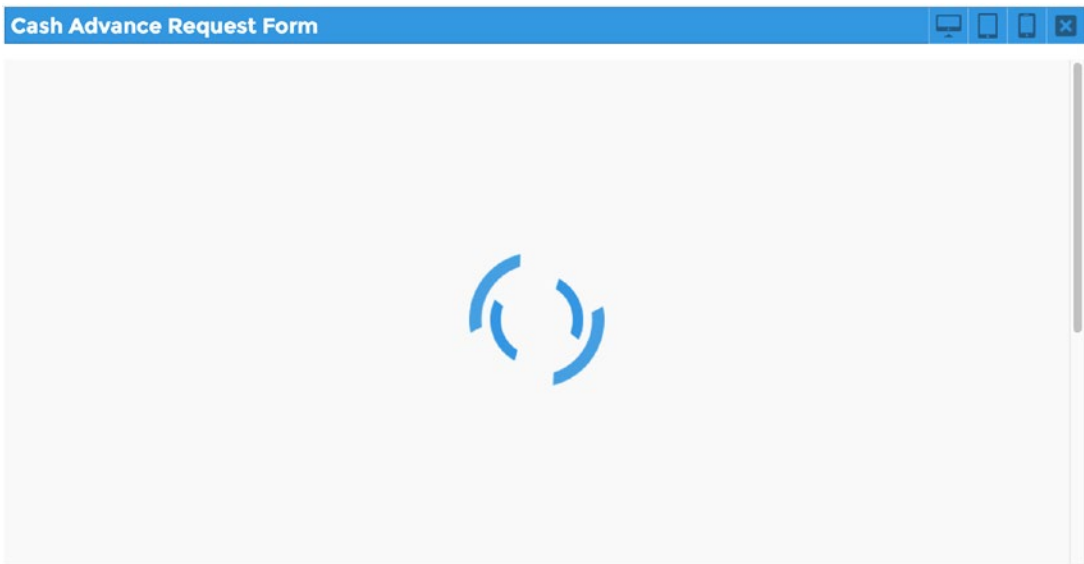
I also don't see much reason to use `.on()` instead of `.click()`, which is a shorthand that does the same thing: `$("#form\[comment_button\]").click(addComments);`

Some people find escaping with jQuery selectors to be very confusing, so they use this code: `$("[id='form\[comment_button\]']").click(addComments);`

No matter which method you choose, I do recommend using `jQuery()` in all instances, so that people get used to using jQuery and won't try to use `document.getElementById().value` and `document.getElementById().onchange`, which will cause problems.

Debugging Errors in JavaScript

Before we continue, let us quickly look at how to troubleshoot errors that might occur from adding JavaScript to the dynaform. If there is an error in your JavaScript code that is executed when the form loads, you will be unable to preview the form, and the preview screen will be stuck in the loading state, as shown in the following image. If the error is in a section of code that will be executed later, the form will appear in the preview, but the functionality expected of the JavaScript code will not work.



The common causes of error in JavaScript code include incorrect variable names, syntax errors, and typos. It is important to note that JavaScript is a case-sensitive language, and you are encouraged to use a consistent case in naming your variables and controls to help avoid errors.

Let us illustrate this with an example. We will introduce a typo into the code we just added to our form. Select the form in the Dynaform Designer and click the Edit button beside the JavaScript property. In the JavaScript editor displayed, make the following change to the code.

Change the line

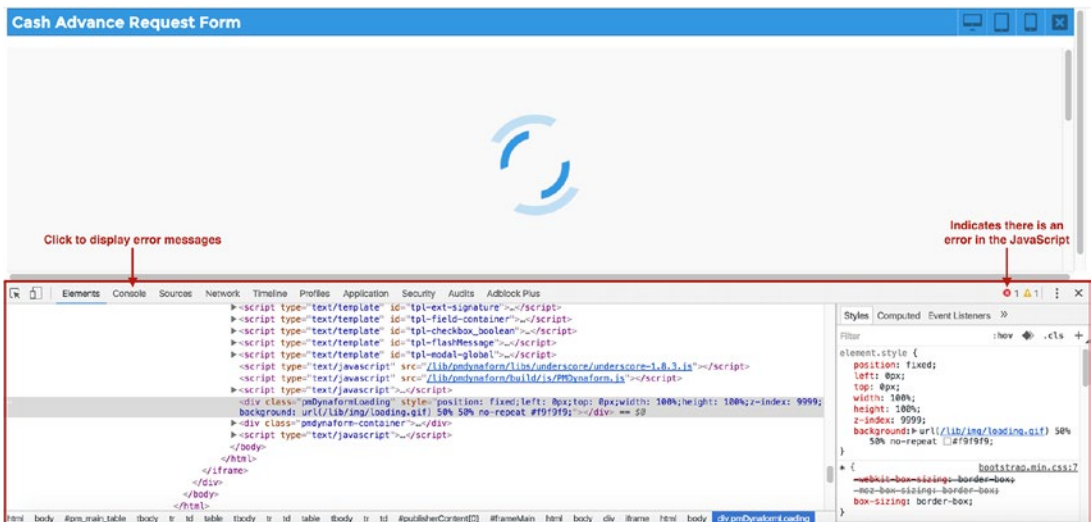
```
$("#comment_button").find("button").on("click", addComments);
```

to

```
$("#commentbutton").find("button").on("click", addComments);
```

We have now added a typo to our JavaScript code (left out the underscore in `comment_button`). Click the Save button and proceed to preview the form. You will observe that the form is stuck in the loading state, indicating an error.

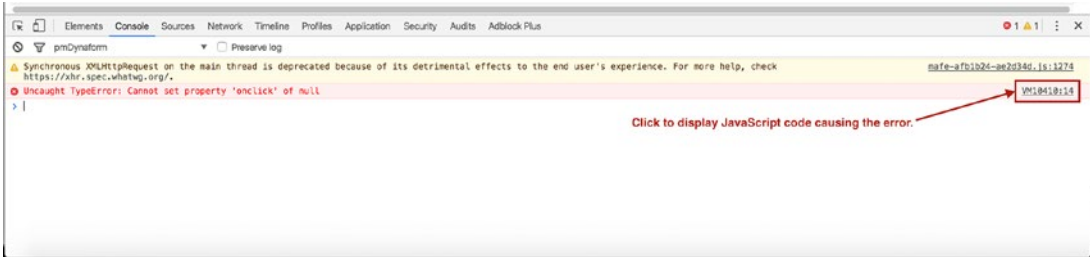
If you're using the Chrome browser, right-click the loading screen and in the context menu, select Inspect and the Chrome Developer Tools should appear as shown next.



Click the Console tab in the Developer Tools window to display the error messages as shown next. Click the link to the right of the error message to view the JavaScript code

CHAPTER 9 ADDING FORMS TO THE PROCESS

causing the error. The number prefixing the link is the line number of the section of code causing the error.



The code is displayed as shown in the following image, and we can see that line 14 contains the section of code where we added the typo.



Armed with this information we can close the preview window and make the corrections to our JavaScript code. Close the Developer Tools by clicking the Close (x) icon in the top-right corner and close the preview mode. Select the dynaform in the editor and correct the code.

Change the line

```
$("#commentbutton").find("button").on("click", addComments);
```

back to

```
$("#comment_button").find("button").on("click", addComments);
```

Save your changes and preview the form again; it should now load correctly. To learn more about how to debug JavaScript code in dynaforms, see this article in the ProcessMaker wiki: http://wiki.processmaker.com/3.0/JavaScript_in_DynaForms#Debugging_JavaScript

Cloning the Form

Export the form by clicking the Export button in the dynaform menu and save the .json file to a location on your system. Close the Dynaform Designer. In the Process Designer, click the Create icon (+) beside Dynaforms in the Main Toolbox (process objects) to create a new blank dynaform. Set the title to **Cash Advance Approval Form** and the description to **This form will be used to approve Cash Advance Requests**. Click the Save & Open button to create the dynaform and open it in the Dynaform Designer.

Click the Import button in the dynaform menu and select the file you just exported. The form is now imported and displayed. One of the features from the 2.x version of ProcessMaker that has yet to make its way to 3.x is the option to save a copy of a dynaform. This would have saved us the effort of exporting it and reimporting it. That does, however, get the job done until the Save As feature makes it to 3.x.

To make this form ready for the approver, we just need a few modifications. First we disable the fields that were filled by the requester so that they cannot be edited by the approver. Then we add a way for the approver to indicate if the request is approved or rejected. We will explore two approaches to doing this: using a dropdown field (no need for JavaScript) and using two buttons and a confirmation prompt.

Select all the controls in the Request Details section of the form (Request Date, Employee Name, Department, Requested Amount, Reason for Expenses), and set their display mode to Disabled. A preview of the section should now look like the following.

Request Details

Request Date *	<input type="text"/>	Employee Name *	<input type="text"/>
Department *	<input type="text" value="Select your department"/>	Requested Amount *	<input type="text"/>
Reason for Expenses *	<input type="text"/>		

Adding Approval Functionality

As mentioned earlier, there are two ways we can add the approval functionality to the approval form we just created. The first approach requires no code. We simply add a control (it can be a dropdown, checkbox, or radio) to the form that allows the user select an approval option and associate the control with a variable.

Approval without Code

To use this approach, select the row containing the Submit button and divide it into two by setting its col-span property to **6 6**. Move the Submit button to the right column of the row.

Next, place a radio control in the left column beside the Submit button. In the Create/Select Variable modal shown, change the variable name to **is_approved**. Expand the settings and change the variable type to Boolean. Click on the options and change the Label of the options from **true** to **Yes** and from **false** to **No**. Click the Apply button and save the variable. Select the radio control on the form and change its label to **Approve this request?** and make it required. Also change the label of the Submit button to Submit. Preview the form and the bottom should look like this.

The screenshot shows a form with a label "Approve this request?" followed by two radio buttons labeled "Yes" and "No". To the right of these controls is a "Submit" button. The form is displayed on a light gray background.

With this design, the supervisor can indicate if the request should be approved or rejected by choosing Yes or No. Let us make a clone of this form to show the other method we can use to add approval functionality to the form. Export the form by clicking the Export button in the dynaform menu and save it to a location on your system. Close the Cash Advance Approval Form to return to the process designer.

Approval with Code

Click the Create (+) button beside Dynaforms in the Main Toolbox (process objects) to add a new dynaform. Set the title to **Cash Advance Approval Form v2** and the description to **An alternative form for the supervisor to approve Cash Advance Request**. Click the Save & Open button to open the form in the Dynaform Designer. Import the form we just exported by clicking the Import button in the dynaform menu and selecting the exported Cash Advance Approval Form.json file you saved to your system.

With the form successfully imported, scroll to the bottom of the form and delete the radio button we added for the approval functionality. To delete a control, click it to select it and then click the X button in the top-right corner of the control.



Now place a Submit button in the column where the radio control used to be. We now have two Submit buttons on the form. Select the button in the left column, set its id and name properties to **approve_button** and its label to **Approve Request**. Select the Submit button on the right and also change its properties. Set its id and name properties to **reject_button** and its label to **Reject Request**. As you may have guessed, the user will be able to approve or reject the request by clicking the appropriate button.

However, we need a way to store the user's decision in a variable so we can know if the request was approved or rejected. To do this, we will use the same `is_approved` variable we created earlier for the previous approval form. This time however, we add it to the form as a hidden control. We already have one hidden control on the form in the row beneath the Submit buttons. Select the row and split it into two by setting its `col-span` to **6 6**. Now place a hidden control in the empty column of the row. In the Create/Select Variable modal, switch to the Select Variable option and select the `is_approved` variable.

Our form is now almost ready. All that is left is to define an event handler for the Approve Request and Reject Request buttons so that they update the `is_approved` variable accordingly when clicked. If the user clicks the Approve Request button, we will set the `is_approved` variable to 1, which means Yes, and set it to 0 meaning No if the Reject Request button is clicked. We will also prompt the user with a confirmation dialog to confirm the action before proceeding to avoid accidental approval or rejection.

Select the form by clicking the gray part of the dynaform container. Click the Edit button beside the JavaScript property and add the highlighted code following to the form beneath the code we already added for the comments, as shown.

```
//Comments handler function addComments() {
var currentComment = $('#comment').getValue();
var currentUser = $('#current_user').getValue();
var comments = $('#comments').getValue();
```

```

if(currentComment.trim() !== '') {
  comments = currentUser + ': ' + currentComment + '\n' + comments;
  $('#comments').setValue(comments);
  $('#comment').setValue('');
}
}

//Register comments handler to button events
$("#comment_button").find("button").on("click", addComments);
$("#form").submit( addComments );

//Approve or Reject handler function approval(action) {
if (confirm('Are you sure you want to ' + action + '?')) {
if (action === 'APPROVE') {
$("#is_approved").setValue(1);
$("#form").submit();
}
if (action === 'REJECT') {
$("#is_approved").setValue(0);
$("#form").submit();
}
}
else return false;
}

//Approve form
$("#approve_button").find("button").on("click", function()
{ return approval('APPROVE');
});

//Reject form
$("#reject_button").find("button").on("click", function() {
return approval('REJECT');
});

```

As before, let's quickly walk through the code and explain what we are doing.

```
function approval(action) {...}
```

First we create a function called `approval` that takes a parameter called `action`.

```
if (confirm('Are you sure you want to ' + action + '?')) {...} else return false;
```

Then we use the built-in JavaScript `confirm` function to prompt a confirm dialog. We are, however, wrapping the `confirm` function execution in an `if` statement. If the user clicks OK in the confirm dialog, the result returned is `true` and we proceed to perform the statements in the braces. However, if the user clicks CANCEL in the dialog, the result is `false`, and we execute the `else` statement, which returns `false`. The `confirm` function has a message parameter, which is displayed to the user. Here, we create a string asking the user if they are sure they want to perform the action passed as the argument to the `approval` function.

```
if (action === 'APPROVE') {
  $("#is_approved").setValue(1);
  $("form").submit();
}
if (action === 'REJECT') {
  $("#is_approved").setValue(0);
  $("form").submit();
}
```

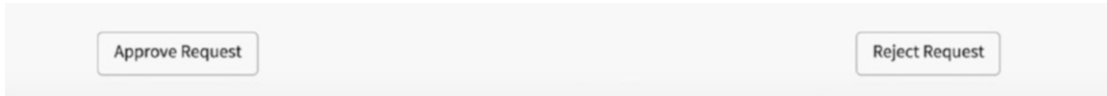
If the result of the `confirm` function is `true`, we check the value that was passed as the argument for the `action` parameter of the `approval` function. If it is `APPROVE`, we use the jQuery selector to select the `is_approved` hidden control and use ProcessMaker's built-in `setValue` function to set its value to 1. Then we select the form and call its `submit` function to submit the form. If the argument passed is `REJECT`, we set `is_approved` to 0 and also submit the form.

```
//Approve form
$("#approve_button").find("button").on("click", function() { return approval('APPROVE'); });

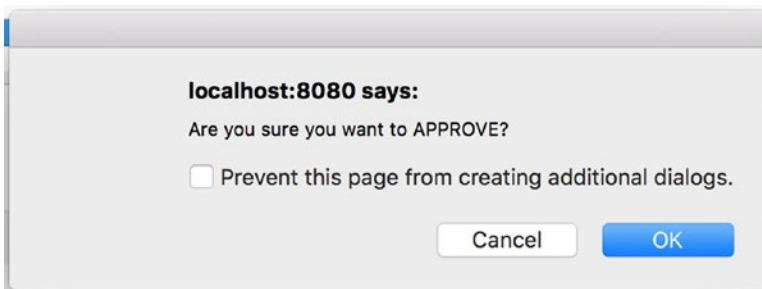
//Reject form
$("#reject_button").find("button").on("click", function() { return approval('REJECT'); });
```


Finally, we register the approval function as the handler for the click event of the `approve_button` and `reject_button` submit controls.

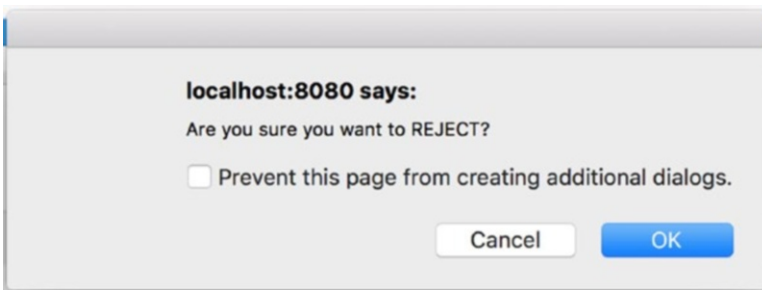
Go ahead and preview the form, and the bottom of the form should look like the following.



Click the Approve Request button and a confirmation dialog like the following should be displayed. Click the Cancel button and the dialog should close without submitting the form.



Click the Reject Request button and a confirmation dialog should appear. Click the OK button this time, and the form should be submitted.



We now have two different forms we can use for the supervisor to approve the request. When building your own processes, you will need only one, but I wanted to show you two ways to achieve the same functionality. I prefer the buttons approach, though; and if you are thinking it is a lot more work than the first approach, remember that once you have it working in one form, you just have to copy the code to any other

form you want to use it and add the submit and hidden controls with the same name. Go ahead and close the preview and the Dynaform Designer.

We are almost done with the forms for the first part of the process. All that is left is a form for the Finance officer to enter the disbursement details.

Another Variant of the Form

For the Finance officer's form, we begin by creating a new dynaform. Click the Create (+) button beside dynaforms in the Main Toolbox (process objects) and set the title to **Cash Advance Disbursement Form** and the description to **Form to be filled by finance officer providing disbursement details..** Click the Save & Open button to create the form and open it in the designer. Import the Cash Advance Request Form.json file we exported earlier. We could also have cloned the Cash Advance Approval Form, since our goal is to have the same fields replicated in this new form with a few modifications.

With the form successfully imported, proceed to disable all the fields that were filled by the requester—that is, all the fields in the Request Details section of the form—so that they cannot be edited. Select each control and set the display mode to Disabled.

Next, we enable the fields that will be filled in by the Finance officer and make them required. They are the Amount Advanced and Date Advanced fields in the Disbursement Details (Finance) section. Select the controls in the form, set the display mode to Edit, and check the Required property. Also change the label of the Submit button at the bottom of the form, from Submit Request to Submit. Preview the form. The form should look like the next image.

Cash Advance Requisition

Request Details

Request Date *	<input type="text"/>	Employee Name *	<input type="text"/>
Department *	<input type="text" value="Select your department"/>	Requested Amount *	<input type="text"/>
Reason for Expenses *	<input type="text"/>		

Disbursement Details (Finance)

Amount Advanced *	<input type="text"/>	Date Advanced *	<input type="text"/>
-------------------	----------------------	-----------------	----------------------

Comments

Enter your comment	<input type="text"/>	<input type="button" value="Add your comment"/>	Comments	<input type="text"/>
--------------------	----------------------	---	----------	----------------------

Signoff/Approval

Requested By	<input type="text"/>	Date and Time	<input type="text"/>
Approved By	<input type="text"/>	Date and Time	<input type="text"/>
Disbursed By	<input type="text"/>	Date and Time	<input type="text"/>

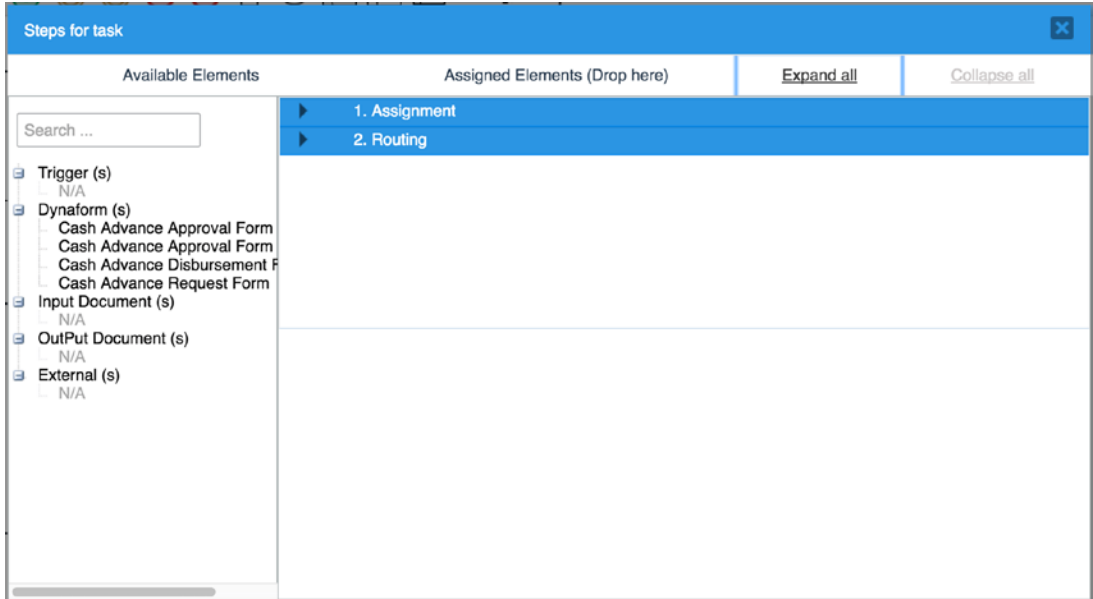
I agree to account for this advance within ten working days of the date advanced as defined in the preceding section, either with adequate receipts, cash or a check for the balance made payable to MSB Corporation. I understand that my failure to account for advanced funds in full within sixty days will result in a Payroll deduction for the balance due. By clicking the submit button below, I agree to allow MSB Corporation to make any such deductions from my pay.

This completes all the forms we require for the Cash Advance part of the process. To see our forms in action, we have to add them to the corresponding tasks. Close the Dynaform Designer and return to the Process Designer.

Assigning a Form to a Task

In order for the forms to be available for a task, we add it as a step in that task. Remember, we defined a step as an action that must be done to complete a task. In our sample process, to complete the first task (Request Advance) of the process, the user must fill out a Cash Advance Request Form.

To add the form to the task, right-click on the Request Advance task in the process map and select the Steps option. This displays the modal shown next. As mentioned earlier in the book, a step can either be a dynaform, trigger, input or output document or an external step. As you can see in the image, we do not yet have any triggers, input or output document defined (available elements are listed on the left). We can, however, see the forms we have created so far.



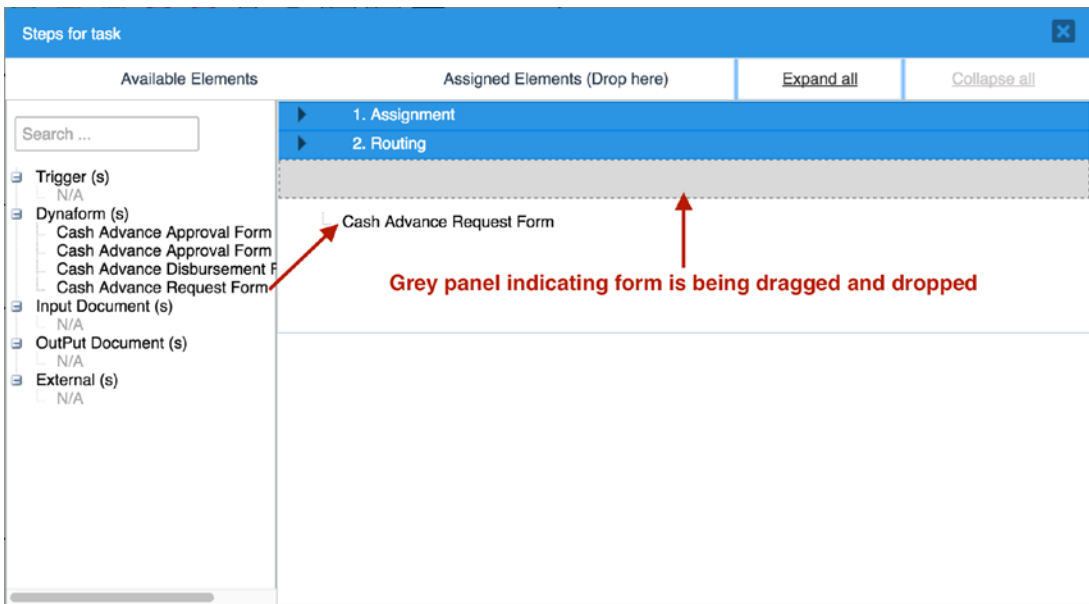
Default Steps in a Task: Assignment and Routing

Before we add the form for this task, let us take a moment to quickly explain the two default steps already added to the task; namely Assignment and Routing. These are the last steps that will be performed for every task. The Assignment step evaluates the rule defined in the Assignment Rules property of the next task to determine the next user that the case will be assigned to at the end of this task. In the Cash Advance and Expense Retirement process for example, the assignment step of the Request Advance task will determine which user to assign as the supervisor that will approve the request in the next task.

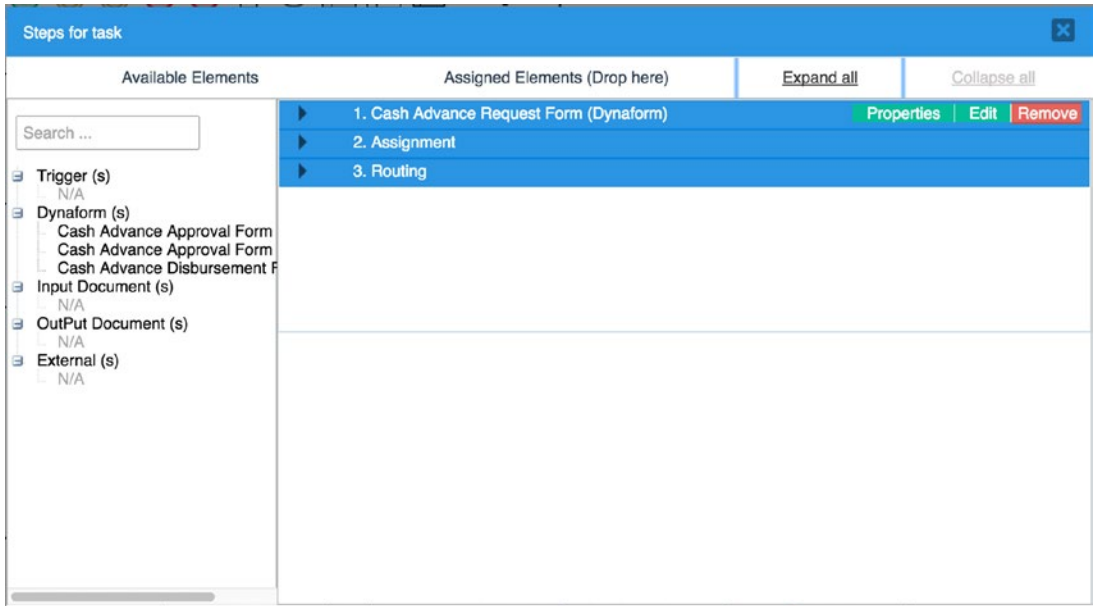
The Routing step works alongside the Assignment step and is used by the ProcessMaker engine to move the case from the current task to the next task. In the case of a decision gateway between tasks, this step evaluates the routing rule defined in the gateway to determine which task comes next. In summary, the assignment step determines the next user, while the routing step determines the next task.

Adding a Dynaform Step

To place a dynaform as a step in a task, simply drag the name of the form from the Available Elements on the left of the Steps for Task modal to the Assigned Elements on the right. Hovering your mouse over the name of the dynaform will display the full name. Let us go ahead and assign the Cash Advance Request Form to this task by dragging and dropping it under the assigned elements as shown in the next image.



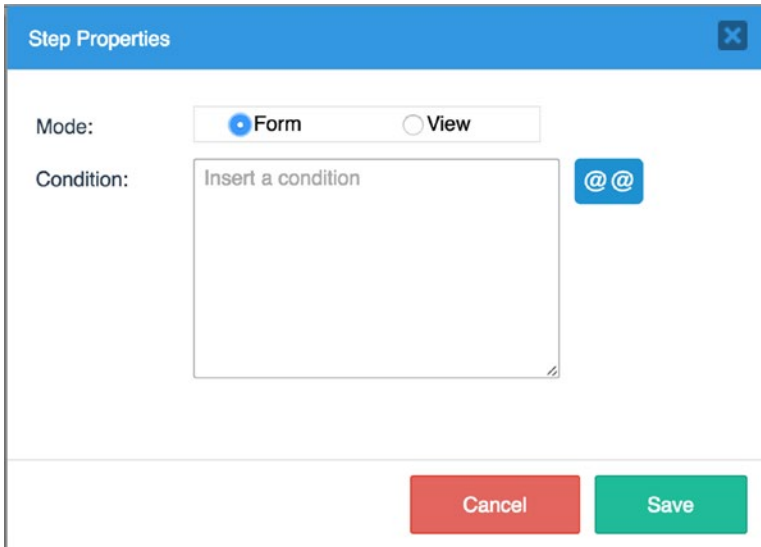
When the form is added to the task, it displays as shown here.



You will observe that the form is automatically moved to the top of the steps and assigned the number 1. This is expected, as we've seen that the Assignment and Routing steps always come last in the list of steps for a task. You will also notice that unlike the default Assignment and Routing steps, it also has some action links on the right of the title (Properties, Edit, and Remove).




Click the Properties link to display the following modal, which allows us to define the Mode the form should be displayed in and the Condition for displaying it.



Changing the Mode property to View will make the form read-only when displayed. Leave it as the default Form (Edit) mode. If the Condition property is blank, the form will always be displayed for that task. If we wanted to show the form only under certain conditions, we could define that condition in this property, and the form would only be shown if the condition defined evaluates to true. Leave it blank and close the Properties modal by clicking the Cancel button.

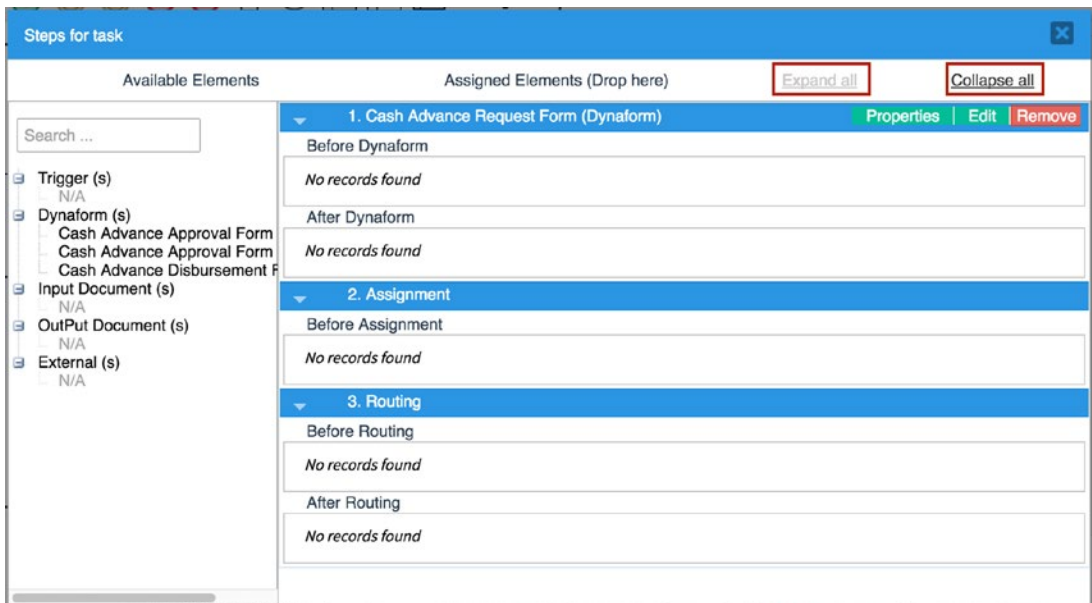
Clicking the Edit button opens the dynaform in the Dynaform Designer, allowing us to edit the form and its controls. Close the Dynaform Designer. Clicking the Remove button removes the dynaform from the assigned elements and places it back under the list of available elements. You will be prompted to confirm that you want to remove the form step.

The elements (steps) assigned to a task can be reordered by dragging and dropping them in the list of assigned elements, with the exception of the two default steps—Assignment and Routing, which are always the last two steps. You can try this out by dragging and dropping the other forms under Assigned Elements to reposition them.

 Do not forget to remove them when done and leave only the Cash Advance Request form when done, as that is the only form required for this task.

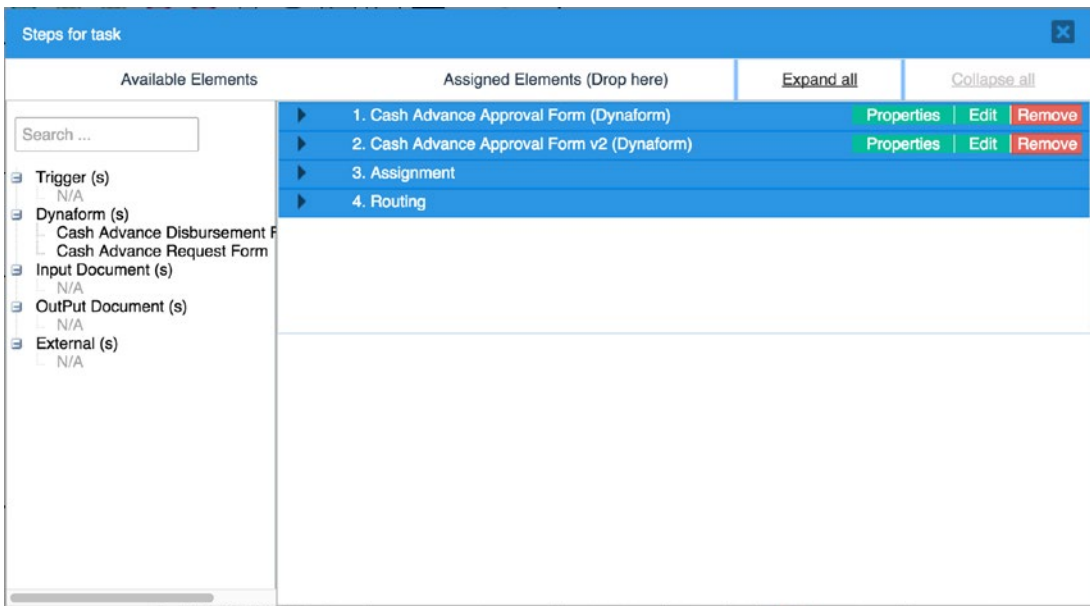
The steps are collapsed by default so that multiple steps can be displayed on the screen at once without having to scroll. Click the **Expand all** link at the top of the screen to expand the steps as shown next. Clicking the **Collapse all** link will return the screen to the default compact view.

If you are wondering about the “Before and After” items (such as Before Dynaform or After Routing) and the “No records found” displayed under the steps, these are locations where we will place triggers. You will learn more about this in Chapter 12 when we explore triggers. Go ahead and close the Steps for Task modal.



Now that we know how to add forms to a task, let us add the Cash Advance Approval Form to the Approve Advance task, and the Cash Advance Disbursement Form to the Disburse Advance task. To illustrate how the Conditions property of the dynaform works, we will add the two variants of the Approval forms to the Approve Advance task and define a condition that shows the first form if the department of the requesting employee is Finance and the other variant for other departments.

Right-click the Approve Advance task and select Steps from the context menu. Drag the Cash Advance Approval Form and the Cash Advance Approval Form v2 to the Assigned Elements.



Click the Properties for the Cash Advance Approval Form and in the Conditions property, paste in the following condition:

```
@@department == 'Finance'
```

This means that the form should be displayed if the value stored in the department variable is equal to Finance. Click the Save button.

Click the Properties for the Cash Advance Approval Form v2 and in the Conditions property, paste the following condition:

```
@@department != 'Finance'
```

This means that the form should be displayed if the value stored in the department variable is not equal to Finance. Click the Save button.

Close the Steps for Task modal.

Right-click on the Disburse Advance task and select Steps from the context menu. Drag and drop the Cash Advance Disbursement Form under Assigned Elements and close the Steps for Task modal.

We have now successfully created forms and assigned them to the tasks in the Advance part of the process. We will create and add the remaining forms later on. The next stage in building our process will be assigning users to its tasks. In order to do that, however, we need to create some users, so in the next chapter we will take a quick detour to learn about administering users before continuing on in Chapter [11](#) to assign them to tasks in the process.

CHAPTER 10

Administering Users in ProcessMaker

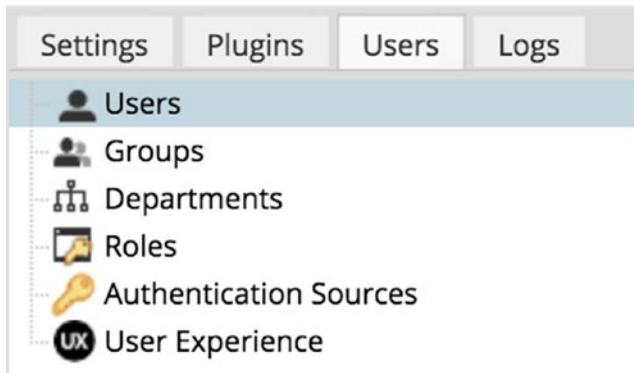
Before we can assign users to our process, we first have to create the users in the system. In this chapter, we will learn about the features available to us in ProcessMaker for managing users. To begin, head over to the Admin section of ProcessMaker by clicking Admin in the main menu.



The workspace in the Admin section of the application is divided into two panes. The left pane contains the Admin menu and submenu, while the right pane displays the view for the admin menu option selected on the left. The admin menu on the left consists of four tabs: Settings, Plugins, Users and Logs.

The Settings tab, as the name implies, contain options for configuring ProcessMaker settings such as the logo, email, and so on. The Plugins tab is used for adding, enabling, and disabling plugins to ProcessMaker. In the Enterprise Edition, it also displays the license details and options to enable and disable the Enterprise features. The Logs tab is where you can view logs from Events, Emails sent, Cron (Scheduled Tasks), and the Case Scheduler. The Users tab is what we are interested in for now and we will learn about the Settings, Plugins, and Logs admin options later.

Click the Users tab in the left pane to display the submenu options described briefly next.



Users: This displays the list of users in the system. It also has options for creating new users, editing existing users, and assigning the users to groups.

Groups: This displays the list of Groups with options to add new groups and assign users to the groups. Groups in ProcessMaker make it easier to assign tasks and process permissions to a set of users as opposed to doing it one by one. For example, we could have more than one Finance Officer in the organization. By creating a group for Finance Officers, we can easily assign that group to the disbursement task of the Cash Advance and Expense Retirement process, and any of the officers can work on the requests routed to that task.

Departments: This allows us to replicate the hierarchical structure of the organization in ProcessMaker. A user can, however, belong to only one department.

Roles: This displays the roles available in ProcessMaker and can be used to add new roles and manage the permissions of the roles. A user can only have one role in ProcessMaker. The permissions assigned to a role control what the user can do in ProcessMaker application. Groups, on the other hand, are used to control what a user can do in a process. Users are generally given a role of

Operator, which allows them to log in and run cases. You can always create new roles to give different levels of access to the application. The System Administrator role has full access to the system and should be assigned cautiously.

Authentication Sources: This is used to define additional sources for authenticating users when logging in to ProcessMaker. The default authentication source is ProcessMaker’s MySQL database. If you have an existing Active Directory or LDAP directory in the organization, you can set it up as an authentication source so that users can use the same credentials to log in to ProcessMaker.

User Experience: This is used to define the way ProcessMaker will be displayed for a user when they log in. The options are Normal, Mobile, Switchable and Single Application. We will illustrate these later after creating a few users.

Users

Let us now look at the Users submenu in depth. Click the Users submenu in the left pane under the Users tab. This displays the Users menu and list of users in the system, as shown here. Currently, we have only one user, which is the admin user (the username will be what you defined when installing ProcessMaker).



User Name	Full Name	Status	Role	Last Login	# Cases	Due Date
user	user, Administrator (user)	Active	System Administrator	2016-09-19 10:08:07	0	2020-01-01 00:00:00

Adding a New User

To add a new user, click the New button to display the form shown next. The form is divided into five sections: Personal Information, Change Password, Account Option, Profile, and Preferences. The required fields for creating a new user are marked with a red asterisk. Most of the fields are self-explanatory. For now, we will proceed with the minimum information required to create a user. The following table provides sample users we will create for illustration purposes. Leave every other field as-is for now and fill in only the required fields.

We are using a generic password for all the users, since this is just an illustration, in a live production deployment, stronger and unique passwords should be used. To test email notifications later in this guide, use your email address for all the users. If you use Gmail, you can use different aliases of your Gmail address for the users' emails, so as to be able to determine which user an email was sent to, when we learn about email notifications. To learn more about aliases in Gmail, see <https://support.google.com/mail/answer/12096?hl=en>.

The screenshot shows a user administration form with the following sections and fields:

- Personal information:**
 - * First Name: [Text input]
 - * Last Name: [Text input]
 - * Username: [Text input]
 - * Email: [Text input]
 - Address: [Text input]
 - Zip Code: [Text input]
 - Country: [Dropdown menu]
 - State or Region: [Dropdown menu]
 - Location: [Dropdown menu]
 - Phone: [Text input]
 - Position: [Text input]
 - Replaced by: [Text input]
 - Expiration Date: 2017-12-09 [Calendar icon]
 - Calendar: [Dropdown menu]
 - Status: Active [Dropdown menu]
 - Role: Operator [Dropdown menu]
- Change Password:**
 - * New Password: [Text input]
 - * Confirm Password: [Text input]
- Account Options:**
 - User must change password at next logon
- Profile:**
 - Photo: Please select a photo [Image upload icon]
 - Maximum upload file size: (20M)
- Preferences:**
 - Default Main Menu Option: HOME [Dropdown menu]
 - Default Cases Menu option: New case [Dropdown menu]

Buttons: Save, Cancel

First Name	Last Name	Username	Email	Password
Wanda	Marshall	wanda.marshall	{your email or email alias}	test123
Nicholas	Williams	nicholas.williams	{your email or email alias}	test123
Karen	Baker	karen.baker	{your email or email alias}	test123
Justin	Sanchez	justin.sanchez	{your email or email alias}	test123
Steve	Bennett	steve.bennett	{your email or email alias}	test123
Philip	Price	philip.price	{your email or email alias}	test123
Billy	Green	billy.green	{your email or email alias}	test123
Carlos	Shaw	carlos.shaw	{your email or email alias}	test123
Amy	Alexander	amy.alexander	{your email or email alias}	test123
Julia	Smith	julia.smith	{your email or email alias}	test123

That was a bit repetitive, but we now have 11 users in the system, which will be useful in exploring the basics of working with users in ProcessMaker. Our Users view now looks like the following image.

User Name	Full Name	Status	Role	Last Login	# Cases	Due Date
user	user, Administrator (user)	Active	System Administrator	2016-09-20 11:57:02	0	2020-01-01 00:00:00
julia.smith	Smith, Julia (julia.smith)	Active	Operator		0	2017-09-20 00:00:00
steve.bennett	Bennett, Steve (steve.bennett)	Active	Operator		0	2017-09-20 00:00:00
wanda.marshall	Marshall, Wanda (wanda.marshall)	Active	Operator		0	2017-09-20 00:00:00
amy.alexander	Alexander, Amy (amy.alexander)	Active	Operator		0	2017-09-20 00:00:00
justin.sanchez	Sanchez, Justin (justin.sanchez)	Active	Operator		0	2017-09-20 00:00:00
nicholas.williams	Williams, Nicholas (nicholas.williams)	Active	Operator		0	2017-09-20 00:00:00
karen.baker	Baker, Karen (karen.baker)	Active	Operator		0	2017-09-20 00:00:00
philip.price	Price, Philip (philip.price)	Active	Operator		0	2017-09-20 00:00:00
carlos.shaw	Shaw, Carlos (carlos.shaw)	Active	Operator		0	2017-09-20 00:00:00
billy.green	Green, Billy (billy.green)	Active	Operator		0	2017-09-20 00:00:00

As you can see, all the new users are given a default role of Operator, have never logged in to the system, and have no cases. The Due Date column shows when the user's account will become inactive and by default is one year from the date created.

Editing a User

To edit a user, simply select the user by clicking in the list and clicking the Edit button. The selected user will be highlighted in yellow, as shown in the following image. You can also right-click the user to display the context menu and select Edit, Or you can double-click the user from the list to display the Edit form.

User Name	Full Name	Status	Role	Last Login	# Cases	Due Date
user	user, Administrator (user)	Active	System Administrator	2016-09-20 11:57:02	0	2020-01-01 00:00:00
julia.smith	Smith, Julio (julia.smith)	Active	Operator		0	2017-09-20 00:00:00
steve.bennett	Bennett, Steve (steve.bennett)	Active	Operator		0	2017-09-20 00:00:00
wanda.marshall	Marshall, Wanda (wanda.marshall)	Active	Operator		0	2017-09-20 00:00:00
amy.alexander	Alexander, Amy (amy.alexander)	Active	Operator		0	2017-09-20 00:00:00
justin.sanchez	Sanchez, Justin (justin.sanchez)	Active	Operator		0	2017-09-20 00:00:00
nicholas.williams	Williams, Nicholas (nicholas.williams)	Active	Operator		0	2017-09-20 00:00:00
karen.baker	Baker, Karen (karen.baker)	Active	Operator		0	2017-09-20 00:00:00
philip.price	Price, Philip (philip.price)	Active	Operator		0	2017-09-20 00:00:00
carlos.shaw	Shaw, Carlos (carlos.shaw)	Active	Operator		0	2017-09-20 00:00:00
billy.green	Green, Billy (billy.green)	Active	Operator		0	2017-09-20 00:00:00

This form is similar to the one used when creating the user, the only exception being that the password fields are no longer required. If you want to reset a user’s password, you simply edit the user and enter a new password in the password fields. Leaving the password field blank when editing a user leaves the user’s password unchanged.

Let us go ahead and edit the user Julia Smith and change the Due Date (Expiration Date) to a different date. Once it is changed, click the Save button at the bottom of the form and the Due Date for Julia Smith is now updated as shown next.

User Name	Full Name	Status	Role	Last Login	# Cases	Due Date
user	user, Administrator (user)	Active	System Administrator	2016-09-20 11:57:02	0	2020-01-01 00:00:00
julia.smith	Smith, Julia (julia.smith)	Active	Operator		0	2022-09-20 00:00:00
steve.bennett	Bennett, Steve (steve.bennett)	Active	Operator		0	2017-09-20 00:00:00

Disabling a User

All the users we just created are Active by default. In a real-world organization, there will often be need to disable a user, either when the user is on vacation or has left the organization. To disable a user, simply select the user and click the Disable button in the top menu. Alternatively, you can edit the user and change the status to Inactive or Vacation.

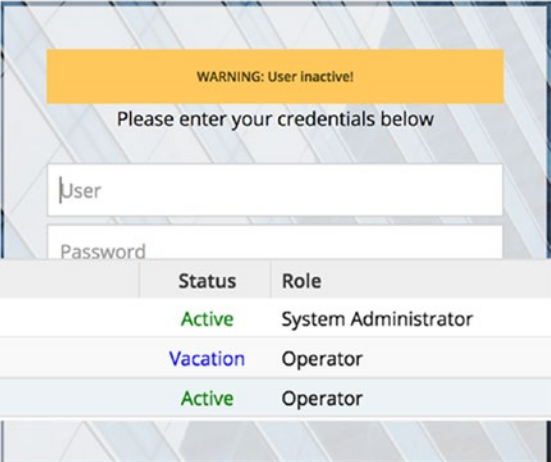
When the user’s status is Inactive, the user will be unable to log in to the system and cannot be assigned to work on any case. The Vacation status, however, disables the user

temporarily. The user cannot be assigned a new case until they log in to ProcessMaker again (implying that the user has returned and status is automatically updated back to Active).

To see this in action, disable the user Julia Smith. Open ProcessMaker in a different browser (not a different window or tab of the browser where you are logged in as Administrator). You can get the login URL by logging out and copying the URL (It should be similar to `http://localhost:8080/sysworkflow/en/neoclassic/login/login`) from the current browser and pasting it in the other browser.

If you try to log in to the other browser as Julia Smith (username: `julia.smith` and password: `test123`) you will see the error message “WARNING: User Inactive!”. Go back to the browser where you are logged in as the admin user and edit the user Julia Smith and change the status to Vacation and click the Save button.

Now try to log in as Julia Smith again in the other browser. You should be able to successfully log in. Returning to the Administrator view, the status of the user should have been changed from Vacation to Active. You might have to refresh the view (click the Users submenu option in the left pane) to see the change.



The screenshot shows a login form with a yellow warning banner at the top that reads "WARNING: User inactive!". Below the banner, it says "Please enter your credentials below" and has two input fields labeled "User" and "Password".

User Name	Full Name	Status	Role
user	user, Administrator (user)	Active	System Administrator
julia.smith	Smith, Julia (julia.smith)	Vacation	Operator
steve.bennett	Bennett, Steve (steve.bennett)	Active	Operator

You will also observe that the view for the user is different from that of the administrator, and there is only one link, Home, in the main menu. This is because the user’s role is set to Operator with permissions only to log in and create or work on cases.

Deleting a User

To delete a user, select the user and click the Delete button. You will be prompted to confirm that you want to delete the user. Clicking Yes deletes the user and No cancels the delete process. You can, however, only delete users that have no cases assigned or completed; that is, users who have never participated in any case. It is recommended to make users inactive rather than deleting them for historical and audit purposes.

User Summary, Group and Authentication

The Summary button shows the summary information of the selected user, while the Authentication button shows the authentication source configured for the user. This is the ProcessMaker MySQL database by default for users created in ProcessMaker. If the user was imported from another source, such as Active Directory or LDAP, it will be displayed in this tab. The Group button displays the groups that the user belongs to. We don't have any groups set up yet in our ProcessMaker instance. Let us create some groups and assign the users we have to different groups.

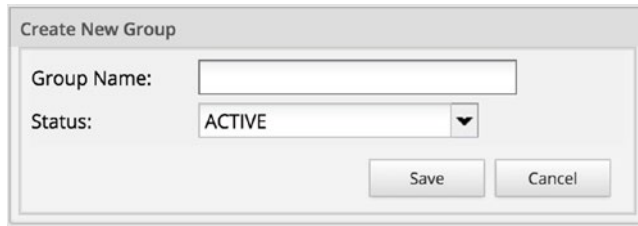
Groups

As mentioned earlier, we use groups to pool users together to make it easy to assign them to tasks and grant them permissions in a process, instead of doing it individually for each user. To begin our exploration of groups, click the Groups submenu in the left panel under the Users tab.



Creating a Group

To add a new group, click the New button and enter the name of the group. Leave the status set to ACTIVE and click the Save button. Go ahead and create in ProcessMaker the groups in the table for the purpose of our illustration.

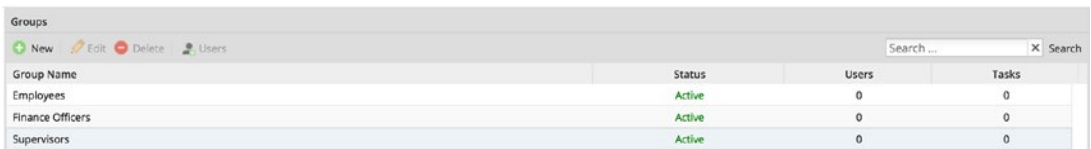


The 'Create New Group' dialog box contains the following fields and controls:

- Group Name:** A text input field.
- Status:** A dropdown menu currently set to 'ACTIVE'.
- Buttons:** 'Save' and 'Cancel' buttons.

Group Name	Purpose
Employees	A general-purpose group for all employees in MSB Corp
Supervisors	A group for supervisors in MSB Corp
Finance Officers	A group for Finance officers in MSB Corp

Once you are done, you should now have the groups listed in the view as shown next. You can see that the groups have no users and no task assigned to them.



The 'Groups' list view shows a table with the following data:

Group Name	Status	Users	Tasks
Employees	Active	0	0
Finance Officers	Active	0	0
Supervisors	Active	0	0

Editing a Group

You can change the name of a group or its status by editing it. Select the group from the list (this highlights it in yellow) and click the Edit button. You can also double-click the group from the list or select Edit from the context menu displayed when the group is right-clicked. This displays the same modal form used when creating the group. Make the desired changes to the name or status, and click the Save button.

Deleting a Group

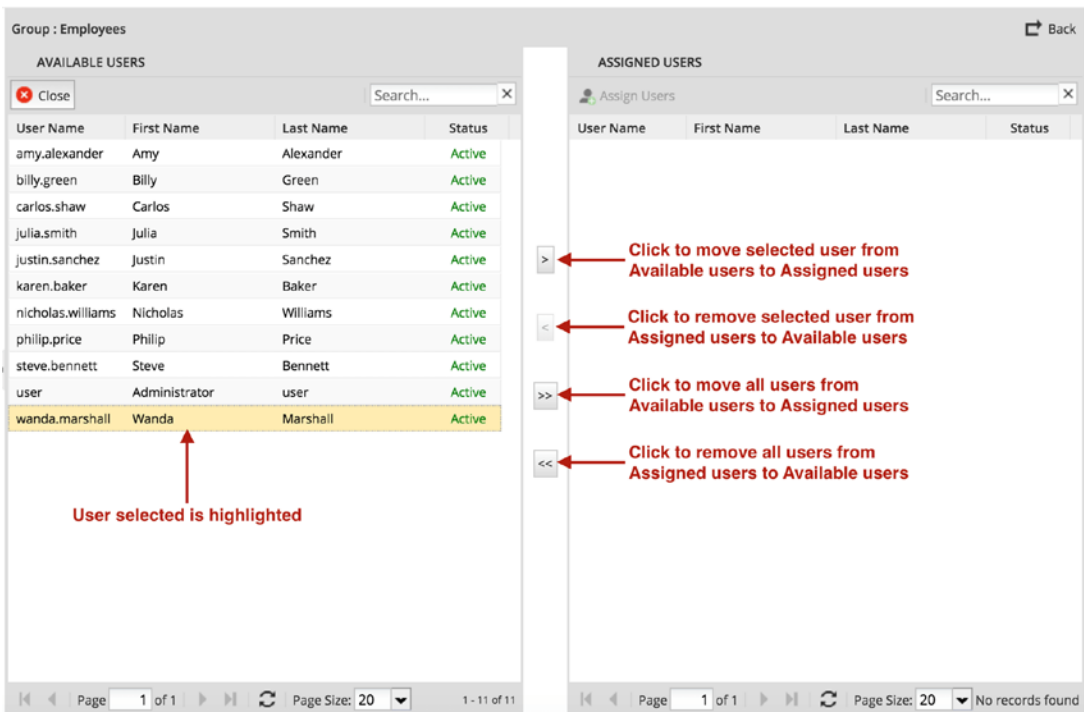
To delete a group, simply select the group and click the Delete button. You will be prompted to confirm that you want to delete the group. If the group is currently assigned to a task, you will not be able to delete it. You first have to unassign it from the task before deleting it.

Assigning Users to a Group

Now that we have created a few groups and users, we can proceed to assign the users to groups. This can be done either from the Users or Groups submenu options. To assign users to a group, select the group and click the Users button in the menu of the groups view. Let us begin by adding all users to the Employees group. Select Employees in the list of groups and click the Users button. This displays the following view.



There are currently no users assigned to the group. Next click the Assign Users button above the list to display this view.



The Available users list (all users in the system that do not belong to the group) is displayed on the left, and Assigned users (users belonging to the group) are displayed on the right. Both lists are divided by the following four buttons:

-
- > This is activated when one or more users (you can select multiple users by holding down the Ctrl/Cmd key on the keyboard and clicking the users) are selected from the Available Users list. Clicking this button moves the selected user(s) to the Assigned Users list, effectively assigning them to the group.
 - < This button is activated when one or more users are selected from the Assigned users list. Clicking this button moves the selected user(s) to the Available Users, list effectively removing them from the group.
 - >> This button when clicked moves all the users in the Available Users list to the Assigned Users list, assigning them to the group. The list can be filtered using the search box on top of the list. Try typing “b” in the search box and pressing the Enter key, Only users whose names contain the letter “b” are displayed. If this button is clicked with a filtered list, only the users displayed in the filtered list will be assigned to the group (in this example, only the three users matching the “b” filter will be assigned).
 - << This works like the previous button, but in reverse. It removes (unassigns) the users displayed in the Assigned Users list from the group and moves them to the list of Available Users.
-

Because our aim is to add all users to the Employees group, click the >> button to move all the users from the Available Users list to the Assigned Users list to assign them to the Employees group. Click the Back button in the top-right corner to return to the list of groups.



Group Name	Status	Users	Tasks
Employees	Active	11	0
Finance Officers	Active	0	0
Supervisors	Active	0	0

We can now see that we have 11 users in the Employees group.

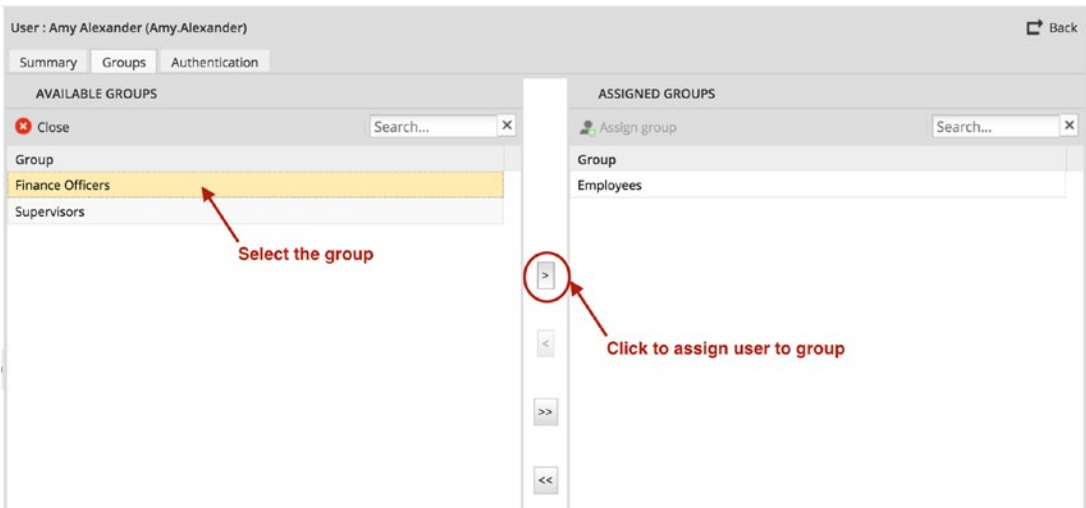
Assigning Groups to a User

To add users to the Finance Officers group, we will use the alternative approach. Click the Users submenu in the left panel to display the list of users. We want to add the users Amy Alexander and Philip Price to the Finance Officers group. Select Amy Alexander from the list of users and click the Groups button from the top menu to display the view

shown next. This shows the groups that Amy Alexander currently belongs to, at this point only Employees.



Click the Assign group button to display the following view. This is very similar to the view we saw earlier for assigning users to a group. We have a list of Available Groups on the left and the list of Assigned Groups on the right. Select the Finance Officers group on the left and click the > button to assign Amy to the Finance Officers group and click the Back button in the top-right corner to return to the list of users. Repeat the same process to add Philip Price to the Finance Officers group.



We now have users assigned to two groups. Return to the list of groups by clicking Groups in the submenu on the left pane. As an exercise, go ahead and assign Steve Bennett, Amy Alexander, Nicholas Williams, Julia Smith and Administrator to the Supervisors group. You can use either of the two approaches.

Next we look at setting up Departments in ProcessMaker.

Departments

The departments feature provides functionality for mirroring the hierarchical structure of an organization within ProcessMaker. We can create departments and subdepartments and assign users to these organizational entities and designate managers for each department. This allows ProcessMaker to automatically assign a case to the manager/supervisor of a user by checking which department a user belongs to and routing the case to the manager of that department.

We will demonstrate this when we look at the various assignment rules that can be configured for a task. For now let us set up the departments we defined earlier in this guide, assign the users to the different departments, and designate managers for the departments.

Click the Departments submenu option from the left pane to display the following view.



Adding a New Department

To add a new department, click the New button and in the modal dialog displayed, enter the name of the department and click Save. Create the following departments: Administration, Finance, Human Resources, Information Technology and Sales.

We can also define subdepartments under a department. To add a subdepartment, select the department and click the New Sub-Department button. This displays a form similar to that used to create the departments, enter a name for the sub-department and click the Save button. Go ahead and add the following subdepartments under the Information Technology department: Application Development and IT Support. The list of departments should now look like the next image.

Department Name	Status	Manager	Users
Information Technology	Active		0
Application Development	Active		0
IT Support	Active		0
Administration	Active		0
Finance	Active		0
Human Resources	Active		0
Sales	Active		0

We now have our departments set up. Next we will add users to the departments.

Assigning Users to a Department

We will illustrate this by assigning our finance officers, Amy Alexander and Philip Price to the Finance department. To assign users to a department, select the department (Finance) and click the Users button in the top menu to display the view shown here.

Full Name	Status
No users are currently assigned to this department.	

There are currently no users in the department. Click the Assign Users button to display the list of available users. Select Amy Alexander and Philip Price from the list and click the > button to assign them to the Finance department.

AVAILABLE USERS		ASSIGNED USERS	
Full Name	Status	Full Name	Status
Alexander, Amy (amy.alexander)	Active		
Baker, Karen (karen.baker)	Active		
Bennett, Steve (steve.bennett)	Active		
Green, Billy (billy.green)	Active		
Marshall, Wanda (wanda.marshall)	Active		
Price, Philip (philip.price)	Active		
Sanchez, Justin (justin.sanchez)	Active		
Shaw, Carlos (carlos.shaw)	Active		
Smith, Julia (julia.smith)	Active		
user, Administrator (user)	Active		
Williams, Nicholas (nicholas.williams)	Active		

Setting a Department Manager

ProcessMaker automatically assigns one of the users as the Manager of the department as shown next (Philip Price in this case). We can use the Set Manager and No Set Manager buttons to designate a user as manager and remove the manager designation, respectively.

ASSIGNED USERS	
Assign Users Set Manager No set Manager <input type="text" value="Search..."/>	
Full Name	Status
Alexander, Amy (amy.alexander)	Active
Price, Philip (philip.price) [Manager]	Active

To set no manager for the department, select Philip Price (if Amy was automatically assigned as manager, select Amy) from the Assigned Users list and click the No Set Manager button. Now, select Amy Alexander and click the Set Manager button to designate Amy as the manager for the Finance department.

ASSIGNED USERS	
Assign Users Set Manager No set Manager <input type="text" value="Search..."/>	
Full Name	Status
Alexander, Amy (amy.alexander)	Active
Price, Philip (philip.price) [Manager]	Active

ASSIGNED USERS	
Assign Users Set Manager No set Manager <input type="text" value="Search..."/>	
Full Name	Status
Alexander, Amy (amy.alexander)	Active
Price, Philip (philip.price)	Active

Click the Back button in the top-right corner to return to the list of departments. We now see that we have two users in the Finance department, and the manager of the department is Amy Alexander.

Department Name	Status	Manager	Users
Information Technology	Active		0
Application Development	Active		0
IT Support	Active		0
Administration	Active		0
Finance	Active	amy.alexander	2
Human Resources	Active		0
Sales	Active		0

Go ahead and assign the remaining users to the departments and designate managers as shown in the following table.

Department	Users	Manager
Administration	Carlos Shaw Steve Bennett	Steve Bennett
Finance	Amy Alexander Philip Price	Amy Alexander
Human Resources	Wanda Marshall Nicholas Williams	Nicholas Williams
Information Technology	Administrator	Administrator
Application Development	Billy Green	None
IT Support	Karen Baker	None
Sales	Julia Smith Justin Sanchez	Julia Smith

We have intentionally not defined any manager for the Application Development and IT Support subdepartments and made the Administrator user the manager of the Information Technology department. ProcessMaker uses the manager of the parent department as the manager of subdepartments that have no managers. In this case, Administrator will serve as the manager for Billy Green and Karen Baker. Our list of departments should now look like the following .

Department Name	Status	Manager	Users
Information Technology	Active	admin	1
Application Development	Active		1
IT Support	Active		1
Administration	Active	steve.bennett	2
Finance	Active	amy.alexander	2
Human Resources	Active	nicholas.williams	2
Sales	Active	julia.smith	2

Deleting a Department

To delete a department, select it from the list and click the Delete button. You can, however, only delete a department that has no users assigned to it. If there are users already assigned to the department, you will have to unassign them before deleting the department.

Roles

Roles in ProcessMaker are used to define a set of permissions available to users. For an exhaustive list of all permissions available in ProcessMaker, see this link (<http://wiki.processmaker.com/3.1/Roles>). A user can have only one role, and this role determines what that user can do in ProcessMaker.

Default Roles

ProcessMaker comes with three default roles, described next.

Code	Name	Status	Active Users	Create Date	Update Date
PROCESSMAKER_ADMIN	System Administrator	Active	1	2007-07-31 19:10:22	2007-08-03 12:24:36
PROCESSMAKER_OPERATOR	Operator	Active	10	2007-07-31 19:10:22	2007-08-03 12:24:36
PROCESSMAKER_MANAGER	Manager	Active	0	2010-03-29 09:14:15	2010-03-29 09:19:53

System Administrator (PROCESSMAKER_ADMIN)

This role is used for system administrators and process designers requiring complete access to all the features and functionality in ProcessMaker. Users with this role have all permissions available in ProcessMaker assigned. This includes the ability to configure the system, design processes, administer users, and much more.

Operator (PROCESSMAKER_OPERATOR)

The operator role is the default for users who simply need to log in to ProcessMaker and run cases.

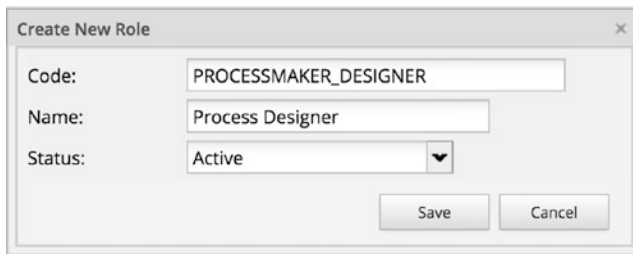
Manager (PROCESSMAKER_MANAGER)

The manager role is for users who are responsible for overseeing cases and users but do not design processes or administer the application.

Creating New Roles

You might find yourself in a situation where the default roles provided do not suffice for your needs. This is easily resolved by creating a new role and assigning it the desired permissions. Assuming we wanted Billy Green in the Application Development subdepartment to be able to design processes but we do not want him to have all the full privileges of a system administrator, we could readily create a new role called Process Designer and assign him this role.

To create a new role, click the New button and, in the modal that appears, define the Code, Name, and Status as shown here.



The screenshot shows a 'Create New Role' dialog box. It has a title bar with the text 'Create New Role' and a close button (X). The dialog contains three input fields: 'Code:' with the value 'PROCESSMAKER_DESIGNER', 'Name:' with the value 'Process Designer', and 'Status:' with a dropdown menu set to 'Active'. At the bottom right are 'Save' and 'Cancel' buttons.

Code - PROCESSMAKER_DESIGNER

Name - Process Designer

Status - Active

The Code is the unique identifier, and we use All CAPS to be consistent with the other ProcessMaker roles. The name is a description of the role and we set the status to Active as we cannot assign an inactive role to users. Click the Save button to create the role.

Viewing and Editing Role Permissions

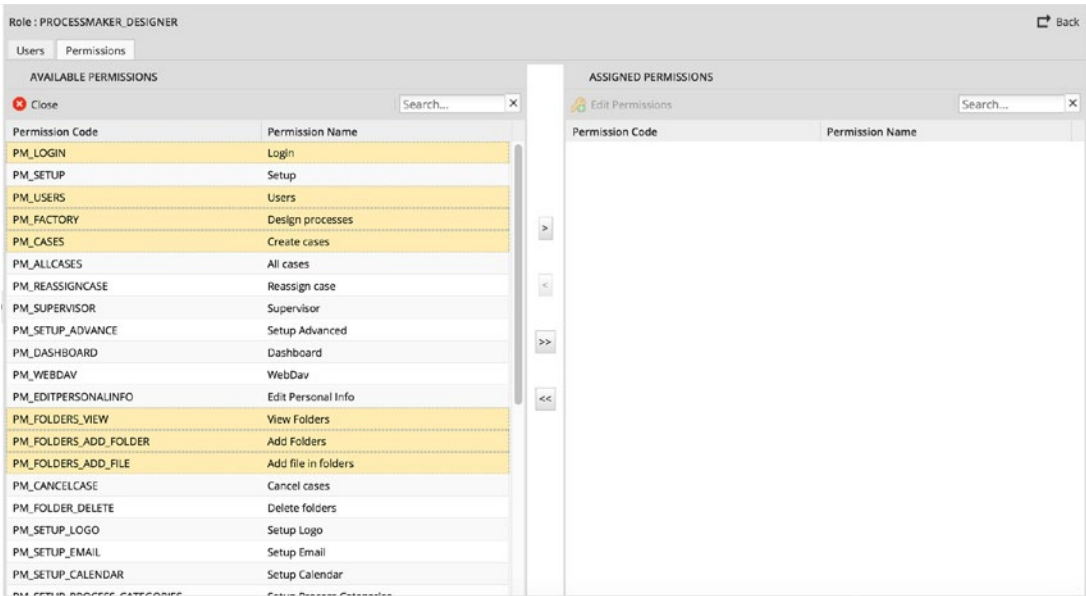
We now have a new role, but the role has no permissions assigned to it, which means that if we assign a user to this role as-is, they will not be able to do anything in ProcessMaker. To view the permissions assigned to a role, select the role and click

the Permissions button. Let us see what permissions the default roles have. Select the PROCESSMAKER_ADMIN role and click Permissions. You should see a long list of permissions. Click the Back button to return to the list of roles. Repeat the same process for the other two default roles. The PROCESSMAKER_OPERATOR role has permissions for logging in, running cases, and editing the user profile.

Now view the permissions of our newly created role PROCESSMAKER_DESIGNER. We can see that the role has no permissions. Let us proceed to define the permissions we want to give this role. To edit the permissions of a role, click the Edit Permissions button to display a list of Available Permissions and Assigned Permissions. Select the following permissions from the list of Available Permissions (you can select multiple permissions by holding down the Ctrl or Cmd key):

- PM_LOGIN
- PM_USERS
- PM_FACTORY
- PM_CASES
- PM_FOLDERS_VIEW
- PM_FOLDERS_ADD_FOLDER
- PM_FOLDERS_ADD_FILE

Click the > button to assign these permissions to the role as shown in the next image.



Click the Back button to return to the list of roles. Now that we have our roles set up, all that is left is to assign the roles to users.

Assigning Users to Roles

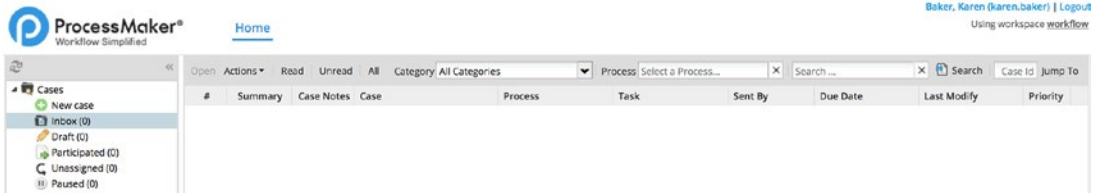
We can assign a user to a role either by editing the user and changing the role field or by adding the user to the list of assigned users for that role. The latter approach is appropriate for assigning multiple users at once. Let us assign the department managers to the Manager role. Select PROCESSMAKER_MANAGER from the list of roles and click the Users button. This displays the list of Assigned Users, which is currently empty. Click the Assign Users button to display the list of Available Users. Select Steve Bennett, Amy Alexander, Nicholas Williams, and Julia Smith from the list of Available Users and click the > button to assign them to the role. Click the Back button to return to the list of roles. We do not change the role of the Administrator, because the Administrator’s role already encompasses all the permissions of the Manager role.

Next, we use the alternative approach to assign Billy Green to the Process Designer role. Click the Users submenu option in the left pane to display the list of users. Select Billy Green from the list and click the Edit button. Under the Personal Information section of the Edit form, change the Role field from Operator to Process Designer and click the Save button. Return to the list of roles and we should now see that the Process Designer role has one active user.

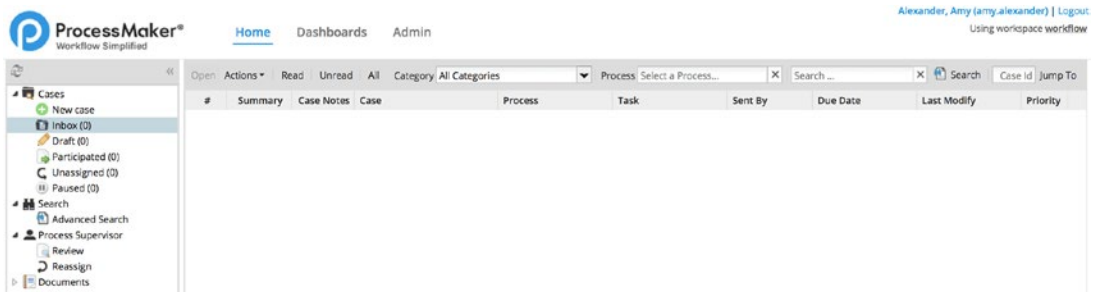
Code	Name	Status	Active Users	Create Date	Update Date
PROCESSMAKER_ADMIN	System Administrator	Active	1	2007-07-31 19:10:22	2007-08-03 12:24:36
PROCESSMAKER_OPERATOR	Operator	Active	5	2007-07-31 19:10:22	2007-08-03 12:24:36
PROCESSMAKER_MANAGER	Manager	Active	4	2010-03-29 09:14:15	2010-03-29 09:19:53
PROCESSMAKER_DESIGNER	Process Designer	Active	1	2016-09-21 06:24:35	2016-09-21 06:24:35

To see the effect of the different roles on the level of access the users have in ProcessMaker, open ProcessMaker in a different browser and log in as the following users to see the different views.

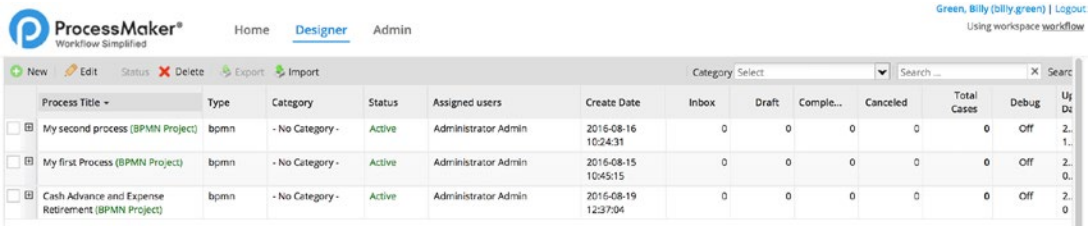
User	Password	Role
karen.baker	test123	Operator



User	Password	Role
amy.alexander	test123	Manager



User	Password	Role
billy.green	test123	Process Designer



Authentication Sources

ProcessMaker also allows us to import users from other authentication sources such as an LDAP (Lightweight Directory Access Protocol) or Active Directory user directories.

i A user directory is a system that stores user information like usernames, passwords, groups, and so on.

If you already have a user directory deployed in your organization, it would make sense to import the users from the directory into ProcessMaker and allow them use the same credentials when logging in, rather than creating them again in ProcessMaker and giving them a new set of credentials.

An extensive discussion of how to set up an LDAP or Active Directory user directory is outside the scope of this book. We will, however, illustrate this functionality by making use of a free online LDAP cloud directory provided by zFlex Integrator (<http://www.zflexldapadministrator.com/index.php/blog/82-free-online-ldap>) to illustrate how we can import users from an LDAP directory into ProcessMaker.

For more information on configuring external authentication sources for LDAP and Active Directory, see http://wiki.processmaker.com/3.0/External_Authentication.

Setting Up an Authentication Source

To set up a new authentication source, select Authentication Sources from the submenu option in the left pane under Users tab. The following view is displayed.



Click the New button. This displays the form shown here. Select LDAP as the provider and click the Continue button.

 The screenshot shows a form titled 'Available Authentication Sources'. It features a 'Provider' dropdown menu with the text 'Choose an option...' and a downward arrow. At the bottom right of the form, there are two buttons: 'Continue' and 'Cancel'.

In the Authentication Source Information form displayed, enter the following values and click the Save button to create the new authentication source.

Name: zFlex LDAP

Type: ldap

Server Address: www.zflexldap.com

Port: 389

Enabled TLS: no **Version:** 3

Base DN: ou=users,ou=developers,dc=zflexsoftware,dc=com

Anonymous: no

Username: cn=ro_admin,ou=sysadmins,dc=zflexsoftware,dc=com

Password: zflexpass

Identifier for an imported user: uid

Additional Filter:

Authentication Source Information

Name: zFlex LDAP

Type: ldap

Server Address: www.zflexldap.com

Port: 389

Enabled TLS: no

Version: 3

Base DN: ou=users,ou=developers,dc=zflexsoftware,dc=com

Anonymus: no

Username: cn=ro_admin,ou=sysadmins,dc=zfl

Password: *****

Identifier for an imported user: uid

Additional Filter:

Save Cancel

Importing Users from an Authentication Source

With our authentication now set up, let us proceed to import users. Since we are using a cloud service, your system will have to be connected to the Internet to import the users. Select the authentication source as shown here and click the Import Users button.



This displays a form to search for users in the directory as shown next.

Search for user

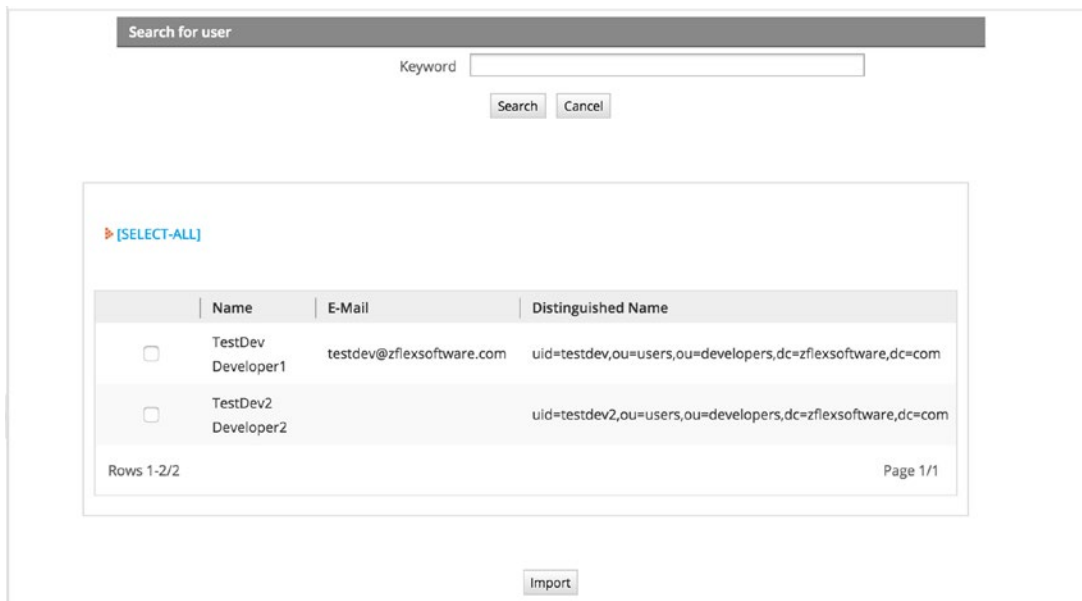
Keyword

Search Cancel

Import

When setting up the zFlex directory as an authentication source for this illustration, we limited the organizational unit to look up users from to the developers OU. Without

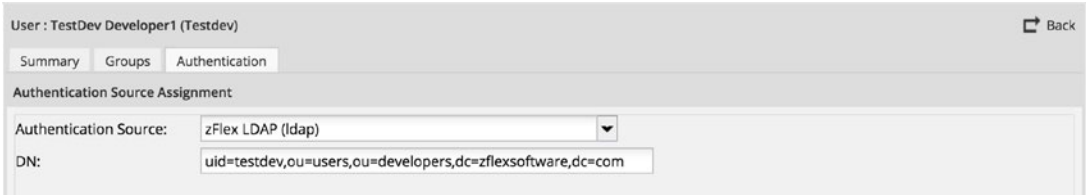
entering any value in the Keyword field, click the Search button. For a larger directory of users, you would use the Keyword field to filter the search results. The users in this organizational unit are displayed as shown next.



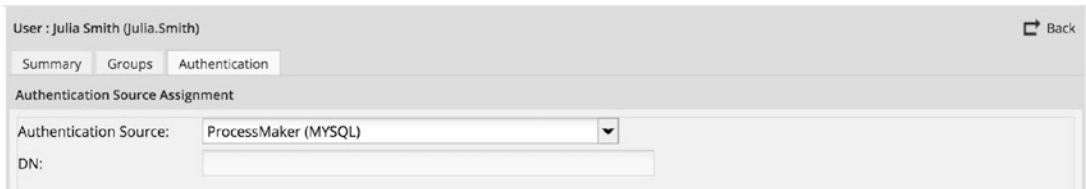
Click the [SELECT-ALL] link above the list to select both users, and click the Import button at the bottom of the list to import the users into ProcessMaker. A successful import should redirect you to a view of the list of users, similar to the image here. We can see the two users, TestDev Developer1 and TestDev2 Developer2, in the list.

User Name	Full Name	Status	Role	Last Login	# Cases	Due Date
admin	Admin, Administrator (admin)	Active	System Administrator	2016-09-21 05:14:55	0	2020-01-01 00:00:00
julia.smith	Smith, Julia (julia.smith)	Active	Manager	2016-09-20 12:57:57	0	2022-09-20 00:00:00
steve.bennett	Bennett, Steve (steve.bennett)	Active	Manager		0	2017-09-20 00:00:00
testdev2	Developer2, TestDev2 (testdev2)	Active	Operator		0	2018-09-21 00:00:00
testdev	Developer1, TestDev (testdev)	Active	Operator		0	2018-09-21 00:00:00
wanda.marshall	Marshall, Wanda (wanda.marshall)	Active	Operator		0	2017-09-20 00:00:00
amy.alexander	Alexander, Amy (amy.alexander)	Active	Manager	2016-09-21 07:24:22	0	2017-09-20 00:00:00
justin.sanchez	Sanchez, Justin (justin.sanchez)	Active	Operator		0	2017-09-20 00:00:00
nicholas.williams	Williams, Nicholas (nicholas.williams)	Active	Manager		0	2017-09-20 00:00:00
karen.baker	Baker, Karen (karen.baker)	Active	Operator	2016-09-21 07:22:34	0	2017-09-20 00:00:00
philip.price	Price, Philip (philip.price)	Active	Operator		0	2017-09-20 00:00:00
carlos.shaw	Shaw, Carlos (carlos.shaw)	Active	Operator		0	2017-09-20 00:00:00
billy.green	Green, Billy (billy.green)	Active	Process Designer	2016-09-21 07:26:08	0	2017-09-20 00:00:00

Select TestDev Developer1 from the list of users and click the Authentication button at the top of the list to show the Authentication source of the user. We see that the authentication source is set to zFlex LDAP, as shown next.



Click the Back button in the top-right corner. Now select one of our previous users (Julia Smith) and click the Authentication button. We see that the authentication source is set to ProcessMaker (MYSQL).



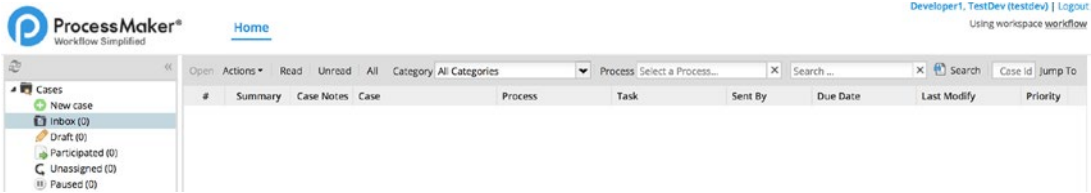
Click the Back button to return to the list of users. Select the Authentication Sources sub-menu option from the left pane. We see that we now have two active users.

Name	Provider	Server Address	Port	Enabled TLS	Active Users
zFlex LDAP	ldap	www.zflexldap.com	389	Disabled	2

Finally, let us try to log in with our newly imported users. Open ProcessMaker in another browser and log in with the following credentials.

User	Password
testdev	password
testdev2	password

The user should be logged in successfully. The default role for imported users is Operator and we can see the same view as expected for an Operator, shown next.

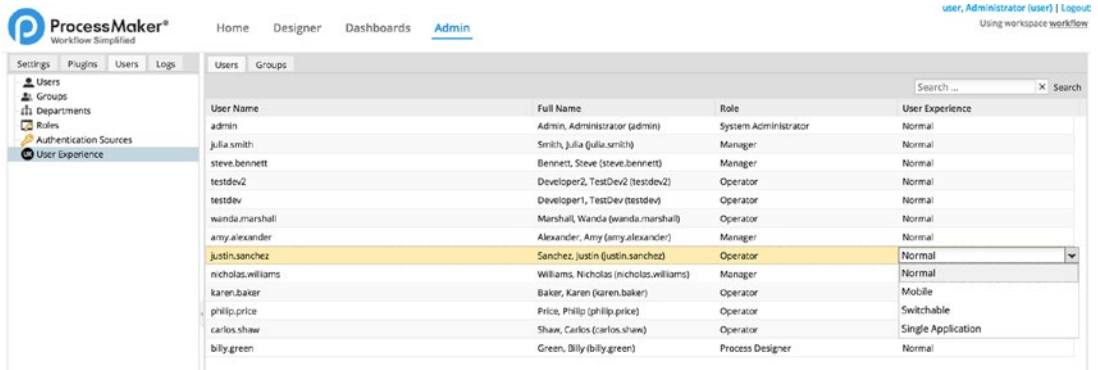


User Experience

To conclude our discussion of user administration in ProcessMaker, we look at the User Experience submenu option. This feature allows us to define the user experience (UX) to be displayed for the user when logged in to ProcessMaker. There are four options available: Normal, Mobile, Switchable, and Single Application.

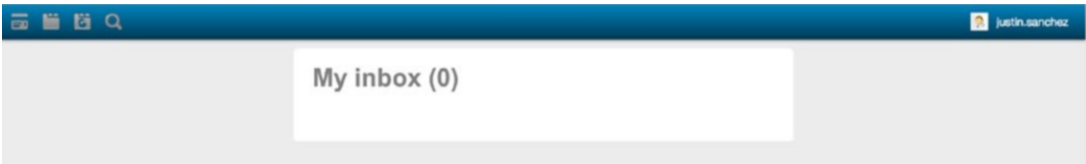
Changing the User Experience for a User

To change the User Experience for a user, click the User Experience submenu option from the left pane. Select the user from the list. We will use Justin Sanchez for our demonstration of this feature. Click on Normal in the User Experience column to display the list of options as shown here.



We are already familiar with the Normal option, so proceed to change the UX to Mobile. Now log in as Justin Sanchez (justin.sanchez) to ProcessMaker in another browser. The view displayed should look like the following image.

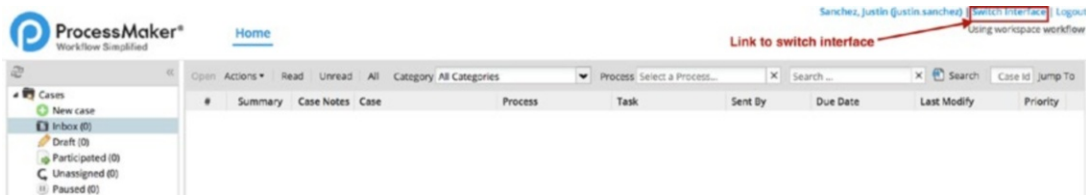
The mobile UX is optimized for users who will be accessing ProcessMaker primarily from their mobile devices. Click the username in the top-right corner to display the sign out link. Click it to log out.



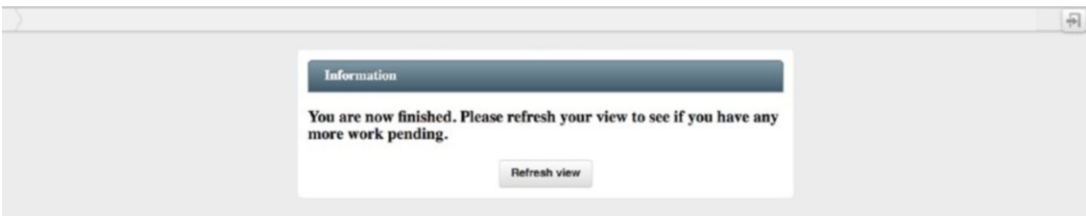
In the admin view, select Justin Sanchez again and change the UX to Switchable. As the name implies, this option allows the user to switch between the Normal UX and the Mobile UX. Log in again in the other browser as Justin Sanchez. The view displayed is the same as we saw for the mobile view, with an additional link under the signout link to Switch Interface as shown here.



Click the Switch Interface link, and the view should change to the Normal view with the addition of a new Switch Interface link, as shown in the next image. This allows you to toggle the view between the mobile and normal interfaces. Click the Logout button.



Return to the admin view and change the User Experience to Single Application. The single application UX displays only the next task assigned to the user. Log in as Justin Sanchez again in the other browser, and the view should look like the following image.



Because we have not yet created any cases and no case is assigned to Justin Sanchez currently, the view shows a message that the user has no work pending. Click the Exit icon in the top-right corner to log out Justin Sanchez.

Change the UX for Justin Sanchez back to Normal. Once we have started running cases, you can try toggling the User Experience for the users to see how they render with cases.

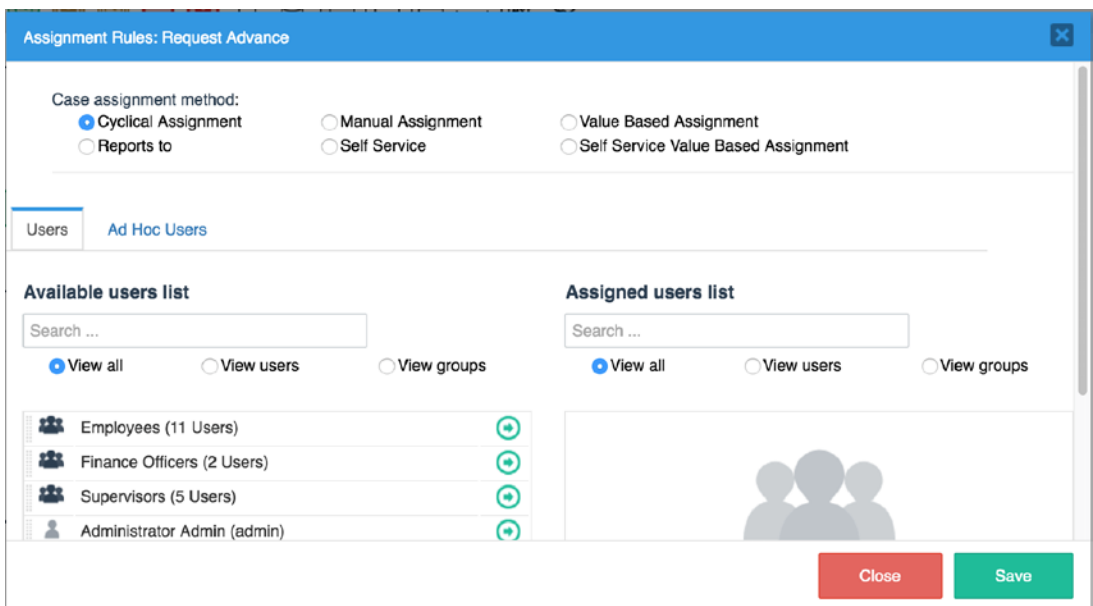
We have covered a lot in this chapter, and we now have users in the system ready to be assigned to the tasks in our Process. In the next chapter, we will set up assignment rules for assigning users and groups to tasks.

CHAPTER 11

Assigning Users to Tasks in a Process

Welcome back from the detour. We are now ready to assign the users we have created to the tasks in the Cash Advance part of the process. We will learn about the various ways users can be assigned to tasks and start a number of cases with different users to demonstrate how the different assignment rules work.

Log in as the admin user and head over to the Process List to open the Cash Advance and Expense Retirement process in the Process Designer. Right-click the Request Advance task and select Assignment Rules from the context menu to display the Assignment Rules modal shown here.



This is where we define how users will be assigned to tasks. The modal is divided into three sections: Case assignment method, Users, and Ad-Hoc Users.

- The case assignment method is used to select which type of assignment should be used for the task. The type of assignment determines how the specific user who will work on the task for a specific case will be selected.
- The Users section allows us to choose which users or group of users should be assigned to the task from the available list of users and the
- The Ad-hoc users section is used for defining additional users or groups to whom a task can be reassigned.

Assignment rules for a task are a matter of *how* the user should be selected and *which* users can be selected. The Users tab is where we define which users are assigned to the task, and the Case Assignment method is where we define how the specific user will be selected from the list of assigned users to work on a specific case. Before we look at the intricacies of the various case assignment methods, let us quickly see how to assign users to the task.

Assigning Users and Groups

The recommended best practice is to assign groups to tasks rather than individual users. This is because when a process is exported and imported into another instance of ProcessMaker, the user assignments are discarded, while the group assignments are preserved. For example, since we want every user to be able to request a Cash Advance, we would assign the Employees group to the Request Advance task instead of assigning each individual user. If we decide to export the process from our system to a live server, we only need to assign users to the Employees group (the group will be created if it does not exist on the server); we don't have to go and edit the assignment rules for the process to assign the users again.

Even if only one user will be assigned to a task, it is still recommended to create a group and assign that group to the task. The groups can be reused in other processes, and as you add more processes to ProcessMaker, you can easily manage which users are assigned to tasks by adding or removing them from the groups instead of editing the assignment rules for each task in every process.

In the Users tab of the Assignment Rules modal, there is a list of Available Users on the left and a list of Assigned Users on the right. To assign a user or group, simply drag and drop the user from the Available Users list to Assigned Users. You can also click the green arrow icon beside a user or group to move it to the Assigned Users list. Drag and drop the Employees group to the Assigned Users list. To remove a user or group from the Assigned Users list, simply drag it back to the Available Users list or click the red close icon beside it.

The tab also has a search box, which can be very useful for filtering the list in situations where we have a large number of users and groups in ProcessMaker. You can also filter the list using the radio options below the search box. The View All option is selected by default and displays all users and groups. The View Users option filters the list to show only users, and View Group filters the list to show only the groups.

Ensure that the Employees group is assigned to the task and click the Save button. Right-click the Approve Advance task and select the Assignment Rules option. Assign the Supervisors group to the task and click Save. Finally, assign the Finance Officers group to the Disburse Advance task.

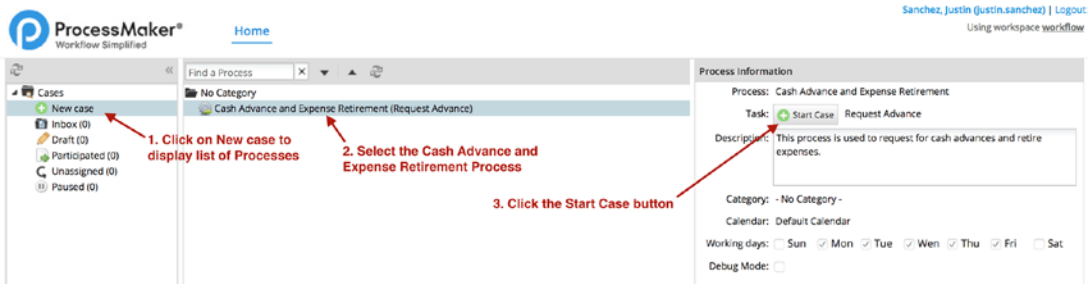
Now that we have defined which users can be assigned to the tasks, we will explore how the specific user for each case will be selected from the list of assigned users. It is important to note that for the first task (the starting task) in a process, the assignment method of the task is not considered when the case is being created, and the user starting the new case is automatically assigned to the task for that case. If later in the process, the case is routed back to the starting task, the case assignment method defined for the task is then used to select the user to assign the case to.

All users assigned to a starting task of a process will be able to initiate a new case for that process. Because we have assigned the Employees group to the starting task (Request Advance) of our Cash Advance and Expense Retirement Process, all users in that group will be able to create a new Cash Advance and Expense Retirement case. But if the case is rejected by the Supervisor, it might be routed back to a different user and not the user who started the case, as expected. We also need to be able to determine the right supervisor to route the case to among the five users in the Supervisors group when the employee submits the request. The case assignment method helps us address this.

Cyclical Assignment

This is the default assignment method in ProcessMaker. It assigns users from the Assigned Users list to a task in a round-robin (sequential) pattern. If we start a new Cash Advance and Expense Retirement case and submit it to a Supervisor, the case will be routed to one of the users in the Supervisors group. The next case created will be routed to the next supervisor, and so on until all supervisors have been assigned a case and then the sixth case will be assigned to the first supervisor.

Let us illustrate this initiating a number of Cash Advance and Expense Retirement cases and submitting them to a supervisor. Log in to ProcessMaker in another browser as Justin Sanchez. In the Cases menu on the left, click the New Case button. This displays the Cash Advance and Expense Retirement process in the list of processes. Select it and click the Start Case button on the right to start a new case. You can also double-click the process to start a new case.



The new case is created, and the Cash Advance Request Form we added as a step for the Request Advance task is displayed as shown next. Fill in the required fields and click the Submit Request button at the bottom of the form. The form could use a few improvements, such as prefilling information such as the Employee’s name, request date, and department. We will learn how to enhance the form to prefill these details after we learn about triggers. Right now, we just want to demonstrate the Cyclical assignment method.

Steps Information Actions Case Notes

Case #: 1 Case #: 1 Title: #1

Next Step

Cash Advance Requisition

Request Details

Request Date * 2016-09-22 Employee Name * Justin Sanchez

Department * Sales Requested Amount * 1000

Reason for Expenses * To test cyclical assignment.

Disbursement Details (Finance)

Amount Advanced Date Advanced

Comments

Enter your comment here. Add Disbursement

The form is submitted and the routing screen is displayed as shown next. The next task is Approve Advance as expected, and the Employee assigned to work on the task is Julia Smith (the user might be different on your system). Click the Continue button to submit the request.

Steps Information Actions Case Notes

Case #: 1

Previous Step

Assign Task

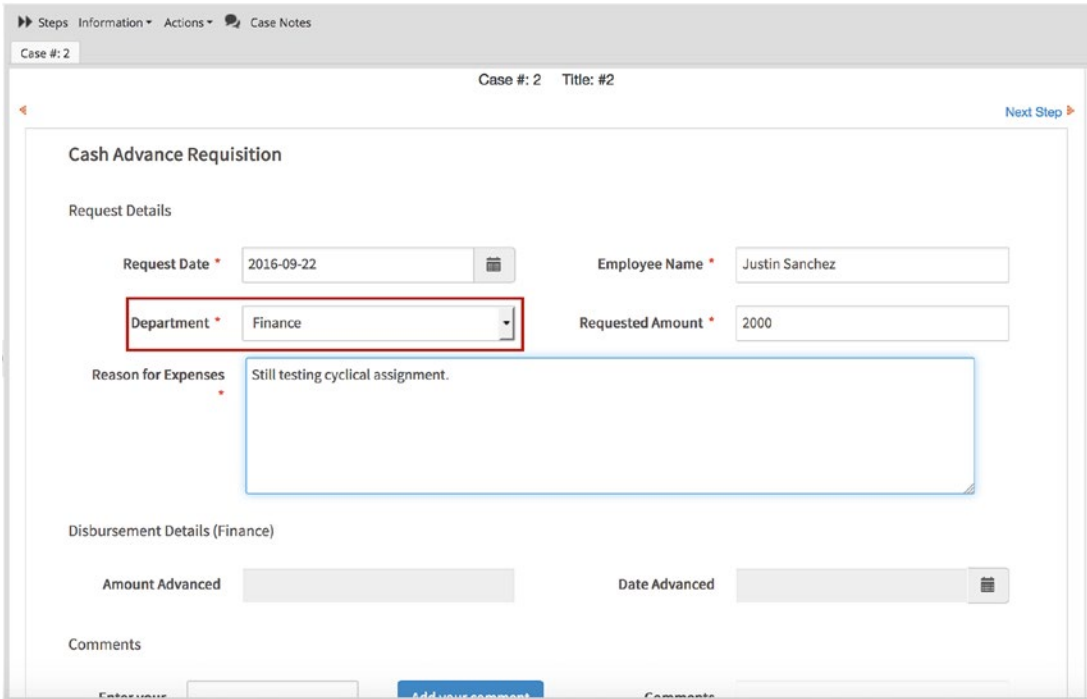
Next Task: Approve Advance

Employee: Smith, Julia

Continue

Start another Cash Advance and Expense Retirement case using different values from the first case. For the second case, set the department to Finance. We will use this to illustrate the conditional display of forms in a task as discussed earlier, when we set the condition property, for the forms we added to the Approve Advance task. You will recall

that we defined a condition to display a different form if the department is Finance. With the required fields filled in, click the Submit Request button to submit the form.



The routing screen is displayed as shown next, and this time, we see that a different employee (Steve Bennett in my case) is being assigned as the Supervisor to approve the request.



Go ahead and create four additional requests with different request details (we have five supervisors, so the sixth request should be assigned to the same user as the first request) to complete the cycle. When you have created the six cases, click the Participated menu under Cases in the left sidebar to display the list of all the cases as shown next.

#	Summary	Case Notes	Case	Process	Task	Current User	Last Modify	Status
6			#6	Cash Advance and Expense Retirement	Approve Advance	Smith, Julia (julia.smith)	2016-09-22 10:17:18	To do
5			#5	Cash Advance and Expense Retirement	Approve Advance	Admin, Administrator (admin)	2016-09-22 10:16:47	To do
4			#4	Cash Advance and Expense Retirement	Approve Advance	Williams, Nicholas (nicholas.williams)	2016-09-22 10:16:20	To do
3			#3	Cash Advance and Expense Retirement	Approve Advance	Alexander, Army (army.alexander)	2016-09-22 10:15:41	To do
2			#2	Cash Advance and Expense Retirement	Approve Advance	Bennett, Steve (steve.bennett)	2016-09-22 10:09:30	To do
1			#1	Cash Advance and Expense Retirement	Approve Advance	Smith, Julia (julia.smith)	2016-09-22 09:57:38	To do

We can see that the first and sixth cases have been assigned to the same user, completing the cycle. This is not, however, an ideal method for selecting a supervisor to approve a request. The Cyclical assignment method is better suited for situations where we have a pool of users assigned to work on a task and we want to distribute cases evenly. A good example is a help desk with a group of technicians. The cyclical assignment method will distribute incoming cases sequentially, ensuring that the workload is evenly distributed.

For our Cash Advance and Expense Retirement process, we want employees to be able at least to choose the specific supervisor to submit their request to. The next assignment method allows us to do this.

Manual Assignment

The manual assignment method allows the user completing a previous task to choose the user to assign the task to from the list of assigned users for that task. Continuing with our example, changing the case assignment method for the Approve Advance task to manual assignment will enable the employee submitting the request to choose which supervisor to send the request to from the list of supervisors.

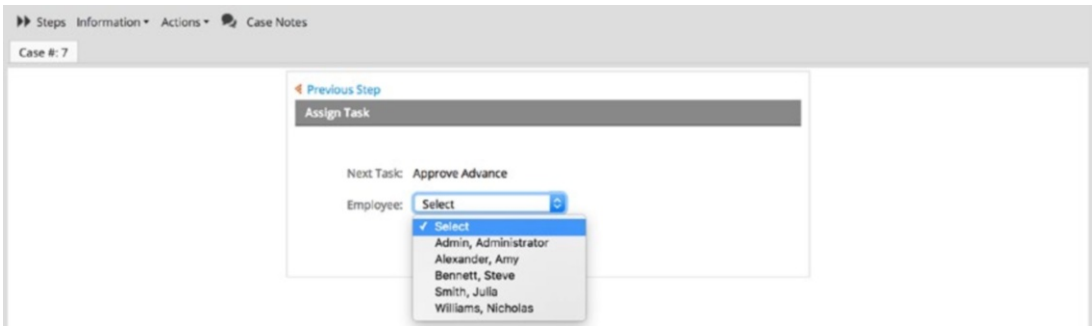
In the browser you are logged in as the admin user, go ahead and change the case assignment method for the Approve Advance task to manual assignment. Right-click the task and select Assignment Rules. Select the Manual Assignment radio option and click Save.

Assignment Rules: Approve Advance

Case assignment method:

- Cyclical Assignment
- Manual Assignment
- Value Based Assignment
- Reports to
- Self Service
- Self Service Value Based Assignment

Now log in to ProcessMaker as Justin Sanchez and start a new Cash Advance and Expense Retirement case. Fill in the required fields and submit the form. The routing screen displayed is quite different from what we saw with cyclical assignment. Instead of seeing a user automatically assigned as supervisor, we are shown a dropdown list of supervisors from which we can select the one to assign the case, as shown in the following image. Justin Sanchez is in the Sales department, and his manager is Julia Smith. Select Julia Smith from the list of Supervisors and click the Continue button.



The request is submitted, and if you go to the Participated menu, you will see that the case has been routed to Julia Smith.

#	Summary	Case Notes	Case	Process	Task	Current User	Last Modify	Status
7			#7	Cash Advance and Expense Retirement	Approve Advance	Smith, Julia (julia.smith)	2016-09-22 10:43:43	To do
6			#6	Cash Advance and Expense Retirement	Approve Advance	Smith, Julia (julia.smith)	2016-09-22 10:17:18	To do
5			#5	Cash Advance and Expense Retirement	Approve Advance	Admin, Administrator (admin)	2016-09-22 10:16:47	To do

We can see that the manual assignment method provides us with better control in choosing how to route a request. There are, however, scenarios where we want the user to select not only the user to assign the next task, but also other tasks after that. The next assignment method—Value Based Assignment—helps us address this scenario.

Comparing Cyclical, Manual, and Value-Based Assignment

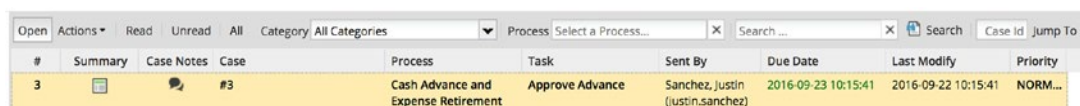
Before we look at the value-based assignment method in detail, you need to see a limitation of the cyclical and manual assignment methods and how value based assignment can help address it.

Assuming that in MSB Corp. a request had to be approved by more than one manager, we want the user to be able to choose which managers to send the request to. But the routing screen for the manual assignment method allows them only to select the user for the next task. We can resolve this by storing the UID (unique identifier) of the users we want to route the case to in variables, which ProcessMaker can then check to determine which user to send the request to.

In our earlier illustration, Justin Sanchez has sent his request to managers he does not report to. If these managers decide to return the case, the case will be routed to just about any employee because the Request Advance task uses cyclical assignment and as we have seen, the cases for the task will be assigned sequentially. If we change the assignment rule for the Advance Request task to manual assignment, the supervisor will be able to select Justin Sanchez from the list of employees before returning the case. While this is a better alternative to just returning the case blindly, it could be quite cumbersome in situations where there are many users. Also, the returning supervisor can make a mistake and select another employee.

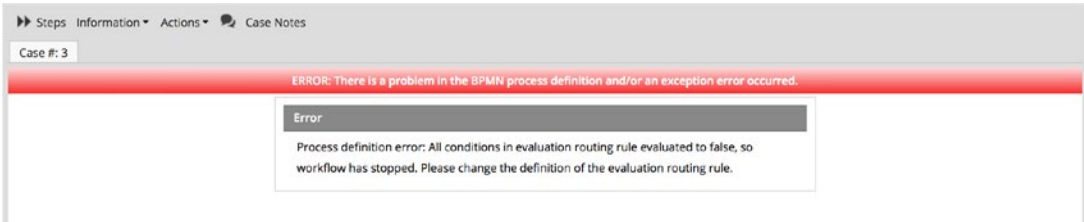
The ideal solution would be to automatically return the case to the user that initiated the request. To do this, we store the UID of the user starting the case in a variable, change the assignment method of the Request Advance task to Value Based Assignment, and set the variable containing the UID as the variable for the value based assignment field. Let us demonstrate the limitation of the current setup and implement the value-based assignment approach.

Log in to ProcessMaker as Amy Alexander in another browser. You should see the request sent by Justin Sanchez in the inbox as shown next. Select the case and click the Open button to open the request.



#	Summary	Case Notes	Case	Process	Task	Sent By	Due Date	Last Modify	Priority
3			#3	Cash Advance and Expense Retirement	Approve Advance	Sanchez, Justin (justin.sanchez)	2016-09-23 10:15:41	2016-09-22 10:15:41	NORM...

The Cash Advance Approval form is displayed with the request details shown as read-only. Scroll to the bottom of the form and click the Reject Request button to send the request back to Justin Sanchez. Click OK in the confirm dialog that asks “Are you sure you want to REJECT?” and BOOM! An error message is displayed.



ProcessMaker is telling us that it cannot determine which way to route the request. This is because the Approve Advance task uses an Exclusive Gateway and we have two routes (return to Request Advance or proceed to Disburse Advance) defined for the gateway in our Process Map, but we have not defined the condition for determining which route to take.

We need to define the condition for the routes in the gateway. We will explore routing in ProcessMaker in more detail later, when we illustrate how to use the different gateway types.

For now, log in as the admin user and in the Cash Advance and Retirement Process Map, right-click the Exclusive Gateway beside the Approve Advance task and select the Properties option. The Routing Rule screen is displayed, as shown next. Define the conditions for the next task in the routing rule as shown here and click the Save button. Also save the process.

Next Task	Condition
Disburse Advance	@@is_approved == '1'
Request Advance	@@is_approved == '0'

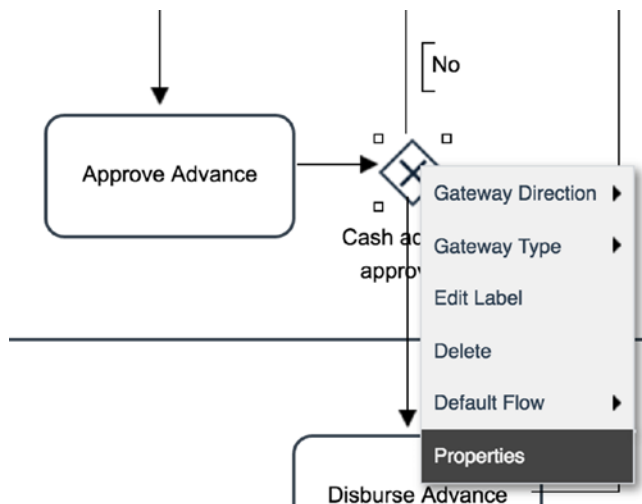
Routing Rule - EXCLUSIVE

Add Routing Rule

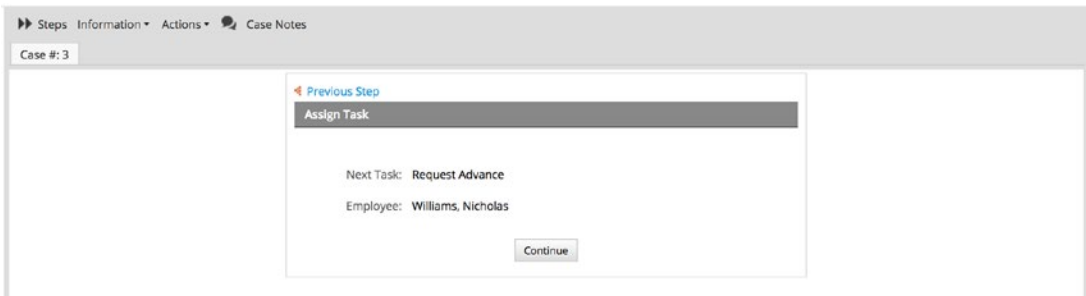
Next Task	Condition		
Disburse Advance	@@is_approved == '1'	@@	Delete
Request Advance	@@is_approved == '0'	@@	Delete

Cancel Save

We have told the routing engine that if the value stored in the `is_approved` variable is `TRUE`, the case should be routed to the `Disburse Advance` task and if the value is `FALSE`, the case should be routed to the `Request Advance` task. The `is_approved` variable is of type `Boolean` and when set to `TRUE`, the string representation is `1` and if set to `FALSE`, the string representation is `0`. You will recall that we created this variable when we learned how to add approval functionality to our forms, and the value of the variable is set when the supervisor approves or rejects the request.



With our routing rule sorted out, return to the browser where you are logged in as Amy Alexander and click the inbox menu. Open the case again and reject the request. The routing screen is displayed, and we can see that the request is being returned to the Request Advance task as expected, but the assigned user should not be Justin Sanchez. (Note that because the Request Advance task is using cyclical assignment, the next user on your screen could be Justin Sanchez. If this is the case, go ahead and click Continue, and then log in as another manager and reject the request in that manager’s inbox. The assigned user will be a different user from Justin Sanchez.)



Click the Continue button to send the request back (to Nicholas Williams in this case) and log out Amy Alexander. We have clearly seen that the cyclical assignment method is inappropriate for the Request Advance task.

In the Process Map, right-click the Request Advance task and change its assignment method to manual assignment. Now log in as the other managers (Administrator, Julia Smith, Steve Bennett, and Nicholas Williams), and go to the inbox and reject the requests sent from Justin Sanchez. The Task column of the cases will be Approve Advance and the Sent By column will be Sanchez, Justin (justin.sanchez). The routing screen should display the list of employees. Select Justin Sanchez from the list and click the Continue button.

While returning the cases, you will observe that the form displayed for Case #2 is different from the others. Instead of the Approve Request and Reject Request buttons, we have radio options. This is because we defined the condition property of the dynaform to show a different form if the department is Finance, when we added dynaforms to the step earlier. Select No and click the Submit Request button to reject the request.

Steps Information Actions Case Notes

Case #: 2

Disbursement Details (Finance)

Amount Advanced Date Advanced

Comments

Enter your comment Comments

Signoff/Approval

Requested By Date and Time

Approved By Date and Time

Disbursed By Date and Time

I agree to account for this advance within ten working days of the date advanced as defined in the preceding section, either with adequate receipts, cash or a check for the balance made payable to MSB Corporation. I understand that my failure to account for advanced funds in full within sixty days will result in a Payroll deduction for the balance due. By clicking the submit button below, I agree to allow MSB Corporation to make any such deductions from my pay.

Approve this request? Yes No

When done, log in as Justin Sanchez and you should see all the returned requests in the inbox. We will now demonstrate how the same situation would be handled using value-based assignment for the Request Advance task.

Value-Based Assignment

Log in as the admin user and open the Cash Advance and Expense Retirement process in the Process Designer. Before we can use the value based assignment, we have to create a variable to store the UID of the user that starts the task and a trigger to help us fetch the value and store it in the variable. We will learn more about triggers later, but for now, just follow along.

Click the Create (+) icon beside Variables in the Main Toolbox (process objects) in the Process Map and, in the modal that appears, set the variable name to **case_owner**. Leave other fields unchanged and click Save.

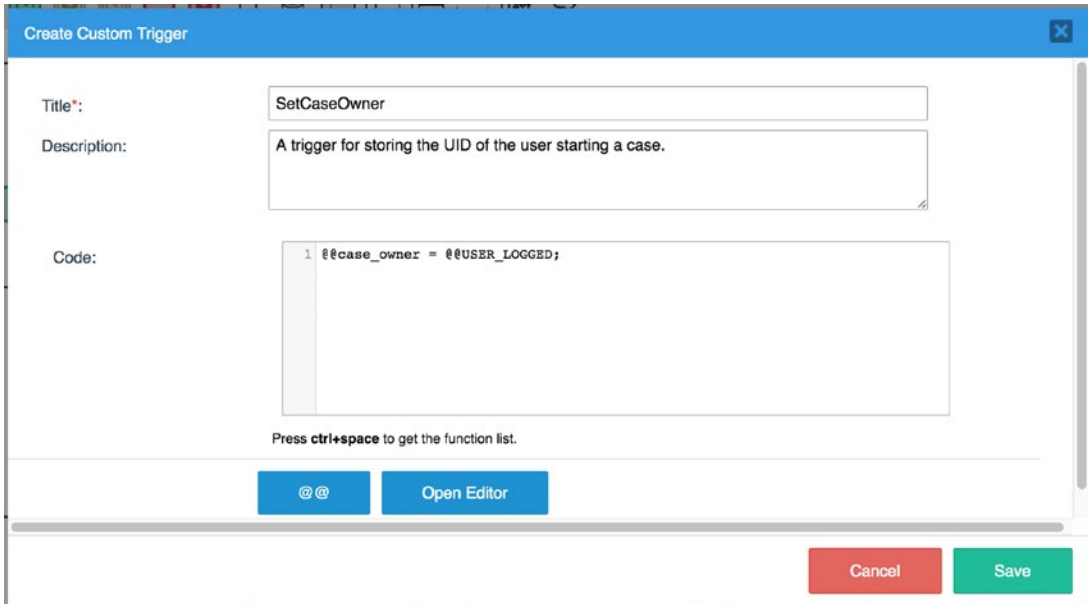
Next, click the Create (+) icon beside Triggers in the Main Toolbox again. The Triggers modal is displayed. Click the Create button in the top right of the screen, and a

Create Custom Trigger modal should be displayed as shown next. Set the fields as shown here and click the Save button. Close the modal to display the list of triggers.

Title: SetCaseOwner

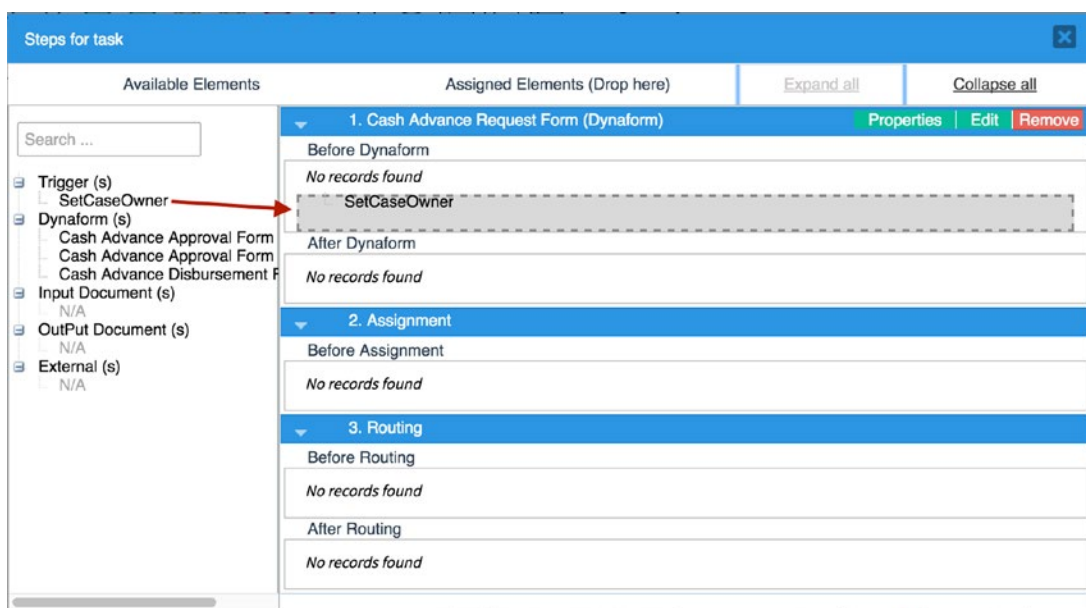
Description: A trigger for storing the UID of the user starting a case.

Code: @@case_owner = @@USER_LOGGED;

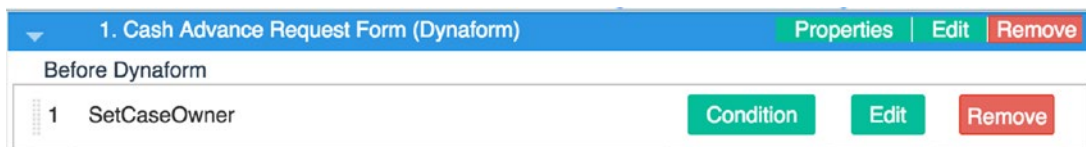


The code in our trigger is very basic. It copies the value of the system variable USER_LOGGED, which contains the UID of the currently logged-in user and assigns it to the case_owner variable we just created. Next we need to set up the trigger to execute when a new case is created. To do this, we add it as a step to the starting task of the process.

Right-click the Request Advance task in the Process Map and select Steps in the context menu. This displays the Steps modal for the Request Advance task, and we can see the Cash Advance Request Form we added as a step earlier. We can also see that our newly created trigger is displayed in the list of Available Elements. Click the Expand All link to show the Before and After sections of the steps. You can also use the right arrow beside the Cash Advance Request Form step to expand the view. Drag and drop the SetCaseOwner trigger to the Before Dynaform section of the Cash Advance Request Form step as shown in the following image. This sets up the trigger to be executed before the first step of the process, Cash Advance Request Form, is loaded.

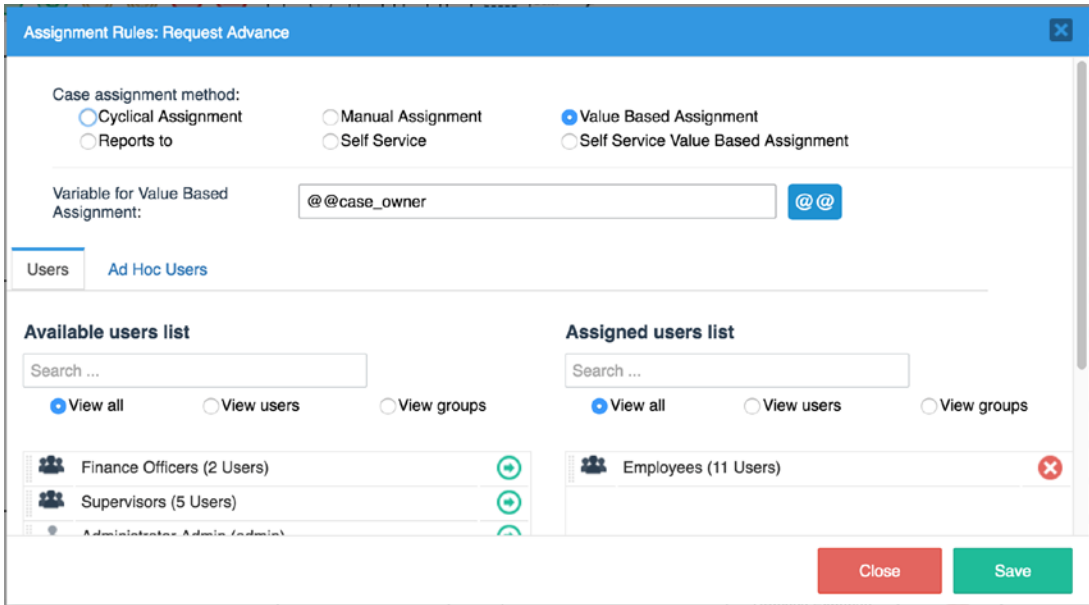


The trigger when successfully assigned displays as shown next. We can use the Condition button to define a condition to be evaluated before executing the trigger. Ideally, we would want to execute the SetCaseOwner trigger only once, when the case is started, and not every time the Cash Advance Request Form is opened in the Request Advance task. We will see how to do this in the next chapter when we discuss triggers, but for now, we will let it execute repeatedly so that we can set the case_owner variable for the cases already in Justin Sanchez's inbox. The Edit button displays the code for the trigger in an editor and the Remove button can be used to remove the trigger. Close the Steps window.



Now that we have our missing pieces, let us proceed to change the assignment rule of the Request Advance task to Value Based Assignment. Right-click the Request Advance task and select Assignment Rules. In the Assignment Rules modal displayed, change the case assignment to Value Based Assignment. This causes a new field, Variable for Value Based Assignment, to be displayed. Delete the @@SYS_NEXT_USER_TO_BE_ASSIGNED

value in the field and replace it with **@@case_owner**, the variable we just created as shown in the following image. Save the changes by clicking the Save button.



The Request Advance task is now configured to use the Value Based Assignment method. A supervisor should now be able to reject the request without having to manually select the employee to return it to, and the request will be correctly assigned to the employee who sent the request. To demonstrate this, log in as Justin Sanchez in another browser. Open three of the cases in the inbox and submit the requests. Select Julia Smith as the supervisor in the routing screen and click Continue to submit the requests.

Log out and log in as Julia Smith. The requests should be displayed in the inbox. Open each one and reject the requests. The routing screen should show that each request is now being assigned correctly to Justin Sanchez, the case owner, without the supervisor having to manually select who to return the case to or ProcessMaker cyclically assigning the case to users in the Employees group. Click the Continue button and return all requests back to Justin Sanchez.



Just as we have helped Julia avoid the hassle of manually selecting the user to return a case to, we can also help Justin avoid the hassle of choosing the supervisor to send a request to. The next assignment method helps us do just that.

Reports To

This assignment method automatically assigns a task to the manager of the department to which the user assigned the previous task belongs. In our example, using a Reports To assignment method for the Approve Advance task will automatically assign the task to the manager of the user assigned to the Request Advance task (the previous task) of the case.

If the previous user is the manager of his department, the task will be assigned to the manager of the parent department if the user's department is a subdepartment. If the department is not a subdepartment, or the parent department does not have a manager, the task will be assigned to the same user.

It is therefore important to ensure that the organizational hierarchy is properly configured in the Departments setup in ProcessMaker before using this assignment method. Let us see how it works with a couple of examples. Right-click the Approve Advance task, select Assignment Rules and change the case assignment method to Reports to. Save your changes.

Log in as Justin Sanchez in another browser and open one of the cases in the inbox. Submit the request and you will now observe that in the routing screen, we no longer see a drop-down list to select a user, but the request is being correctly routed to Justin Sanchez's supervisor—Julia Smith. Click the Continue button and log out.

Department : Sales

ASSIGNED USERS

Assign Users Set Manager No set Manager

Full Name

Sanchez, Justin (justin.sanchez)

Smith, Julia (julia.smith) [Manager]

Steps Information Actions Case Notes

Case #: 1

Previous Step

Assign Task

Next Task: Approve Advance

Employee: Smith, julia

Continue

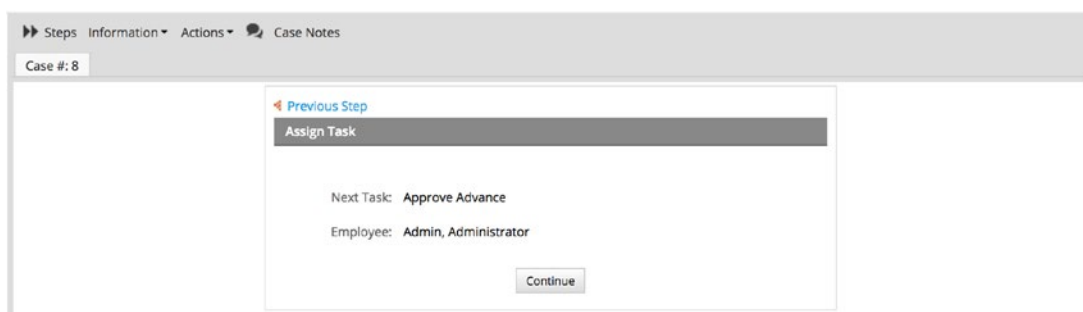
Next, let us see a scenario where the user belongs to a subdepartment (IT Support) that does not have a manager.

Departments

New New Sub-Department Edit Delete Users

Department Name	Status	Manager	Users
Information Technology	Active	admin	1
Application Development	Active		1
IT Support	Active		1
Administration	Active	steve.bennett	2
Finance	Active	amy.alexander	2
Human Resources	Active	nicholas.williams	2
Sales	Active	julia.smith	2

Log in as Karen Baker (karen.baker) and start a new Cash Advance and Expense Retirement case. Fill in the required fields and submit the request. In the routing screen, we can see that the Approve Advance task is being assigned to Administrator, who is the manager of the parent department, Information Technology. Click the Continue button to submit the request and log out.



Finally, let us see one more example of a scenario where the user is a manager of a department without a parent department. Log in as the manager of the Administration Department, Steve Bennett (steve.bennett), and start a new Cash Advance and Expense Retirement case. Fill in the required fields and submit the request. In the routing screen, we can see that the Approve Advance task is being assigned to the same user—Steve Bennett—making the request. Click the Continue button to submit the request and log out.



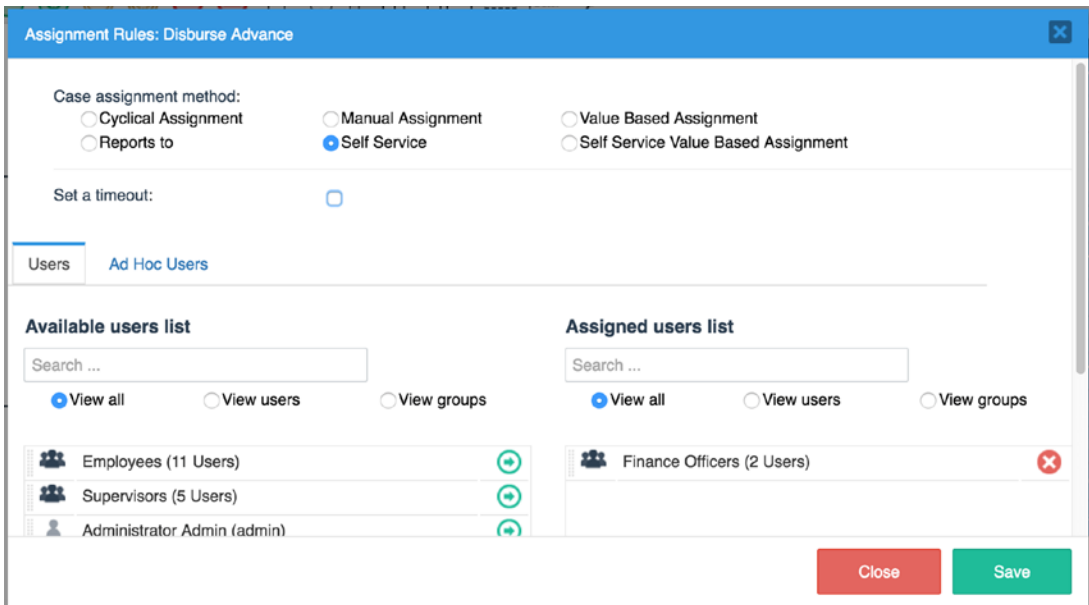
That concludes our discussion of the Reports To assignment method.

Self Service

The Self-Service assignment method allows a user from the pool of assigned users for a task to claim a case to work on it. Rather than the system choosing a user to assign the case to, the case is displayed to all the users in the assigned users list of the task, and one of the users can claim the case to work on it, thereby assigning it to herself or himself. Once a user claims the case, other users will no longer be able to see or claim the case.

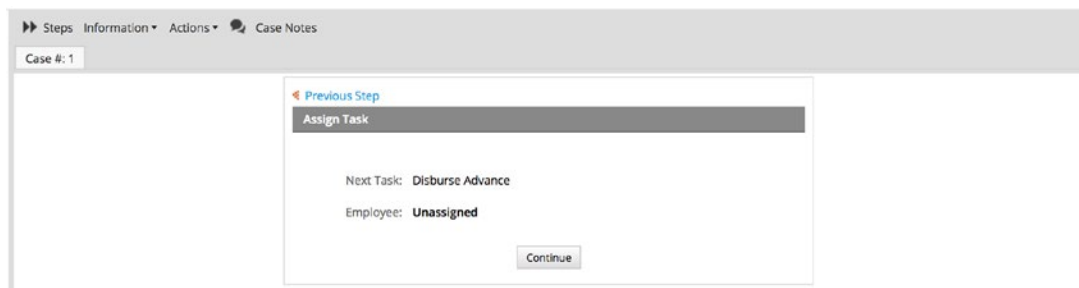
This assignment method is ideal for situations where any member of a group of users can work on a case, for example a help desk with a group of technicians. However, unlike the cyclical assignment, new cases are not distributed among the group cyclically, but placed in a general pool where the technicians can pick cases to work on. This can be useful for days when a member of the group is unavailable and cannot work on cases as it avoids cases being assigned to the user, but rather placed in the general pool where the technicians available can work on it.

We illustrate how this assignment method works by applying it to the Disburse Advance task. We want any of the finance officers to be able to work on the Disburse Advance task. Right-click the Disburse Advance task, select Assignment Rules, and change the case assignment method to Self Service. A “Set a timeout” field is displayed. Checking this box allows us to set a time limit for the case to be claimed by one of the assigned users and define a trigger that will be executed if the case is not claimed when the time elapses. Leave the field unchecked for now. We will return to it later in the book to send a reminder to the Finance manager when a case has not been claimed within a specified period. Save your changes.



Now log in as Julia Smith and open the Cash Advance request sent by Justin Sanchez. This time around, click the Approve Request button to approve the request it and route it to the Disburse Advance task. The routing screen displayed shows that the task is being

routed to the Disburse Advance task, but the Employee is unassigned. Click the Continue button and log out.

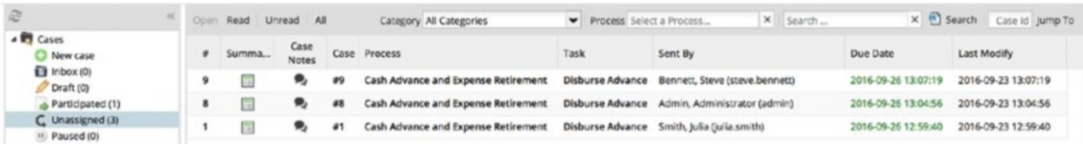


Unlike the other assignment methods we have seen so far, there is no employee assigned for us and we are not choosing an employee to assign the case to. Rather we are sending it to all employees assigned to the task (the Finance Officers group in this instance), and any of them can claim the case to work on it. Before we log in as the Finance Officers, let us approve two more requests so we can have more requests in the pool. Log in as the administrator and click Home in the main menu to display the inbox. Open the case sent by Karen Baker and approve the request. It should route to the Disburse Advance task with Employee unassigned. Log out and log in as Steve Bennett to approve the request in the inbox. We should now have three requests waiting for our Finance officers to disburse.

Log in as the first finance officer, Amy Alexander. Surprise! The cases are not in the inbox even though they have been routed to the finance officer group which Amy belongs. In the left pane, we can however see that the Unassigned link has (3) suffixed to it. The number in the brackets indicates the number of cases in that link. The inbox only shows cases to which a user is currently assigned.



Click the Unassigned link under Cases in the left pane to display the list of unassigned cases as shown next.

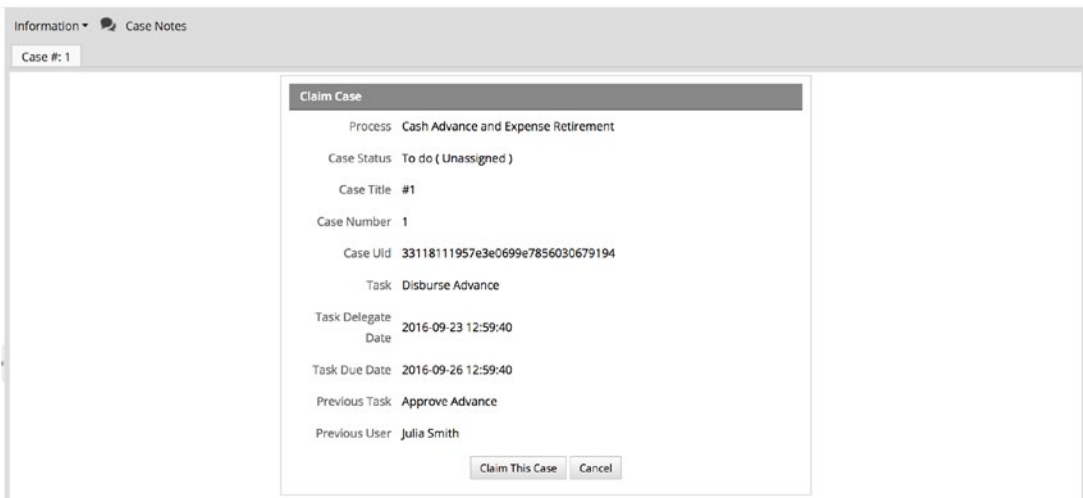


The screenshot shows a web application interface for case management. On the left, there is a navigation pane with 'Cases' selected, showing sub-items: 'New case', 'Inbox (0)', 'Draft (0)', 'Participated (1)', 'Unassigned (3)', and 'Paused (0)'. The main area displays a table of unassigned cases. The table has columns for '#', 'Summa...', 'Case Notes', 'Case', 'Process', 'Task', 'Sent By', 'Due Date', and 'Last Modify'. Three rows are visible, all with a status of 'Unassigned'.

#	Summa...	Case Notes	Case	Process	Task	Sent By	Due Date	Last Modify
9			#9	Cash Advance and Expense Retirement	Disburse Advance	Bennett, Steve (steve.bennett)	2016-09-26 13:07:19	2016-09-23 13:07:19
8			#8	Cash Advance and Expense Retirement	Disburse Advance	Admin, Administrator (admin)	2016-09-26 13:04:56	2016-09-23 13:04:56
1			#1	Cash Advance and Expense Retirement	Disburse Advance	Smith, Julia (julia.smith)	2016-09-26 12:59:40	2016-09-23 12:59:40

Log out and log in as the second finance officer, Philip Price. You will see that the inbox is also empty, but there are three cases in the Unassigned list. Click the Unassigned link and you will see that they are the same three cases we saw when we logged in as Amy Alexander.

Let us claim one of the cases to work on. Select the case sent by Julia Smith from the list to highlight it and click the Open button. (You could also right-click it and select Open from the context menu or double-click the case to open it.) Instead of opening a form as we have seen before, we are shown a Claim Case screen like the following.



The screenshot shows a 'Claim Case' dialog box. At the top, it says 'Information Case Notes' and 'Case #: 1'. The dialog contains the following information:

- Process: Cash Advance and Expense Retirement
- Case Status: To do (Unassigned)
- Case Title: #1
- Case Number: 1
- Case Uid: 33118111957e3e0699e7856030679194
- Task: Disburse Advance
- Task Delegate Date: 2016-09-23 12:59:40
- Task Due Date: 2016-09-26 12:59:40
- Previous Task: Approve Advance
- Previous User: Julia Smith

At the bottom of the dialog, there are two buttons: 'Claim This Case' and 'Cancel'.

To claim the case, click the Claim This Case button. Clicking the Cancel button closes this screen and returns to the list of unassigned cases. Go ahead and claim the case. The Cash Advance Disbursement Form is now opened and we can see the details of the request. We will not be submitting the request yet as we have not set up steps or assignment rules for the next task (Report Expense) in the process and submitting the form now will throw an error. Click the Inbox link on the left pane and the request we just

claimed should be listed in the inbox. You will also notice that the Unassigned list is now reduced to two cases.

Log out and log in again as Amy Alexander. Go to the Unassigned list and you will see that the list is also reduced to two cases, and the case claimed by Philip Alexander is no longer in the list. Claim the case sent by Administrator to reduce the unassigned cases to one. Do not submit it after claiming it. Return to the Inbox and you should see the case listed there. Go ahead and log out.

Self-Service Value-Based Assignment

The last assignment method we will look at is a hybrid between the Value-Based and Self-Service assignment methods. It allows us to store the UID of a specific user or group of users that can claim the case from the unassigned pool. Let us assume that we have two groups of Finance Officers in MSB Corp.—one group for requests sent from the sales team, and a separate group for other departments' requests. We still want to retain the Self-Service method of assigning the cases, but we also want only a specific group or user to be able to *claim* the case, that is, assign it to themselves.

To illustrate this assignment method, we need to do some ground work. First we create two new Finance groups and assign users to the groups. Log in as the administrator and go to Admin. Select the Users tab and the Groups submenu. Create two groups and assign users to the group as shown here:

Group Name	Users
Finance – Sales	Amy Alexander
Finance – General	Philip Price

Return to the designer and open the Cash Advance and Expense Retirement process. We will create a variable to store the UID of the group to use, just as we did when exploring Value-Based assignment. Click the Create (+) icon beside Variables in the main menu and create a variable named `finance_group` with a string variable type. Next we create a trigger that will update this variable with the right group UID based on the department selected in the request form.

Click the Create (+) icon beside Triggers in the main menu and create a trigger with the following properties and save it:

Title: SetFinanceGroup

Description: Sets the finance group based on department selected

Code:

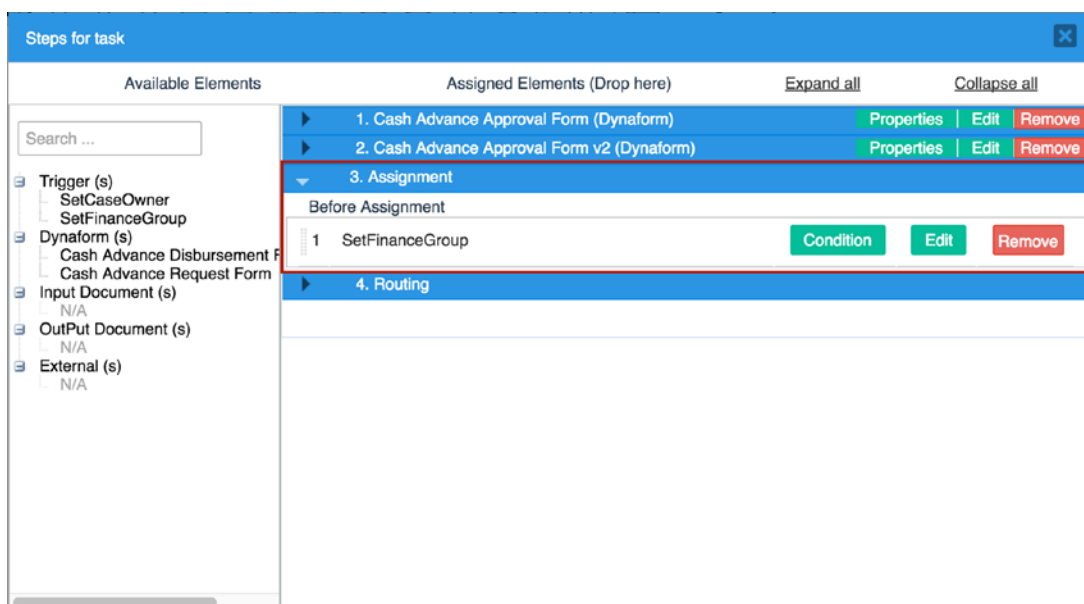
```
if (@@department == "Sales") {
    @@finance_group = PMFGetGroupUID("Finance - Sales");
}
else {
    @@finance_group = PMFGetGroupUID("Finance - General");
}
```

This code checks the value stored in the department variable. If it is Sales, we use the built-in ProcessMaker function `PMFGetGroupUID` to get the UID of the new Finance – Sales group we just created and store it in the `finance_group` variable. Otherwise, we store the group UID of the Finance – General group. We will explore the ProcessMaker built-in functions when we explore triggers in the next chapter.

Finally, we need to execute the trigger in a step that occurs before the case is routed to the Disburse Advance task and after the department has been selected. Can you think of an appropriate step to put this trigger? There are a number of options available to us, such as these:

- After the Cash Advance Request Form step in the Request Advance task
- Before or after the Cash Advance Approval Form step in the Approve Advance task
- Before the Assignment step in the Approve Advance task

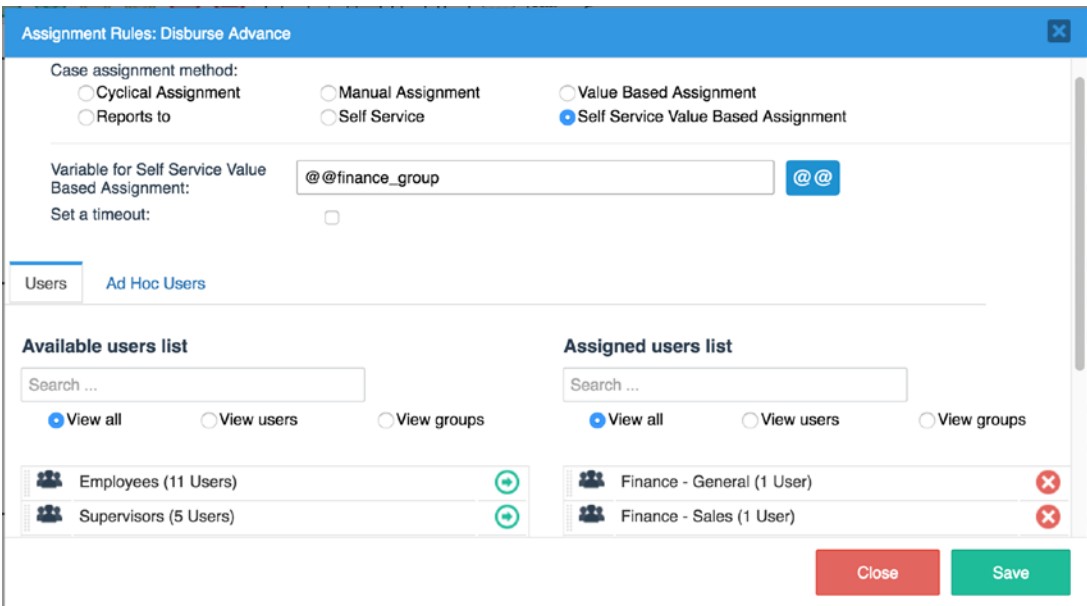
The goal is to ensure that the department variable has been set and the assignment rule has not yet been evaluated. We will place it before the Assignment step in the Approve Advance task. Right-click the Approve Advance task and select Steps option. Drag and drop the SetFinanceGroup trigger to the Before Assignment step as shown in the following image and close the modal.



That concludes the preparatory ground work required for the Self Service Value Based assignment method. Right-click the Disburse Advance task and change the Assignment Rule to Self Service Value Based Assignment. A new field, Variable for Self Service Value Based Assignment is displayed. Set the value to @@finance_group. Leave the Set a Timeout field unchecked. We also need to add the newly created groups to the Assigned Users list in the Users section. Drag the Finance – General and Finance – Sales groups from the Available Users list to the Assigned Users list. Click the Save button. BOOM! An error message is displayed, reading “Bad Request. Current Activity has cases and cannot be deleted.”

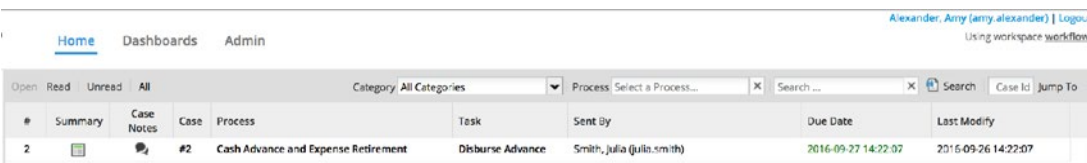
What is going on? Remember that we still have an unclaimed case in the Disburse Advance task, and changing the assignment rule could leave us unable to assign the case if the case does not match the requirements for the new rule. In this example, the unclaimed case in the Disburse Advance task does not have a value set for the finance_group variable we just created, and ProcessMaker will not be able to assign the case.

All hope is not lost. Simply log in as Amy Alexander or Philip Price and claim the case. Return to the Process Designer and change the assignment rule again to Self Service Value Based Assignment, ensure that the new groups (Finance – General and Finance – Sales) are assigned, and save the changes. The Assignment rule should now be correctly saved.



Let us see the effect of our changes. Log out as Amy Alexander in the other browser and log in as Justin Sanchez. Open one of the cases in the inbox and ensure the department is set to Sales. Note the case number (Case #2) and submit the request. Open another case and make sure the department is *not* Sales. Note the case number (Case #4) and submit the request. Log out and log in as Julia Smith and Approve the two requests. The routing rule should still show the next employee is Unassigned.

Now we have two unassigned cases in the Disburse Advance task. Log out as Julia Smith and log in as the first finance officer Amy Alexander, who is assigned to the Finance – Sales group. Go to the Unassigned menu under Cases, and there should be only one case (Case #2) in the list – —the case with the department set to Sales. Even though we have two unassigned cases at the Disburse Advance task, only the users in the group stored in the `finance_group` variable will be able to see the case in their unassigned and be able to claim the case. Go ahead and claim the case but do not submit it yet. Log out as Amy Alexander.



Log in as Philip Price, and you should see the case with the department not set to Sales (Case #4) in the unassigned cases list. Go ahead and claim the case but do not submit it yet.

#	Summary	Case Notes	Case	Process	Task	Sent By	Due Date	Last Modify
4			#4	Cash Advance and Expense Retirement	Disburse Advance	Smith, Julia (julia.smith)	2016-09-27 14:25:09	2016-09-26 14:25:09

We can now see that the case is being assigned using a combination of Self Service and Value Based assignment as expected.

Now that we are familiar with the different case assignment methods available to us in ProcessMaker, we can apply that knowledge to address the different case assignment requirements we encounter when automating business processes. We have also been able to route a case as expected in the Cash Advance part of the Cash Advance and Expense Retirement process. Before we proceed to build the second half and complete the process, let us take a closer look at triggers to understand them better as we will be using more complex ones in the second half of the process. We will also explore input and output documents, as we will also be using them in the other half of the process.

CHAPTER 12

Triggers

As robust and flexible as ProcessMaker is, it cannot do everything we will require in building a business process right out of the box as we have seen in some of the scenarios in the earlier chapters of this guide. Fortunately, it provides us a powerful tool for adding the necessary custom behavior, called *triggers*. We saw in the previous chapter how triggers can be helpful in adding custom logic to the business process in the Self-Service Value-Based Assignment method example. We were able to fetch and assign a specific group based on input filled in by the user.

Triggers are basically snippets of PHP code that we use to perform complex calculations and add custom functionality to processes such as fetching data from a remote web service or evaluating complex business logic. The triggers have access to the case variables and can read and manipulate their values. They can also access the system variables created by ProcessMaker for every case such as the UID of the case, the current task, and so on.

Because it is basically a PHP script, we can also create regular PHP variables in a trigger and import PHP libraries, including access to the Gulliver Framework used to build ProcessMaker. Gulliver is a high-quality open source framework for developing web applications and web services with PHP, based on the MVC pattern. We should note, however, that the PHP variables created in a trigger are temporary and are discarded once the trigger execution completes. To retain a value after the execution of a trigger, you should store it in a process variable.

Once a trigger has been created, it must be added to a step in the process to be executed. A trigger can be reused several times in a step and in multiple tasks in the same process. When setting up the Self Service Value Based assignment, we identified three possible locations where we could place the SetFinanceGroup trigger for the same result. It is important to understand the lifecycle of a step in order to time the execution of a trigger properly.

Trigger Timing

For a trigger to be executed, it has to be added to a step, and this can be in any of the following positions in the task lifecycle.

Before a Step

A step here refers to a dynaform, input document, output document, or external step assigned to a task. Placing a trigger before a step is useful when the trigger will be used to set values of variables that will be used in the step. For example, in our Cash Advance and Expense Retirement case, we would like to set the Employee Name and Request Date when a new case is initiated, without the employee having to fill out the information on the form. To achieve that, we will need a trigger to get the values, assign them to the appropriate variables, and then set the trigger to execute before the dynaform.

After a Step

Placing a trigger after a step causes the trigger to be executed when the step is completed. Triggers placed here have access to the values (if any) set by the just concluded step and can use them. For example, suppose we had two dynaform steps, Form A and Form B, and we wanted to show form B only if a condition is met. However, we need to fetch a value to be evaluated in the condition from an external source based on input entered in Form A. We could use a trigger placed after Form A to fetch the value and set it so that it can be available to be evaluated as a condition for displaying Form B.

Before Assignment

Before assignment is a good place to put a trigger that sets the values of variables that are used in determining the next user to assign a case. We saw an example of this when we used a trigger for the Self-Service Value-Based assignment in the previous chapter.

Before Routing

Triggers placed before routing will be executed before the case is routed to the next task. This is therefore a good location for placing triggers that will set variables that will be evaluated to determine what task or sub-process the case will be routed to next.

After Routing

This is the point after which the next user and task for the case have been determined and saved to the database. Any trigger requiring this information should be placed after routing. For example, if we wanted to send an email to the next assigned user that includes the title of the task the user has been assigned to, we could use a trigger placed here to get the information about the user and next task and send the mail.

Case and System Variables

Earlier in the book, we learned about variables and how to create them. These variables are case variables and are used to store data for a case such as values entered in Dynaform controls or data fetched from a database or web service. We can read and manipulate the values stored in case variables in a trigger. We have seen examples of this in the triggers we created in earlier chapters. Just as PHP variables are identified by a \$ prefix (PHP variables are written as `$variable_name`), case variables are identified with a prefix, which can be any of the following:

`@@`

`@%`

`@#`

`@?`

`@$`

`@=`

Case Variable Prefixes

The prefixes determine the data type in which the value stored in the variable will be parsed before it is rendered.

Prefix	Type	Sample	Notes
@@	String	@@name	This parses the value stored in the variable as a string.
@%	Integer	@%count	This parses the value stored in the variable as an integer. If the value stored in the variable is a decimal number, it will be rounded up. For example, if you have a variable named <code>amount</code> containing a value <code>24.50</code> , using it as <code>@%amount</code> in a trigger will return <code>25</code> .
@#	Float	@#amount	This parses the value stored in the variable as a float. This will be the appropriate prefix to use in the <code>amount</code> variable example earlier.
@?	URL	@?params	This parses the value as a URL encoded string using PHP's <code>urlencode</code> function. For example when passing parameters in a URL, spaces should be rendered with a <code>+</code> sign. If we had a variable <code>params</code> , containing a URL parameter with value <code>"user=justin sanchez"</code> , using it as <code>@?params</code> would return <code>"user%3Djustin+sanchez"</code> .
@\$	SQL Query	@\$query	This parses the value stored in the variable as an SQL string escaping any quotes. If we had a variable <code>title</code> with value <code>Gulliver's Travels</code> , using it as <code>@\$</code> would return <code>Gulliver\'s Travels</code> .
@=	Original type	@=grid	This returns the value stored in the variable as the original type. It is ideal for objects and arrays.

System Variables

In addition to the case variables, ProcessMaker provides us with a default set of variables whose values are automatically set for every case and hold information about the system, process, case, and its status. The following system variables are available in ProcessMaker.

System Variable	Description
@@SYS_LANG	Stores the current system language in two letter ISO 639-1 code, which by default is “en” (English).
@@SYS_SKIN	Stores the current system skin (the theme), which by default is “neoclassic” in ProcessMaker 3.0.x.
@@SYS_SYS	Stores the current workspace name, which by default is “workflow”.
@@PROCESS	Stores the UID of the current process.
@@TASK	Stores the UID of the current task.
@@APPLICATION	Stores the UID of the current case.
@@APP_NUMBER	Stores the current case number. Available Version: From 3.0.1.8 on.
@@USER_LOGGED	Stores the UID of the current user.
@@USR_USERNAME	Stores the username of the current user.
@%INDEX	Stores the delegation index number, which is a positive integer which counts tasks in a process, starting from 1. If multiple tasks are operating in parallel, then each task will have a different index number.
@@PIN	Stores a four-character PIN, which can be used to access information about the case without being a registered user when accessing the ProcessMaker Case Tracker URL.
@@ ERROR	If a ProcessMaker error occurs, this system variable will be created, containing the error message. Note that this system variable only exists after a ProcessMaker exception occurs. It will not be created by syntax errors in PHP or JavaScript or by errors which ProcessMaker doesn’t know how to handle.

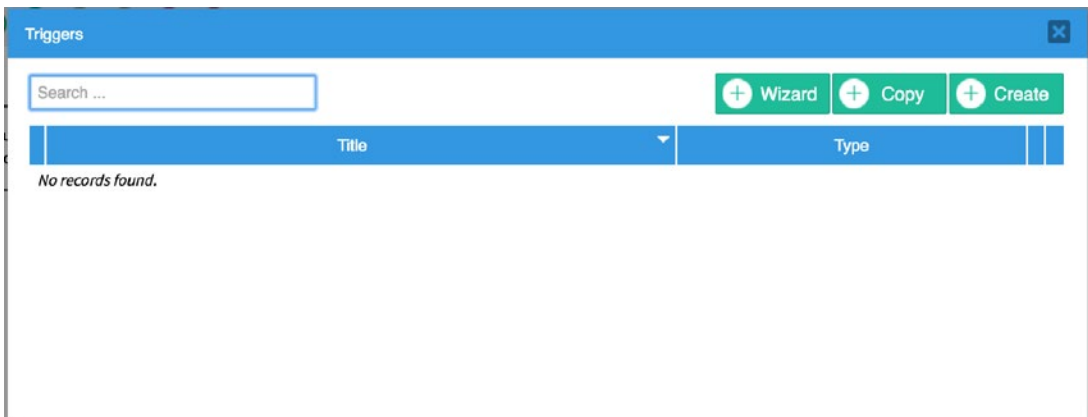
Variable Selector

You might have noticed buttons labeled @@ in earlier chapters of this book and wondered what they are. These are *variable selectors* that allow you select a variable to be used in the specified context. They come in very handy when we have a sizable amount of variables in our process and help us find the variable we need. The feature also helps avoid typographical errors when entering a variable name in a trigger or other use cases. We will illustrate how to use the variable selector by creating a few triggers and exploring them.

Creating Triggers

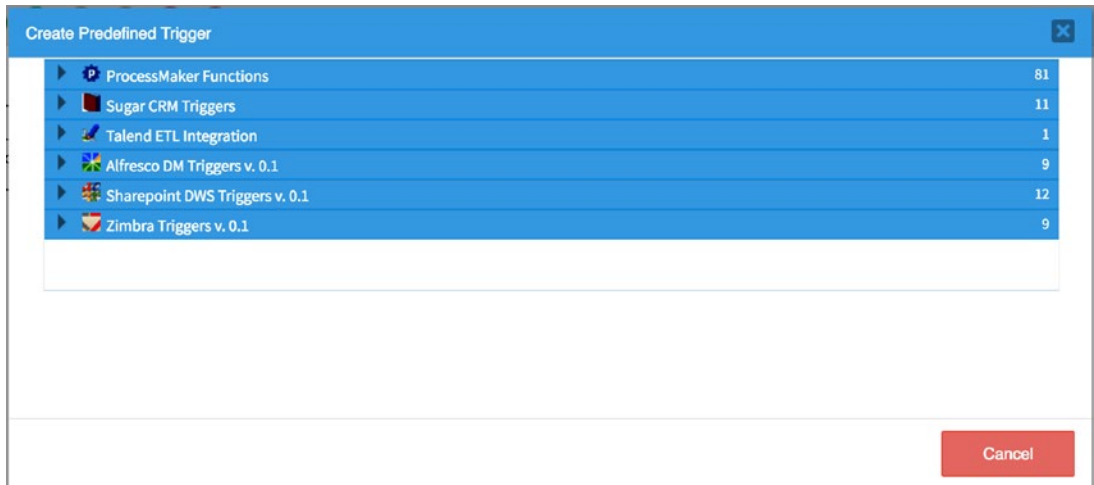
There are two types of triggers in ProcessMaker, *predefined triggers* and *custom triggers*. Predefined triggers are a set of triggers already defined in ProcessMaker and are created using a wizard that allows us select the type of trigger, and fill in the required parameters, and set a variable to hold the result. The wizard then automatically generates the code for the trigger. Custom triggers, on the other hand, are triggers for which we write the code ourselves and are used for functionality not provided out of the box or by predefined triggers.

Let us return to the playground we used when we learned about dynaform controls. Log in as Administrator to ProcessMaker and from the list of processes, open “My first process.” Click on Triggers in the Main Toolbox (Process Objects) to display the Triggers screen as shown next. We currently have no triggers. Let us create a predefined trigger.



Predefined Triggers

Click the Wizard button in the top-right of the Triggers screen to display the different categories of predefined triggers as shown here.



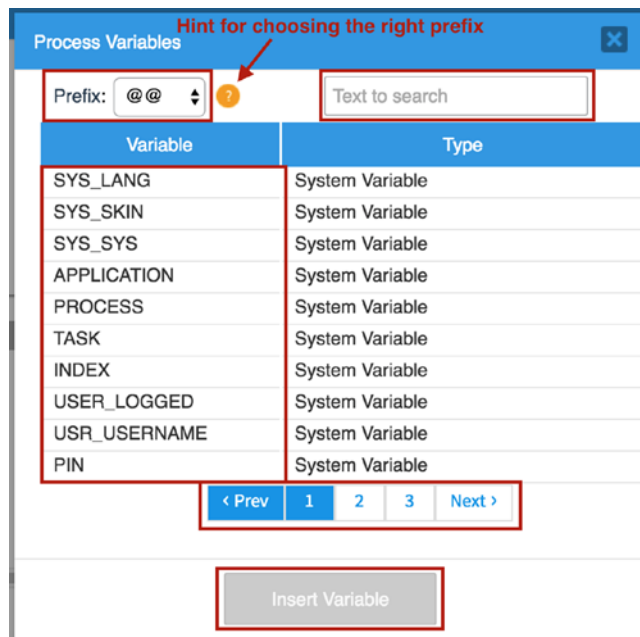
We can see the number of triggers available under each category on the right. Click the ProcessMaker Functions category to expand it and display the list of triggers. This category of triggers consists of PHP functions predefined in ProcessMaker to help with common tasks. For a detailed list of the functions and what they do, see http://wiki.processmaker.com/3.1/ProcessMaker_Functions. For the purpose of this demonstration, we will keep it simple. Select PMF Get Group UID from the list and the Create ProcessMaker Function screen should be displayed.

Fill in the following fields:

Title: GetGroupUID

Description: Sample predefined trigger that gets the UID of a Group and stores it in a variable.

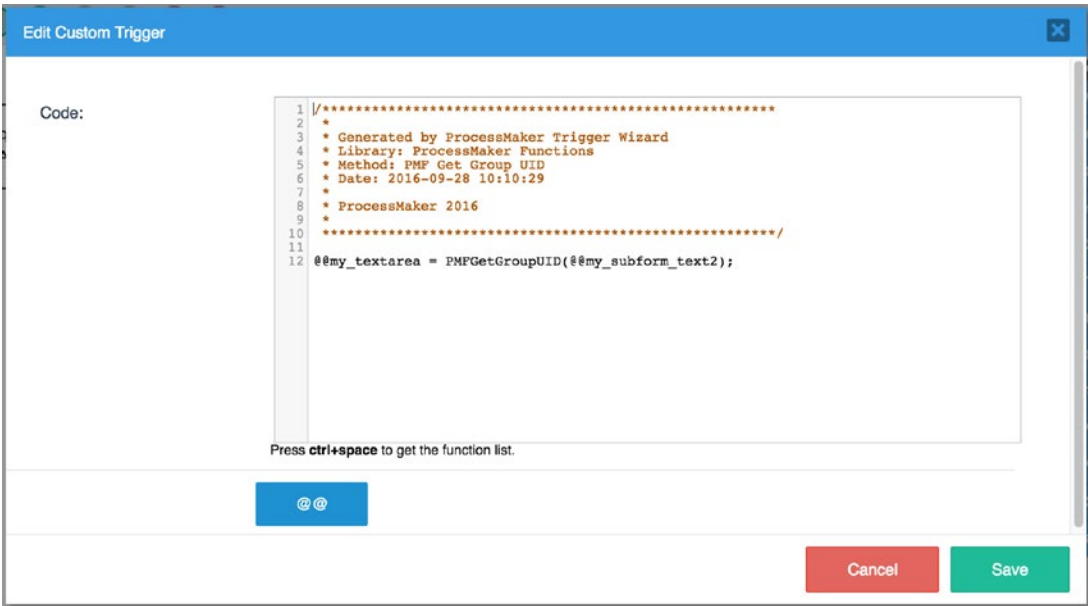
This predefined trigger requires the name of the group as a parameter. We can see a variable selector beside the **Name group** and **Variable to hold return value** fields, so instead of typing, let us set it using the variable selector. Click the @@ button to display the selector, which shows a list of the system and case variables.



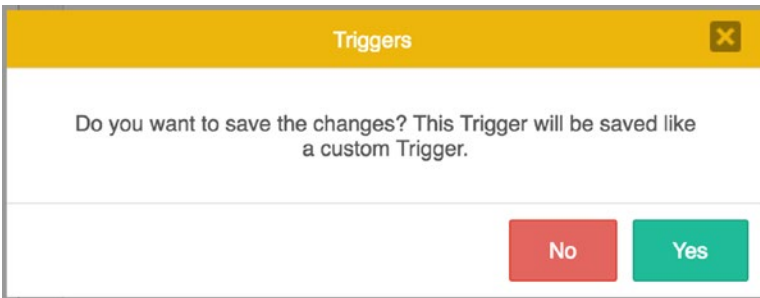
The selector has a search box for filtering the list of variables. A Prefix dropdown lets you choose the type of prefix to add to the variable name. Beside the Prefix dropdown is a hint about what each prefix does. Once the desired variable is found, click it to select it and click the Insert Variable button at the bottom (it will be enabled when a variable is selected) to insert it into the field.

For the Name Group field, select the `my_subform_text2` variable with a `@@` prefix and for the “Variable to hold return value” field select `my_textarea` with a `@@` prefix. Click the Save button to create the trigger and close the wizard. The trigger is now displayed in the list of triggers.

Click the Edit button beside the trigger to view the code that was generated by ProcessMaker. We see a similar form to the one filled when creating the trigger, but with an additional Edit Source Code button in the bottom-right corner. Click this button, and you should see the code generated by ProcessMaker.



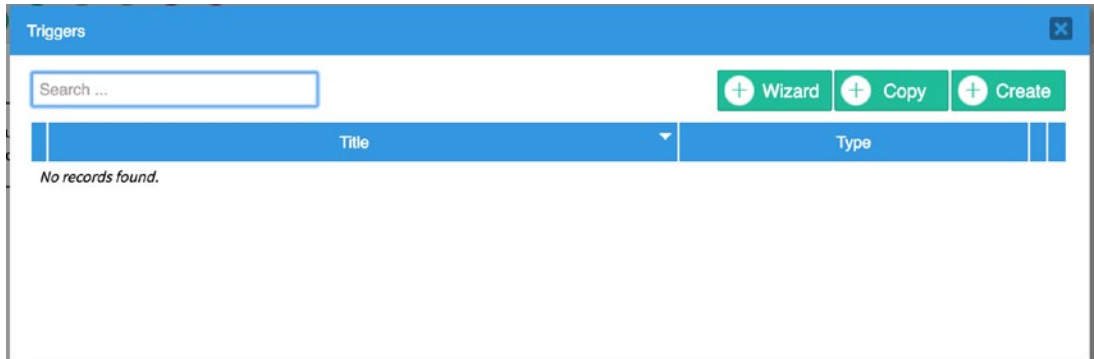
If we make changes to the code and save it, the trigger will become a custom trigger. Try entering a new line in the code by placing your cursor after the semicolon on line 12 and pressing the Enter key. Click the Save button, and you will see a confirmation dialog as shown next. Clicking Yes will change the predefined trigger to a custom trigger. Click No to preserve it as a Predefined trigger and discard the changes by clicking Cancel in the Edit trigger screen.



This shows us that we can always use the predefined functions as a starting point for our own custom triggers when necessary, instead of starting from scratch. Now let us create a custom trigger.

Custom Triggers

We have already seen how to create custom triggers in earlier chapters, when we created triggers to set the case owner and the finance groups in the Cash Advance and Expense Retirement process. We did so by clicking the (+) Create button in the Triggers screen.



This opens the code Create Custom Trigger screen as shown next. As we already know, we can make use of ProcessMaker’s functions in our custom triggers. These functions can be easily accessed by pressing the Ctrl+Space keys. This displays the list of functions, and double-clicking a function from the list adds it to the code. Pressing Ctrl+Space again immediately after the function is added displays the parameters for the function, if any. Let us try it out.

Click the Create button in the Triggers screen and, in the Create Custom Trigger screen, set the Title to **MyCustomTrigger** and Description to **Sample custom trigger that uses a ProcessMaker function**. Place your cursor in the code field and press Ctrl+Space. In the list of functions displayed, double-click generateCode. Then press Ctrl+Space again. This will display the parameters required for the function as PHP variables, as shown next.

Create Custom Trigger

Title*: MyCustomTrigger

Description: Sample custom trigger that uses a ProcessMaker function

Code: 1 generateCode(\$size,\$type)

Press **ctrl+space** to get the function list.

@@ Open Editor

Cancel Save

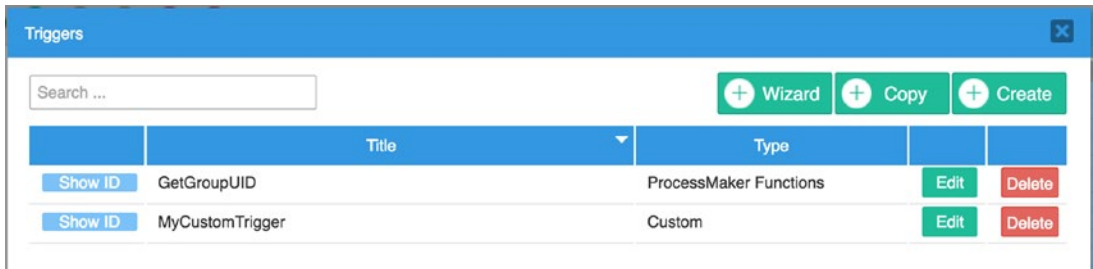
The `generateCode` function generates a random string of letters and/or numbers of a specific length and requires two parameters. Size determines the length of the code, and type determines whether the code should consist of only letters (ALPHA), only numbers (NUMERIC), or a mixture of both (ALPHANUMERIC).

ProcessMaker triggers also allow us to create case variables on demand by assigning the variable in the trigger. Variables created this way are not added to the variable selector but can still be used in other parts of the process, such as conditions. Let us modify the code section of the trigger to create a new case variable and store the generated code in it. We also use the variable selector to select the `my_subform_text1` variable and store another generated code in it.

```
$size = 8;
$type = 'ALPHANUMERIC';
@@my_subform_text1 = generateCode($size,$type);
@@on_demand = generateCode($size,$type);
```

In this code, we are creating a PHP variable named `$size` and assigning it a numeric value of 8. We also create another PHP variable named `$type` and assign it a string value of 'ALPHANUMERIC'. Next we use the variable selector to choose the `my_suggest` variable and assign the result of the ProcessMaker `generateCode` function, which uses the PHP variables created as its parameters. We then create a new case variable `on_demand` and assign it another generated code from the `generateCode` function.

Click the Save button to create the custom trigger, and it should appear in the list of triggers as shown next.



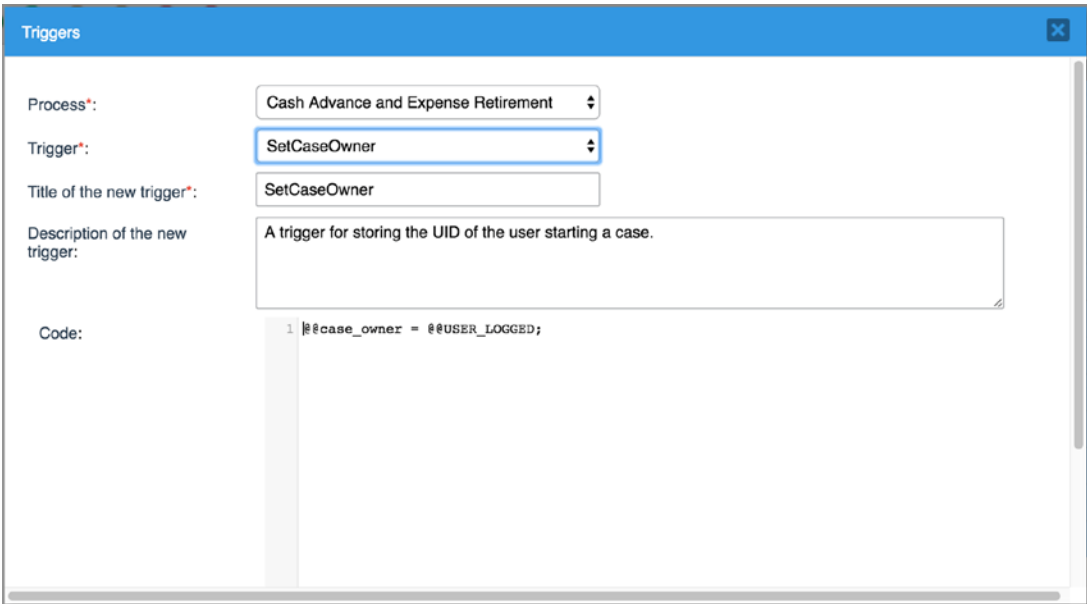
The screenshot shows a window titled "Triggers" with a search bar and three buttons: "Wizard", "Copy", and "Create". Below is a table listing triggers.

	Title	Type		
Show ID	GetGroupUID	ProcessMaker Functions	Edit	Delete
Show ID	MyCustomTrigger	Custom	Edit	Delete

Copying Triggers

When building processes, we often need to reuse the same logic defined in triggers for multiple processes. We may also have triggers that are simply variations of others. The Copy button in the trigger list allows us to easily create a copy of any custom trigger that exists in the system. To illustrate, let us create a copy of the `SetCaseOwner` trigger from the Cash Advance and Expense Retirement process for use in My First process.

Click the Copy button, and the Copy Trigger screen is displayed as shown next. Select Cash Advance and Expense Retirement in the Process field and `SetCaseOwner` in the Trigger field. The title, description, and code fields are immediately populated with the values defined in the `SetCaseOwner` trigger created earlier. Click the Copy Trigger button to create a copy of the trigger in My first process.



Testing the Triggers

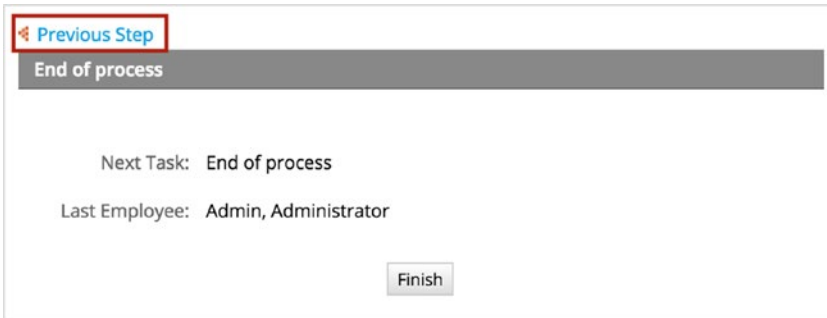
When designing a process that uses triggers, it is important to test the triggers and ensure that they work as expected. Let us run a case to execute the triggers created so far and see the effect they have.

First we need to assign the triggers to the process. Right-click the Request Cash Advance task we created in My First Process and select the Steps option. Assign the My First Form Dynaform to the task, expand it, and place the SetCaseOwner and MyCustomTrigger triggers to execute before the form, and GetGroupUID to execute after it. Do you know why we place the GetGroupUID to execute after the form and not before, like the others? You will see why soon. Close the Steps screen.

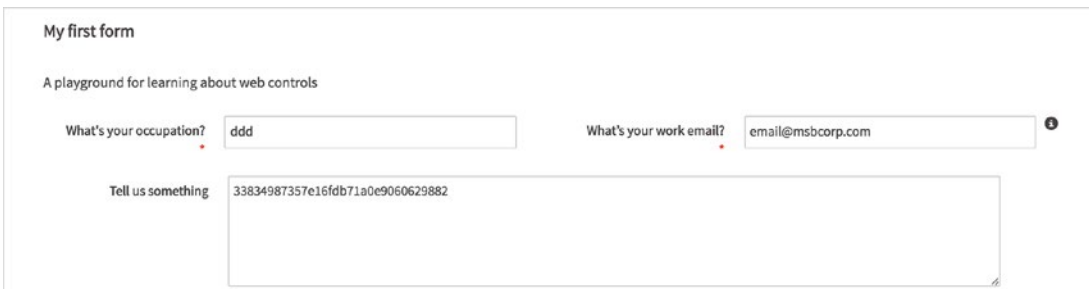
Next click the Request Cash Advance task and, in the quick toolbar, select the End event as shown in the image on the right. This is to avoid errors when we submit the form in the task, as ProcessMaker will not know how to route the case. For now we just want the case to end when we submit the form.

Go to the Home section of the main menu and select New Case under Cases. Select My First Process (Request Cash Advance) and start a new case. The My First Form we created earlier in the guide is displayed and if you scroll to the bottom of the form, you should see a generated code in the Subform Text 1 field as shown here.

We can see that MyCustomTrigger that we set to execute before the form executed successfully. In the Subform Text 2 field, type the name of any of the groups we created earlier, such as Supervisors; also fill out any other field marked as required and submit the form. The routing screen is displayed and shows that the Next Task is End of Process. We would like to know if the GetGroupUID trigger was executed successfully by retrieving the UID of the group we typed in the Subform Text 2 field and storing it in Tell me something textarea. Click the Previous Step link in the routing screen to return to the form.



We should now see the UID of the group displayed in the Tell Us Something textarea, as shown next. Submit the form again and click the Finish button to end the case.



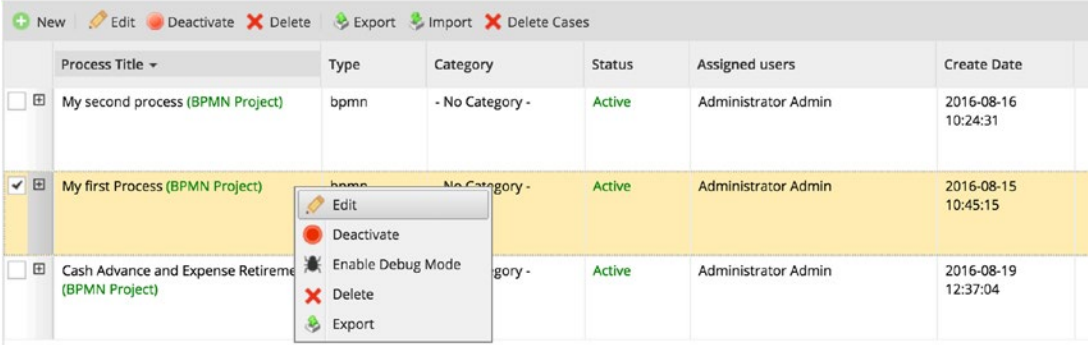
Remember that we also created a new variable in the trigger called `on_demand`. However, we do not have any control on our form associated with it and could not see the value set by the trigger just executed. We also executed the `SetCaseOwner` trigger that set the `case_owner` variable. It would be nice to see that these values were set correctly. ProcessMaker provides a means for us to see what is happening behind the scenes by debugging the process.

Debugging Triggers

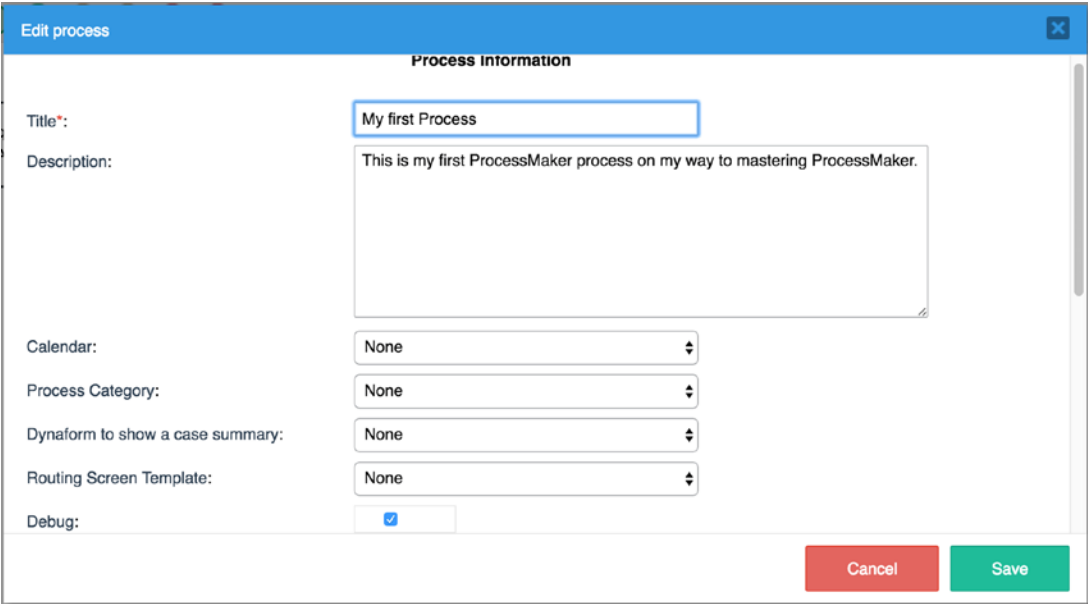
The ability to debug a process is very valuable and can provide insights into why a process is not behaving as expected. This could be as a result of a trigger not executing or throwing an error, or a wrong value being set in a variable. Debug mode should only be enabled when designing the process and not in production, as it would affect the user experience and can expose data that should not be seen by the user.

Enabling and Disabling Debug Mode

Debug mode can be enabled on a process by right-clicking the process from the list of processes and selecting Enable Debug Mode. The Debug column is updated to On. To disable the debug mode, right-click it again and select Disable Debug Mode.

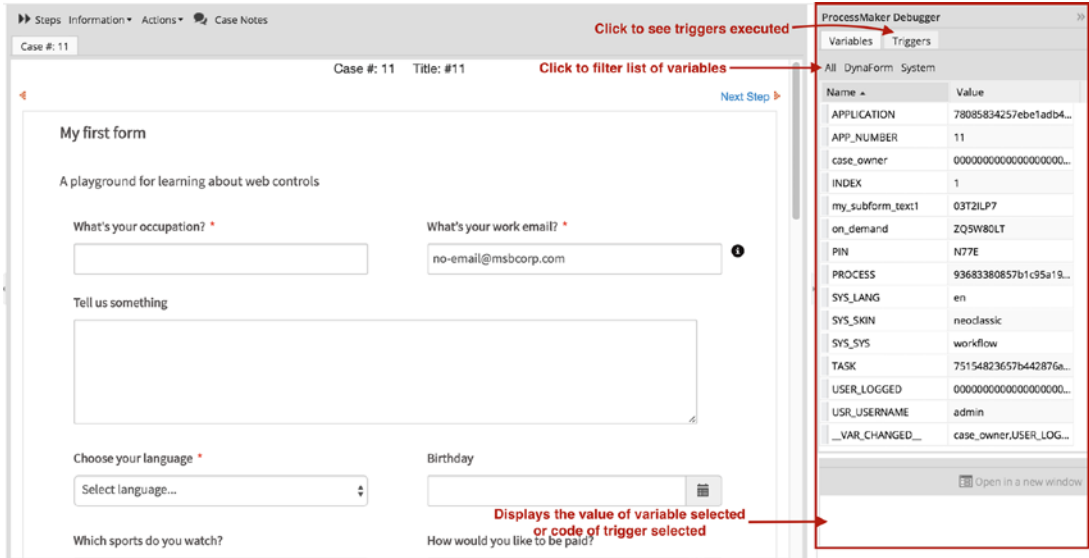


Alternatively, open the process in the designer and right-click in a blank area of the process map. Select Edit Process in the context menu to display the Process Information. Check the Debug field and click the Save button. You can also disable debug mode by unchecking the Debug field. Enable debugging for My First Process.



ProcessMaker Debugger

With debugging enabled, start a new My First Process (Request Cash Advance) case as before. You will notice that the ProcessMaker Debugger window is now opened beside the form as shown here.



The debugger has two tabs, Variables and Triggers. The details of the variables whose values have been set so far in the case are displayed in this tab. Clicking Dynaform under Variables filters the list to show only variables specific to the case. Click it and you should see that the variables we set in our triggers have been assigned values. Clicking on the variables displays the values in the box at the bottom of the debugger. This is useful for variables with long values, like case_owner. Next click the Triggers tab and you should see the two triggers that were executed before the form. Click the trigger and the code is displayed in the box at the bottom of the debugger. The box has an Open in New Window button to display the code in a popup. This can be useful for triggers with many lines of code.

Name ▲	Value
APP_NUMBER	11
case_owner	00000000000000000000...
my_subform_text1	03T2ILP7
on_demand	ZQ5W80LT
PIN	N77E
__VAR_CHANGED__	case_owner,USER_LOG...

ProcessMaker Debugger	
Variables Triggers	
Name ▲	Execution
☑ Execution: before (2 Triggers)	
MyCustomTrigger	before
SetCaseOwner	before

Just as we did with the previous case, fill in the required fields and enter the name of a Group in the Subform Text 2 field and submit the form.

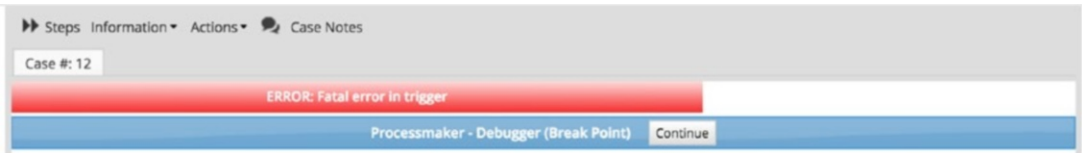


Instead of moving straight to the routing screen, we see a ProcessMaker Debugger Breakpoint. This allows us to examine triggers that were fired after the form. In this case, we can see the GetGroupUID trigger that we set to execute after the form. We can also click the Variables tab and lookup the my_textarea variable. We will see that its value has been set to an array containing the group UID, which shows that our trigger executed successfully.

We can also see the values set for the other variables in the form. Click the Continue button on the breakpoint to continue to the routing screen. Click the Finish button to end the case. We also see another breakpoint displayed which can be used to see the effect of triggers executed after routing. Click the Continue button.

Identifying Errors

The ProcessMaker Debugger also helps identify the cause of errors in a trigger. When an error is encountered, the execution of the trigger is halted and an error message is displayed if the error is fatal.



In these cases, a system variable `_ERROR_` will be added to the variables list in the debugger and contain information about the error.

As we have seen, triggers play an integral role in adding business-specific functionality and logic to the processes built in ProcessMaker. To learn more about triggers in ProcessMaker, see <http://wiki.processmaker.com/3.1/Triggers>.

CHAPTER 13

Input and Output Documents

Business processes often require supporting documents to aid the users assigned to a task perform the task. For example, in our Cash Advance and Expense Retirement process, we will require users to provide receipts for the expenses, which serve as proof of transaction and help the Supervisor and Finance officers ascertain that the transaction indeed took place. This class of documents is referred to as *input documents* in ProcessMaker.

Also, other processes produce documents that are either stored for record purposes or shared with other parties within and outside the organization. For example, an Asset Requisition process will need to produce a Purchase Order that will be sent to the vendor or supplier. Documents that are generated in the course of a process are referred to as *output documents* in ProcessMaker.

Let us return to our playground process and explore how input and output documents are created and used in ProcessMaker.

Input Documents

We begin our discussion with input documents. These are basically files or documents that contain information that serves as input to the process. Input documents can be digital or printed (hard copy). Digital input documents can be uploaded into ProcessMaker while for printed documents, the user will be asked to provide a description of the document. Let us create a couple of input documents to explore them in detail. Open My First Process in the Process Designer.

Creating an Input Document

To create an input document, click the Create (+) icon beside Input Documents in the Main Toolbox. The Create Input Document screen is displayed, as shown next. The form allows us to define and configure the behavior of the input document. Let us briefly look at each field.

Title: This is the title of the input document.

Document Type: This refers to the type of input document to be created. The options are Digital, Printed, or both. Digital documents are expected to be uploaded along with the case, while only a description of the printed document is required. If the Printed or Digital/Printed option is selected, you will be shown an additional field to specify the format of the document, whether it is the Original, Legal Copy, or Copy.

Description: This field is to enter a description of the document that will be displayed to the user when requesting the document to be uploaded. This can help the user know what type of document is expected.

The screenshot shows a 'Create Input Document' dialog box with the following fields and values:

- Title*: [Empty text box]
- Document Type: Digital (dropdown menu)
- Description: [Empty text area]
- Enable Versioning: NO (dropdown menu)
- Destination Path: [Empty text box] with @@ icon
- Tags: INPUT (text box) with @@ icon
- Allowed file extensions (Use .* to allow any extension)*: .* (text box)

At the bottom right, there are two buttons: a red 'Cancel' button and a green 'Save' button.

Enable Versioning: This is used to indicate whether versioning should be enabled for the document. If versioning is enabled, different copies of the input document uploaded for a case will be preserved. If versioning is disabled, new uploads of the input document for a particular case will overwrite the documents previously uploaded for the case. Versioning can be used when you want to preserve an audit trail of all documents uploaded as the input document for the case, especially if different users in the life cycle of the case can upload the input document.

Destination Path: This is used to define the path in the built-in ProcessMaker Document Management system where the uploaded document will be stored. This is useful for organizing documents being uploaded into the system. You will observe that there is a variable selector beside the field. This allows us to select variables to make up the path to store the document.

For example, if we have an input document for receipts, we might want to separate it by the department and the case the receipt belongs to. We can therefore specify the path as **Receipts/@@department/@@APP_NUMBER**. This way, if a user in Sales uploads a receipt saved as `taxi.jpg` for case #12, the document will be stored in `Receipts/Sales/12/taxi.jpg` in the Document Management System.

Tags: As the name implies, this is used to specify a comma-separated list of tags to associate with the input document. This can be useful when searching for the document later. The tags can be plain text or obtained from case variables.

Allowed file extensions: This is used to restrict the type of files that can be uploaded as the input document. For example, we would expect receipts to be either a picture (`.jpg`, `.png`, `.bmp`, `.gif`) or PDF (`.pdf`) document. The file extensions are prefixed with an asterisk (*) and separated by commas. To allow any file type, simply use an asterisk.

Maximum file size and Unit: These are useful fields for restricting the size of the document that can be uploaded. It is important

to set this to a reasonable limit, but not too large to avoid users uploading files that are too large and can slow down the system. To remove the size limit, enter 0 in the maximum file size field.

Go ahead and create an input document as specified in the following table. Fill in the fields and click the Save button.

Title	My input doc one
Document Type	Digital
Format	
Description	Sample digital input document
Enable Versioning	Yes
Destination Path	Playground/input/@@my_radio/@@APP_NUMBER
Tags	digital,doc one
Allowed File Extensions	*.jpg,*.pdf
Maximum File Size	1000
Unit	KB

With our input document created, the next thing we have to do before we can use it is assign it to the process. There are two ways we can make input documents available for use within a process. We can assign the input document to a file or multiple files control in a dynaform, or add it as a step to a task. Let us try out both.

Adding Input Documents to a Dynaform

Click on Dynaforms in the main toolbox (Process Objects) and in the list of dynaforms displayed, click the Edit button beside “My First Form” to open it in the dynaform editor. You will recall that when we added a file control to this form earlier in the guide, we were unable to associate it with a variable because the variable had to be a file type and required an input document. Now we have an input document, so we can associate the file control with a variable.

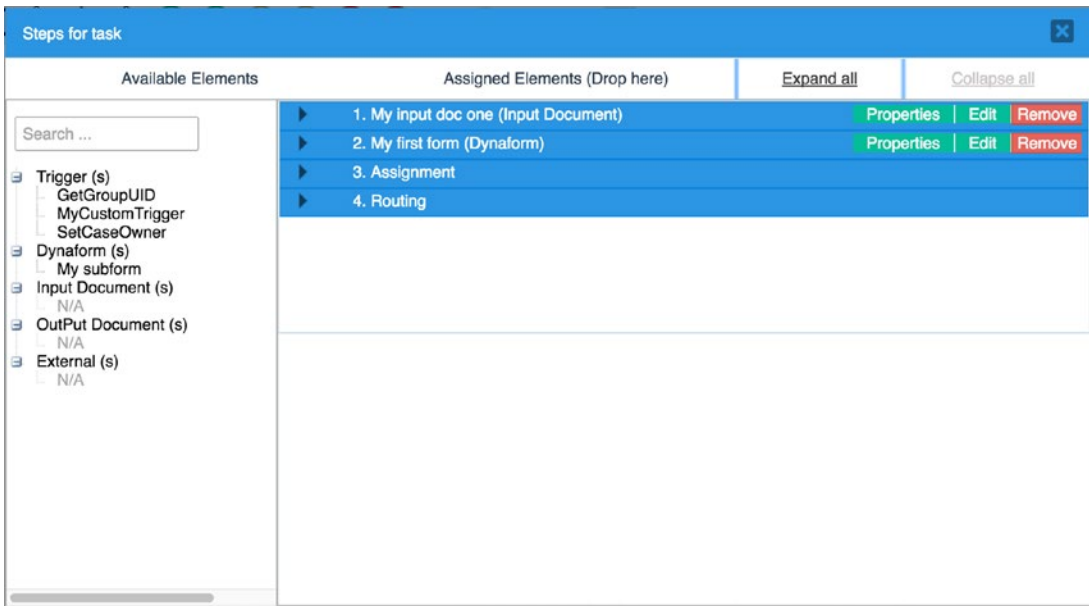
Select the file control (Attach Receipt) from the dynaform by clicking it, and in the properties panel on the left, click the ellipsis beside the Variable property and in the Create/Select Variable modal displayed, enter **my_file** as the variable name. Expand the settings and click the ellipsis to select My input doc one as the input document. Click the Save button.

Also select the multiple files upload control (Upload supporting documents), and in the properties panel, click the ellipsis in front of the Input document property. In the modal that appears, select My input doc one. Save the dynaform and close the Dynaform Designer.

Now when a file is uploaded to either control, it will be saved to the input document—My input doc one.

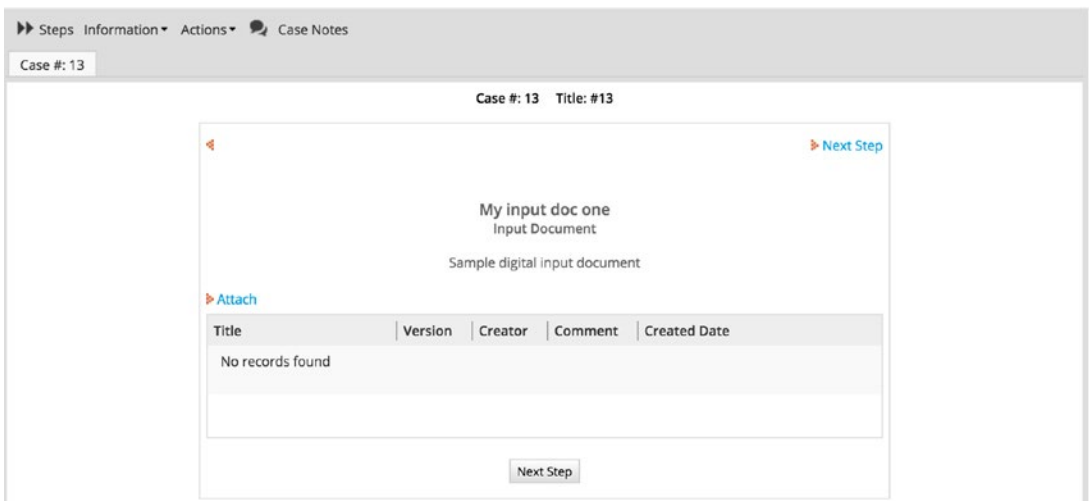
Adding Input Documents as a Step

We can also make an input document available for use in the process by making it a step for the task. This is similar to the way we add dynaforms as steps to a task. Let us proceed to add the input documents as steps to the Request Cash Advance task. Right-click the task and select Steps. The input document is displayed in the list of available elements. Drag and drop the input document to the Assigned Elements on the right, placing My input doc one before My First Form, as shown here.

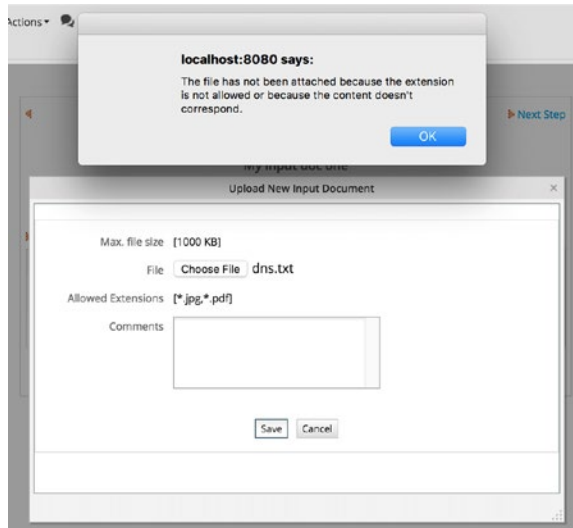


All done. Let us initiate a new case to see how the input documents work. We will also illustrate versioning in input documents.

Still logged in as the Administrator, go to Home in the main menu and under Cases, select New Case. Select My first process (Request Cash Advance) and click Start Case. The input document upload screen is displayed as shown next. We can see the title and description we specified for the first input document we created.



Click the Attach link to display the interface for uploading the document. You can see the specified max file size and allowed extensions displayed. Try uploading a file that is not `.jpg` or `.pdf`, and an error message will be displayed as shown in the image on the left. Dismiss the error message and proceed to upload a `.jpg` or `.pdf` file less than 1000 KB, also type a comment in the Comments field and save the document.



Title	Version	Creator	Comment	Created Date		
input_doc.pdf	1	Administrator Admin	Comment on pdf input document	2016-10-03	Download	New Version
input_doc.jpg	1	Administrator Admin	Comment on document	2016-10-03	Download	New Version

Rows 1-2/2 Page 1/1

You can upload more than one input document. Upload one `.pdf` and `.jpg` file each, and the documents should be displayed as shown next. Note that if you still have debug mode enabled for the process, you will notice that a breakpoint is encountered every time a document is uploaded. If we had any trigger placed after the input document step, it will be executed every time a document is uploaded.

Looking at the image here, we observe that there is a New Version button beside the uploaded documents. This is because we have versioning enabled. If we want to upload a new version of any of the documents uploaded, we should use the New Version button. Uploading a file with the same name does not update the already uploaded file but is

added as a new document as shown in the image, where we have uploaded the `input_doc.jpg` file again.

Title	Version	Creator	Comment	Created Date		
input_doc.jpg	1	Administrator Admin	Same document uploaded again	2016-10-03	Download	New Version
input_doc.pdf	1	Administrator Admin	Comment on pdf input document	2016-10-03	Download	New Version
input_doc.jpg	1	Administrator Admin	Comment on document	2016-10-03	Download	New Version

Rows 1-3/3 Page 1/1

Let us create a new version by clicking the New Version button beside the document. This displays the input document upload screen; select the file, enter comments, and upload it. The document version is updated to 2 and a Version History button is placed beside it as shown here.

Title	Version	Creator	Comment	Created Date			
input_doc.jpg	1	Administrator Admin	Same document uploaded again	2016-10-03	Download	New Version	
input_doc.pdf	1	Administrator Admin	Comment on pdf input document	2016-10-03	Download	New Version	
input_doc.jpg	2	Administrator Admin	New version of document uploaded	2016-10-03	Download	New Version	Version History

Rows 1-3/3 Page 1/1

Click the Version History button to see the different versions of the document as shown in the following image. To view the contents of the uploaded documents, click the Download button beside it. Now that we have seen how versioning works, let us proceed to see how the input document attached to a file control works. Close the input Document History screen and click the Next Step button below the table to open the dynaform.

In the displayed dynaform, click the Attach Receipt file control and upload a `.pdf` or `.jpg` file. Also upload a couple of files to the Upload supporting document multiple file control. Select an option from “How would you like to be paid” field (`my_radio` control), fill out the other required fields in the form, and submit it. The end of process routing screen is displayed. Before we close the case, how can we view the input document we uploaded using the file control?

Title	Version	Creator	Created Date	
input_doc.jpg	1	Administrator Admin	2016-10-03	Download
input_doc.jpg	2	Administrator Admin	2016-10-03	Download

Rows 1-2/2 Page 1/1

Click the Previous Step link on the Routing Screen and also on the dynaform to return to the input document screen we saw at the beginning of the case. We should now see the form uploaded via the file and multiple file controls as shown next.

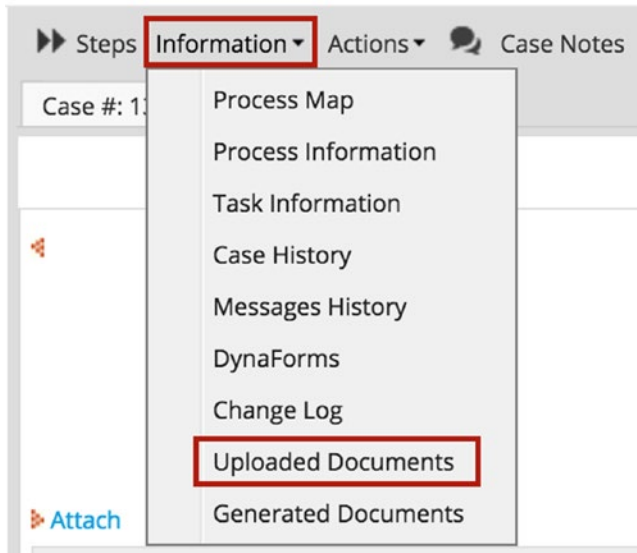
Title	Version	Creator	Comment	Created Date		
receipt.pdf	1	Administrator Admin		2016-10-03	Download	New Version
input_doc.jpg	1	Administrator Admin	Same document uploaded again	2016-10-03	Download	New Version
input_doc.pdf	1	Administrator Admin	Comment on pdf input document	2016-10-03	Download	New Version
input_doc.jpg	2	Administrator Admin	New version of document uploaded	2016-10-03	Download	New Version Version History

Rows 1-4/4 Page 1/1

This is not the ideal way, however. ProcessMaker provides us with an Information menu with which we can view information about a case. Click the Information button as shown in the image on the right and select Uploaded Documents from the drop-down displayed. This displays a list of the uploaded documents for the case as shown in the next image . To return to the case, click the Close button beside the Uploaded Documents tab or click the tab with the Case number.

In a process with multiple steps, as will be the case most of the time, users in other tasks where the input document is not available as a step can always view the documents that have been uploaded for a case using this approach.

Go ahead and complete the case by clicking the Next Step button, submitting the dynaform, and clicking the Finish button.



Steps Information Actions Case Notes

Case #: 13 Uploaded Documents

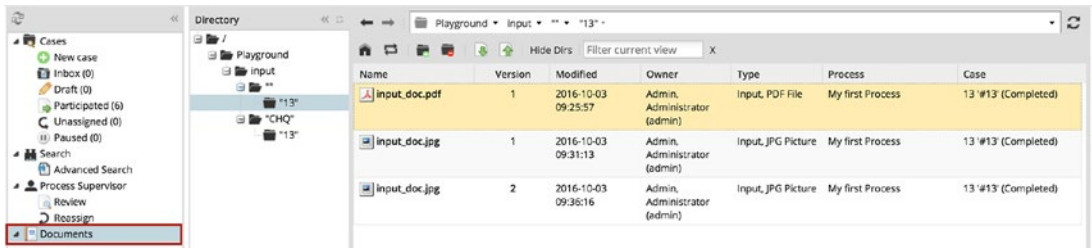
Download

Filename	Comments	Type	Version	Origin Task	Created By	Create Date
<input type="checkbox"/> receipt.pdf		Input	1	Request Cash Advance	Admin, Administrator (admin)	2016-10-03 09:52:42
<input type="checkbox"/> input_doc.jpg	New version of document uploaded	Input	2	Request Cash Advance	Admin, Administrator (admin)	2016-10-03 09:36:16
<input type="checkbox"/> input_doc.jpg	Same document uploaded again	Input	1	Request Cash Advance	Admin, Administrator (admin)	2016-10-03 09:31:13
<input type="checkbox"/> input_doc.pdf	Comment on pdf Input document	Input	1	Request Cash Advance	Admin, Administrator (admin)	2016-10-03 09:25:57

Viewing the Documents in the Document Management System

When defining the input document, we specified a destination path for the uploaded documents to be stored in the basic Document Management System (DMS) built into ProcessMaker. Let us check how the files we uploaded were stored.

Click on Documents in the left menu to display the DMS directory structure. Expand the Playground directory and you should see that the files we uploaded were saved based on the path specified when creating the Input Document. We can also see that the case variables @@my_radio and @@APP_NUMBER were correctly read and used in creating the folder structure, as shown in the following image. The documents we uploaded before filling the form do not have a value for @@my_radio and are placed in the "" folder, while the one uploaded in the form after we had set a value for the variable CHQ is placed in a folder with the matching name.



We have seen how input documents can be created, used, and stored in ProcessMaker. Next we will look at output documents and then use that knowledge to complete our Cash Advance and Expense Retirement process.

Output Documents

As we mentioned earlier, output documents are those generated by a process. The document can either be a .doc or a .pdf document. Let us return to the Process Designer and select My First Process to create an output document to see how it works.

Creating an Output Document

We begin by clicking the Create (+) icon beside Output Documents in the Main Toolbox to display the form as shown here.

The fields in the form allow us to configure how the output document will be generated. Let us look at each field.

Title: This is the title of the output document used to identify it in the list of output documents.

Filename generated: This is the name of the file that will be generated by the output document, and it can be composed using variables from the variables selector or just given a static name.

Description: A description of the output document that will be shown to users to give a general idea of the content of the document that has been generated.

Report Generator: This is used to select the library that will be used to generate the PDF document. There are two options, TCPDF and HTML2PDF. The HTML2PDF version was used in earlier versions of ProcessMaker but had a number of limitations such as large file size and limited number of pages. For version 3.0 and above it is recommended to use the TCPDF library, which addresses these challenges.

Media: This is used to specify the page size of the document that will be generated. The options consist of common paper sizes such as Legal, Letter, A1, A2, A3, A4, and so on.

Orientation: Used to specify if the document should be portrait or landscape.

Margin: This is used to specify the distance of the content of the document from the edges of the paper in all directions. The value of the margin is measured in millimeters.

Output Document to generate: This indicates the format in which the document will be generated and the options are PDF, Doc or both.

PDF Security: This allows you enable security features for the generated PDF document if required. When set to Enabled, the following additional fields are displayed to set up the security features:

- **Open Password:** sets a password required for opening the document.
- **Owner Password:** sets a password for changing the document permissions.
- **Allowed Permissions:** sets the default permissions for the document..

Enable versioning: Just like the input documents, this is used to allow multiple versions of the document to be generated. This is useful when the document is generated multiple times during a case with changing data. With versioning enabled, you will be able to see the different versions and the changes in the document over time.

Destination Path: This is also the same as we saw for input documents. It specifies the path in the DMS (Document Management System) to store the generated document and can be composed with variables from the case using the variable selector.

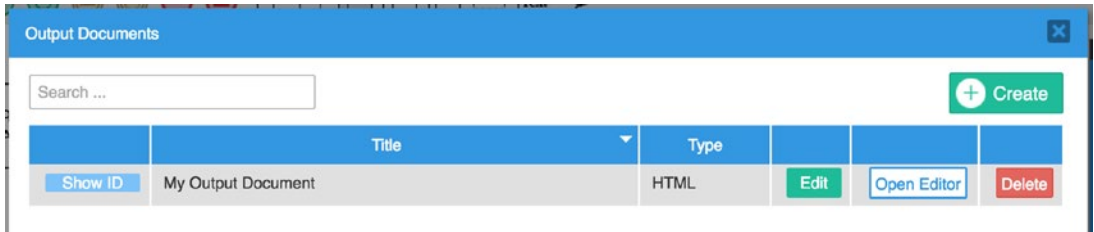
Tags: Also like the input document, these are useful for adding metadata to the document to make it easier to find during searches.

By clicking on the generated file link: This is used to specify the default behavior of the browser when the link to the generated document is clicked. The options are to either download or open the file. With the latter option, the browser must have a default application configured for the file type, such as Adobe Acrobat for PDF files. It is generally recommended to set it to download file.

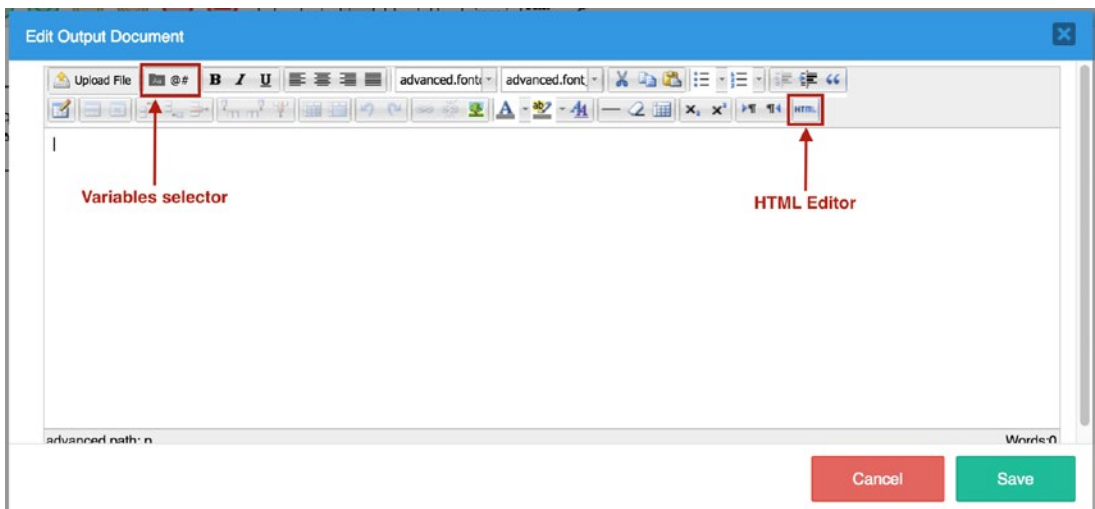
If you are wondering how the output document gets its content, that is defined after the document is created. Let us create an output document using the details in the following table, and then we will see the editor for defining the content of the generated document. In My first process, click the Create (+) icon beside Output Documents in the Main Toolbox and fill in the following values:

Title	My Output Document
Filename generated	demo-@@my_suggest
Description	Sample output document
Report Generator	TCPDF
Media	A4
Orientation	Portrait
Margin	Leave all as 20
Output document to generate	Both
PDF Security	Enabled
Open Password	test123
Owner Password	test123
Allowed Permissions	Leave all unchecked
Enable Versioning	Yes
Destination Path	Playground/output/@@my_radio/@@APP_NUMBER
Tags	demo, output doc
By clicking on the generated file link	Download the file

When you're done, click the Save button to create the output document. The output document is displayed in the Output Documents list as shown here.



Beside it are three buttons. The Edit button opens the form we filled when creating the document and can be used to edit the settings of the document. The next button, Open Editor, opens a WYSIWYG (What you see is what you get) editor that is used to define the contents of the output document as shown next.



The content of the document can be interpolated with the values stored in the variables for the case. To access the variables, click the variable selector in the editor toolbar and a list of variables available in the process will be displayed. You can also upload an HTML file as a template by clicking the Upload File button.

Since the template is basically an HTML file, you can click the HTML button in the editor to open the HTML source code and edit as desired. The other icons in the toolbar should be familiar from common editors. They include formatting options for the text such as font type, font size, bold, underline, and so on. There are also icons for inserting tables and images.

In the list of output documents, click the Open Editor button if you have not already done so, and click the HTML icon in the toolbar of the editor. Paste the following code in the HTML Source Editor and click the Update button.

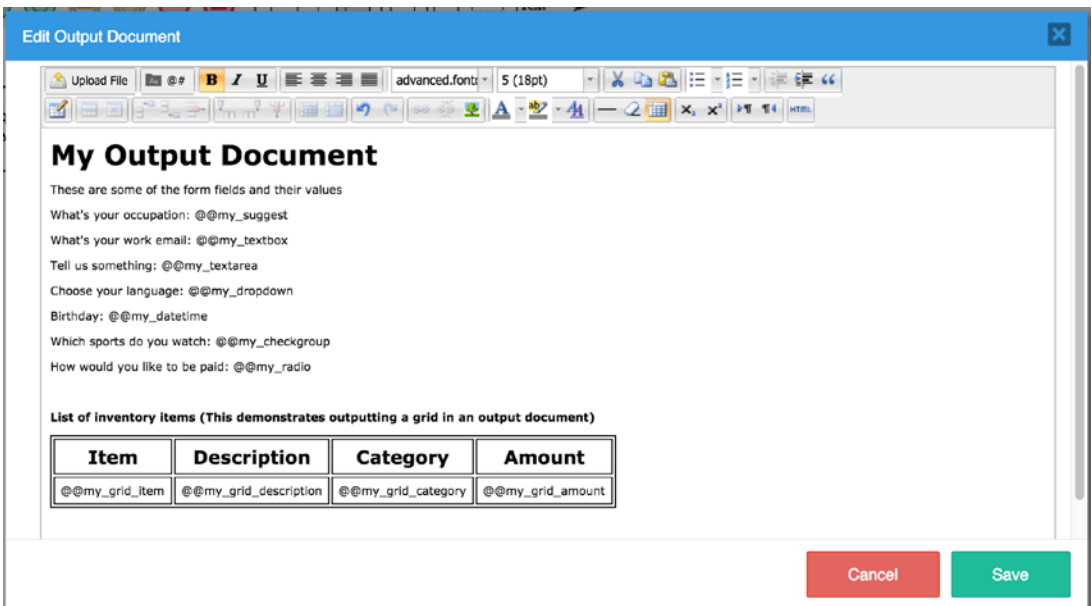
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
  </head>
  <body>
    <p><b><font size="5">My Output Document</font></b></p>
    <p>These are some of the form fields and their values</p>
    <p>What's your occupation: @@my_suggest</p>
    <p>What's your work email: @@my_textbox</p>
    <p>Tell us something: @@my_textarea</p>
      <p>Choose your language: @@my_dropdown</p>
    <p>Birthday: @@my_datetime</p>
    <p>Which sports do you watch: @@my_checkgroup</p>
    <p>How would you like to be paid: @@my_radio</p>
    <p></p>
    <p>
      <b>List of inventory items
      (This demonstrates outputting a grid in an output document)</b>
    </p>
    <table border="1" cellpadding="5"
    style="border-color: #000000; border-width: 1px; border-style: solid;">
    <thead>
      <tr>
        <th>Item</th>
        <th>Description</th>
        <th>Category</th>
        <th>Amount</th>
      </tr>
    </thead>
    <tbody>
      <!--@>my_grid-->
```

```

<tr>
  <td>@@my_grid_item</td>
  <td>@@my_grid_description</td>
  <td>@@my_grid_category</td>
  <td>@@my_grid_amount</td>
</tr>
<!--@my_grid-->
</tbody>
</table>
<p></p>
<p></p>
</body>
</html>

```

The editor should display the template of the output document as shown next. It is important to note that you do not need to use the HTML editor when creating your output document. We are using the HTML editor because we want to display a grid in the output document. Unlike other variables, the value in a grid variable is nested, and we require a special syntax in order for the output document to recognize it and render it appropriately.



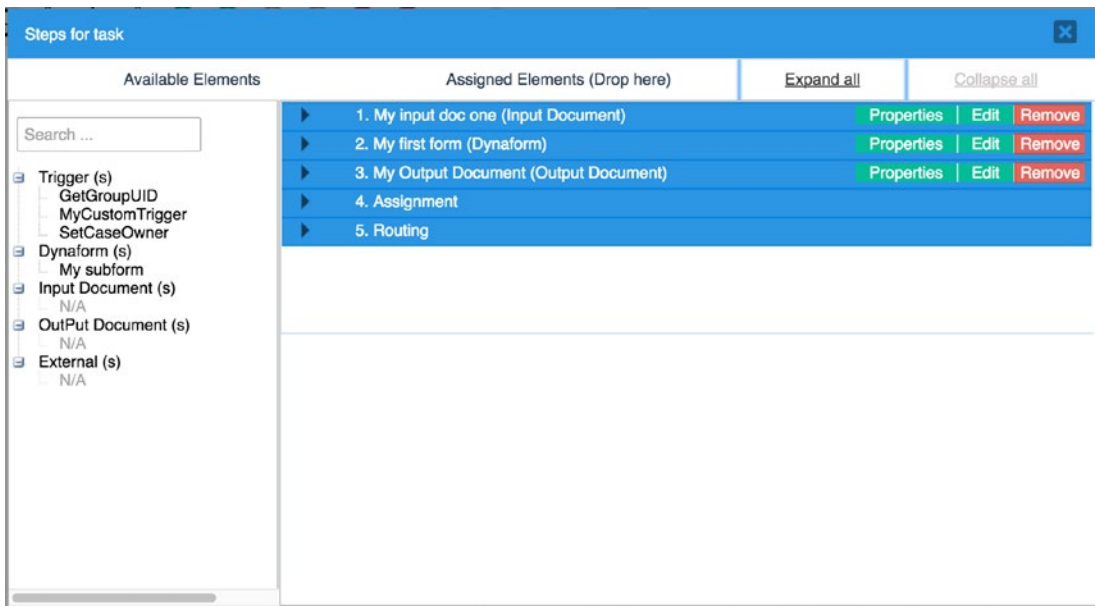
To quickly explain the content of the output document, we have a title in the first line with a larger font and bold text. In the following lines, we have the labels of some of the controls in My First Form, written in plain text, and we use the variable selector to add the variable name to the line. When the output document is generated, the variables (such as `@@my_suggest`) will be replaced by the actual value filled in the case.

At the bottom of the document, we insert a table with columns representative of some of the field in the `my_grid` variable in the form. The header for each column is typed as plain text and we add a row for the actual values that will be filled in the form. Since a grid can have multiple rows, we need to be able to represent that data in an iterative manner.

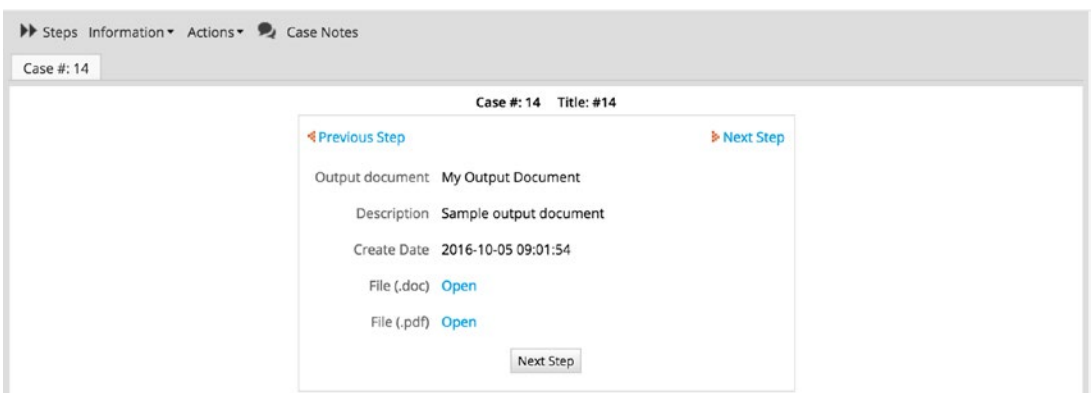
ProcessMaker provides us a syntax for representing grids in dynaform, and you will observe that in the code sample earlier, we have highlighted a section. This is the portion that displays the table that renders the grid. If you are familiar with HTML, the code is basically the same for a table, but the only difference is the `<!--@my_grid-->` and `<!--@<my_grid-->` tags that we use to wrap the row containing the variables for the field in the grid. These wrappers tell ProcessMaker that the variables between these two tags are grid variables and should be iterated through the number of rows in the grid.

The opening tag is `@>NAME_OF_GRID`, and the closing tag is `@<NAME_OF_GRID`. However, because the HTML editor does not recognize incomplete tags like `< or >`, the tags are wrapped as HTML comments, making it `<!--@>NAME_OF_GRID-->` and `<!--@<NAME_OF_GRID-->`.

Click the Save button to save the changes and complete the output document setup. Return to the Process Map and add the output document as a step after the dynaform in the Request Cash Advance task as shown in the following image. With that done, we are all set to test out the output document.



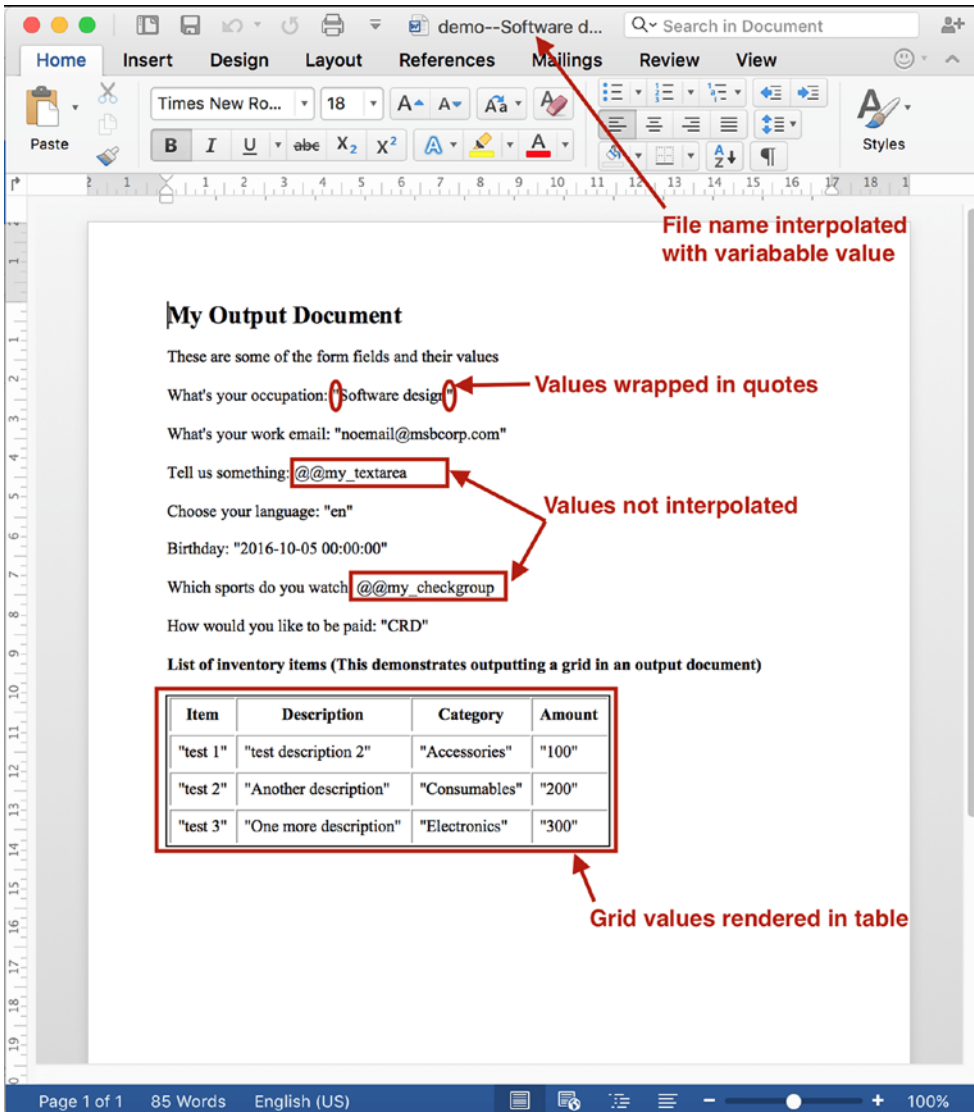
Go to Home in the main menu and under Cases select New Case and initiate a new case for My First Process (Request Cash Advance). Disregard the upload document step and click the Next button. On the dynaform, fill out all the fields in the form with the exception of the file controls. For the grid, add a couple of rows and fill in values for the item, description, category and amount columns. Click the Submit Form button and if you still have debug mode enabled, click Continue. You should now see the output document screen as shown next.



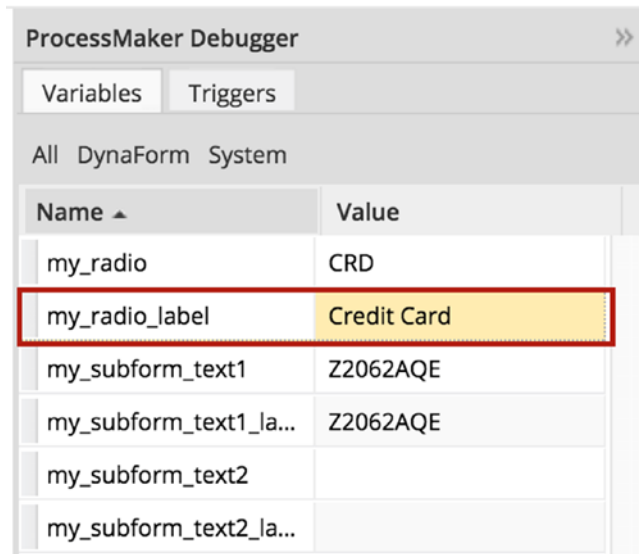
We can see that both .doc and .pdf versions of the document were generated. Click the Open link beside each type to download the generated document. The name of the

downloaded file should include the value entered in the my_suggest variable. If you try opening the PDF document, you should be prompted to enter a password. This is expected, since we enabled security for the output document. You will also notice that the values are all being wrapped in double quotes. That is because we used the @@ prefix for the variable names. To display the variable values without the quotes, use the @= prefix to return the values in their original format.

You will also observe that the values for @@my_checkgroup and @@my_textarea were not interpolated in the template.



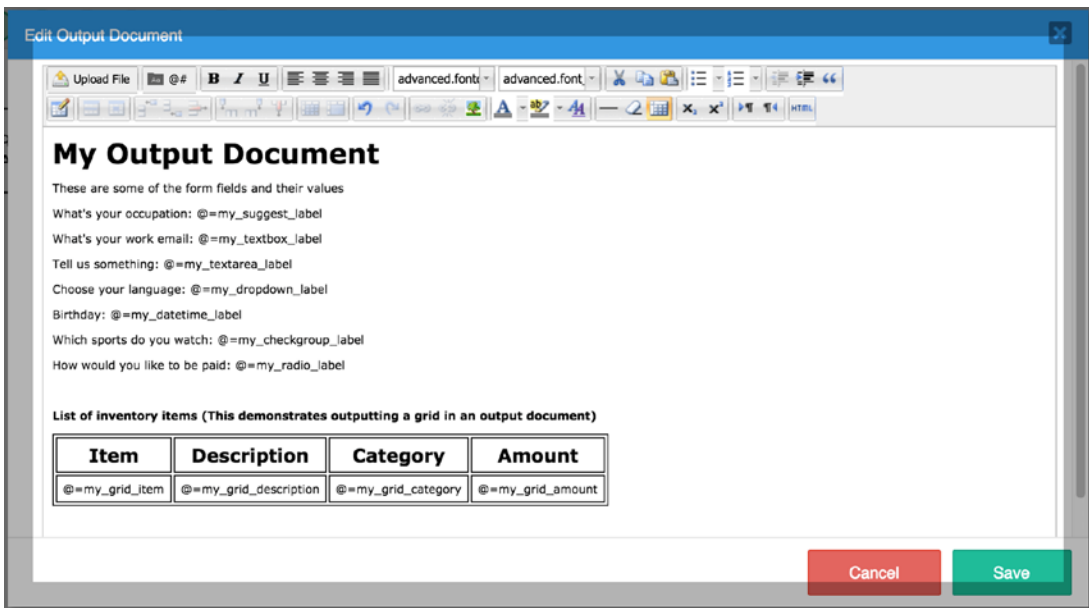
If we check the values of these variables in the debugger, we will notice that they are arrays (the textarea shows as an array because we are setting its value in a trigger—`GetGroupUID`—that is returning an array). You will also observe in the debugger that most of the variables in our form have an associated label variable that contains a text description of the values stored in the variable. For example, the `my_radio` variable in the following image has a value `CRD`, and the associated label variable, `my_radio_label`, stores the label of the value stored in the variable, `Credit Card`.



Name ▲	Value
my_radio	CRD
my_radio_label	Credit Card
my_subform_text1	Z2062AQE
my_subform_text1_la...	Z2062AQE
my_subform_text2	
my_subform_text2_la...	

Armed with this new knowledge, it is obvious that we can improve the documents generated in the output document by making a few changes to the output document template. Click the Next Step and Finish buttons to close the current case.

Open My First Process in the designer and click Output Documents in the main toolbox. Open the output document we just created in the editor and change the prefix for the variables to `@=` and use the associated label variable variant for the first section of the document as shown in the following image. For example, `@@my_suggest` becomes `@=my_suggest_label`. When you're done, save your changes.



Now return to Home and run another case. Disregard the input document step as before and in the dynaform, fill out the fields we have listed on the output document. Create multiple rows for the grid and fill out the item, description, category, and amount columns in the grid as we did before. When done, click the Submit Form button, and the output document step is displayed. Click the Open links to download the files. Open the files (enter the password for the .pdf document) and you will see that all the variables are now properly interpolated and the double quotes are gone, as shown in the next image.

My Output Document

These are some of the form fields and their values

What's your occupation: Database administrator

What's your work email: noemail@msbcorp.com

Tell us something: Something in the text area

Choose your language: Spanish

Birthday: October 5th, 2016 00 00 AM

Which sports do you watch: ["Tennis", "Soccer", "Hockey"]

How would you like to be paid: Cheque

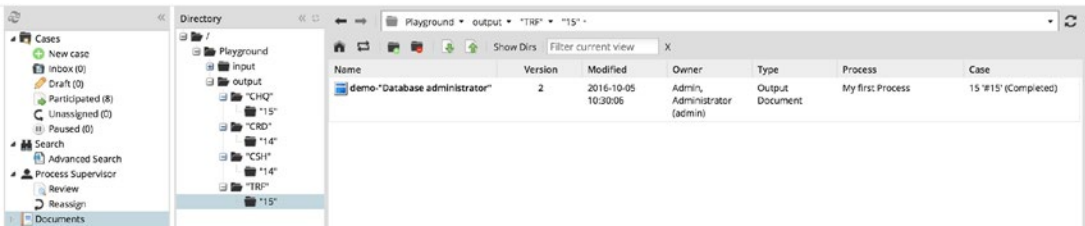
List of inventory items (This demonstrates outputting a grid in an output document)

Item	Description	Category	Amount
item 1	item 1 description	Accessories	100
item 2	item 2 description	Consumables	200
item 3	item 3 description	Electronics	300

Page 1 of 1 94 Words English (US) 100%

You will agree that this is a better output than the previous one. Before completing the case, click the Previous Step link to go back to the form and make changes to the value of the fields in the form and submit it again. This will generate a new version of the output document. Click the Next Step, Finish, and Continue buttons to complete the case.

We set up the output document to be saved to the Playground folder in the DMS just as we did for the input document. Click Documents in the left sidebar and expand the Playground folder. You should see a new output folder created as shown here.



If you changed the value of my_radio variable when we went back to make changes to the form, there will be more than one folder with the same case number, but the document will be stored in the folder with the most recent value of my_radio variable. We can also see that the version of the generated document has been updated to 2.

That concludes our discussion of output documents, but you can explore the topic further on the ProcessMaker wiki, at http://wiki.processmaker.com/3.0/Input_Documents and http://wiki.processmaker.com/3.0/Output_Documents. In the next chapter, we will return to our Cash Advance and Expense Retirement process and use the knowledge we have gained from this detour into triggers, input documents, and output documents to complete the design of the process.

CHAPTER 14

Completing the Process

A quick recap of what we have done so far. We began by modeling the Cash Advance and Expense Retirement process. In our model, we split the process into two parts: the Cash Advance Request and the Expense Reporting. We then proceeded to build the forms for the cash advance part of the process and assign them to the tasks. We also set up users, groups, and departments in the system to explore the different case assignment methods and ran a few cases to see how they worked.

In the process, we created a couple of basic triggers to help us with some of the scenarios we explored. We stopped the cases we initiated at the Disburse Advance task, however, because the next task, Report Expense, is the first task in the second half of the process and was yet to have any steps or users assigned to it.

When designing the process, one consideration was that we will want a user to be able to report an expense without having requested an advance. To make this possible, we made the Report Expense task a starting task too by linking it to a start event in the Process Map. To complete the process, we will build the forms for the remaining tasks, define additional triggers, and setup assignment rules.

You might have noticed that as we progressed through the test cases, we were not updating the Signoff/Approval section of the form with the details of the user. We will create triggers to help do this and set conditions to make sure that the triggers only execute when the user has approved the request. In addition, we will use triggers to pre-fill some of the information on the form such as the requesting officer and date of the request. We will also setup input documents for the user to upload receipts and generate Expense Reports as output documents. Once we have completed the process, we will add a few enhancements like email notifications, escalations, theming, and more. Let us dive in.

Building the Additional Forms

By now, you are already proficient with working with dynaforms, so we will not be as explicit with the instructions as we were in earlier chapters. We will be building three additional forms for the Expense Reporting tasks:

Expense Report Form: This will be filled in by the user to report expense details.

Expense Report Approval Form: This will be used by the supervisor to approve the expense report.

Expense Report Processing Form: This will be used by the Finance officers to process the expense report.

The forms will be based on the Expense Retirement form we saw in the Sample workflow at the beginning of the book.

Since the Expense retirement may be a continuation of a cash advance request, we will provide a way for data already filled when requesting the cash advance to be readily available for the expense report but not editable. We will not be creating the variables in advance as we did with the Cash Advance forms; rather we will create them as needed in the Dynaform Designer. Let us get started.

Open the Cash Advance and Expense Retirement process in the Process Designer and click the Create (+) icon beside Dynaforms in the Main Toolbox. Create a new dynaform with the following properties and open it in the Dynaform Designer:

Title: Expense Report Form

Description: This form will be filled in by the employee reporting an expense

Next, import the Cash Advance Request form we exported earlier so we don't have to start from scratch. This way, we already have the comments functionality and a basic layout in place for the new form.

Modifying the Imported Form

With the form successfully imported, make the following changes to the form to convert it to the Expense Report Form:

1. Select the title control in the first row and change the id property to **expense_form_title** and label property to **Expense Retirement**.
2. Select the subtitle control in the second row and change the id property to **expense_subtitle1** and label property to **Expense Details**.
3. Select the Request Date datetime control in the third row and in the properties panel on the left, clear the request_date variable associated with the control by clicking the X icon beside it. Click Yes in the confirm dialog displayed. Next, click the ellipsis beside the Variable property and create a new datetime variable named **report_date**. Save the variable. Lastly, change the label property to **Report Date**.
4. Leave the Employee Name and Department fields unchanged.
5. Delete the Requested Amount field in the right column of the fourth row.
6. Drag the Amount Advanced field from the Disbursement Details section to the right column of the fourth row where we just deleted the Requested Amount field. Change the display mode property of the Amount Advanced field to Edit and make it required.
7. Leave the Reason for Expenses text area unchanged.
8. Place a new row below the Reason for Expense row by dragging and dropping the empty row at the bottom of the row and above the Disbursement Details section.
9. Place a grid control in the row you just added (it should be the sixth row). In the Create/Select Variable modal, name the variable **expense_grid** and save it. Select the grid and change its

Title property to **Expense Breakdown**. Also change the Layout property to Static.

10. Place a textarea control in the grid and set the id property to **item_description**. Change the label to **Description**, make it required, and set the Rows property to 2 and the column width to 500.
11. Place a checkbox control beside the textarea in the grid and set the id property to **item_receipt**. Change the label to **Receipt Attached?** and the column width to 150.
12. Place a textbox control beside the checkbox in the grid and set the id property to **item_amount**. Change the label to **Amount**, make it required, and change the Function property to Sum and the column width to 150.
13. Place a new row under the grid and set its col-span property to **6 6**.
14. Place a textbox control in the left column of the newly added row. In the Create/Select Variable modal, create a new float variable named **amount_to_reimburse** and save it. Set the label to **Amount to be reimbursed** and make it required.
15. Place a textbox control in the right column of the new row. In the Create/Select Variable modal, create a new float variable named **amount_to_refund** and save it. Set the label to **Amount to be refunded** and make it required.
16. Select the Disbursement Details (Finance) subtitle and change its id property to **expense_subtitle2** and its label to **Reimbursement/Refund Details (Finance)**.
17. Place a textbox control in the left column of the ninth row (the row below the subtitle) and in the Create/Select Variable modal, create a float variable named **amount_reimbursed** and save it. Set the label to **Amount Reimbursed** and the display mode to Disabled.
18. Delete the Date Advanced datetime control in the right column.

19. Place a textbox control in the right column and in the Create/Select Variable modal, create a float variable named **amount_refunded**, and save it. Set the label to **Amount Refunded** and the display mode to Disabled.
20. Place another empty row above the Comments subtitle and set its col-span property to **6 6**.
21. Place a radio control in the left column and in the Create/Select Variable modal create a string variable named `payment_mode` and in the variable settings, create two options as shown here and save it.

Key	Label
CSH	Cash
CHQ	Cheque

Set the label to **Payment Mode** and the display mode to Disabled.

22. Place a textbox control in the right column beside the Payment Mode field and in the Create/Select Variable modal, name the variable **transaction_ref** and save it. Set the label to **Transaction Reference** and its display mode to Disabled.
23. Scroll down to the Signoff/Approval section of the form and select the Requested By field. In the properties panel, clear the `requestor_name` variable associated with the control. Click the ellipsis beside the Variable property, create a new string variable named **prepared_by** and save it. Change the label of the control to **Prepared By**.
24. Select the Date and time field in the same row and clear the `requestor_datetime` variable associated with it. Click the ellipsis, create a new string variable named **preparer_datetime**, and save it.

25. Select the Approved By field and in the properties panel, clear the `approver_name` variable associated with the control. Click the ellipsis, create a new string variable named **approved_by**, and save it.
26. Select the Date and time field in the same row and clear the `approver_datetime` variable associated with it. Click the ellipsis, create a new string variable named **approved_datetime**, and save it.
27. Select the Disbursed By field and in the properties panel, clear the `disbursed_by` variable associated with the control. Click the ellipsis, create a new string variable named **processed_by**, and save it. Change the label to **Processed By**.
28. Select the Date and Time field in the same row and clear the `disbursed_datetime` variable associated with it. Click the ellipsis, create a new string variable named **processed_datetime** and save it.
29. Delete the label control above the Submit button and its containing row.
30. Select the Submit Request button and change the label to Submit Report.

Save your changes and preview the form. It should now look like the following image. Close the preview mode and export the form. Save the Expense Report Form.json file to your system. We will be importing it shortly to create the remaining two forms. Close the Dynaform Designer.

Expense Retirement

Expense Details

Report Date *

Employee Name *

Department *

Amount Advanced *

Reason for Expenses *

Expense Breakdown

[+ New](#)

#	Description *	Receipt Atta...	Amount*
1	<div style="border: 1px solid #ccc; height: 20px; width: 100%;"></div>		<input type="text"/>
Σ : 0			

Amount to be reimbursed *

Amount to be refunded *

Reimbursement/Refund Details (Finance)

Amount Reimbursed

Amount Refunded

Payment Mode Cash Cheque

Transaction Reference

Comments

Enter your comment [Add your comment](#)

Comments

Signoff/Approval

Prepared By

Date and Time

Approved By

Date and Time

Processed By

Date and Time

[Submit Report](#)

Clone the Form

Just as we did with the forms we created for the Cash Advance part of the process, we will create the other two forms by cloning the first and making a few modifications to it. Create a new dynaform with the title **Expense Report Approval Form** and description **This form will be used by the supervisor to approve or reject the expense report.** Save the form and open it in the Dynaform Designer.

Import the Expense Report Form.json file we just exported. With the form successfully imported, we will proceed to make all the fields in the Expense Details section read-only by setting their display mode to Disabled. This is to ensure that the

supervisor cannot change the information entered by the employee submitting the report. Select the following fields in the form and change their display mode property to Disabled:

1. Report Date (datetime).
2. Employee Name (textbox).
3. Department (dropdown).
4. Amount Advanced (textbox).
5. Reason for Expense (textarea).
6. Expense Breakdown (grid): We only have to disable the grid and the child controls will be disabled, since their display mode is set to inherit the parent display mode.
7. Amount to be reimbursed (textbox).
8. Amount to be refunded (textbox).

Next we need to add functionality for the supervisor to be able to approve or reject the report. We will be using the button-and-JavaScript approach.

1. Select the row containing the Submit button at the bottom of the dynaform and change its col-span property to **6 6**.
2. Select the row under it containing the hidden control and change its col-span property to **6 6**.
3. Select the Submit Report button and change its id and name properties to **approve_button** and its label to **Approve Report**.
4. Place a submit control in the right column beside the Approve Report button. Select it and change its id and name properties to **reject_button** and its label to **Reject Report**.
5. Split the row containing the current_user hidden control in to two. Place a hidden control in the right column beside the current_user hidden control. In the Create/Select Variable modal, switch to the Select Variable option and select the is_approved variable.

6. Select the form by clicking the gray border of the dynaform and click the Edit button beside its JavaScript property to open the JavaScript editor. Append the following code after the code already in the editor and click the Save button.

```
//Approve or Reject handler
function approval(action) {
  if (confirm('Are you sure you want to ' + action + '?'))
  {
    if (action === 'APPROVE') {
      $("#is_approved").setValue(1);
      $("form").submit();
    }
    if (action === 'REJECT') {
      $("#is_approved").setValue(0);
      $("form").submit();
    }
  }
  else return false;
}

//Approve form
$("#approve_button").find("button").on("click", function() {
  return approval('APPROVE');
});

//Reject form
$("#reject_button").find("button").on("click", function() {
  return approval('REJECT');
});
```

Click the Preview button to view and save the changes, and the form should display as shown next.

Expense Retirement

Expense Details

Report Date *

Employee Name *

Department *

Amount Advanced *

Reason for Expenses *

Expense Breakdown

Description *	Receipt Atta...	Amount *
1 <input style="width: 95%; height: 20px;" type="text"/>	<input type="checkbox"/>	<input type="text"/>
Σ : 0		<input type="text"/>

Amount to be reimbursed *

Amount to be refunded *

Reimbursement/Refund Details (Finance)

Amount Reimbursed

Amount Refunded

Payment Mode Cash Cheque

Transaction Reference

Comments

Enter your comment

Comments

Signoff/Approval

Prepared By

Date and Time

Approved By

Date and Time

Processed By

Date and Time

Close the preview and close the Dynaform Designer. Create another dynaform with the title **Expense Report Processing Form** and description **This form will be used by the finance officers to process the expense report.** Save it and open it in the designer. Import the Expense Report Form.json file again and make the following modifications to the form.

Disable all the fields in the Expense Details section of the form as we did for the Expense Report Approval Form earlier. Next, change the display mode of the following fields in the Reimbursement/Refund Details section to Edit and make them required.

1. Amount Reimbursed (textbox)
2. Amount Refunded (textbox)

3. Payment Mode (radio)
4. Transaction Reference (textbox)

Preview the form to view and save the changes. Close the preview and close the Dynaform Designer. Our forms are ready, and the next step is to use them as steps in the appropriate tasks.

Assign the Forms to Tasks

In the Process Map, right-click the Report Expense task and select Steps. In the Steps for Task screen, drag the Expense Report Form under Assigned Elements and it should be the first step in the task. Since the Report Expense task can also be a starting task, we will want to set the `case_owner` variable for cases that start at this task and are not a continuation of the Cash Advance Request. To do that, we will set the `SetCaseOwner` trigger to execute before the Expense Report Form.

However, we do not want to execute the trigger if the `case_owner` variable is already set. To ensure that the trigger is only fired if `case_owner` is empty, we add a condition to the trigger. Click the Condition button beside the `SetCaseOwner` trigger in the Before Dynaform section of the Expense Report Form step, and in the modal that appears, enter the following code and click Save.

```
@@case_owner == ''
```

Remember that a trigger or step with a condition will only be executed or performed if the condition evaluates to True. In this case, the `SetCaseOwner` trigger will only be executed if the `case_owner` variable is empty. Triggers with a condition are identified with an asterisk on the Condition button. Close the Steps for Task screen.

Next, right-click the Approve Expense Report task and select Steps. Drag the Expense Report Approval Form under Assigned elements and close the modal. Repeat the same process and assign the Expense Report Processing Form to the Process Expense Report task.

Define the Routing Rule

This part of the process also has a decision gateway after the Approve Expense Report task, and we need to define the conditions for each route. Right-click the Exclusive Gateway element beside the Approve Expense Report task and select properties. Set the condition for the Report Expense route to

```
@@is_approved == '0'
```

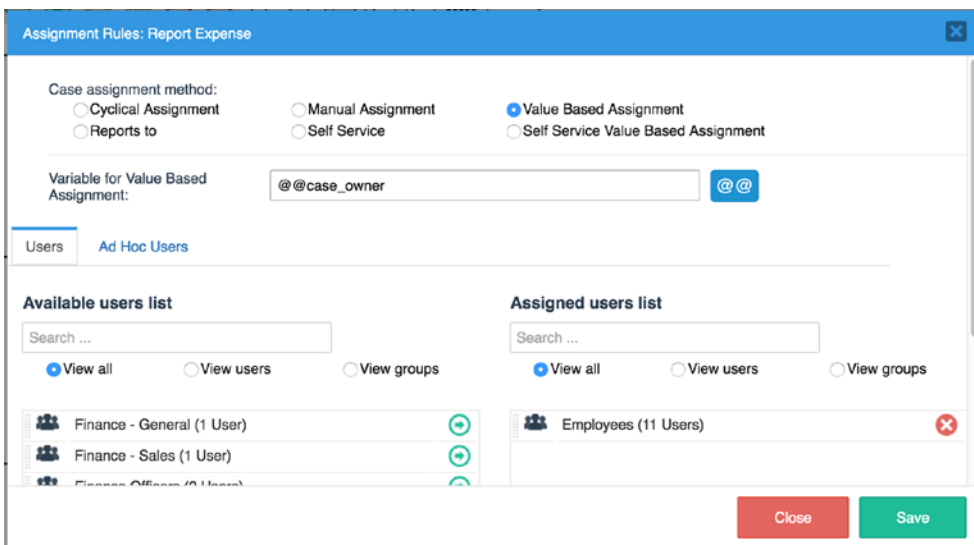
And the condition for the Process Expense Report route to

```
@@is_approved == '1'
```

Click the Save button.

Configure Assignment Rules

Next we define which users can work on each task and how the specific user for each case should be selected. Beginning with the Report Expense task, we will set the Case Assignment method to Value Based Assignment and the variable for value based assignment field to @@case_owner just as we did for the Request Cash Advance task. This is to ensure that the case is always returned to the user who started the case if it is rejected. We also make the task available to all users by assigning the Employees group to the task under Users. Click the Save button.



With the assignment rule for the task set up as shown here, let us proceed to the next task. For the Approve Expense Report task, set the Case Assignment method to Reports to, and under the Users tab, place the Supervisors group in the Assigned users list. Save your changes.

Finally, for the Process Expense Report task, set the case assignment method to Self-Service and in the Users tab, place the Finance Officers group in the Assigned Users list. Save your changes. We now have a process that can be run from start to finish, but we are not yet done. We need to provide a way for the users to attach receipts as supporting documents to their expense reports.

Set Up Receipt Upload

To enable receipt uploads for the process, we first have to create an input document for the purpose. In the Main Toolbox of the Process Map, click the Create (+) icon beside Input Documents and create a new input document with the following properties.

Title	Receipts
Document Type	Digital
Description	Digital (scanned/images) copies of receipts for items listed in the expense report.
Enable Versioning	No
Destination Path	Expense Reports/Case @=APP_NUMBER/Receipts
Tags	receipt
Allowed File Extensions	*.jpg, *.png, *.pdf
Maximum File Size	1000
Unit	KB

With the input document created, we need to assign it to the Report Expense task as a step. Right-click the Report Expense task, select Steps, and drag the Receipts input document to Assigned Elements after the Expense Report Form, making it the second step for the task.

Generate the Expense Report

With our input document all set up, let us configure the output document. As per our requirement, we want to generate a soft copy of the processed report. To do so, we will require an output document. Create a new output document from the Main Toolbox with the following properties.

Title	Expense Report
Filename generated	ExpenseReport-Case-@=APP_NUMBER
Description	Generated expense report for filing
Report Generator	TCPDF
Media	A4
Orientation	Portrait
Margin	Leave all as 20
Output document to generate	Pdf
PDF Security	Disabled
Enable Versioning	No
Destination Path	Expense Reports/Case @=APP_NUMBER
Tags	report
By clicking on the generated file link	Download the file

Once the output document is created, proceed to edit the template by clicking the Open Editor button. We will be getting a bit fancy here. The objective is to replicate the fields on the form in the output document as shown next. We also add a logo and background colors to the subtitles. Can you try to create this output document using the editor? You can get the URL of the logo by right-clicking it in the browser and select Copy Image Address (in Chrome) or Copy Image Location (in Firefox) from the context menu. Also remember to wrap the grid variables with the special syntax.

Alternatively, you can click the HTML button in the editor and paste the following code into the HTML Editor and save your changes. Remember to update the logo URL to match the URL for your installation if different.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
  </head>
  <body>
    <p>
      
    </p>
    <p></p>
    <p><b><font size="5">Expense Report @#APP_NUMBER</font></b></p>
    <table border="0" cellpadding="5" style="width:100%;">
```

```

<tbody>
  <tr style="background-color: #3397e1;">
    <td colspan="4"><font color="#FFFFFF" size="4">
      <b>Expense Details</b></font>
    </td>
  </tr>
  <tr>
    <td><b>Report Date</b></td>
    <td>@=report_date_label</td>
    <td><b>Employee Name</b></td>
    <td>@=employee_name_label</td>
  </tr>
  <tr>
    <td><b>Department</b></td>
    <td>@=department_label</td>
    <td><b>Amount Advanced</b></td>
    <td>@=amount_advanced_label</td>
  </tr>
  <tr>
    <td><b>Reason for Expenses</b></td>
    <td colspan="3">@=expense_reason_label</td>
  </tr>
  <tr>
    <td colspan="4"><b>Expense Breakdown</b></td>
  </tr>
  <tr>
    <td colspan="4">
      <table border="1" cellpadding="5">
        <tbody>
          <tr style="background-color: #eeeeee;">
            <td><b>Description</b></td>
            <td><b>Receipt Attached</b></td>
            <td><b>Amount</b></td>
          </tr>
          <!--@>expense_grid-->

```

```

        <tr>
            <td>@=item_description_label</td>
            <td>@=item_receipt_label</td>
            <td>@=item_amount_label</td>
        </tr>
        <!--@<i>expense_grid</i-->
    </tbody>
</table>
</td>
</tr>
<tr>
    <td><b>Amount to be reimbursed</b></td>
    <td>@=amount_to_reimburse_label</td>
    <td><b>Amount to be refunded</b></td>
    <td>@=amount_to_refund_label</td>
</tr>
<tr style="background-color: #3397e1;">
    <td colspan="4"><font color="#FFFFFF" size="4">
        <b>Reimbursement/Refund Details (Finance)</b></font>
    </td>
</tr>
<tr>
    <td><b>Amount Reimbursed</b></td>
    <td>@=amount_reimbursed_label</td>
    <td><b>Amount Refunded</b></td>
    <td>@=amount_refunded_label</td>
</tr>
<tr>
    <td><b>Payment Mode</b></td>
    <td>@=payment_mode_label</td>
    <td><b>Transaction Reference</b></td>
    <td>@=transaction_ref_label</td>
</tr>
<tr style="background-color: #3397e1;">
    <td colspan="4"><font color="#FFFFFF" size="4">

```

```

        <b>Signoff/Approval/Comments</b></font>
    </td>
</tr>
<tr>
    <td><b>Prepared By</b></td>
    <td>@=prepared_by_label</td>
    <td><b>Date and Time</b></td>
    <td>@=preparer_datetime_label</td>
</tr>
<tr>
    <td><b>Approved By</b></td>
    <td>@=approved_by_label</td>
    <td><b>Date and Time</b></td>
    <td>@=approved_datetime_label</td>
</tr>
<tr>
    <td><b>Processed By</b></td>
    <td>@=processed_by_label</td>
    <td><b>Date and Time</b></td>
    <td>@=processed_datetime_label</td>
</tr>
<tr>
    <td><b>Comments</b></td>
    <td colspan="3">@=comments_label</td>
</tr>
</tbody>
</table>
</body>
</html>

```

With the output document created, we then set it up as a step after the Expense Report Processing Form in the Process Expense Report task. Right-click the Process Expense Report task, select Steps, and drag the Expense Report Output document under Assigned Elements, ensuring that it is the second step.

We now have a complete process that can be run from beginning to end, but it is still missing an important component. We need to be able to update the forms with the name

of the user who performed a specific task and when. This information is automatically logged by ProcessMaker, but we would also like to display it on the form as a kind of signature.

Add Some Triggers

To update the signoff/approval sections of the form, we will use a trigger to get the name of the currently logged-in user and the current date and time, and update the appropriate fields in the form with the values. For the Approval tasks, we will only execute the triggers if the `is_approved` variable is set to true (1). Go ahead and create the following triggers in the process.

Title: UpdateCashAdvanceRequestor

Description: Timestamps the date and time a request is submitted and the name of the requestor.

Code:

```
$data = userInfo(@@USER_LOGGED);
@@requestor_name = $data['firstname'] . ' ' . $data['lastname'];
@@requestor_datetime = getCurrentDate() . ' ' . getCurrentTime();
```

Title: UpdateCashAdvanceApprover

Description: Timestamps the date and time a request is approved and the name of the approver.

Code:

```
$data = userInfo(@@USER_LOGGED);
@@approver_name = $data['firstname'] . ' ' . $data['lastname'];
@@approver_datetime = getCurrentDate() . ' ' . getCurrentTime();
```

Title: UpdateCashAdvanceDisburser

Description: Timestamps the date and time a request is disbursed and the name of the disburser.

Code

```
$data = userInfo(@@USER_LOGGED);
@@disbursed_by = $data['firstname'] . ' ' . $data['lastname'];
@@disbursed_datetime = getCurrentDate() . ' ' . getCurrentTime();
```

Title: UpdateExpenseReportPreparer

Description: Timestamps the date and time a report is prepared and the name of the preparer.

Code

```
$data = userInfo(@@USER_LOGGED);
@@prepared_by = $data['firstname'] . ' ' . $data['lastname'];
@@preparer_datetime = getDate() . ' ' . getTime();
```

Title: UpdateExpenseReportApprover

Description: Timestamps the date and time a report is approved and the name of the approver.

Code

```
$data = userInfo(@@USER_LOGGED);
@@approved_by = $data['firstname'] . ' ' . $data['lastname'];
@@approved_datetime = getDate() . ' ' . getTime();
```

Title: UpdateExpenseReportProcessor

Description: Timestamps the date and time a report is processed and the name of the processor.

Code

```
$data = userInfo(@@USER_LOGGED);
@@processed_by = $data['firstname'] . ' ' . $data['lastname'];
@@processed_datetime = getDate() . ' ' . getTime();
```

You will observe that the triggers are pretty much the same, the only difference being the variables in which we are storing the values. The code uses the built-in `UserInfo` function, which takes the UID of the currently logged-in user to get the details of the user and stores it in a temporary PHP variable `$data`. The result of the function is an array, and we then look up the `firstname` and `lastname` of the user, concatenate them and store it in a case variable (e.g. `processed_by`). Finally, we use the built-in `getDate` and `getTime` functions to get the current date and time, concatenate them and store it in another case variable (`processed_datetime`).

We will also need a trigger to update the value of the `current_user` variable used for the comments on our forms to the `firstname` and `lastname` of the logged-in user for every task. Create another trigger with the following details:

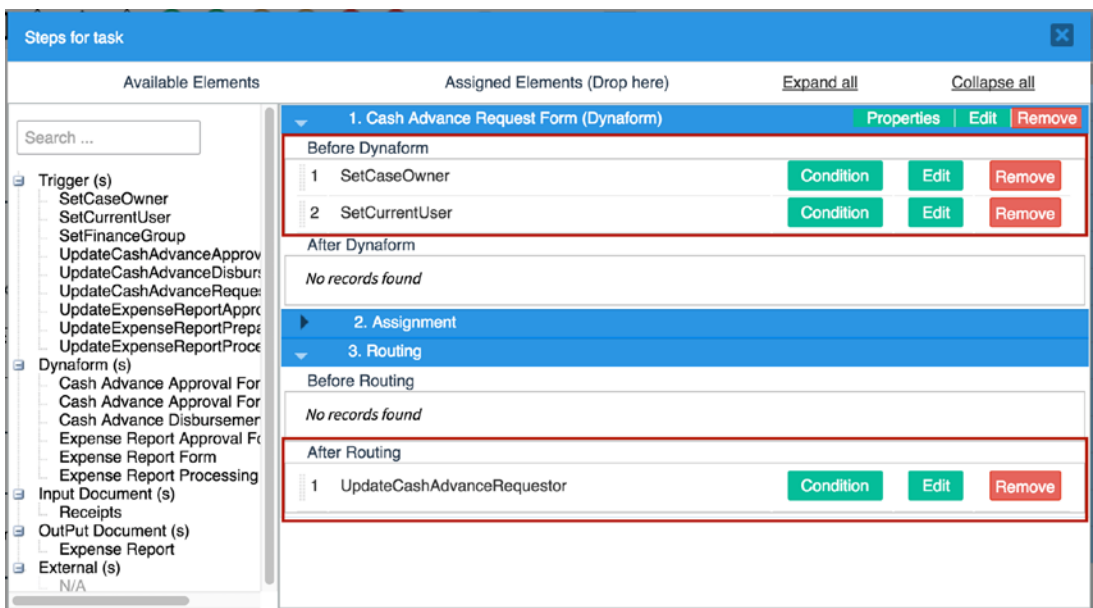
Title: SetCurrentUser

Description: Sets the name of the currently logged on user for use in comments.

Code:

```
$data = userInfo(@@USER_LOGGED);
@@current_user = $data['firstname'] . ' ' . $data['lastname'];
```

With our triggers ready, we need to add them as steps in the respective tasks. Since the triggers are signoff triggers, we want them to be executed as the last step in the task, so we will place them to execute After Routing. Right-click the Request Cash Advance task and place the SetCurrentUser trigger in the Before Dynaform step and the UpdateCashAdvanceRequestor trigger in the After Routing step as shown next.



Next, for the Approve Advance task, place the SetCurrentUser trigger in the Before Dynaform step of the two dynaforms. Remember that we have two dynaforms for this task, and either of them could be used, so in order to ensure that the trigger is executed for both forms, we have to place it in both Before Dynaform steps.

Then place the UpdateCashAdvanceApprover trigger in the After Routing step of the Approve Advance task. Because we want to update the approver only if the advance was approved, click the Condition button beside the UpdateCashAdvanceApprover trigger, enter the following condition, and save it:

```
@@is_approved == '1'
```

Next, we place the `SetCurrentUser` trigger in the Before Dynaform step and the `UpdateCashAdvancedDisburser` trigger in the After Routing step of the Disburse Advance task.

Moving to the Expense Reporting tasks, place the `SetCurrentUser` trigger in the Before Dynaform step and the `UpdateExpenseReportPreparer` trigger in the After Routing step of Report Expense task.

Just as we did with the Cash Advance Approval, place the `SetCurrentUser` trigger in the Before Dynaform step and the `UpdateExpenseReportApprover` trigger in the After Routing task of the Approve Expense Report task and set the following condition for the `UpdateExpenseReportApprover` trigger.

```
@@is_approved == '1'
```

Finally, place the `SetCurrentUser` trigger in the Before Dynaform step and the `UpdateExpenseReportProcessor` trigger in the After Dynaform step of the Process Expense Report task.

We are placing the trigger after the form because we want the name of the processor to be displayed on the output document generated in the next step. And we are done! That was quite repetitive, but on the brighter side, we are now ready to give our process another spin and this time, run the cases to completion.

Test the Changes

Returning to where we left off at the end of the first part of the process, log in in another browser as Amy Alexander and open one of the cases in the inbox with the Disburse Advance task. If you scroll to the bottom of the form, you will see that the signoff/approval section has not been updated. We have, however, fixed this with the triggers we just created. To confirm that our fix works, let us run a new case from the beginning.

Log out as Amy Alexander and log in as Justin Sanchez and open one of the returned cases in the inbox. Alternatively, you can also start a new Cash Advance and Expense Retirement (Request Advance) case. With the case open, ensure that all required fields are filled. Also set the department to Sales (Justin Sanchez is in the Sales department, and it would be nice to automatically set this without the user filling it out.). Type a comment in the field labeled “Enter your comment” and click the Add Your Comment

button beside it. The comment should be added to the Comments box with the name of the current user (Justin Sanchez in this case) prefixed to it, as shown here. This demonstrates that our `SetCurrentUser` trigger works as expected.

Comments

Enter your comment

Add your comment

Comments Justin Sanchez: Testing current user

First test passed. Submit the request, click the Continue button on the routing screen, and log out as Justin Sanchez. Log in as Julia Smith, the supervisor, and open the case just sent by Justin Sanchez in the inbox. Scroll to the bottom of the form and you should see that the Requested By field in the Signoff/Approval section has been updated.

Signoff/Approval

Requested By	JUSTIN SANCHEZ	Date and Time	2016-10-11 07:57:31
Approved By		Date and Time	
Disbursed By		Date and Time	

Enter a comment stating a reason for rejection and reject the request to send it back to Justin Sanchez. Log out as Julia Smith. We want to test that the Approved By field is not updated when the request is rejected. Log in again as Justin Sanchez and open the case that was rejected by Julia Smith. Scroll to the bottom, and the Approved By field should still be empty and in the comments section, we should see the rejection reason entered by the supervisor as shown next. Second test passed. Submit the request again and log out as Justin Sanchez.

Comments

Enter your comment

Add your comment

Comments Julia Smith: Testing rejection
Justin Sanchez: Testing current user

Signoff/Approval

Requested By	JUSTIN SANCHEZ	Date and Time	2016-10-11 08:18:26
Approved By		Date and Time	
Disbursed By		Date and Time	

Log in as Julia Smith, open the case from the inbox, enter some comments and this time, approve the request. The request should be routed to the Finance - Sales group as we configured earlier. Log out as Julia Smith and log in as Amy Alexander.

Go to the Unassigned Cases list and claim the case just sent by Julia Smith. Once the case opens, scroll to the bottom and you should see that the Approved By field is now updated when the request is approved. This shows that the condition set on the trigger works as expected. Fill in the required fields (Amount Advanced and Date Advanced), add comments, and submit the request. The routing screen should show that the case is now being routed back to Justin Sanchez to fill out an expense report. Click the Continue button and log out Amy Alexander.

So, we assume Justin Sanchez has received the advance, made some expenses and is now ready to submit an expense report. Log in as Justin Sanchez, go to the inbox, and open the case with the task set to Report Expense sent by Amy Alexander. We should see the Expense Report Form we designed earlier. You will also notice that some of the fields have been prefilled with the details from the Cash Advance such as the Amount Advanced and the Reason for Expenses. However, the user can edit these fields and make changes, which should not be the case. We will fix that when we explore ways to enhance the process in the next chapter. For now, we will trust that Justin will not alter any figures.

Fill in the remaining required fields in the Expense Details section of the form, entering a couple of values in the Expense Breakdown grid. The total amount is automatically calculated and displayed. If the total amount is less than amount advanced, then Justin Sanchez should fill the difference in the Amount to be refunded field, else the difference should be entered in the Amount to be reimbursed field. You might be wondering if this calculation should not be done automatically to avoid errors, and you are right. We will enhance our forms later to do that.

Expense Retirement

Expense Details

Report Date * 2016-10-11 Employee Name * Justin Sanchez

Department * Sales Amount Advanced * 7000

Reason for Expenses * Testing cyclical assignment case 6

Expense Breakdown

+ New

Description *	Receipt Attache...	Amount *
1 Lunch with client at Special Restaurant	<input checked="" type="checkbox"/>	5000
2 Taxi to lunch	<input checked="" type="checkbox"/>	1500
		Σ: 6500

Amount to be reimbursed * 0 Amount to be refunded * 500

With the fields filled in, add some comments if desired and submit the report. We should now see the Receipts upload screen. This shows that the configured input document step works as expected. Go ahead and upload some receipts and add the comments as shown next.

[Previous Step](#) [Next Step](#)

Receipts
Input Document

Digital (scanned/images) copies of receipts for items listed in the expense report.

Attach

Title	Version	Creator	Comment	Created Date	
receipt.pdf	1	Justin Sanchez	Lunch receipt	2016-10-11	Download
taxi receipt.jpg	1	Justin Sanchez	Taxi receipt	2016-10-11	Download

Rows 1-2/2 Page 1/1

[Next Step](#)

Even though a user may have indicated that receipts were attached in the grid, the user can move past this step without uploading any documents. We could use a trigger to check whether any input documents have been uploaded and show an error message. Alternatively, we could add the input document as file controls to the grid and use some JavaScript to check if a file is uploaded for rows where the receipt uploaded checkbox is checked. All that, however, is outside the scope of this book. Click the Next Step button, and the case should be routed to the supervisor to Approve the report. Click Continue and log out as Justin Sanchez.

Before we approve the report, you will recall that we also made the Report Expense task a starting task. This is to allow users to report out-of-pocket expenses for which an Advance was not given. Let us log in as Wanda Marshall in HR and report such an expense. Once logged in, start a new Cash Advance and Expense Retirement (Report Expense) case. We can see that the Expense Retirement form is displayed, just like the one we saw for the continued cash advance, with the only difference being that the fields from the cash advance are not pre-filled. Leave the form unfilled for now and log out as Wanda Marshall. We will return to complete the form later when we have added the enhancements.

Log in as Julia Smith and open the case with the Approve Expense Report task sent by Justin Sanchez. The Expense Report Approval form is displayed with buttons to approve or reject the report at the bottom. To view the uploaded documents, click the Information button at the top of the case and select Uploaded Documents in the dropdown. This displays the uploaded documents tab, but—Surprise!—there are no uploaded documents. This is because the supervisor does not have permission to view the documents. We will remedy this later when we explore *case permissions*. Go ahead and close the Uploaded documents tab and approve the report. The routing screen should show that the case is being routed to the Process Expense Report task and is unassigned. Click Continue and log out as Julia Smith.

Finally, log in as Amy Alexander and claim the case from the Unassigned cases list. We are assuming that the Finance officer will have posted the transaction on the accounting system and then filled out the transaction reference from the accounting system on the form. Go ahead and fill in the required fields for the Finance officer: Amount Reimbursed, Amount Refunded, Payment Mode, and Transaction Reference. You can enter comments if desired and click the Submit button. The Output Document screen for our generated expense report is displayed as shown here.

Previous Step Next Step

Output document Expense Report

Description Generated expense report for filing

Create Date 2016-10-11 12:08:15

File (.pdf) [Open](#)

Next Step

Click the Open link to download the generated PDF file. The generated output document should look like the following image.



Expense Report 6

Expense Details

Report Date 2016-10-11 **Employee Name** Justin Sanchez
Department Sales **Amount Advanced** 7000
Reason for Expenses Testing cyclical assignment case 6

Expense Breakdown

Description	Receipt Attached	Amount
Lunch with client at Special Restaurant	true	5000
Taxi to lunch	true	1500

Amount to be reimbursed 0 **Amount to be refunded** 500

Reimbursement/Refund Details (Finance)

Amount Reimbursed 0 **Amount Refunded** 500
Payment Mode Cash **Transaction Reference** PV-001

Signoff/Approval/Comments

Prepared By JUSTIN SANCHEZ **Date and Time** 2016-10-11 09:34:43
Approved By JULIA SMITH **Date and Time** 2016-10-11 10:43:14
Processed By Amy Alexander **Date and Time** 2016-10-11 12:05:48

Comments
 Amy Alexander: Cash received from Justin Sanchez
 Julia Smith: Okay
 Justin Sanchez: Balance will be refunded as cash.
 Amy Alexander: Please pick up cash at finance office
 Julia Smith: Okay to treat
 Julia Smith: Testing rejection
 Justin Sanchez: Testing current user

Click the Next Step and then Finish buttons to complete the process. The generated and uploaded documents for the case should also have been successfully saved to the DMS, and if you click the Documents menu in the left pane, you should see a new Expense Reports folder, which when expanded should show a folder with the case number containing the generated document and the uploaded receipts in a receipts folder.

We have now successfully run the Cash Advance and Expense Retirement process from beginning to the end. At this point, you should be able to think of some processes you can try your hands at to practice what you have learned so far. Although the process we have built is complete, it could use a number of improvements, and I am sure you can think of some already. In the next chapter, we will add the improvements we hinted at when testing and more.

CHAPTER 15

Enhancing the Process

Now that we have a complete Cash Advance and Expense Retirement process designed on ProcessMaker, there has been an eager adoption of the process as employees are able to make requests and report expenses from the comfort of their seats without having to carry forms around seeking approvals. We have, however, received some feedback on what users feel are the shortcomings of the new process. Some of the feedback follows.

Feedback

Feedback came from Finance Officers, supervisors, and employees.

Finance Officers

We have no way of knowing a particular employee's request just by looking at the cases in our queue, and when there is a need to treat a specific request urgently, we have to open each case one after the other to find the exact case. We would like to be able to know the owner of a request and the amount without having to claim or open the request.

Also, when disbursing cash advances, some of our officers mistakenly pick a date in the future as date advanced. We would like to ensure that an officer cannot select a future date. Lastly, some employees reporting expense on a cash advance make changes to the expense reason and amount advanced, making it difficult to reconcile. Can we make the fields carried over from the cash advance request immutable in the expense report form?

Supervisors

We would like to be notified when a case is assigned to us. We often have our direct reports calling us to let us know that there is a case they need us to approve. Also, when reviewing expense reports, we are unable to see the documents uploaded by the employee.

We would also like to know if it is possible to send a message to the requesting employee without having to send back the request or report. Sometimes we just need some clarification, and it would be great if the correspondence can be linked to the case, just like the comments on the form.

Employees

It would be nice if some of the fields, such as employee name and department, can be automatically filled when making a request or submitting a report. Also, when submitting the expense report, can the calculations for reimbursement/refund be done automatically? Finally, we notice that Finance officers sometimes do not claim cases on time. We would like an escalation email to be sent to the Finance manager if a case is not claimed by an officer within the stipulated SLA (Service Level Agreement).

Reviewing the feedback from our users, we can distill the required improvements to the process into the following areas:

1. Case labels: adding information to cases in the case list
2. Email notifications: notifying the next assigned user
3. Prefilling form fields with triggers
4. Setting datetime control properties
5. Dynaform logic in JavaScript: performing calculations and conditionally disabling fields
6. Case Permissions and Case Notes
7. Escalating unclaimed cases

Let us proceed to look at these areas in detail and address the feedback from our users. We will add the enhancements first and when done, test them together.

Case Labels

Case labels are used to add customizable title and description to cases to make them easy to identify in a list of cases. The label of a case can change over the life of the case, but is often left unchanged for most processes. In our Cash Advance and Expense Retirement process, we will set an initial case label for the Cash Advance part of the process and then change it for the Expense Report part.

Let us set the case label for the Cash Advance request. Right-click on the Request Advance task and select the Properties option. In the Activity properties screen, select the Case Label tab as shown here.

The screenshot shows the 'Activity Properties' dialog box with the 'Case Labels' tab selected. The 'Title' field is populated with 'Cash Advance Request for @=amount_requested by @=employe' and the 'Description' field is populated with '@=employee_name is requesting @=amount_requested for this purpose: @@expense_reason'. Both fields have a blue '@@' icon to their right. The dialog box has a blue header with a close button, a left sidebar with navigation options (Definitions, Case Labels, Timing Control, Notifications), and a bottom bar with 'Cancel' and 'Save' buttons.

Enter the following values for the title and description and click the Save button:

Title: Cash Advance Request for @=amount_requested by @=employee_name

Description: @=employee_name is requesting @=amount_requested for this purpose:
@@expense_reason

Now when a Cash Advance request is submitted, we should be able to see the name of the employee and amount requested directly from the case list. Next, let us update the case label for the Expense Report part of the process. Right-click on the Report Expense task and select the Properties option. In the Activity properties screen, select the Case Label tab and enter the following values for the title and description:

Title: Expense Report for @=employee_name

Description: Report on expense incurred for the following purpose prepared by @=employee_name:@=expense_reason Click the Save button.

Email Notifications

We truly cannot automate business processes without email notifications. We need to be able to inform users when cases are assigned to them so that they can log in and treat the cases. The Enterprise edition of ProcessMaker also has plugins integrating ProcessMaker directly with email clients like GMail, Outlook, and Zimbra that allow users to see and treat cases assigned to them directly from their email clients.

There are two ways we can send email notifications to users assigned to a task. The first is to use the Notification property of the task, and the second is using a trigger that executes the built-in `PMFSendMessage` function. We shall explore both approaches.

Using the Task Notification Property

The simplest way to send a notification to a user assigned to a task is to use the Notification property of the previous task. Let us set up a notification message to the supervisor using this approach. Right-click the Request Advance task and select properties. In the Activity properties screen, select the Notifications tab and check the “After routing notify the next assigned user(s)” field to display the settings as shown next.

The screenshot shows the 'Activity Properties' dialog box with the 'Notifications' tab selected. At the top, there is a checkbox labeled 'After routing notify the next assigned user(s)' which is checked. Below this, the 'Email From Format' is set to 'Assigned User'. The 'Subject*' field contains 'Insert a title' and has a blue '@@' icon to its right. The 'Content Type' is set to 'Plain Text'. The 'Message*' field contains 'Insert a message' and also has a blue '@@' icon to its right. At the bottom of the dialog, there is a note: 'Fields marked with asterisk (*) are required.' The 'Cancel' button is red and the 'Save' button is green.

The first field, Email From Format, allows us select which user the notification email should be sent from. The options are Assigned User and Email Account Settings. If set to Assigned User, the email notification sender will be the user currently assigned to the task, while Email Account Setting will use the email address we set up in the SMTP configuration. We will set it to Assigned User so the supervisor can know which employee the notification is from.

The message content can either be plain text or HTML. If using HTML, you will need to set up a template for the message. Let us use plain text for now. Leave the Content Type as Plain Text and enter the following values for the Subject and Message:

Subject: Cash Advance Request for @=amount_requested by @=employee_name

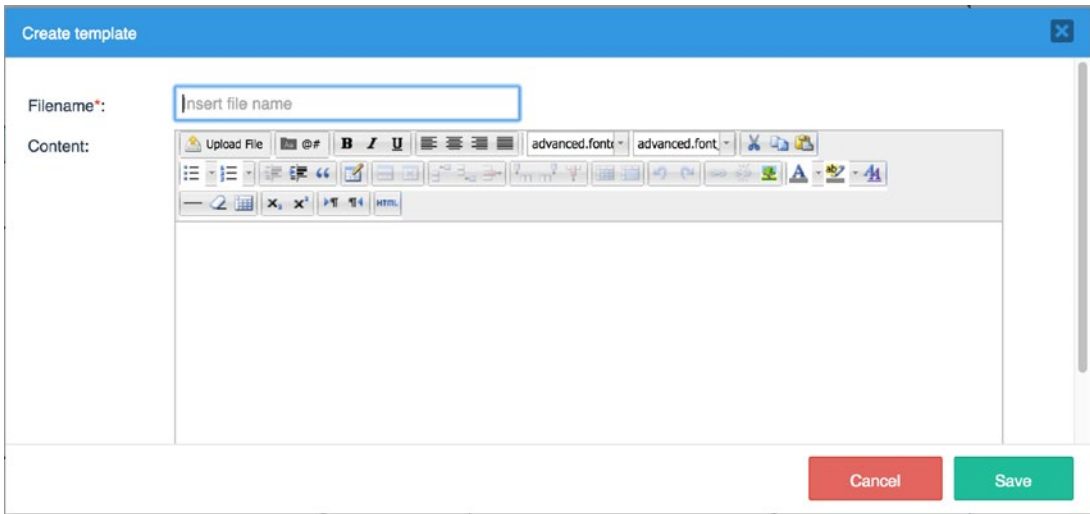
Message: A Cash Advance Request for @=amount_requested by @=employee_name has been assigned to you for approval on ProcessMaker. Please log in to treat.

You can see how we have used the case variable selector to add variables to be interpolated into the subject and message of the notification. Click the Save button. Now when a Cash Advance Request is submitted, a notification will automatically be sent to the supervisor's email.

Creating a Template for Email Notification

The notification message shown earlier is very basic, which is why we can use plain text. If we had a requirement to include the company’s logo in the message or details of a grid in the notification message, we would not be able to do so with a plain text message. We can use an HTML template for this purpose; however, we must create the template first.

To create a template, click the Create (+) icon beside Templates in the Main Toolbox, and the Create Template screen is displayed as shown next.



It is very similar to the editor we used when creating the template for the output document earlier. Let us create a template for notifying the supervisor of an expense report containing the expense breakdown. In the Filename field, enter **ExpenseReportNotifySupervisor**. Click the HTML button in the Content toolbar and paste in the following code, which displays the grid in a message to the supervisor.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
  </head>
  <body>
    <p>An Expense Report by @=employee_name has been assigned to you.</p>
    <p><b>Expense Breakdown</b></p>
```

```

<table border="1" cellpadding="5">
  <tbody>
    <tr style="background-color: #eeeeee;">
      <td><b>Description</b></td>
      <td><b>Receipt Attached</b></td>
      <td><b>Amount</b></td>
    </tr>
    <!--@>expense_grid-->
    <tr>
      <td>@=item_description_label</td>
      <td>@=item_receipt_label</td>
      <td>@=item_amount_label</td>
    </tr>
    <!--@<expense_grid-->
  </tbody>
</table>
<p>Please log in to ProcessMaker to treat.</p>
</body>
</html>

```

Click the Update button and then save the template. The list of templates is displayed as shown next. Since it is basically an HTML file, the file can be downloaded by clicking the Download button beside a template. The Upload button can also be used to create additional templates by uploading an HTML file with the content of the message.



If using an image in a template, the URL of the image must be publicly available on the Internet for the image to be rendered in the email message.

With the template created, let us use it for the notification sent to the supervisor when an expense is reported. Right-click the Report Expense task and select Properties. In the Activity Properties screen, select the Notifications tab and check the “After routing

notify the next assigned user(s)” field. Enter the following value for the fields and click the Save button.

- Email From Format:** Assigned User
- Subject:** Expense Report for @=employee_name
- Content Type:** HTML Template
- Template:** ExpenseReportNotifySupervisor.html

Using PMFSendMessage in a Trigger

We sometimes have situations where the notification property of a task might not suffice, such as if we need to send different messages based on the path a case is routed. For example, we might want to send a different message to the requesting employee when a request is rejected and another message to the Finance officers when a request is approved. In this scenario, we can use triggers to send the notifications and determine which trigger to execute after routing the case by evaluating conditions.

The PMFSendMessage function (learn more about the parameters of the function here: http://wiki.processmaker.com/3.0/ProcessMaker_Functions#PMFSendMessage.28.29) requires a template for the message content, so we must first create two templates—one for the rejected request and another for the approved request. Go ahead and create two templates with the following details. You do not need to use the HTML button, since we are not working with grids. You can use the WYSIWYG editor to format as desired.

Filename	Content
CashAdvanceNotifyRejection	Your Cash Advance Request for @=amount_requested has been rejected by your manager. Please log in to ProcessMaker to view.
CashAdvanceNotifyFinance	A Cash Advance Request for @=amount_requested by @=employee_name has been approved and assigned to your group for disbursement on ProcessMaker. Please log in to claim case and treat.

Next, we need to create the triggers to send the message. We will be using two separate triggers for simplicity and use conditions to determine which trigger to execute, but the logic can also be combined into a single trigger. Create two triggers; details for the first are shown here:

Title: NotifyCashAdvanceRejection

Description: Send a message to user when cash advance is rejected

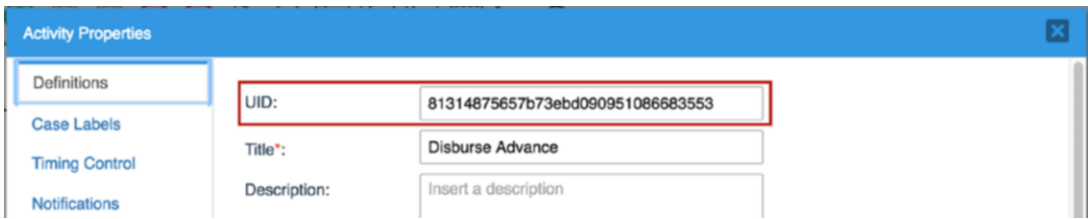
Code:

```
if (isset(@@case_owner) and !empty(@@case_owner)) {
    $aUserTo = userInfo(@@case_owner);
    $aUserFrom = userInfo(@@USER_LOGGED);
    PMFSendMessage(@@APPLICATION, $aUserFrom['mail'], $aUserTo['mail'], '',
        '', 'Cash Advance Request Rejected', 'CashAdvanceNotifyRejection.html');
}
```

This code checks to see if we have a value stored in the `case_owner` variable (you will recall that the variable stores the UID of the user that started the case). If the variable is set, we use the `userInfo` function to get the user's details and store the information in the `$aUserTo` variable. We also get the details of the currently logged-in user (which would be the supervisor) and store them in the `$aUserFrom` variable.

We then call the `PMFSendMessage` function and pass the following parameters: the case UID (`@@APPLICATION`), the sender email address (`$aUserFrom['mail']`), the recipient email address (`$aUserTo['mail']`), the CC and BCC are set to empty, the subject of the message ('Cash Advance Request Rejected') and the name of the template containing the message to be sent ('CashAdvanceNotifyRejection.html'). The sender email address could also have been set to a default email address, such as `processmaker@domainname.com` or `notifications@domainname.com` if such addresses exist.

For the second trigger, we will need the UID of the Disburse Advance task to get a list of users assigned to the task dynamically from the database. To get the UID, right-click the Disburse Advance task and select Properties. In the Activity properties screen the UID of the task is displayed as shown next. Copy the UID and paste it in a text editor for reference later. Close the screen and proceed to create the trigger.



In the following trigger code, replace the value of the `$taskId` variable (81314875657b73ebd090951086683553) with the UID you just copied and pasted in the text editor before saving the trigger.

Title: NotifyCashAdvanceApproval

Description: Send a message to users in finance officers group when cash advance is approved

Code:

```
$taskId = "81314875657b73ebd090951086683553";

$assignedUsers = array();

$userQuery = "SELECT USR_UID FROM TASK_USER
WHERE TAS_UID = '$taskId' AND TU_RELATION = 1";

$groupQuery = "SELECT GU.USR_UID FROM GROUP_USER GU, TASK_USER TU
WHERE TU.TAS_UID = '$taskId' AND TU.TU_RELATION = 2
AND TU.USR_UID = GU.GRP_UID";

$users = executeQuery($userQuery);

if (is_array($users) and count($users) > 0) {
    foreach ($users as $user)
        $assignedUsers[] = $user['USR_UID'];
}

$users = executeQuery($groupQuery);

if (is_array($users) and count($users) > 0) {
    foreach ($users as $user)
```

```

        $assignedUsers[] = $user['USR_UID'];
    }
$assignedUsers = array_unique($assignedUsers);

$emailTo = "";

foreach ($assignedUsers as $assignedUser) {
    $aUser = userInfo($assignedUser);
    $emailTo .= (empty($emailTo) ? "" : "," ) . $aUser['mail'];
}
$aUserFrom = userInfo(@@USER_LOGGED);

if (!empty($emailTo)) {
    PMSendMessage(@@APPLICATION, $aUserFrom['mail'], $emailTo, '', '',
    "New Cash Advance Request case to be claimed", 'CashAdvanceNotifyFinance.
    html');
}

```

Now a quick walkthrough of the code. We begin by storing the UID of the Disburse Advance task in the `$taskId` variable. Then we create an empty array and store it in `$assignedUsers` variable. Then we compose an SQL query to look up the UID of all users that have been assigned to the task and store the query in the `$userQuery` variable. We also compose a similar query to get the UIDs of the users in the groups assigned to the task and store them in the `$groupQuery` variable.

We then use the `executeQuery` function to execute the query against the ProcessMaker database and store the result in the `$users` variable. We check to see if there are any records returned by the database query; if any, we loop through the results and store the UIDs in the `$assignedUsers` array we created in line 2. We then repeat the process for the `$groupQuery`, also storing the resulting UIDs in the `$assignedUsers` variable.

Now that we have all the UIDs, we use the PHP function `array_unique` to eliminate duplicates in the array. We then create a new variable `$emailTo` and assign it an empty string. Next we loop through the UIDs stored in the `$assignedUsers` array and use the ProcessMaker `userInfo` function to get the details of each of the users and storing it in the `$aUser` variable. Then we check whether the `$emailTo` variable is empty using the ternary operator (`?`). If it is empty, we append an empty string to it before concatenating

it with the user's email; if not, we append a semicolon, which serves as a separator for the email addresses.

With all the assigned users' email addresses extracted, we get the details of the currently logged-in user and store them in `$aUserFrom`. Finally, we check to see we have email addresses in the `$emailTo` variable and if so, use the `PMFSendMessage` function to send a message to the users, as we did in the previous trigger.

The code in the triggers just shown is adapted from the ProcessMaker wiki, and you can learn more about notifications in ProcessMaker and explore different triggers that can be used in different scenarios here: <http://wiki.processmaker.com/3.0/Notifications>.

Assign the Triggers to Tasks

To use the triggers, we have to assign them to the appropriate task. Right-click the Approve Advance task and place the `NotifyCashAdvanceApproval` and `NotifyCashAdvanceRejection` triggers in the After Routing step of the task. The `NotifyCashAdvanceApproval` trigger should only be executed if the request was approved by the supervisor, so we need to set the condition accordingly. Click the Condition button for `NotifyCashAdvanceApproval` and set the condition to

```
@@is_approved == '1'
```

Save your changes. Repeat the same process for the `NotifyCashAdvanceRejection` trigger, setting the condition to

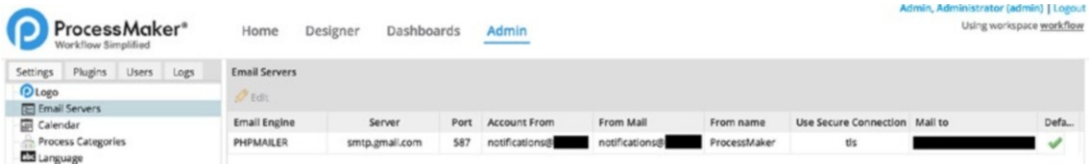
```
@@is_approved == '0'
```

and save your changes.

Check that Email Sending is Configured

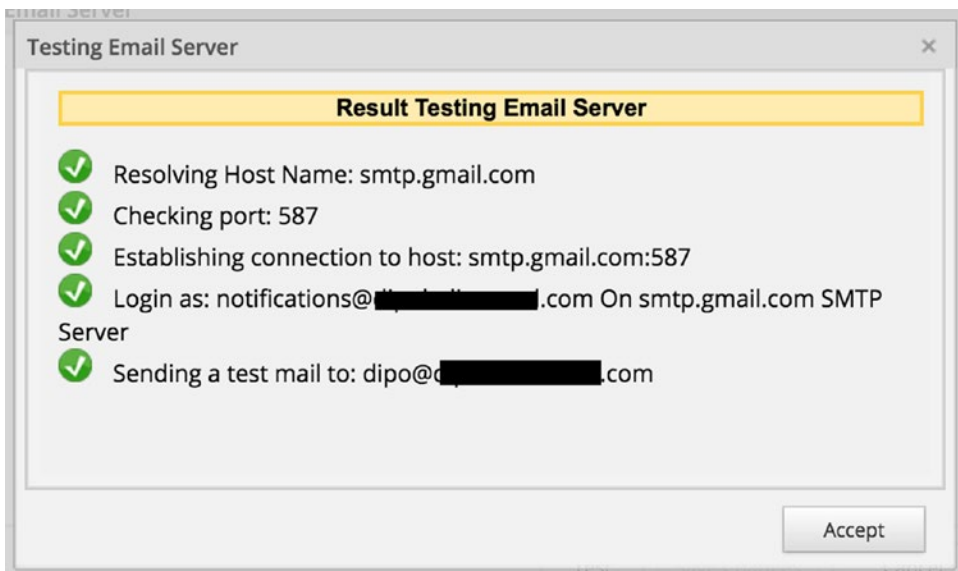
In concluding our discussion of email notifications, we need to verify that email sending is configured and working in ProcessMaker to ensure that the messages will be sent.

Click on Admin in the main menu and in the Settings tab, select Email Servers. If you configured SMTP settings during the installation, then you should have an email server configured similar to the image shown here; otherwise, the fields will be blank.



Click the Edit button and if you do not have a server configured, proceed to fill in the details in the Edit Email Server screen displayed. The ProcessMaker wiki has a detailed explanation of the settings and examples of configuring the common Email providers—Google, Yahoo, Hotmail and Outlook—at http://wiki.processmaker.com/3.0/Email_Settings.

Next click the Test button at the bottom of the form. If the email settings are properly configured, you should see a success message similar to the following. Your system must be connected to the Internet for emails to be successfully sent.



Click the Accept button and Save Changes. You should receive a test email at the address used for the test. And that is a wrap. Let us move on to the next enhancement to our process.

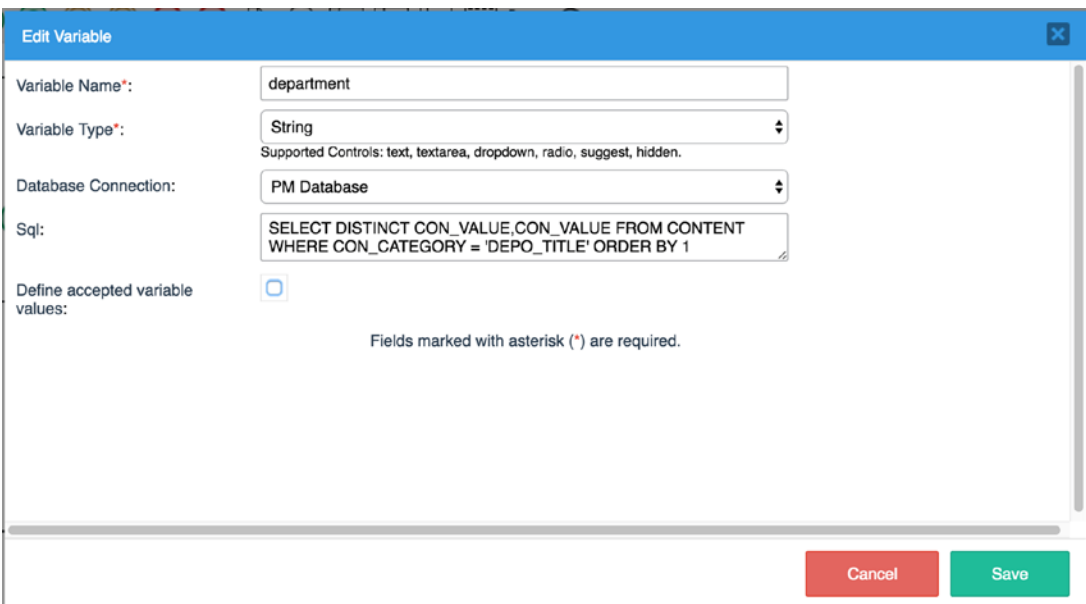
Prefilling Form Fields with Triggers

The next improvement we want to make to the process is to save employees some time when filling in the Cash Advance Request form or the Expense Report Form by prefilling the following fields:

- Request/Report Date (defaults to current date)
- Employee Name (defaults to currently logged in user’s firstname and lastname)
- Department (defaults to current user’s department)

To achieve this, we will create a trigger that executes before the form is loaded and looks up these values and stores them in the corresponding variables. We will, however, need to make a change to the department variable to use an SQL query to fetch the list of department instead of defining them statically in the list of accepted values. Click Variables in the Main Toolbox to display the list of variables and search for the department variable. Click the Edit button beside it and in the Edit Variable screen, paste the following SQL query in the Sql field and uncheck the “Define accepted variable values” field as shown in the screen and save your changes.

```
SELECT DISTINCT CON_VALUE, CON_VALUE FROM CONTENT  
WHERE CON_CATEGORY = 'DEPO_TITLE' ORDER BY 1
```



This SQL query simply selects the distinct name of departments from the database table where they are stored (CONTENT). We are selecting the name (CON_VALUE) twice because we are using it as both the Key and the Label for the field. Ideally, we should select the UID of the department as the key and the name of the department as the label so that if the name of the department is edited, the value stored in a case would still be valid. We are, however, using the name as the key to be consistent with the initial design and not affect cases that are already running.

With that out of the way, we are now ready to create the triggers. Click the Create (+) icon beside Triggers in the Main Toolbox and create the following custom trigger.

Title: SetCashAdvanceInitialData

Description: Prefills the request date, employee name and employee department in the Cash Advance Request form

Code:

```
@@request_date    =    getCurrentDate();
$data = PMFInformationUser(@@USER_LOGGED);
@@employee_name = $data['firstname'] . ' ' . $data['lastname'];
$query = "SELECT DISTINCT CON_VALUE FROM CONTENT
WHERE CON_CATEGORY = 'DEPO_TITLE'
AND CON_ID = '" . $data['department'] ."'";
$aDepartment    =    executeQuery($query);
@@department     =    $aDepartment[1]['CON_VALUE'];
```

The trigger code here uses the built-in ProcessMaker function `getCurrentDate` to get the current date and stores the value in the `request_date` variable. We then use the `PMFInformationUser` function to get the details of the currently logged-in user. The function returns an array of the user's information.

We then concatenate the `firstname` and `lastname` and store the result in the `employee_name` variable. Next, we compose an SQL query to look up the name of the user's department and execute the query, storing the result in `$aDepartment`. Finally, we store the returned result in the `department` variable.

The second trigger for prefilling the expense report fields we will create is similar, but we will only look up the employee name and department if the case is not a continuation of a cash advance. We can determine whether a case is a continuation by checking the

value of a variable that can only be set during the Cash Advance part of the process, such as the `disbursed_by` variable. Let us create the trigger with these details:

Title: SetExpenseReportInitialData

Description: Pre-fills the report date, employee name and employee department in the Expense Report form.

Code:

```
@@report_date = getCurrentDate();

if(@@disbursed_by == ""){
    $data = PMFInformationUser(@@USER_LOGGED);
    @@employee_name = $data['firstname'] . ' ' . $data['lastname'];

    $query = "SELECT DISTINCT CON_VALUE FROM CONTENT
    WHERE CON_CATEGORY = 'DEPO_TITLE'
    AND CON_ID = '" . $data['department'] . "'";

    $aDepartment = executeQuery($query);

    @@department = $aDepartment[1]['CON_VALUE'];
}
```

With the triggers created, proceed to add them to the respective tasks. Place the `SetCashAdvanceInitialData` trigger in the Before Dynaform step of the Cash Advance Request Form in the Request Advance task.

Next, place the `SetExpenseReportInitialData` trigger in the Before Dynaform step of the Expense Report Form of the Report Expense task. And that's a wrap. Now when a user starts a Cash Advance and Expense Retirement case, the request/report date, employee name and department will be automatically prefilled.

We can take this a step further by making these fields noneditable, so that the user does not make any changes to what the system automatically generates. Edit the Cash Advance Request Form and change the display mode property of the following fields to Disabled:

- Request Date
- Employee Name
- Department

Also edit the Expense Report Form, changing the display mode property of the following fields to Disabled:

- Report Date
- Employee Name
- Department

Setting Datetime Control Properties

The next enhancement we want to add to our process is the request from the Finance department to enforce that Finance officers cannot choose a future date for disbursement. To achieve this, we need to set the max date property for the Date Advanced field to the current date. We will create a trigger that stores the current date in a `max_date` variable and execute it before the Cash Advance Disbursement Form in the Disburse Advance task. We will also set the max date property of the Cash Advance Disbursement Form to the `max_date` variable.

Create a new trigger with the details shown here:

Title: SetMaxDisburseDate

Description: Sets the value of the `max_date` variable to limit the date value for Date Advanced.

Code: `@@max_date = getCurrentDate();`

With the trigger created, edit the Cash Advance Disbursement Form, select the Date Advanced field and set its max date property to `@@max_date` as shown next and save your changes.

min date	<input type="text"/>
max date	<input type="text" value="@@max_date"/>
initial selection date	<input type="text" value="true"/>

Finally, place the SetMaxDisburseDate trigger in the Before Dynaform step of the Cash Advance Disbursement Form in the Disburse Advance task. That's all.

Dynaform Logic in JavaScript

Our next task is to automatically calculate the values for the amount to be reimbursed/refunded based on the total amount of the items in the expense breakdown grid, and to make the amount advanced field noneditable if the case is a continuation of a cash advance when reporting an expense. Let us get down to it. Open the Expense Report Form in the Dynaform Designer.

First we add a hidden control to the form and, in the Create/Select Variable screen, switch to the Select option and select the `disbursed_by` variable. We know that this variable will be empty if the case is a fresh expense report, but if the case is a continuation of the cash advance, then the value will be set to the name of the Finance officer who disbursed the advance. Check the Protected Value property of the hidden control.

Next, select the form and click the Edit button beside the JavaScript property. Leave one or two lines after the code already in the editor and append the following code:

```

if ($("#disbursed_by").getValue() !== ""){
    $("#amount_advanced").getControl().attr('disabled', true);
}

function computeReimburseRefund(fieldId, newVal, oldVal) {
if (fieldId.search(/^\[expense_grid\]/) == 0 ||
    fieldId == 'amount_advanced') {

    var amountAdvanced = parseFloat($("#amount_advanced").getValue() ?
        $("#amount_advanced").getValue() : 0);

    var totalExpense = $("#expense_grid").getSummary("item_amount");
    var diff = amountAdvanced - totalExpense;

if (diff > 0){
        $('#amount_to_reimburse').setValue(0);
        $('#amount_to_refund').setValue(diff);
    }
}

```

```

    else {
        $('#amount_to_reimburse').setValue(Math.abs(diff));
        $('#amount_to_refund').setValue(0);
    }
}
}

$("form").setOnChange( computeReimburseRefund );

```

A quick explanation of the code. We get the value of the `disbursed_by` hidden control and if it is not an empty string, we get the `amount_advanced` control and disable it.

Then we create a function `computeReimburseRefund`, which serves as a handler for the `onChange` event of the form. The function takes three parameters: `fieldId` (the id of the field that changed), `newVal` (the new value of the field) and `oldVal` (the old value of the field). In the function, we check that the fields that changed is either the `amount_advanced` field or a field in the expense grid to avoid executing the code for every change in the form.

We then get the value of the `amount_advanced` field. Here, we use a ternary operator to check if the field has a value and if not, we return 0 as the value. We then use the JavaScript `parseFloat` function to convert it to a number and store it in `amountAdvanced` variable. We also use the `ProcessMaker` `getSummary` function to get the total amount of the items in the grid and store it in the `totalExpense` variable.

Next, we get the difference of the two amounts and save it in the `diff` variable. We check whether the difference is greater than zero and if so, we set the value of the `amount_to_reimburse` field to 0 and the `amount_to_refund` field to the difference. If the difference is greater than zero, we get the absolute value of the difference (remove the negative) and set it as the `amount_to_reimburse` value and the `amount_to_refund` is set to 0.

Finally, we make the `computeReimburseRefund` function the event handler for the form's `onChange` event.

Save the code and set an arbitrary value as the default value of the `disbursed_by` hidden control. Proceed to preview the form. Enter a couple of rows in the expense grid and enter different amounts. You should see the values of the Amount to Be Reimbursed and Amount to Be Refunded fields updated as the value of the grid total changes. Also, the Amount Advanced field is disabled because we have set a value in the `disbursed_by` field.

The screenshot shows an 'Expense Report Form' with the following elements:

- Department:** A dropdown menu with the text 'Select your department'.
- Reason for Expenses:** A large text area for input.
- Amount Advanced:** A text input field.
- Expense Breakdown:** A table with the following structure:

	Description	Receipt Attached?	Amount
1		<input type="checkbox"/>	500
2		<input type="checkbox"/>	400
			Σ: 900
- Amount to be reimbursed:** A text input field containing the value '900'.
- Amount to be refunded:** A text input field containing the value '0'.
- Reimbursement/Refund Details (Finance):** A section with two sub-fields: 'Amount Reimbursed' and 'Amount Refunded'.

Since we are now computing the Amount to Be Reimbursed and Amount to Be Refunded values automatically, let us make the fields read-only so that users cannot change the computed values.

Select the fields in the Dynaform Designer and set their display mode property to Disabled. Also delete the default value set for the `disbursed_by` hidden control.

Now our form is all set. Save the form and close the Dynaform Designer. Our next task is to enable permissions for the supervisors to be able to view uploaded documents and add case notes.

Case Permissions and Case Notes

Case permissions in ProcessMaker enable us to determine what access users have to a case and its associated objects, such as uploaded and generated documents, case notes, message history, and dynaforms. As noted in the supervisor’s feedback, supervisors were unable to view the documents uploaded by the employee. To remedy this, we will grant the supervisors the permission to view uploaded documents and add case notes.

To set up case permissions, click the Create (+) icon beside Permissions in the Main Toolbox (Process Objects), and the case permissions screen is displayed as shown next.

ProcessMaker offers a lot of flexibility when creating permissions which enables us to set up fine-grained access control to cases and its related objects. Let us quickly explain each field in the case permissions form and then proceed to grant supervisors the required access.

Status Case: This indicates the status the case must be in order for the permission to be activated. The options are All, DRAFT, TO DO, PAUSED and COMPLETED. Draft cases have not yet been sent out by the initiator, while To Do cases are currently in progress. Paused cases are those in which the workflow has been temporarily suspended, and Completed cases have reached the end of the workflow.

Target Task: This is the task where the case must currently be for the permission to be activated. The options are All Tasks and a list of every task defined in the process. For example, selecting Approve Expense task will make the permission active only when the case is at the Approve Expense task. To grant the permission irrespective of the current task, select All Tasks.

Group or User: This is the group or user to which the permission is being granted. The field is a suggest box, and you type in the name of the group or user and select it from the options shown in the suggestions.

Origin Task: This indicates the task that the object (dynaform, input document, and so on) for which the permission is being configured must be assigned. For example, to grant access to the Receipts input document, we would select the Report Expense task, as that is the task the input document is assigned to. To grant the permission irrespective of the task the object is assigned to, use All Tasks.

Participation Required: this is used to indicate whether the user needs to have participated in the case. Selecting Yes requires the user to have had the case assigned to him/her at one point in the process flow, while No allows the user to access the object whether he/she was involved in the case or not.

Type: Indicates which object the permission is being defined for; the options are All (for all the objects), Dynaform, Input Document, Output Document, Case Notes ,and Message History. If Dynaform, Input Document, or Output Document is selected, a field is displayed allowing you select a specific dynaform/input document/output document or All.

Permission: Indicates which permission to grant on the object; the options are View and Block. View grants the user access to the object, while block denies access. If the object selected is an input or output document, there is an additional option, Delete, which allows the user to delete the document. If the selected object is Message History, there is an additional option, Resend, which allows the user resend a message. If the object selected is Case Notes, the Permission field is not displayed, and access to case notes is implicitly granted.

We want the supervisors to be able to view only the documents in a case they participate in, but also be able to view those documents after the case is completed. Create the permission with the following values and click Save when done.

Status Case	All
Target Task	All Tasks
Group or User	Supervisors
Origin Task	All Tasks
Participation Required	Yes
Type	Input Document
Input Document	Receipts
Permission	View

The newly created permission is now displayed, as shown next. Now when the supervisor checks the uploaded documents tab for a case, they should see all uploaded receipts.

Permissions							
Search ...							
Group or User	Participation	Type	Object	Permission	Status		
Supervisors	Yes	INPUT	Receipts	VIEW	ALL	Edit	Delete

We also want the supervisors to be able to send case notes. This will also require the requesting employee to respond to the case notes. We will therefore grant the permission to everyone participating in the case. Create a permission granting access to case notes with the following values.

Status Case	All
Target Task	All Tasks
Group or User	Employees
Origin Task	All Tasks
Participation Required	Yes
Type	Case Notes

Escalating Unclaimed Cases

The final enhancement required by our users is the ability to send an escalation email to the Finance Manager when a case is not claimed on time by a Finance Officer. To achieve this we will create a trigger that looks up the email address of the Finance Manager and sends them an email containing the case details. This trigger will then be set to execute using the Set a Timeout fields of the Disburse Advance and Process Advance task assignment rules.

Finally, we will need to set up or execute the ProcessMaker cron script when the time elapses to execute the trigger. Sounds like a lot of complex stuff, but trying it out will help us better understand the concepts involved, so let us get started.

We begin by creating the email notification template, which contains the message to be sent to the manager. In the Main Toolbox, click the Create (+) icon beside templates and create a template with the following details and save it.

Filename: EscalateFinanceUnclaimed

Content: Dear Manager,

A Cash Advance and Expense Retirement case with case number @=APP_NUMBER assigned to your department has not been claimed. Kindly assign a member of your team to claim and treat the case.

Next create a trigger with the following details:

Title: EscalateFinanceUnclaimedCase

Description: Sends an email to the Finance Department Manager for unclaimed cases. Used together with the Self-Service assignment rule.

Code:

```
$query = "SELECT DEP_MANAGER FROM DEPARTMENT WHERE DEP_UID =
(SELECT DISTINCT CON_ID FROM CONTENT
WHERE CON_CATEGORY = 'DEPO_TITLE' and CON_VALUE = 'Finance')";

$result = executeQuery($query);
$managerUID = $result[1]['DEP_MANAGER'];
$manager = userInfo($managerUID);
$caseOwner = userInfo(@@case_owner);

PMFSendMessage(@@APPLICATION, $caseOwner['mail'], $manager['mail'], '', '',
'Unclaimed Case for Finance Department', 'EscalateFinanceUnclaimed.html');
```

In the trigger we compose an SQL query to get the UID of the Finance department from the CONTENT table and then use the returned UID to look up the department in the DEPARTMENT table and get the UID of the manager. We execute the query and store the resulting UID in the \$managerUID variable. We then use ProcessMaker's userInfo function to look up the details of the manager. We also look up the details of the user who initiated the case. Finally, we use the PMFSendMessage function to send the message to the manager, using the email address of the case owner as the sender.

With the template and trigger in place, the next line of action is to configure the assignment rule to execute the trigger if the case is not claimed after a while. For the purpose of our illustration, we will set the timeout to five minutes. In a real use case, this will be a longer period and be dependent on the SLA (Service Level Agreement) the department has agreed to for attending to requests.

Right-click the Disburse Advance task and select Assignment Rules. Check the set timeout checkbox to display additional configuration fields as shown in the following image.

The screenshot shows a configuration window titled "Assignment Rules: Disburse Advance". It contains the following fields and options:

- Case assignment method:**
 - Cyclical Assignment
 - Reports to
 - Manual Assignment
 - Self Service
 - Value Based Assignment
 - Self Service Value Based Assignment
- Variable for Self Service Value Based Assignment:** @@finance_group
- Set a timeout:**
- Time*:** 5
- Time unit:** Minutes
- Trigger to execute*:** EscalateFinanceUnclaimedCase
- Execute Trigger:** Every time scheduled by cron

At the bottom, there are tabs for "Users" and "Ad Hoc Users", and buttons for "Close" and "Save".

The displayed fields allow us define how much time to wait, what trigger to execute when the time elapses, and how often to execute the trigger. In this case, we will set the timeout to 5 minutes and select the newly created EscalateFinanceUnclaimedCase as the trigger to be executed. We also want the trigger to be executed every time the scheduled cron is run. Update the assignment rule as shown in the image and save your

changes. We also want to apply the escalation to the Process Advance task too. Right-click the task and select Assignment Rules. Check the Set a Timeout field and set the Time to 5, Time Unit to minutes, Trigger to execute to EscalateFinanceUnclaimedCase, and Execute trigger to “Every time scheduled by cron.” Save your changes.

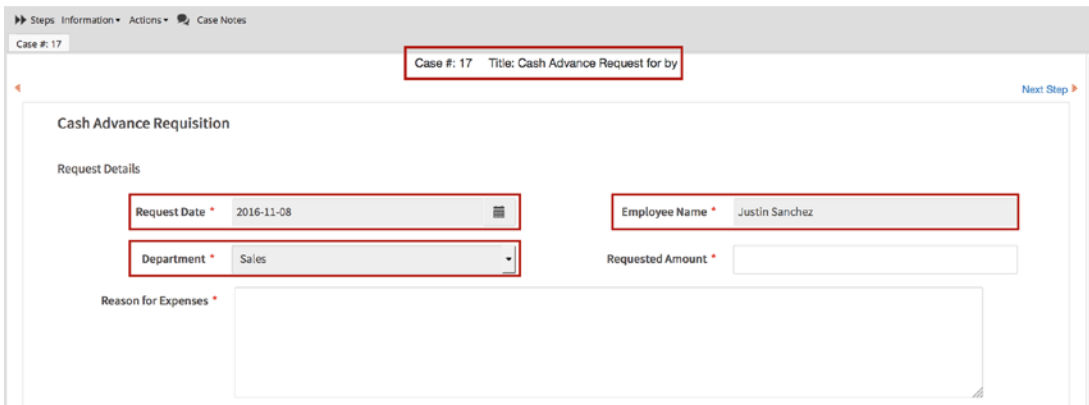
We are done setting up the escalation as requested by the employees. However, because ProcessMaker is a web application, actions such as trigger executions are done in response to user interactions. To handle actions that do not require any actions from the user, such as the timeout we just configured, ProcessMaker provides a number of cron scripts that can be executed in the background to trigger these actions. In a production instance, these scripts will be set up to run as scheduled tasks in a Windows installation or cron jobs in a Unix/Linux installation. To learn more about executing cron scripts in ProcessMaker, see http://wiki.processmaker.com/3.0/Executing_cron.php.

For the purposes of this book, we will execute the cron scripts manually to test that our configured escalation script works as intended. When deployed to production, the scheduled cron will execute periodically (say every 15 minutes) and trigger different actions such as resending pending emails, timer events, timeout triggers, and so on.

We have made quite a number of changes to the process, and it is now time to see the effect of these changes.

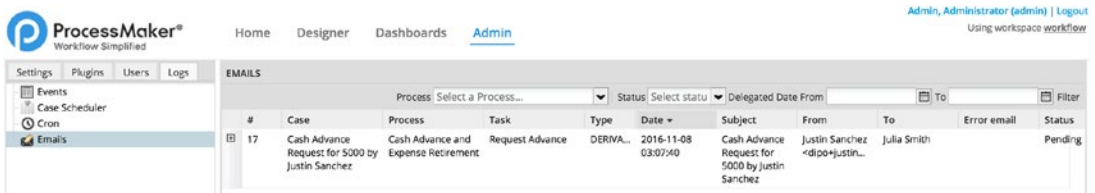
Testing the Enhancements

Log in using another browser as Justin Sanchez and initiate a new Cash Advance and Expense Retirement(Request Advance) case. When the dynaform is loaded, you will observe that the Request Date, Employee Name, and Department fields have been filled in and are disabled. This shows that the SetCashAdvanceInitialData trigger works as expected.



You will also notice that the case title now shows the text we entered in the Case Labels property of the Request Advance task, although incomplete (the values of the variables have not been interpolated into the case title). Fill in the form and submit it to the Supervisor. This should trigger an email notification to be sent to the assigned supervisor (Julia Smith in this case).

In the browser where you are logged in as administrator, click Admin in the main menu and go to the Logs tab in the side-menu. Select Emails and you should see a log of the email that was generated to be sent to the supervisor. You will also notice that the case title is now showing the requested amount and the name of the employee that made the request. (There appears to be a bug affecting alias email addresses, like `dipo+julia@domainname.com`, of the recipient resulting in the email not being sent.)



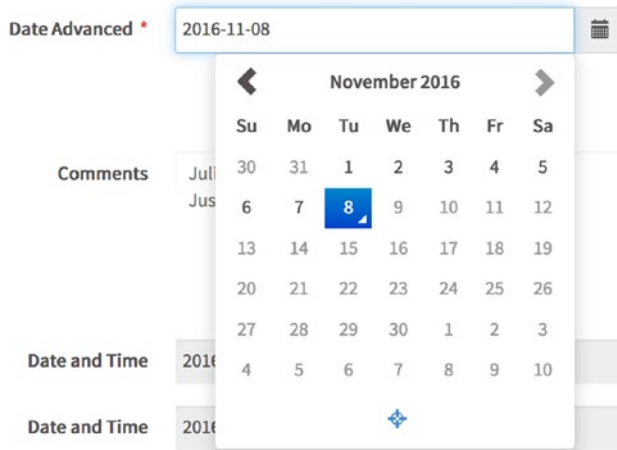
Log out as Justin Sanchez and log in as Julia Smith. Open the case from the inbox and proceed to reject the request. This should execute the `NotifyCashAdvanceRejection` trigger, generating an email notification to Justin Sanchez that the request was rejected. If you check the email log as we did earlier, you should see the record of the email that was generated. Log out Julia Smith and log in as Justin Sanchez. Resubmit the request that was rejected and log out as Justin Sanchez. Log in as Julia Smith and approve the request this time. This should trigger the `NotifyCashAdvanceApproval` trigger, generating an email to the members of the Finance Group. A quick look at the email logs should show the generated email message in the log.

Log out as Julia Smith and log in as Amy Alexander in the Finance department. Once logged in, claim the case from the list of unassigned cases. When the dynaform loads, click the datepicker for the Date Advanced field and you will notice that the future dates cannot be selected. The max date property we set is working as expected. Select a date, enter amount advanced and submit the case.

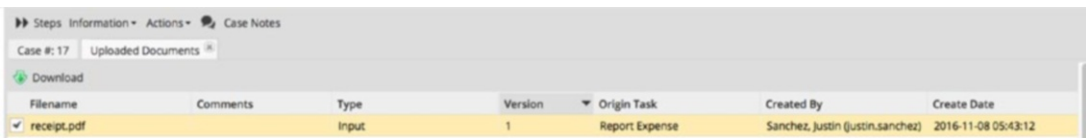
The case is now routed back to Justin Sanchez for the Retirement part of the process.

Log in as Justin Sanchez and go to the inbox; you will observe that the case title has been updated to “Expense Report for Justin Sanchez” from “Cash Advance Request for {amount} by Justin Sanchez”. Open the case and you will see that the Amount

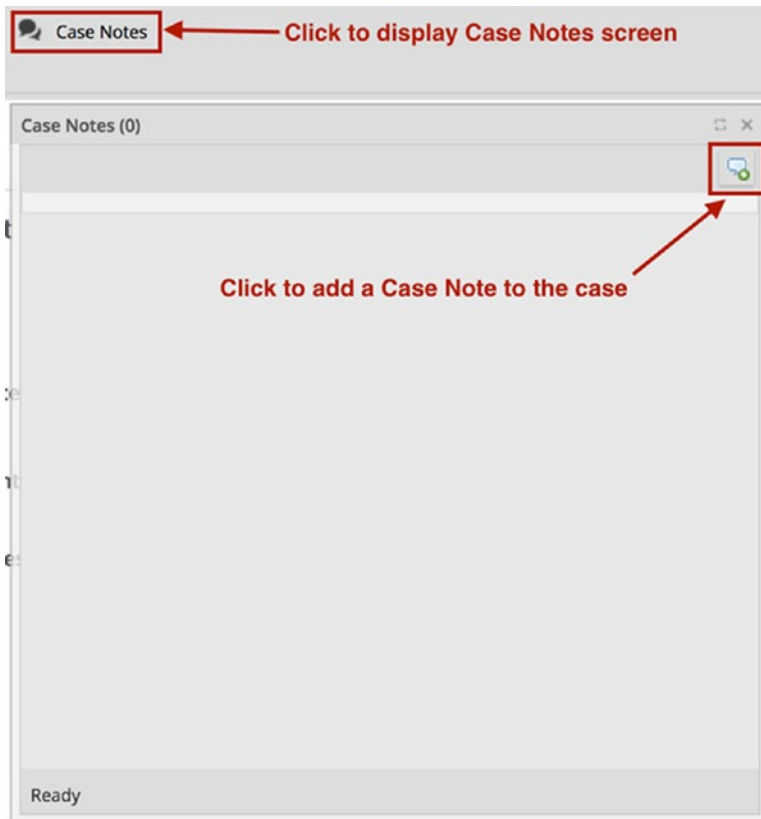
Advanced field is disabled and its value cannot be changed. Enter values in the expense breakdown grid, and the Amount to Be Reimbursed and Amount to Be Refunded should be calculated automatically. Submit the report when done and in the Receipts input document screen displayed next, upload a document. Click the Next step and Continue buttons to send the request to the supervisor and log out.



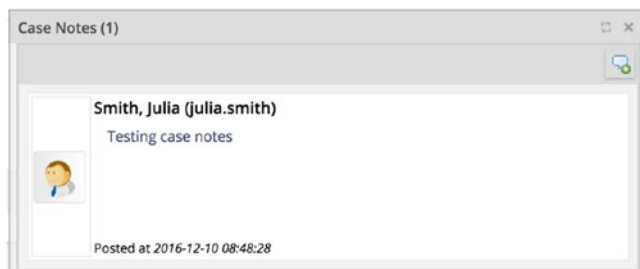
Log in as Julia Smith and open the case from the inbox. Click the Information button at the top of the case and select Uploaded Documents to display the tab. We can now see the receipt uploaded by Justin Sanchez, as shown next.



Let us assume that Julia Smith requires some clarification on the case and does not want to send it back to Justin to add comments. She can use the Case Notes feature in ProcessMaker for just that. Click the Case Notes button on top of the form as shown in the following image. This displays the Case Notes screen.

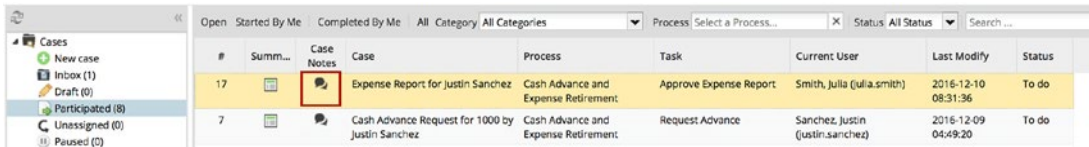


Click the comment icon in the top-right corner of the screen to add a new note. When adding a case note, you can choose to send an email to everyone that has participated in the case by checking the “Send email (Case Participants)” checkbox under the Notes textarea. Type in a message, leave the Send Email option checked, and click the Post button. The note is added to the case as shown next.



Close the Case Notes screen and without approving or rejecting the case, log out Julia Smith. If you check the email log, you will see that an email was sent to Justin Sanchez.

Now log in as Justin Sanchez and reply to the case note. Since the case is still in Julia’s inbox, Justin will have to view the case from the participated cases list. Click Participated under Cases menu in the left pane and click the Case Notes icon for the case as shown next.



The Case Notes screen is displayed. Add a reply, post it, and log out as Justin Sanchez. Log in as Julia Smith and open the case from the inbox. Click the Case Notes button on top of the case and you should see the reply sent by Justin. This way, users can communicate on a case without having to send it back and forth.

Close the Case Notes screen and approve the Report to send it to the Finance department for processing. We will wait for at least 5 minutes after sending the case to the Finance department and then we will manually execute the cron script to escalate the case to the Finance Manager. As we mentioned earlier, in a production deployment, the cron will be scheduled to run automatically periodically and we will not have to manually execute the script.

To execute the script on a Windows installation, open the command prompt and enter the following command, replacing processmaker-3.x.x.x with the actual folder name in your installation (the command should be entered on a single line):

```
"C:\Bitnami\processmaker-3.x.x.x\php\php.exe" -f
"C:\Bitnami\processmaker-3.x.x.x\apps\processmaker\htdocs\workflow\engine\
bin\cron.php"
```

On Mac, open a terminal and enter the following command, replacing processmaker-3.x.x.x with the actual folder name in your installation:

```
php -f /Applications/processmaker-3.x.x.x/apps/processmaker/htdocs/
workflow/engine/bin/cron.php
```

This should cause the escalation trigger to be executed and send the email to the Finance department manager, Amy Alexander. (This appears to work only in the Enterprise Edition).

Finally, log in as Amy Alexander and claim the case. Fill in the fields for the Finance department and submit the case. Download the generated output document if desired and click Continue. Click the Finish button to complete the case.

Our Cash Advance and Expense Retirement Process is now a much improved version with the enhancements we have added in this chapter. Try thinking of ways you can improve the process and try implementing them. In the next chapter, we take a deep dive to explore more complex routing scenarios.

CHAPTER 16

Complex Routing with Gateways

At this point, you have become proficient in designing and building processes in ProcessMaker. So far we have built a complete process using only one type of gateway element: the *exclusive* gateway. You will recall from our discussions of gateways in Chapter 3 that we identified three gateway types:

- Exclusive (XOR) gateway
- Parallel (AND) gateway
- Inclusive (OR) gateway

In the course of designing and building processes in ProcessMaker, we will encounter more complex workflows that will require us to use the other gateway types or a combination of the different gateway types.

To demonstrate, let us assume that the HR department at MSB Corp. has been very impressed with the Cash Advance and Expense Retirement Process we have built and is considering using ProcessMaker to automate aspects of the employee onboarding process.

Currently, the HR officer has to follow up with different stakeholders in different departments to put the necessary things in place for a new employee's first day. This process is currently managed by a checklist that the HR officer ticks as the steps are completed. Following are some of the items on the checklist:

1. IT Department
 - Assign PC to employee
 - Set up employee email account
 - Grant employee access to necessary software

2. Admin Department
 - Print employee business cards
 - Set up employee desk and chair
3. HR Department
 - Collate required forms to be filled by employee
 - Set up employee on payroll
 - Create employee's ID badge
4. Employee's Department Supervisor
 - Assign peer buddy to new employee
 - Define employees first tasks

Our HR officer would like to have a process that is routed from HR to the other departments or stakeholders for them to confirm that the task is completed and fill in details such as the employee's email address or staff ID, and when done is routed back to the HR department for use when the new employee starts.

Looking at the process just described, we observe that a lot of these tasks will have to be done concurrently to make it efficient. If we had to wait for each department to complete their tasks before routing to the next department, the process would take much longer.

The Parallel Gateway in ProcessMaker provides us with just the right tool to address such a problem. We could also consider that some departments might have no input for certain employees (an employee that will not be using any PC or require an email). For such scenario, the inclusive gateway helps us choose which of the tasks to run in parallel.

Our focus in this chapter is on the gateway types and how they can help us handle complex routes, so the process we will design will not be as detailed as the Cash Advance and Expense Retirement process. In the following sections of this chapter, we explore the different gateway types in more detail.

Exclusive (XOR) Gateway

In the Cash Advance and Expense Retirement Process we have been working on in the preceding chapters, we used the Exclusive Gateway type when routing the workflow from the Approve Advance and Approve Expense tasks.

As we saw, the exclusive gateway allows us to determine the next route in a workflow by evaluating a set of conditions. This gateway type is ideal for situations where we need to select exactly one route from two or more route options. The route to take is determined by evaluating conditions defined for each possible route and selecting the one that evaluates to True. When using this gateway type, it is important to ensure that only one condition evaluates to True; otherwise, an error will be thrown and the case will not be routed.

Since we are already familiar with this routing option and have used it in the Cash Advance and Expense Retirement process, we will not be elaborating further on it.

Parallel (AND) Gateway

The *parallel gateway* is used to split the flow into multiple parallel tasks. It is often referred to as a *fork*. Let us illustrate the use of a Parallel gateway using the employee onboarding process described earlier.

Sample Process

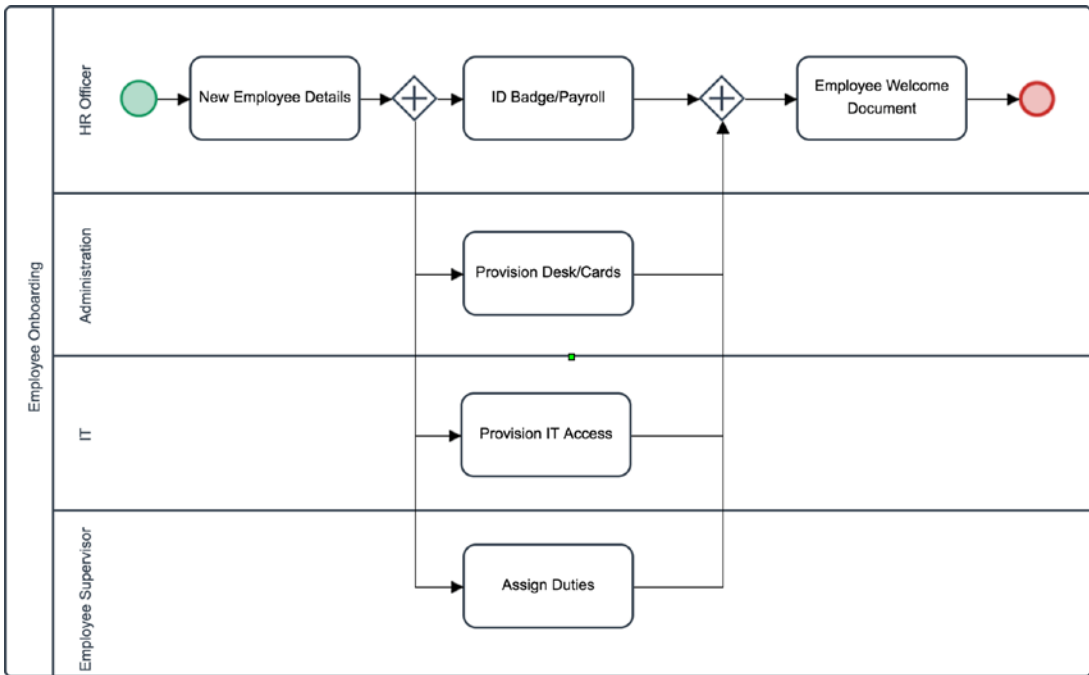
Log in as the Admin user and create a new process with the following details:

Title: Employee Onboarding (Parallel)

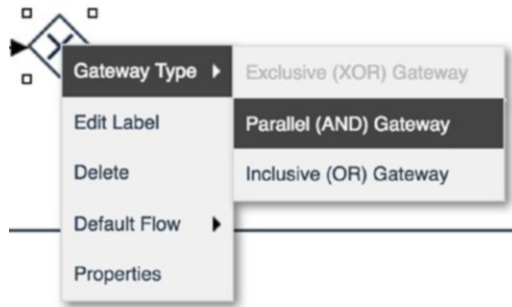
Description: A sample process to illustrate Parallel Gateway Routing.

Process Map

Design the Process Map for the process as shown next. We begin by adding a pool labeled **Employee Onboarding**. We then add four lanes—HR Officer, Administration, IT and Employee Supervisor—to the pool. In the HR officer lane, add a Start event and link it to a task labeled **New Employee Details**.



Next add a gateway element to the New Employee Details task using the quick toolbar, and right-click it as shown next to change the Gateway Type to Parallel. Alternatively, you can drag a parallel gateway from the Shapes toolbox and then link the New Employee Details task to it. This is the Diverging (Fork) gateway.



With the gateway in place, use its quick toolbar to add a task to each of the lanes as shown in the Process Map image and label the tasks accordingly as shown in the following table. Next, add another parallel gateway, the converging gateway, after the ID Badge/Payroll task and join the tasks in the other lanes to it.

Lane	Task
HR Officer	ID Badge/Payroll
Administration	Provision Desk/Cards
IT	Provision IT Access
Employee Supervisor	Assign Duties

Finally add a task labeled **Employee Welcome Document** to the converging gateway and add an End event to it. That completes our Process Map.

When working with gateways, it is important that the diverging gateway and converging gateway are of the same type. Next we create a few simple forms for the process. Remember, we are keeping things very simple. At this point, you should be comfortable working with forms, so we are just going to display the images of the forms and you will create them as an exercise.

Process Dynaforms

Go ahead and create the forms described as follows. When designing the form, assign the fields to variables as shown here.

Field	Variable
Employee Name	emp_name
Resumption Date	resumption_date
Department	department
Employee Type	employee_type
ID Badge Printed?	id_badge_printed
ID Badge No.	id_badge_no
Added to payroll?	payroll_account
Business Cards Printed?	cards_printed
Assigned Email Address	email_address
PC Provided?	pc_provided
Assigned Peer Buddy	peer_buddy
First Tasks	first_tasks

(continued)

Field	Variable
HR Officer	hr_officer
Date and Time	hr_date
Admin Officer	admin_officer
Date and Time	admin_date
IT Officer	it_officer
Date and Time	it_date
Supervisor	supervisor
Date and Time	supervisor_date

Title: Employee Details

Description: A form to capture the new employee’s details

Employee Onboarding

Employee Details

Employee Name * Resumption Date *

Department * Employee Type *

The Department dropdown field should select the list of departments from the database as we did in the Cash Advance and Expense Retirement process. The Employee Type dropdown field should use the following predefined options set on the variable. All the fields should be required.

Key	Label
Full	Full Time
Part	Part Time
Vol	Volunteer

The next form we will create will be used by the departments and supervisor of the employee to provide feedback on the onboarding tasks assigned to them. The form will

contain the data from the Employee Details form in read-only and then add a section for the user or department to provide feedback on the on-boarding task assigned to them.

Because we will be reusing fields from the Employee Details form, we can save time by exporting the form and importing it into the next form we create. We will create a general form with sections for all the tasks which will serve as a master form that will be displayed in the final task to be printed, Employee Welcome Document.

Title: General Onboarding Feedback

Description: A form showing feedback from assigned users on on-boarding tasks.

Employee Onboarding

Employee Details

Employee Name <input style="width: 90%;" type="text"/>	Resumption Date <input style="width: 90%;" type="text"/>
Department <input style="width: 90%;" type="text" value="Select Department"/>	Employee Type <input style="width: 90%;" type="text" value="Select Type"/>

Human Resources

ID Badge Printed? <input type="checkbox"/>	ID Badge No. <input style="width: 90%;" type="text"/>	Added to Payroll? <input type="checkbox"/>
--	---	--

Administration

Desk Provided? <input type="checkbox"/>	Business Cards Printed? <input type="checkbox"/>
---	--

Information Technology

Assigned Email Address <input style="width: 90%;" type="text"/>	PC Provided? <input type="checkbox"/>
---	---------------------------------------

Employee Supervisor

Assigned Peer Buddy <input style="width: 90%;" type="text" value="Select..."/>	First Tasks <input style="width: 90%;" type="text"/>
--	--

Sign Off

HR Officer <input style="width: 90%;" type="text"/>	Date and Time <input style="width: 90%;" type="text"/>
Admin Officer <input style="width: 90%;" type="text"/>	Date and Time <input style="width: 90%;" type="text"/>
IT Officer <input style="width: 90%;" type="text"/>	Date and Time <input style="width: 90%;" type="text"/>
Supervisor <input style="width: 90%;" type="text"/>	Date and Time <input style="width: 90%;" type="text"/>

Quick notes on the form just shown. We made all the fields disabled and removed the Required property on the fields in the Employee Details section. For the Assigned Peer Buddy field in the Employee Supervisor section, the dropdown is associated with a variable that uses an SQL Query to select all users from the database. The query to use is this:

```
SELECT USR_UID, CONCAT(USR_FIRSTNAME, ' ', USR_LASTNAME) FROM USERS
```

We use a number of checkboxes and for those checkboxes, we set the label of the Boolean options to Yes and No, replacing the default True and False.

If you are wondering how we got the subtitles to have a dark background, we used a little JavaScript to add some CSS styling. In your form's JavaScript editor, add the following code:

```
$('.pmdynaform-field-subtitle').css('background', '#666').css('color', '#fff');
```

The code uses the jQuery selector `$('.pmdynaform-field-subtitle')` to select all subtitles on the form, and we use the `.css` function to add a gray background:

```
.css('background', '#666')
```

and set the font color:

```
.css('color', '#fff')
```

Next, we will clone the general form we just created for the forms to be used by the various departments and supervisor. On the cloned forms, we will make the respective field for each task editable. Alternatively, we could use the same form and toggle the display property from Disabled to Edit using JavaScript. We are keeping things simple, though, so we will go the route of cloning the forms.

First export the General Onboarding Feedback form and save it to your PC. When done, go ahead and create the following forms and import the exported General Onboarding Feedback form into the newly created forms.

Title: HR Onboarding Feedback

Description: A form showing feedback from assigned HR officer on onboarding tasks.

After importing the General Onboarding Feedback form, change the display mode of the fields in the Human Resources section to Edit and make them required as shown here.

Title: Administration Onboarding Feedback

Description: A form showing feedback from assigned Admin officer on onboarding tasks.

After importing the General Onboarding Feedback form, change the display mode of the fields in the Administration section to Edit and make them required as shown next.

Title: IT Onboarding Feedback

Description: A form showing feedback from assigned IT officer on onboarding tasks.

After importing the General Onboarding Feedback form, change the display mode of the fields in the Information Technology section to Edit and make them required as shown here. Also enable email validation for the Assigned Email field.

Title: Supervisor Onboarding Feedback

Description: A form showing feedback from assigned Supervisor on onboarding tasks.

After importing the General Onboarding Feedback form, change the display mode of the fields in the Employee Supervisor section to Edit and make them required as shown next.

With our forms ready, the next thing we need to do is create triggers that we will use to update the user who completed a task in the Sign Off section of the General Onboarding Feedback Form.

Process Triggers

Go ahead and create the following triggers in the process.

Title: UpdateHROfficer

Description: Timestamps the date and time a request is processed by the HR Officer.

Code:

```
$data = userInfo(@@USER_LOGGED);
@@hr_officer = $data['firstname'] . ' ' . $data['lastname'];
@@hr_date = getDate() . ' ' . getcurrentTime();
```

Title: UpdateAdminOfficer

Description: Timestamps the date and time a request is processed by the Admin Officer.

Code:

```
$data = userInfo(@@USER_LOGGED);
@@admin_officer = $data['firstname'] . ' ' . $data['lastname'];
@@admin_date = getDate() . ' ' . getcurrentTime();
```

Title: UpdateITOfficer

Description: Timestamps the date and time a request is processed by the IT Officer.

Code:

```
$data = userInfo(@@USER_LOGGED);
@@it_officer = $data['firstname'] . ' ' . $data['lastname'];
@@it_date = getDate() . ' ' . getcurrentTime();
```

Title: UpdateSupervisor

Description: Timestamps the date and time a request is processed by the Supervisor.

Code:

```

$data = userInfo(@@USER_LOGGED);
@@supervisor = $data['firstname'] . ' ' . $data['lastname'];
@@supervisor_date = getCurrentDate() . ' ' . getCurrentTime();

```

Assign Forms and Triggers to Tasks

The next step is to assign the forms and triggers we have created to the tasks in our process. Remember that we assign forms and triggers to a task by right-clicking the task and selecting Steps.

1. Assign the **Employee Details** form to the **New Employee Details** task.
2. Assign the **HR Onboarding Feedback** form to the **ID Badge/Payroll** task. Also assign the **UpdateHROfficer** trigger to the After Routing step of the task.
3. Assign the **Administration Onboarding Feedback** form to the **Provision Desk/Cards** task. Also assign the **UpdateAdminOfficer** trigger to the After Routing step of the task.
4. Assign the **IT Onboarding Feedback** form to the **Provision IT Access** task. Also assign the **UpdateITOfficer** trigger to the After Routing step of the task.
5. Assign the **Supervisor Onboarding Feedback** form to the **Assign Duties** task. Also assign the **UpdateSupervisor** trigger to the After Routing step of the task.
6. Finally, assign the **General Onboarding Feedback** form to the **Employee Welcome Document** task.

Assign Users/Groups to the Tasks

With our forms and triggers all assigned, the next step is to assign users to the tasks. As you will recall, we do this by selecting Assignment Rules from the context menu of each task.

First let us create user groups for HR, IT, and Admin. We are doing this because it is best practice to assign tasks to groups rather than specific users. There are two benefits to this. First, we can manage the users that can work on a task by assigning

or unassigning them from a group. This is good for separation of concerns as in most organizations, user access and control is often managed by a different team and they can carry out their duties without having to edit the process.

The second benefit is that this approach allows us to move processes easily from one instance to another without having to reassign the users to the tasks. When a process is imported, the group assignments are preserved, but user assignments are discarded.

Go to Admin in the main menu, create the following groups, and assign the users shown here to the corresponding group.

Group	Users
Human Resources	Wanda Marshall Nicholas Williams
Administration	Steve Bennett Carlos Shaw
IT	Billy Green Karen Baker

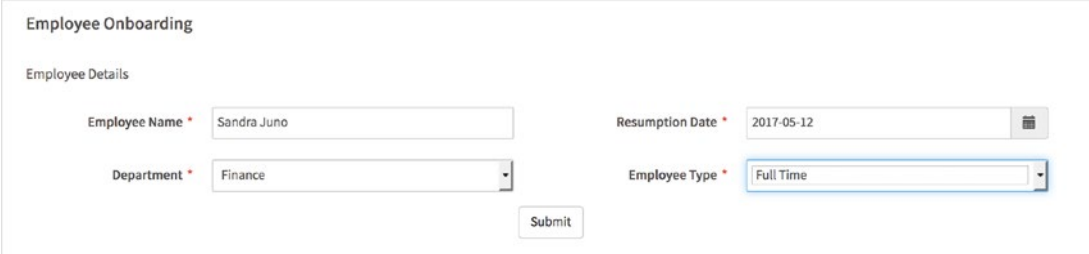
With our groups in place, return to the process and set the assignment rules as described here.

1. Set the Assignment rule for Employee Details task to Cyclical, and assign it to the Human Resources group.
2. Set the Assignment rule for the ID Badge/Payroll task to Self Service and assign it to the Human Resources group.
3. Set the Assignment rule for the Provision Desk/Cards task to Self Service and assign it to the Administration group.
4. Set the Assignment rule for the Provision IT Access task to Self Service and assign it to the IT group.
5. Set the Assignment rule for the Assign Duties task to Manual Assignment and assign it to the Supervisors group.
6. Set the Assignment rule for the Employee Welcome Document task to Self Service and assign it to the Human Resources group.

Our process is now ready to be run. Let us add one more detail to make it better. Set the Case Label of the New Employee Details task to Employee Onboarding for @=emp_name.

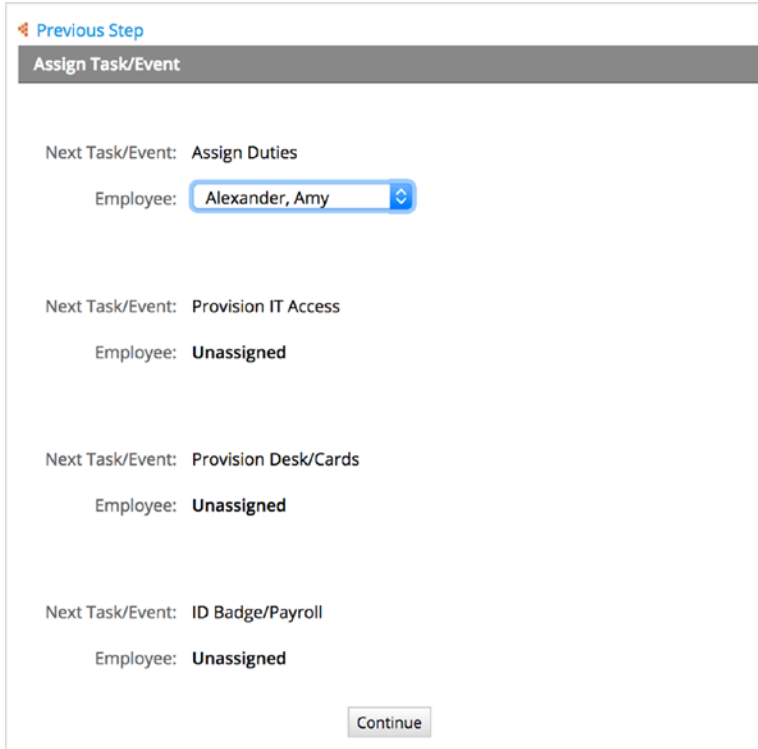
Testing the Process

Now that our process is complete, let us run a case and see the parallel gateway routing in action. In another browser, log in as one of the users in the Human Resources group. Go to New Case and you should see our newly created Employee Onboarding (Parallel) process. Start a new case.



The screenshot shows a web form titled "Employee Onboarding". Under the heading "Employee Details", there are four input fields: "Employee Name" with the value "Sandra Juno", "Resumption Date" with the value "2017-05-12", "Department" with a dropdown menu showing "Finance", and "Employee Type" with a dropdown menu showing "Full Time". A "Submit" button is located at the bottom right of the form.

Fill in the form as shown here and submit it. You should next see the parallel routing screen, as shown here.



The screenshot shows a parallel routing screen. At the top, there is a "Previous Step" link and a header "Assign Task/Event". Below this, there are four task/event entries, each with an "Employee" field:

- Next Task/Event: Assign Duties
Employee: Alexander, Amy
- Next Task/Event: Provision IT Access
Employee: Unassigned
- Next Task/Event: Provision Desk/Cards
Employee: Unassigned
- Next Task/Event: ID Badge/Payroll
Employee: Unassigned

A "Continue" button is located at the bottom right of the screen.

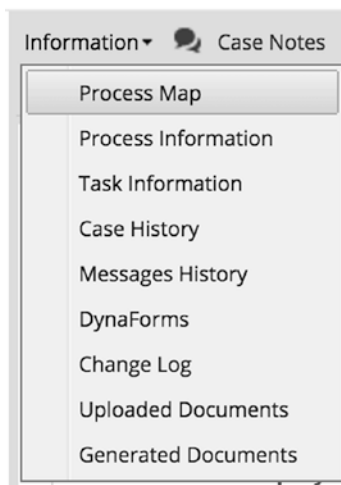
As you can see, this is different from the usual routing screen, as the case is being routed at the same time to the next four tasks. Since we are using manual assignment for the Assign Duties task, we select the supervisor of the department we have chosen for the new employee. In this case I am selecting Amy Alexander, the Finance department manager. When done, click the Continue button.

If you go to the Advance Search screen as the Admin user and search for cases for the Employee Onboarding (Parallel) process, you will see that we have four instances of the same case as shown next.

#	Summa...	Case Notes	Case	Process	Task	Current User	Last Modify	Delegation Date	Due Date	Status
18			#18	Employee Onboarding (Parallel)	Assign Duties	Alexander, Amy (amy.alexander)	2017-05-10 16:56:05	2017-05-10 16:56:05	2017-05-11 16:56:05	To do
18			#18	Employee Onboarding (Parallel)	Provision IT Access	[UNASSIGNED]	2017-05-10 16:56:05	2017-05-10 16:56:05	2017-05-11 16:56:05	To do
18			#18	Employee Onboarding (Parallel)	Provision Desk/Cards	[UNASSIGNED]	2017-05-10 16:56:05	2017-05-10 16:56:05	2017-05-11 16:56:05	To do
18			#18	Employee Onboarding (Parallel)	ID Badge/Payroll	[UNASSIGNED]	2017-05-10 16:56:05	2017-05-10 16:56:05	2017-05-11 16:56:05	To do

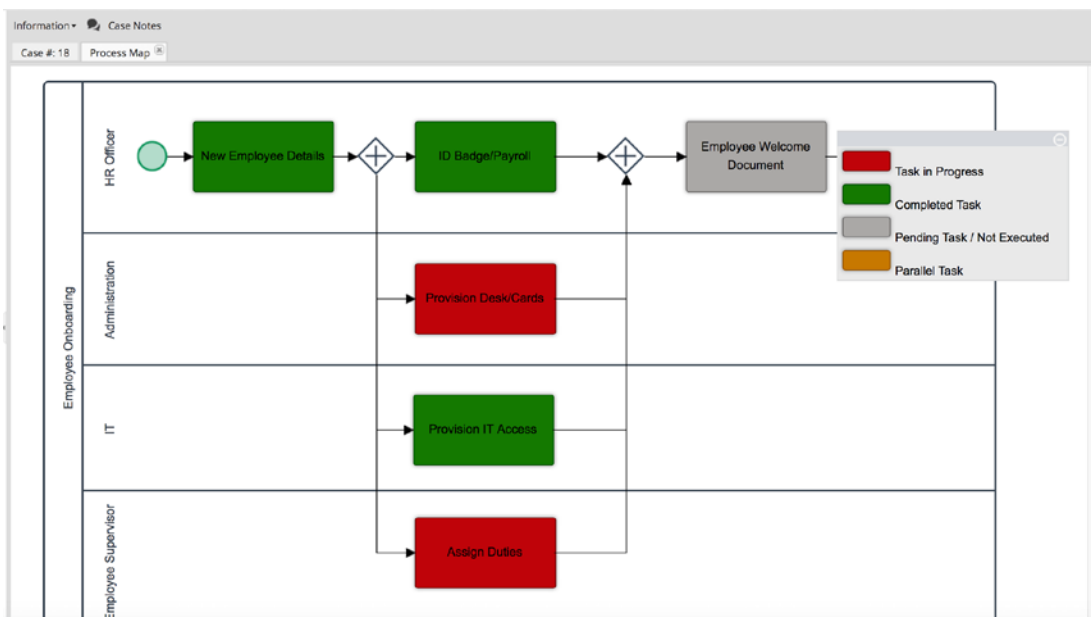
The next thing to do is to log in as a user for each assigned task and update the case. When all four tasks have been completed, the case should converge back to a single instance for the final task.

Since we are still logged in as an HR officer, go to Unassigned Cases and claim the case. Fill in the fields in the Human Resources section of the form and submit it. The next task is displayed as Employee Welcome Document, which is the task after the parallel tasks have completed. Click Continue to submit the task. If you look at the cases in Advanced Search now, you will see that only three instances are listed.



Log in as an IT officer (Billy Green or Karen Baker), claim the case from unassigned, update the form, and submit it. Next we will repeat the same process, logging in as an Administration Officer (Steve Bennett or Carlos Shaw). However, this time, to further illustrate that the different tasks are being run in parallel, after claiming the case, open the Process Map to see the current status of the case on the map. To view the Process Map, click the Information button on top of the case and select Process Map. The Process Map is shown as displayed next.

As we can see in the Process Map, the two tasks we updated are shown as completed, while the other two are shown as in progress. The key thing to note is that all four tasks are being run concurrently. Go back to the case form, and update and submit it. Finally log in as the Supervisor (Amy Alexander in this case) and complete the last parallel task.



When all four tasks have been completed, look up the case in the Advanced Search screen and you should now see that the instances of the case have now converged into one case, and the current task is Employee Welcome Document. Log in as an HR Officer and you should now see the completed form containing all the updates from the various tasks. Go ahead and complete the case.

As you can see, using the parallel routing gateway allows us to design processes that enable multiple tasks to run concurrently and can be an effective tool in the process

design toolbox for creating efficient processes that significantly reduce the turnaround time for completing a process.

Looking at our process, let us consider a situation where some new employees do not need a desk or IT access. For the sake of simplicity, let us assume that onboarding cases for Volunteers require only HR and Supervisor review, but we still want the tasks to run in parallel. We can quickly see that the parallel gateway will not be a fit. Fortunately for us, there is a tool for the job: the *inclusive gateway*.

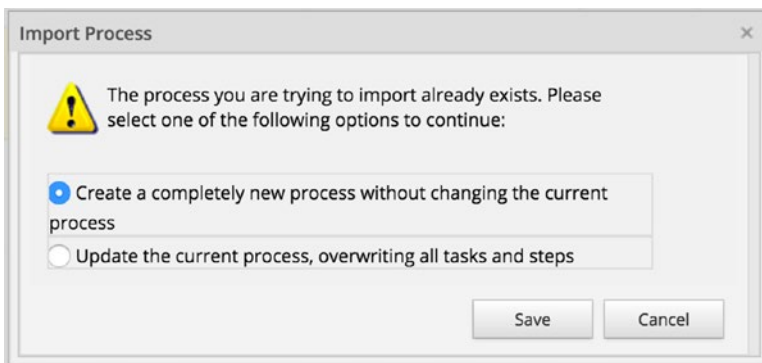
The Inclusive (OR) Gateway

The inclusive gateway is used to split the flow of a case into one or more parallel routes based on the evaluation of a condition. If the condition for a route evaluates to True, the route will be run concurrently with any other path in the gateway whose condition evaluates to True.

To illustrate this, let us make a clone of our previous process and change the gateways to the inclusive type.

Cloning the Process

Log in as an administrator and click Designer in the main menu. Select the Employee Onboarding (Parallel) process and export it. Still on the list of processes page, click the Import button in the top menu and browse to the location you saved the exported process to import it. Click the Upload button and you should be prompted with a screen asking if you want to update the existing process or create a new process, as shown here.

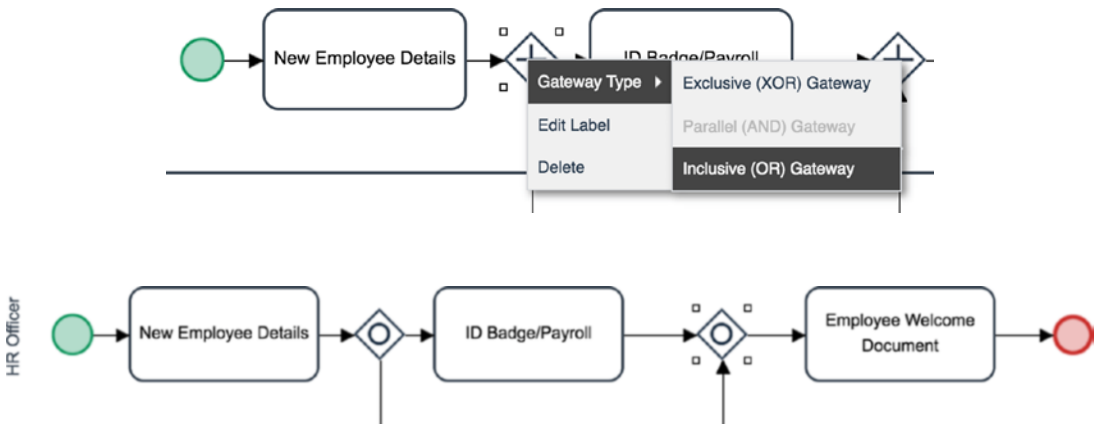


Choose the option to create a completely new process without changing the current process and click Save. A copy of the process is created and opened in the Process Designer. Right-click on an area in the designer outside the process map and select Edit Process. Rename the process to Employee Onboarding (Inclusive) and update the description as shown next.

Now that we have a copy of our process, let us proceed to change the gateways to inclusive.

Changing the Gateway

In the Process Map, right click on the gateway after the New Employee Details task and select Gateway Type ► Inclusive (OR) Gateway as shown in the image on the right. Repeat the same process for the gateway after the ID Badge/Payroll task, changing the Gateway Type also to Inclusive (OR) Gateway. The HR Officer lane of the Process Map should now be similar to the following.



It is important to reiterate that when using an inclusive gateway, the diverging gateway and the converging gateway must be of the same type.

Applying the Conditions

For the purpose of our illustration, as stated earlier, we need to define a condition to omit the Provision Desk/Cards and Provision IT Access tasks for Volunteer employees. To accomplish this, right-click on the diverging inclusive gateway and select Properties. This displays the Inclusive Routing Rule screen shown here.

Routing Rule - INCLUSIVE
✕

Add Routing Rule

Next Task	Condition		
Assign Duties		@@	Delete
Provision IT Access	@@employee_type != "Vol"	@@	Delete
Provision Desk/Cards	@@employee_type != "Vol"	@@	Delete
ID Badge/Payroll		@@	Delete

Cancel
Save

Set the Condition for the Provision Desk/Cards and Provision IT Access tasks to

```
@@employee_type != "Vol"
```

This condition will evaluate to True only if the new employee is not a Volunteer. Save the changes.

Testing the Process

Let us take our new process for a spin and see how the inclusive gateway differs from the parallel gateway. Log in as an HR Officer (Wanda Marshall or Nicholas Williams) and create two cases for our new process, Employee Onboarding (Inclusive). In the first case, set the Employee Type to Volunteer and for the second case, use either Full Time or Part Time as the Employee Type.

You will observe that after you submit the case with Employee Type set to Volunteer, the routing screen shows that the case will be routed to two tasks concurrently, as shown next.

Previous Step

Assign Task/Event

Next Task/Event: Assign Duties

Employee:

Next Task/Event: ID Badge/Payroll

Employee: Unassigned

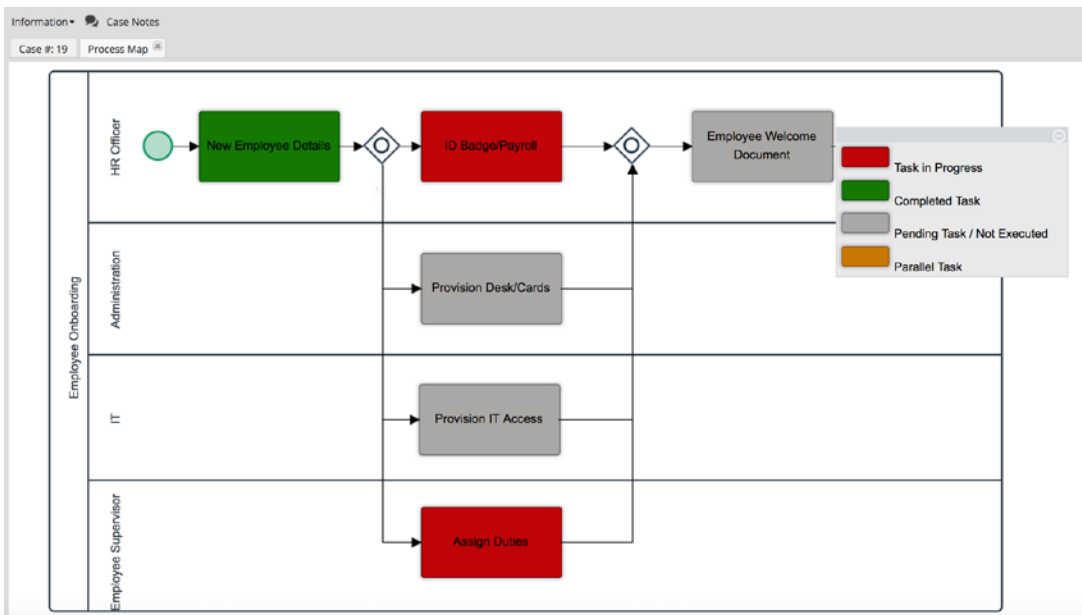
Continue

However, for the second case with Employee Type set to Full Time or Part Time, the case is routed to all four tasks concurrently just as we had with the Parallel routing in the previous section of this chapter. Also, if you go to Advanced Search and search for Employee Onboarding (Inclusive) process cases, you will see that the case with Employee Type set to Volunteer has two instances while the other has four instances, as shown next.

CHAPTER 16 COMPLEX ROUTING WITH GATEWAYS

#	Summary...	Case Notes	Case	Process	Task	Current User	Last Modify	Delegation Date	Due Date	Status
20			#20	Employee Onboarding (Inclusive)	Assign Duties	Smith, Julia (julia.smith)	2017-05-12 12:49:13	2017-05-12 12:49:13	2017-05-15 12:49:13	To do
20			#20	Employee Onboarding (Inclusive)	Provision IT Access	[UNASSIGNED]	2017-05-12 12:49:13	2017-05-12 12:49:13	2017-05-15 12:49:13	To do
20			#20	Employee Onboarding (Inclusive)	Provision Desk/Cards	[UNASSIGNED]	2017-05-12 12:49:13	2017-05-12 12:49:13	2017-05-15 12:49:13	To do
20			#20	Employee Onboarding (Inclusive)	ID Badge/Payroll	[UNASSIGNED]	2017-05-12 12:49:13	2017-05-12 12:49:13	2017-05-15 12:49:13	To do
19			#19	Employee Onboarding (Inclusive)	Assign Duties	Smith, Julia (julia.smith)	2017-05-12 12:48:25	2017-05-12 12:48:25	2017-05-15 12:48:25	To do
19			#19	Employee Onboarding (Inclusive)	ID Badge/Payroll	[UNASSIGNED]	2017-05-12 12:48:25	2017-05-12 12:48:25	2017-05-15 12:48:25	To do

Still logged in as the HR Officer, go to the Unassigned menu and claim the case for the Volunteer employee. Go to the Process Map and you will observe that the ID Badge/Payroll and Assign Duties tasks are being run concurrently, as shown next.



Update the case and submit it. Log in as the assigned supervisor (Julia Smith in my case) and submit the case. If you look in Advanced Search thereafter, you will see that the case has now converged into a single instance without going through the other two tasks. Log in as the HR Officer and complete the case.

As you have seen, the inclusive gateway is a very useful tool in our arsenal for optimizing business processes by allowing tasks to run concurrently while still allowing us to assign tasks conditionally as the business case may require.

Default Flow

When using the exclusive gateway or the inclusive gateway, ProcessMaker provides us with an option to define a default flow. For the parallel gateway, this is not required as all paths in the flow are taken by default.

The default flow sets up the path to be taken when all routing conditions evaluate to False. This can be useful in preventing errors where all routing rules route to False.

In this chapter, we have explored the various ways we can route a process and the different gateway types that can help get the job done. In the next chapter we will explore the ProcessMaker Admin settings to learn more about some of the settings that can be configured in ProcessMaker.

CHAPTER 17

Admin Features

Earlier in this book when exploring the Process Designer and creating our first process, we did not specify a category for the process, and I promised you would learn how to create categories when we explore the Admin features. Well, here we are. We previously explored the Users section of the Admin features when we learned how to administer users in ProcessMaker, and in this chapter we will explore the Settings, Plugins, and Logs tab. To begin, log in as the admin user and click Admin in the main menu at the top of the page.



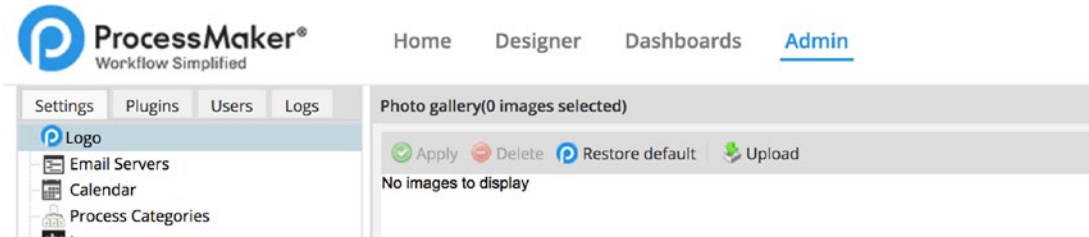
Learning More For more in-depth coverage of the Admin features, visit http://wiki.processmaker.com/3.1/sys_administration.

Settings

The first tab in the Admin section is Settings, which as the name implies provides us with features for configuring the settings in ProcessMaker. We will walk through each option in the Settings tab sequentially, providing a broad overview of what it does and in some cases illustrate with examples.

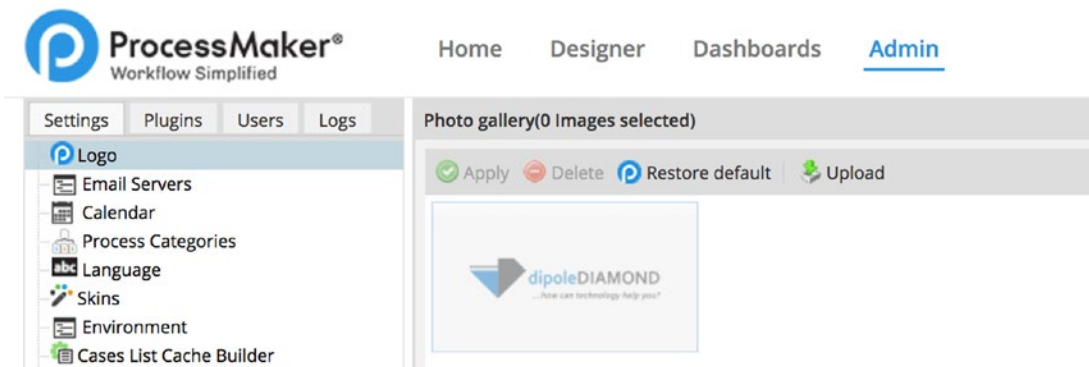
Logo

As you might have guessed, this allows us to change the default logo displayed in the top left of the ProcessMaker application when logged in. Click on the Logo menu in the Settings tab to display the page as shown next.



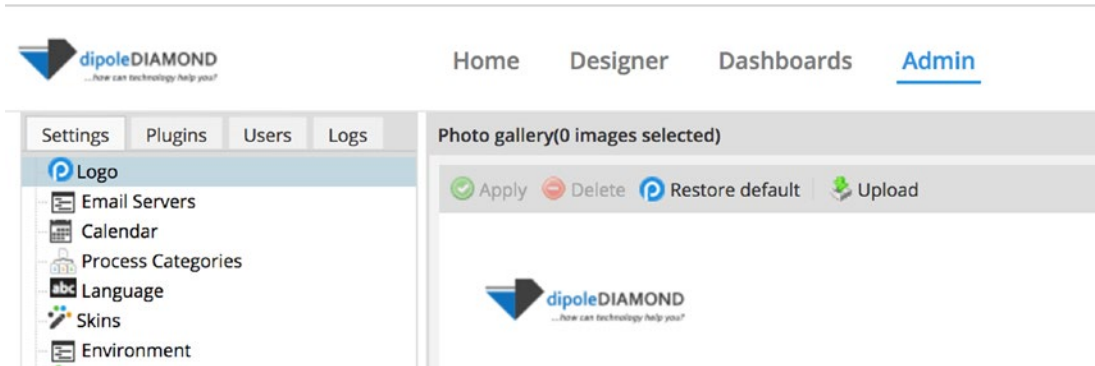
The default logo setting

To change the logo, click the Upload button to upload an image file (the supported formats are JPEG, PNG and GIF). A file selection modal is displayed. Browse to the file on your system to select it and click the Upload button. A “file uploaded successfully” message is displayed, and the uploaded logo is now listed in the Photo Gallery as shown next. Note that ProcessMaker automatically converts the uploaded image to a JPEG file with 80px height, so some skewing may occur in the process. For best results, upload an image with height of 80px or similar proportions.



The logo setting with a logo uploaded

Even though the logo has been uploaded, it is not yet applied to the application. To set the logo, select it and click the Apply button. The applied logo is now displayed as shown next.



The logo applied to ProcessMaker

To restore the default ProcessMaker logo, simply click the Restore Default button.

Email Servers

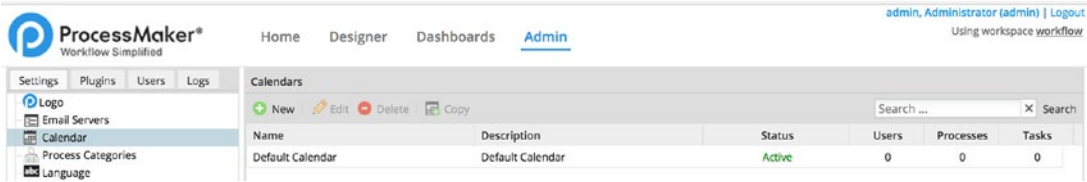
The Email Servers setting allows us to configure the system for sending email notifications. ProcessMaker provides two options for configuring the email engine, SMTP and Mail. We have already explored how to test whether the email configuration is properly set up in a previous chapter so we will not be going into it again.

You can explore the ProcessMaker wiki for detailed explanations of the settings and examples of how to configure the common Email providers: Google, Yahoo, Hotmail, and Outlook (http://wiki.processmaker.com/3.0/Email_Settings).

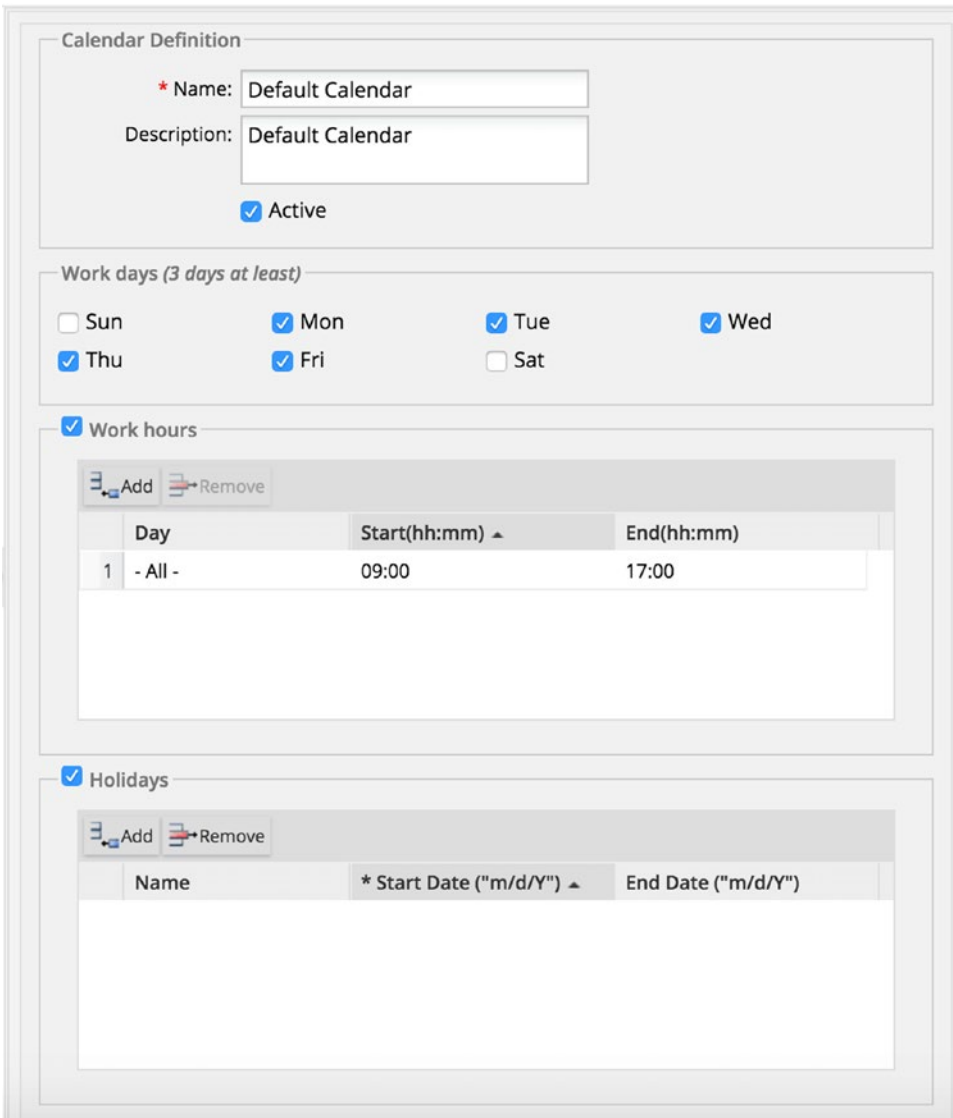
Calendar

The Calendar settings allow administrators to define work hours for users, tasks, and processes. This is especially important for computing task due dates. For example, most countries have a five-day work week and eight- to nine-hour work days. So if a task is assigned to a user at the close of business at the end of the work week, and the task duration is one day, if the Calendar is not configured to account for the weekend break, the task will appear as overdue when the user resumes work at the beginning of the next week.

Let us take a look at the default calendar defined out of the box in ProcessMaker. Click the Calendar menu in the Settings tab, and the Default Calendar should be shown as displayed in the following image.



Select the calendar and click the Edit button to see the values defined for the calendar. The Calendar Configuration screen is displayed as shown here.



As we can see in this image, the defined work days are Monday to Friday and work hours are from 9:00 am to 5:00 pm. The system also provides us the option to define Holidays. This comes in handy when Public Holidays are declared and the users will be absent from work during the regular work days. Unlike the work days and work hours, the Holidays are not recurrent and have to be defined for each holiday every year.

Calendars Order of Precedence

A calendar can be assigned to a user, task, or process. If no calendar is specified for a user, the calendar assigned to the task is used. If the task has no calendar assigned, the process calendar is used. If neither process, task, nor user has a calendar assigned, the default calendar is used. The order of precedence is

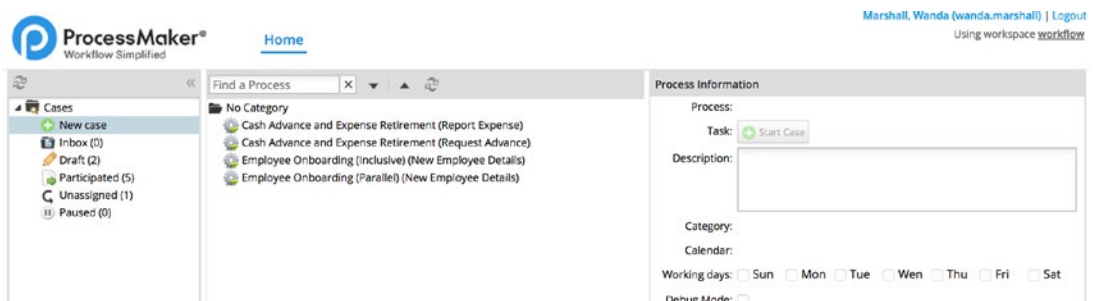
User ► Task ► Process ► Default

To learn more about configuring and assigning calendars, check out the ProcessMaker wiki (<http://wiki.processmaker.com/3.0/Calendars>).

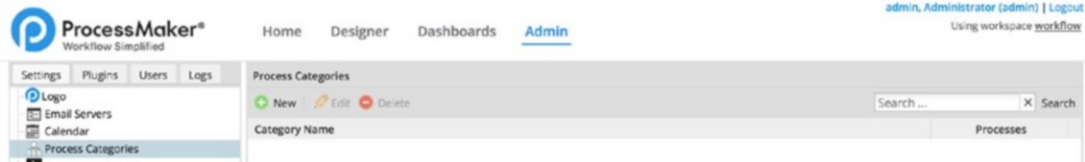
Process Categories

Process Categories provides a handy feature for categorizing processes in ProcessMaker. As more and more processes are added to the system, the New Case screen can become cluttered, making it hard for users to find the processes they need.

Remember that for all the processes we have created so far, we have left the category set to No Category. To illustrate Process Categories, log in as Wanda Marshall (wanda.marshall) in the Human Resources Group in another browser and go to New Case. The list of processes (starting tasks) available to the user is displayed as shown next. We can see that all the processes are listed under the No Category folder.

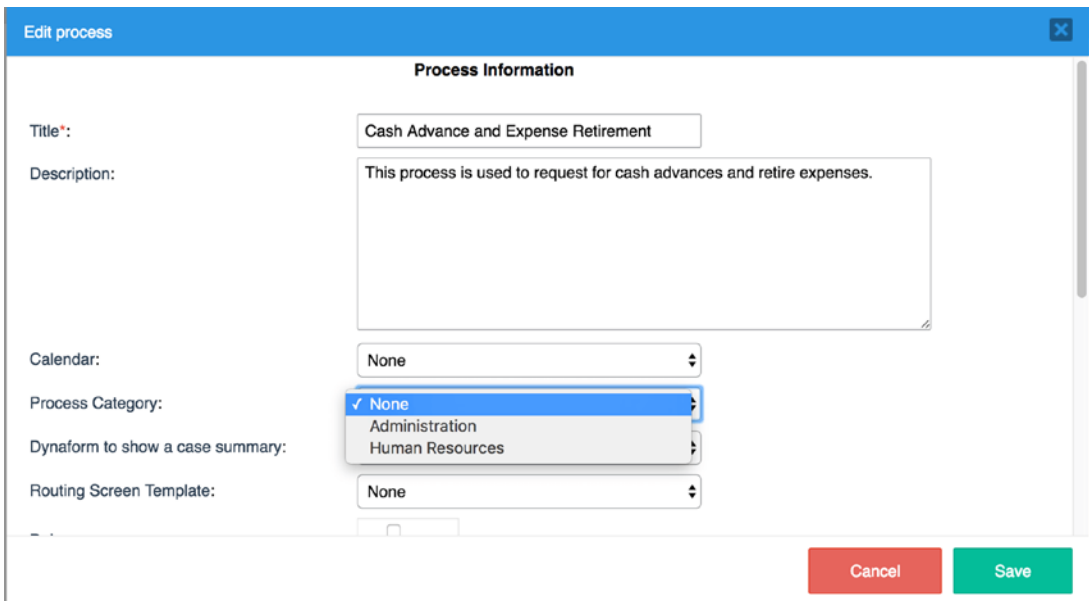


Let us create some categories and see the effect. In the browser where you are logged-in as Admin, click the Process Categories menu in the Settings tab. The page is displayed as shown here.



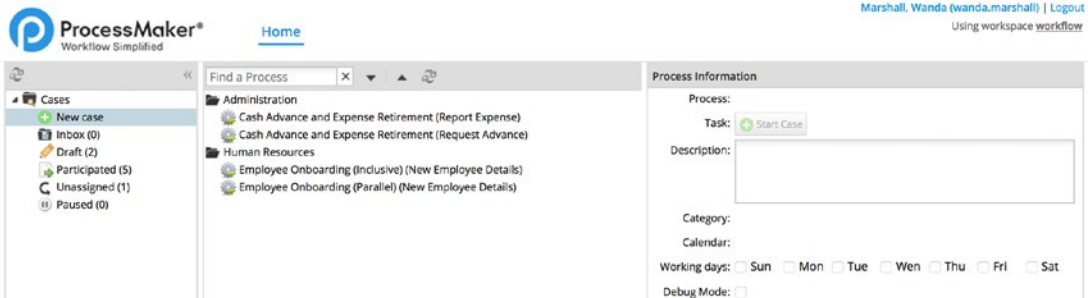
Click the New button and in the dialog that pops up enter **Administration** as the Category Name. Click Save. Repeat the process and add another category named **Human Resources**.

The next step is to assign the categories to the processes we created earlier. Go to the Designer tab and edit the Cash Advance and Expense Retirement Process. Right-click on an area outside the Process Map and select Edit Process from the context menu. The Process Information screen is shown.



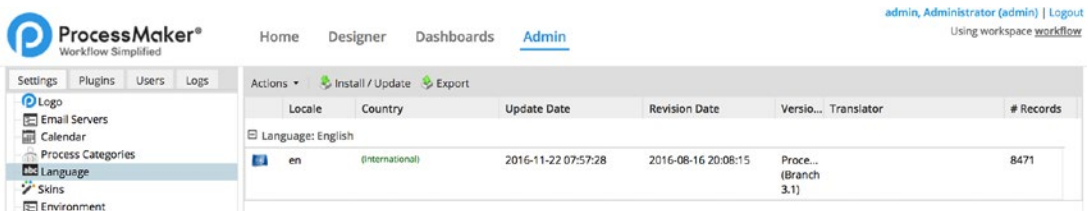
We can see that the defined categories are now included in the dropdown options of the Process Category field. Set the category to Administration and save. Repeat the process for the Employee Onboarding (Parallel) and Employee Onboarding (Inclusive)

processes, setting their category to Human Resources. When done, refresh the New Case page in the other browser where you are logged in as Wanda Marshall, and the processes should now be categorized as shown next.



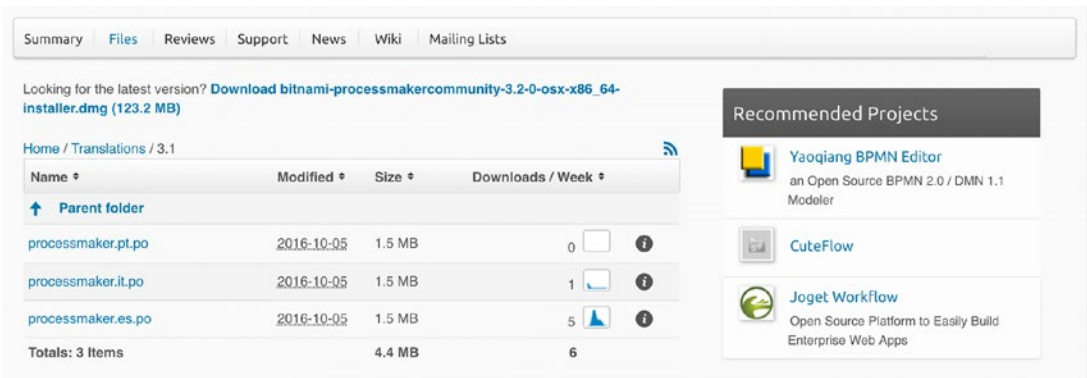
Language

Though the default language for ProcessMaker is English, the Language setting allows us to add other language translations. To try this, let us add another translation to our instance of ProcessMaker. Click the Language menu in the Settings tab to display the page as shown here.

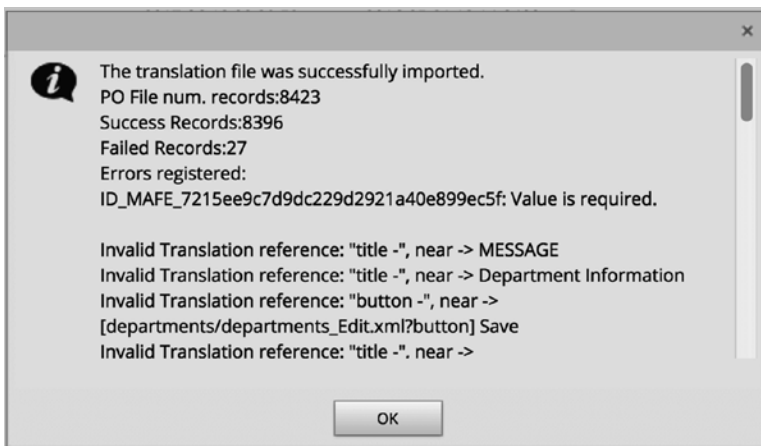


As we can see, we have just one language available in the system, English. The language translations are stored as .po files and ProcessMaker provides translations for other languages that can be downloaded from Sourceforge (<https://sourceforge.net/projects/processmaker/files/Translations/>).

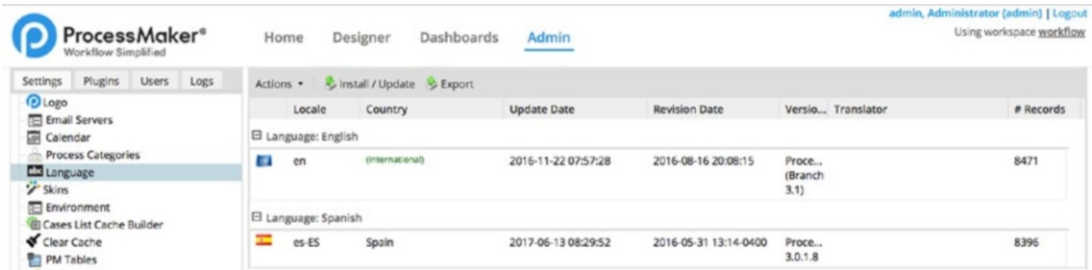
Head to the Sourceforge page and select the folder matching your version of ProcessMaker (3.1 in this case), and the list of translation files should be displayed as shown next.



Let's install the Spanish translation. Click the processmaker.es.po link to download the translation file and save it to your system. Once the file is downloaded, click the Install/Update button in the Language Settings page and locate the downloaded file to select it. Click the Upload button to import the new translation. When done, a message showing the result of the import should be displayed as shown next.



We can see the number of records that were successfully imported and the errors for those that failed. For the purpose of our demonstration we can disregard the errors (in a production instance, you would definitely want to review the errors and update the translation file to fix the identified issues). Click OK, and our new language is now listed on the page as shown here.



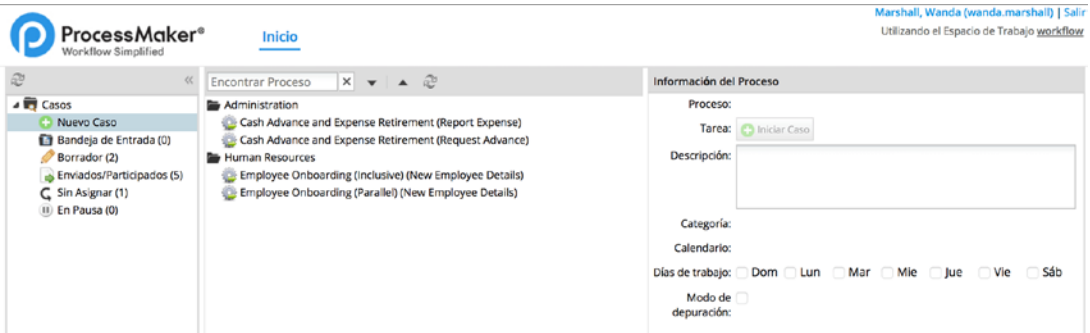
The screenshot shows the ProcessMaker Admin interface. The top navigation bar includes 'Home', 'Designer', 'Dashboards', and 'Admin'. The user is logged in as 'admin, Administrator (admin)'. The left sidebar shows various settings categories, with 'Language' selected. The main content area displays a table of language configurations.

Locale	Country	Update Date	Revision Date	Version	Translator	# Records
Language: English						
en	(internacional)	2016-11-22 07:57:28	2016-08-16 20:08:15	Proce... (Branch 3.1)		8471
Language: Spanish						
es-ES	Spain	2017-06-13 08:29:52	2016-05-31 13:14-0400	Proce... 3.0.1.8		8396

In another browser, open the login page and you should observe that the login language field of the form now includes Spanish as one of the language options.



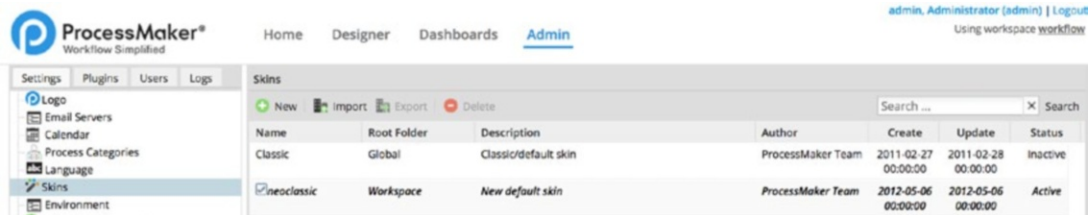
Log in as Wanda Marshall (wanda.marshall) from Human Resources, but this time, select Spanish as the language. The New Cases screen is now displayed in Spanish as shown next.



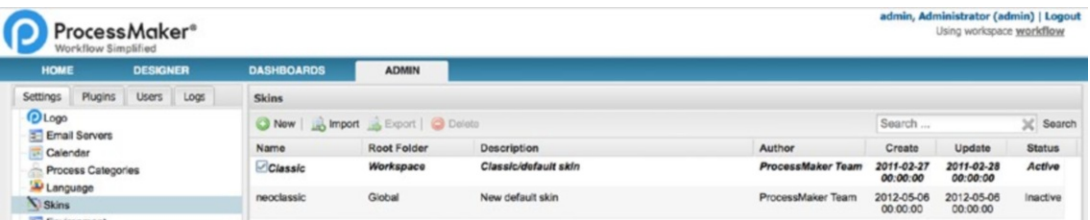
Skins

The Skins setting allows us to change the look and feel of the ProcessMaker application. The application comes with two skins out-of-the box, Neoclassic and Classic. You can also create your own custom skin and import it into the system, but that is beyond the scope of a beginner’s guide.

Let’s try switching between the two skins available. Go to the Skins Setting page and the two skins should be displayed.



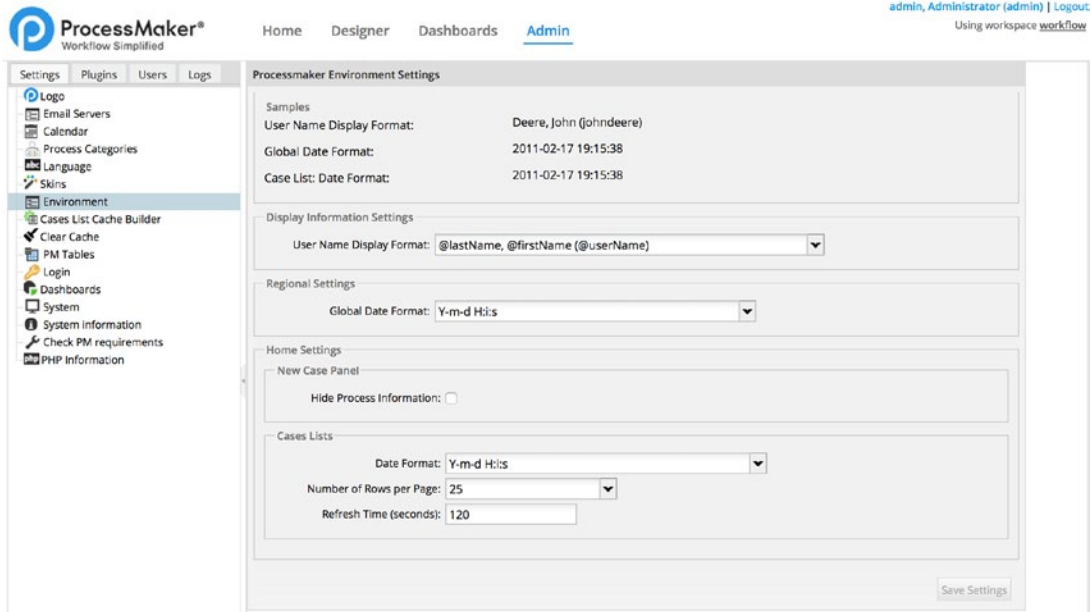
To change the skin to Classic, double-click the Classic skin. The ProcessMaker application should now be wearing the Classic look, as shown next.



Feel free to keep the Classic look or switch back to the Neoclassic.

Environment

The Environment settings provide options for configuring how the names of users, date formats, and case lists are displayed. You can try toggling some of the options to see the resulting effect.



ProcessMaker Environment Settings page

Cases List Cache Builder

The information displayed in the list of cases for a user is aggregated from multiple tables, and since this data is frequently accessed, querying all the tables frequently can create performance issues. To prevent this problem, ProcessMaker stores all this data in a table called APP_CACHE_VIEW. However, in rare instances, this data can become out of sync, resulting in inaccurate information displayed in the users' cases list. The Cases List Cache Builder setting allows the administrator to rebuild the cache at the click of a button.

Clear Cache

ProcessMaker is a web application, and the user interface is built from a large number of PHP, HTML, CSS and JavaScript files. To speed up the application performance, these files are precompiled and stored in the application's compiled directory. After an upgrade or change to the source code, it is recommended to clear the compiled files. This setting provides the interface for clearing the precompiled files.

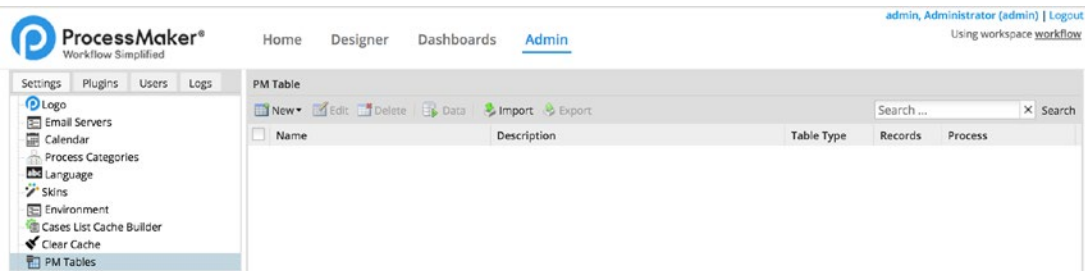
PM Tables

The ProcessMaker PM Tables screen provides an interface for us to create tables in the ProcessMaker database directly from the web application. This is a very useful feature as it allows us to maintain data that can be shared across multiple processes and even external applications such as Business Intelligence Reporting tools.

There are two types of PM tables that can be created in ProcessMaker:

1. PM table
2. Report table

The PM table is an empty table to which we can add our own data while report tables are created from case data of the selected process. Let us create one of each to illustrate the difference and use cases. Head over to the PM Tables menu in the Settings tab.



Creating a Report Table

Let us create a report table showing the list of expenses from the Cash Advance and Retirement Process. Click the New button, and in the context-menu select New Report Table. The New Report Table screen is displayed. Select Cash Advance and Expense Retirement from the list of processes to display the dynaform fields as shown here.

New Report Table

Process: **Cash Advance and Expense** ▼

Table Name (Auto Prefix "PMT_"):

Description:

Type: **Global** ▼

DB Connection: **Workflow** ▼

Search ... Filter

Dynaform Fields ▲

- amount_advanced
- amount_advanced_label
- amount_refunded
- amount_refunded_label
- amount_reimbursed
- amount_reimbursed_label
- amount_requested
- amount_requested_label
- amount_to_refund
- amount_to_refund_label
- amount_to_reimburse
- amount_to_reimburse_label
- approved_by
- approved_by_label

Report Table: (0 columns)

Dynaform Field	Field Name	Field Label	Type	Size	Auto Increm...

Page 1 of 2 | 1 - 50 of 6

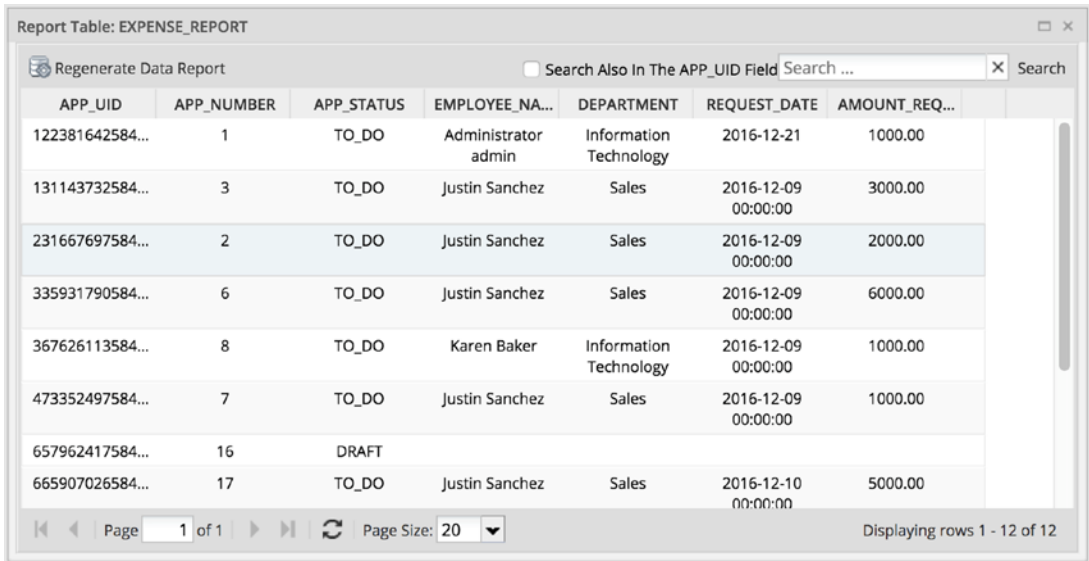
1. Set the table name to `EXPENSE_REPORT`.
2. Give it a description.
3. Leave the type as Global. The other option is Grid; it allows us to create a table of the grid data, which is not included in the report table by default, because of the data structure.
4. Leave the DB Connection as Workflow.
5. From the Dynaform fields, select the following fields and move them to the Report Table by clicking the > button in the middle:
 - `employee_name`
 - `department`

- request_date
 - amount_requested
6. Click the Create button.

The newly created report table is now displayed as shown next.



Select the newly created table and click the Data button above it. The data from the cases we have run so far are displayed. Go ahead and close the data screen.



Creating PM Table

Next let us create a PM table. Click the New button and in the context menu select New PM Table. The New PM Table screen is displayed as shown here.

1. Set the table name to EMPLOYEE_TYPE.
2. Give it a description.
3. Click the Add Field button.

This displays the Add Field interface.

Create the fields as specified in the following table. After filling in the fields, click the Update button to save your changes. When done, click the Create button.

Field Name	Field Label	Type	Size	Null	Primary Key	Auto Increment
ID	ID	BIGINT	11	uncheck	check	check
CODE	Code	VARCHAR	20	uncheck	uncheck	uncheck
LABEL	Label	VARCHAR	255	uncheck	uncheck	uncheck

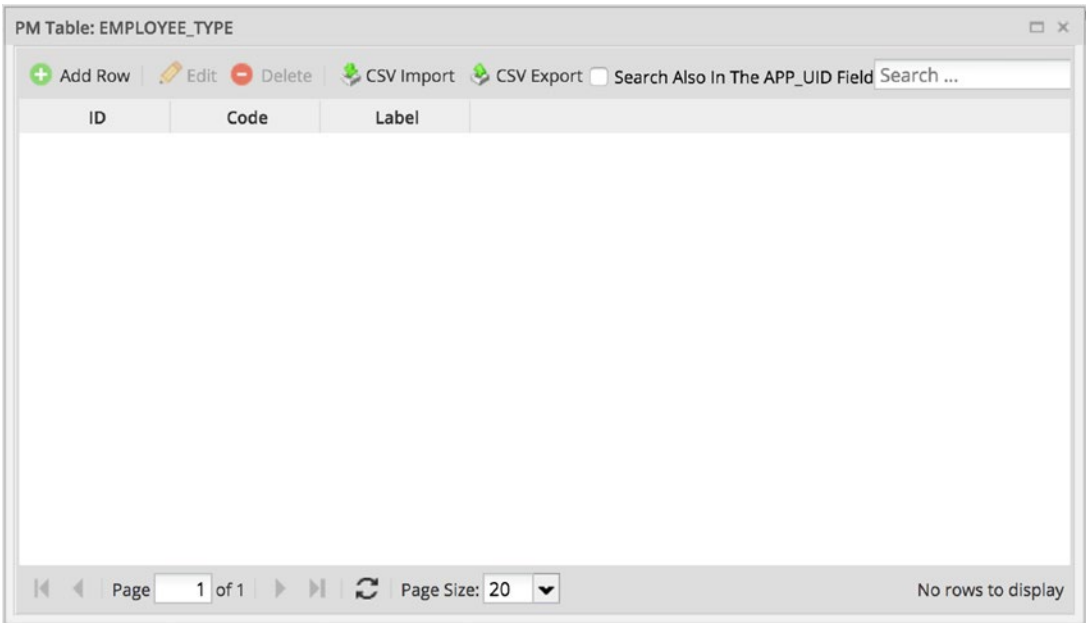
The newly created table should now be displayed as shown here.



The screenshot shows a window titled "PM Table" with a menu bar containing "New", "Edit", "Delete", "Data", "Import", and "Export". A search bar is located in the top right corner. Below the menu bar is a table with the following data:

Name	Description	Table Type	Records	Process
<input type="checkbox"/> EMPLOYEE_TYPE	Sample PM Table	PM Table	0	
<input type="checkbox"/> EXPENSE_REPORT	A sample report table	Report Table	12	Cash Advance and Expense Retirement

To add data to our table, select it and click the Data button. The following screen is displayed. Unlike the Report table, its data is empty.



The screenshot shows a window titled "PM Table: EMPLOYEE_TYPE" with a menu bar containing "Add Row", "Edit", "Delete", "CSV Import", "CSV Export", and "Search Also In The APP_UID Field". A search bar is located in the top right corner. Below the menu bar is an empty table with the following columns:

ID	Code	Label
----	------	-------

At the bottom of the window, there is a pagination bar showing "Page 1 of 1", "Page Size: 20", and "No rows to display".

Let us add some sample employee types to our table as depicted in the next image. Click the Add Row button to add a new row and fill in a value for the Code and Label columns. The ID column is auto-generated by the database because we checked the Auto Increment option when creating the column.

ID	Code	Label
1	Full	Full Time
2	Part	Part Time
	Vol	Volunteer

This data can now be incorporated into our processes by using an SQL Query to fetch the data. That way, we can reuse the same data across multiple processes. As an exercise, try modifying the Employee Onboarding processes to use the data from the PM Table instead of defining values in the options of the `employee_type` variable.

Login

The Login settings allow us enable the Forgot Password feature on the login page and set the default language.

Login Settings

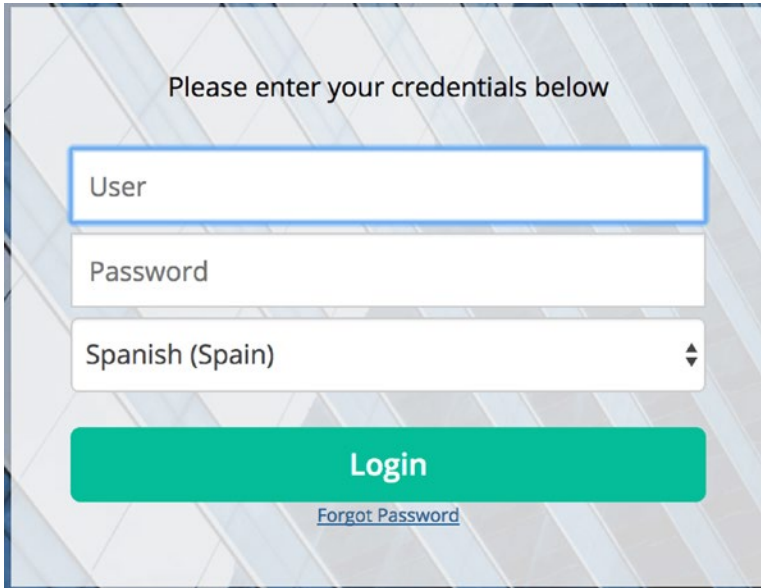
Default Language: English

Enable Forgot password:

The default language set here is for the "Language" dropdown box displayed on the login screen. This configuration is set for each workspace.

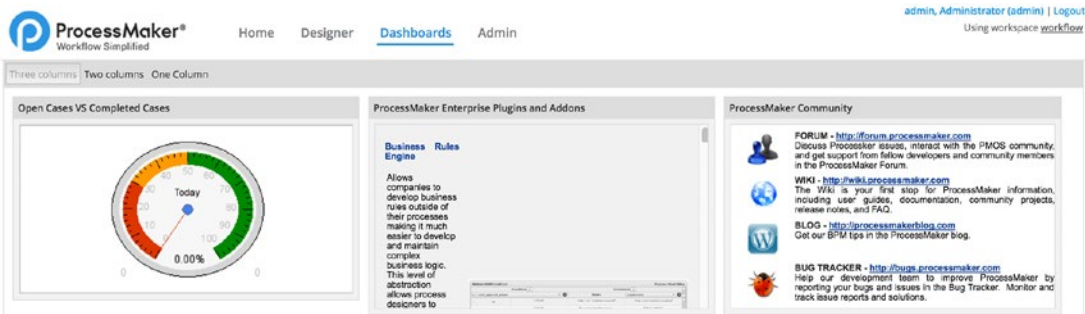
Save Settings

Try changing the default language to Spanish and enabling the Forgot Password option. Remember to click the Save Settings button. Log out, and the login form should now show the default language as Spanish with a link to Forgot Password below the Login button, as shown here.



Dashboards

The Dashboards settings are used to configure the dashlets displayed on the Dashboards page of the main menu.



As an example, let us configure the Open Cases vs Completed Cases dashlet. In the Dashboard Settings page, select the Open Cases vs Completed Cases dashlet and click the Edit button to display the Dashlet Instance Configuration as shown next.

Dashlet Instance Configuration

General

Title:

Dashlet: ▼

Assign To: ▼

Other

Period: ▼

Red Starts In:

Red Ends In:

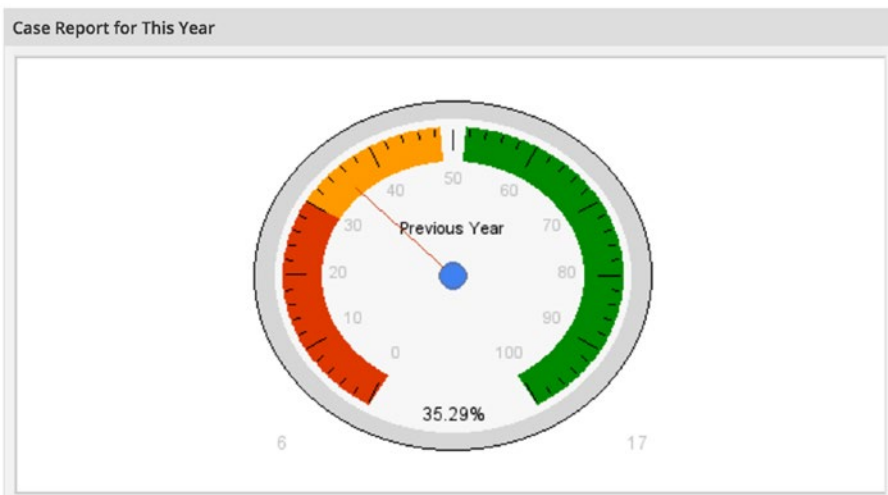
Yellow Starts In:

Yellow Ends In:

Green Starts In:

Green Ends In:

Set the Title to **Case Report for This Year** and the Period to **This Year**. Save your changes. Now return to the Dashboard page, and the dashlet should be updated and show a comparison report similar to the one shown here.



System

The System settings page provides an interface for defining system settings such as the default Time Zone used when working with dates and the memory limit available to each logged-in session. The Cookie Lifetime setting controls how long a session can be idle before being logged out. Other settings include the Default user expiry date, Default Skin, and Default language.

System Information, Check PM Requirements and PHP Information

The remaining three settings are informational and provide insights into the environment in which ProcessMaker is being run. This can be useful for troubleshooting issues when administering the system. Click on each of the settings to view the information displayed.

That concludes our exploration of the Admin Settings tab; up next is the Plugins tab.

Plugins

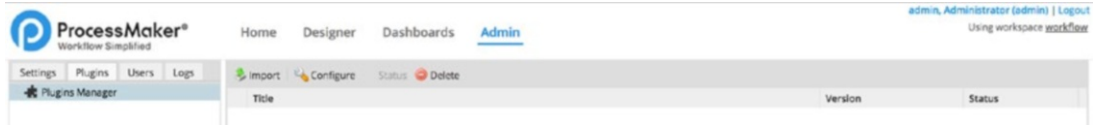
ProcessMaker is a very extensible system, and the way to extend the features is by adding new plugins. A plugin can add a new menu to the ProcessMaker interface or modify the database. To learn more about developing plugins, visit the ProcessMaker Wiki (http://wiki.processmaker.com/3.0/Plugin_Development).

To illustrate how the plugins work, we will import a sample plugin that manages draft cases. Download the plugin file using the link <https://github.com/dipolediamond/draftManager/raw/master/draftManager-1.tar>.



DraftManager Plugin The code for the plugin is available on Github (<https://github.com/dipolediamond/draftManager>). Feel free to clone and improve it.

Now let us proceed to import the plugin. In the Admin section, go to the Plugins tab, the Plugin Manager is displayed.



Click the Import button to display the Import Plugin form as shown here.

Import Plugin

Maximum upload file size in bytes 20971520 (20M)

File draftManager-1.tar

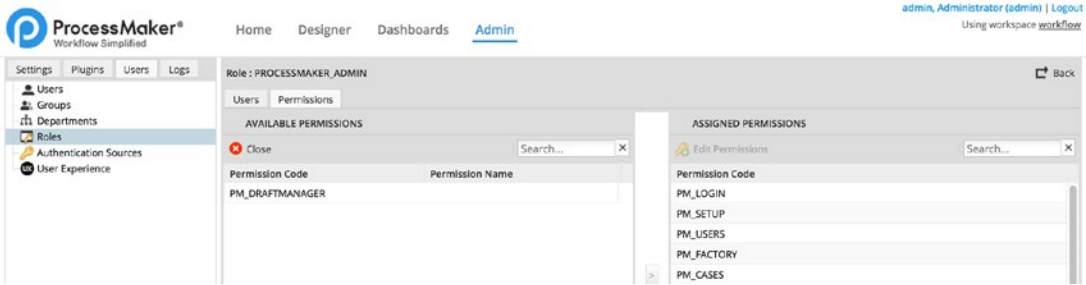
Browse to the draftManager-1.tar file downloaded earlier to select it and click the Import button. The imported plugin is shown.

Title	Version	Status
draftManager Plugin (draftManager.php)	1	Disabled

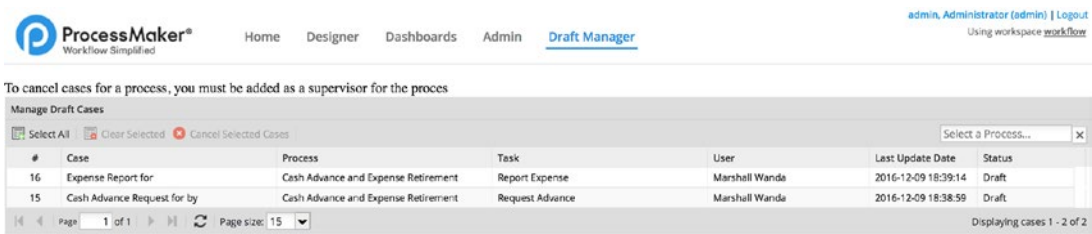
Next thing we need to do is enable the plugin. Select the plugin and click the Enable button. The plugin status should change from Disabled to Enabled.

Title	Version	Status
draftManager Plugin (draftManager.php)	1	Enabled

Our plugin is now enabled, but we cannot see any visible change in the system. The draft plugin we are using for this illustration creates a new permission and allows only users belonging to roles that have that permission to access the plugin page. To remedy this, let us add the newly created permission to the PROCESSMAKER_ADMIN role. Go to the Users tab and select Roles from the menu. Select the PROCESSMAKER_ADMIN role and click Permissions. In the Permissions page click the Edit Permissions button to display the new permission as shown next.



Proceed to add the permission to the Assigned Permissions list. When done, reload the page or log out and log in again. You should now see the plugin page added by our new plugin as shown here.



Logs

The last tab in the Admin features is the Logs tab, and it provides an interface for users with the PM_SETUP_LOGS permission to view the following logs generated by the system.

- Events Log
- Case Scheduler Log
- Cron Log
- Email Log

We have seen an example of the email log in an earlier chapter of this book and a deeper exploration of configuring the other logs is beyond our scope.

That brings us the end of our exploration of the ProcessMaker Admin features. At this point, you should be pretty comfortable finding your way around ProcessMaker and automating processes to meet business needs. In wrapping up this guide, we will explore the ProcessMaker mobile app in the next chapter and beginner options for deploying ProcessMaker to a production environment.

CHAPTER 18

Going Mobile

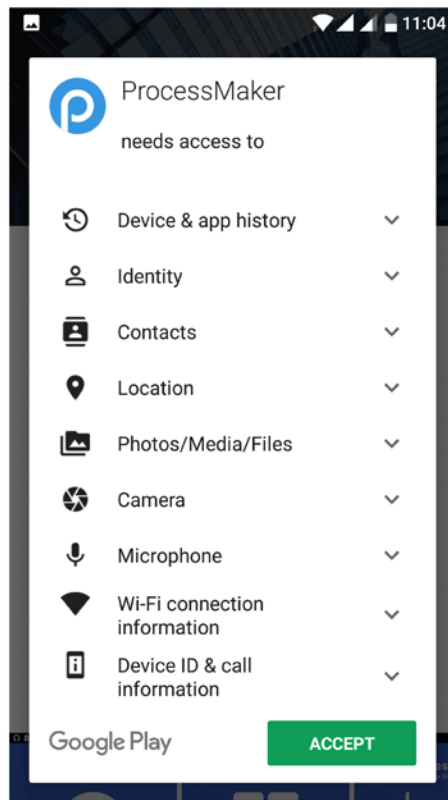
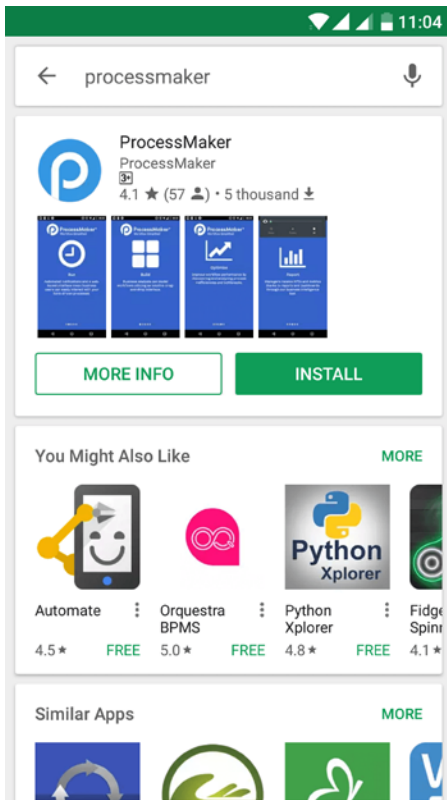
It has been an enlightening journey thus far, and we have seen how ProcessMaker can help automate paper-based approval processes. All we have done so far has been on our local PC, but in a real use case, we would deploy our business processes to a system that can be accessed by others. In this chapter, we begin with a look at the ProcessMaker mobile app and how it allows us to access our processes on the go and then in Chapter 19 we'll set up ProcessMaker on a cloud server so it can be accessed from anywhere with Internet access. Let's get started.

ProcessMaker Mobile Apps

ProcessMaker comes with mobile apps for both iOS and Android devices. The mobile apps allow users to run cases on the go. The mobile apps are limited to running cases for end users and do not include the administrative features. These are only available in the web app. We begin our exploration of the mobile apps by downloading the app from the Google Play store or the Apple App store, depending on the operating system of your mobile device.

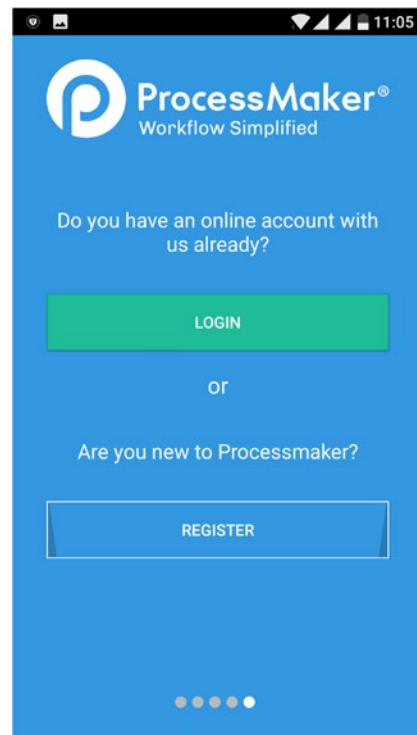
Install the App

On the store for your platform, search for ProcessMaker and install the app. For this guide, I will be using an Android device, but the configuration steps are similar for both Android and iOS. The app page is shown as in the following image. Click the Install button and accept the permissions requested.



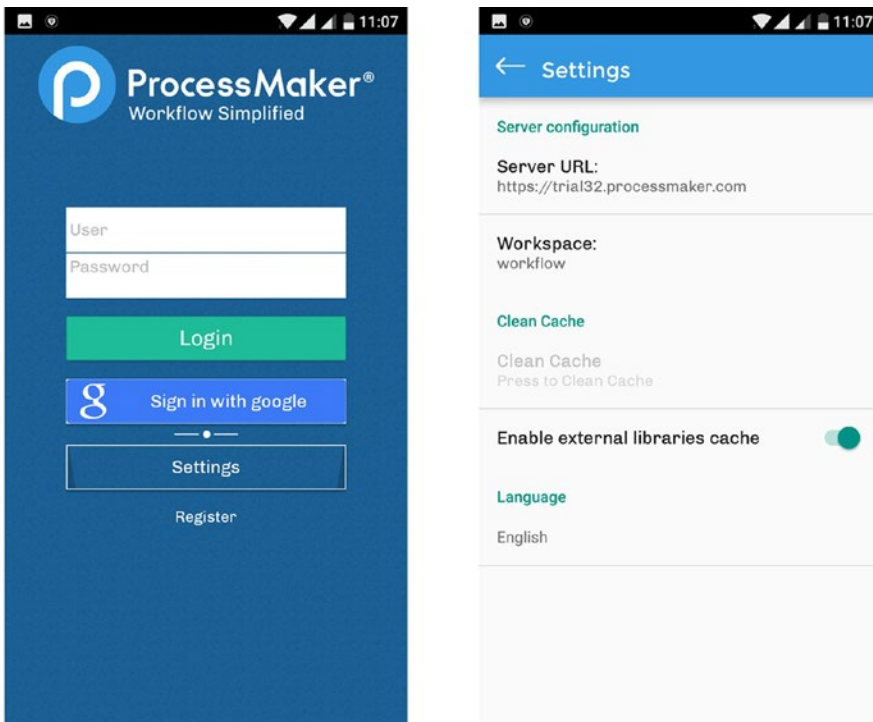
Installing the app

Once the app is installed, the intro slide is displayed as shown next. Swipe to the last slide, which displays the options to log in or register. The Register option allows you to sign up for a trial of the ProcessMaker Enterprise Edition in the cloud. However, for the purpose of our exploration, we will be connecting to our already installed local instance.



Newly installed app intro slides

Click the Login button to display the login screen as shown here.



Login and Settings Configuration

The Sign In with Google button allows you log in with a Google account that is already registered on the ProcessMaker instance the app is connecting to and is only available in the Enterprise edition.

The Settings button launches the setting screen shown on the left in the image above that allows us configure the instance (that is, the ProcessMaker installation) to which the app should be connected. By default, this is set to the ProcessMaker Enterprise trial URL. We can also configure the workspace on the instance which is set to the default workflow workspace.

To connect to our local ProcessMaker installation, we will change the Server URL in the settings to the URL of our installation. You might be wondering how we make our local PC available over the Internet for the mobile app to connect to. We will use a nifty service called ngrok that allows us “create a secure public URL (<https://yourapp.ngrok.io>) to a local webserver on your machine.”

Install ngrok for Remote Access

Ngrok is a service that allows you make a local web server or app running on your local computer, just like our installed Bitnami ProcessMaker stack, available over the Internet. By using ngrok, we will be able to configure our mobile app to access all the processes we have built on our local ProcessMaker installation. You can also share your work with friends and colleagues over the Internet.

To install ngrok, open the ngrok download page (<https://ngrok.com/download>) and this displays the links to download the installer for a variety of operating systems and architecture as shown next.

ngrok

Home Download Docs Product FAQ Login

Download and Installation

ngrok is easy to install. Download a single binary with *zero run-time dependencies* for any major platform. Unzip it and then run it from the command line.

Step 1: Download ngrok

Mac OS X 64-Bit	Download
Windows 64-Bit	Download
Linux 64-Bit	Download
Linux ARM	Download
FreeBSD 64-Bit	Download

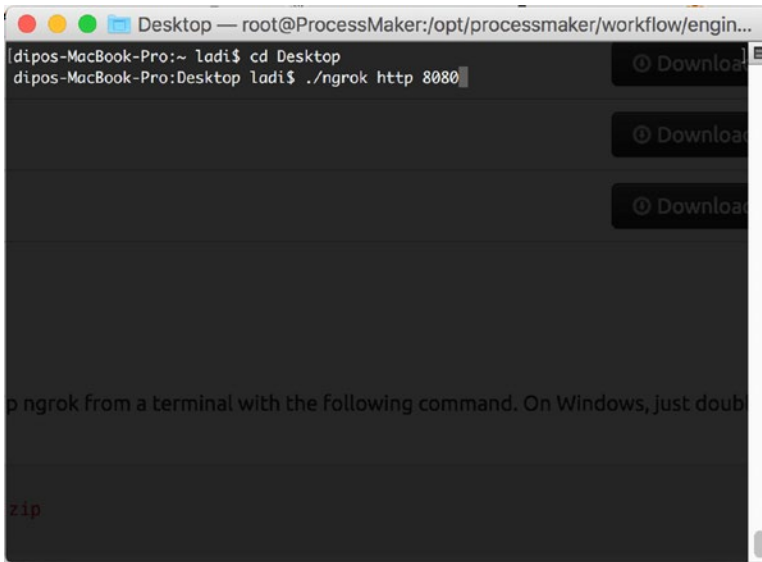
[32-bit platforms](#)

The ngrok download page

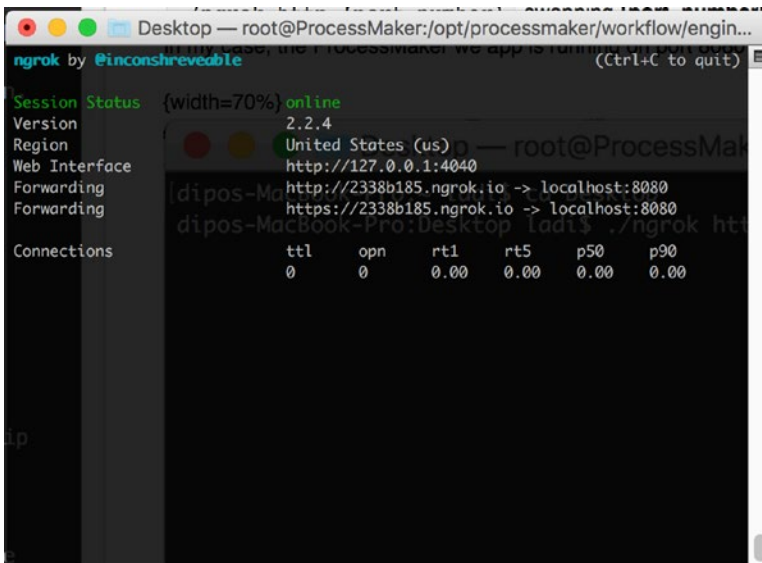
Download the appropriate installer for your Operating System. Save the downloaded installer to a path on your system and unzip it.

Launch ngrok on Mac OS X

If you're using Mac OS X, in a terminal change to the directory to which you unzipped ngrok and enter the command `./ngrok http {port number}`, replacing `{port number}` with your corresponding port number and pressing Enter. In my case, the ProcessMaker web app is running on port 8080, as shown in the following image.



This launches ngrok as shown next.



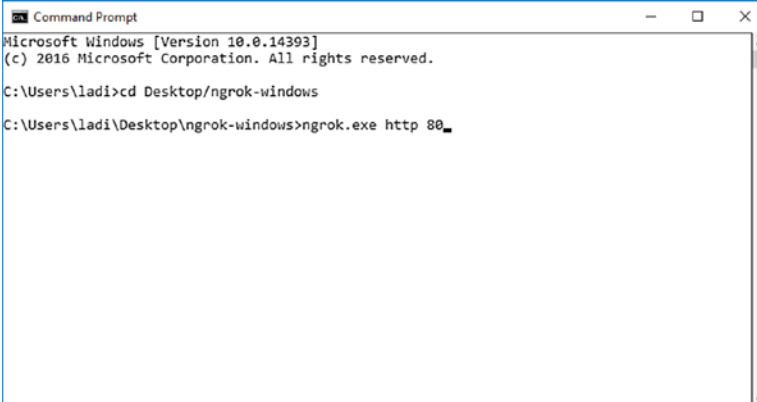
Once launched, ngrok provides us with a forwarding address that we can use to access the web server. In the screen shot, the assigned URL is <https://2338b185.ngrok.io> and we can see it can be accessed on both HTTP and HTTPS protocols.

To verify that our ProcessMaker instance is now available over the Internet, open a web browser on any device with an Internet access and enter the URL assigned by ngrok.

🔑 Opening the URL on a Nonstandard Port If you're using a port other than port 80, the URL might try redirecting to the port on the ngrok-assigned URL, <http://2338b185.ngrok.io:8080/sysworkflow/en/neoclassic/login/login>. If this happens, remove the port portion (:8080) of the URL.

Launch ngrok on Windows

If you're using Windows, unzip the downloaded installer and launch the Command Prompt. In the Command Prompt window, change directory to the folder where you unzipped ngrok and enter the command `ngrok.exe http {port number}`, replacing `{port_number}` with the corresponding port number and pressing Enter.



```


Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\ladi>cd Desktop/ngrok-windows

C:\Users\ladi\Desktop\ngrok-windows>ngrok.exe http 80_

```

This launches ngrok as shown next.



```

ngrok by @inconshreveable (Ctrl+C to quit)

Session Status      online
Version             2.2.4
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://26ecca99.ngrok.io -> localhost:80
                    https://26ecca99.ngrok.io -> localhost:80

Connections         ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00  0.00  0.00  0.00

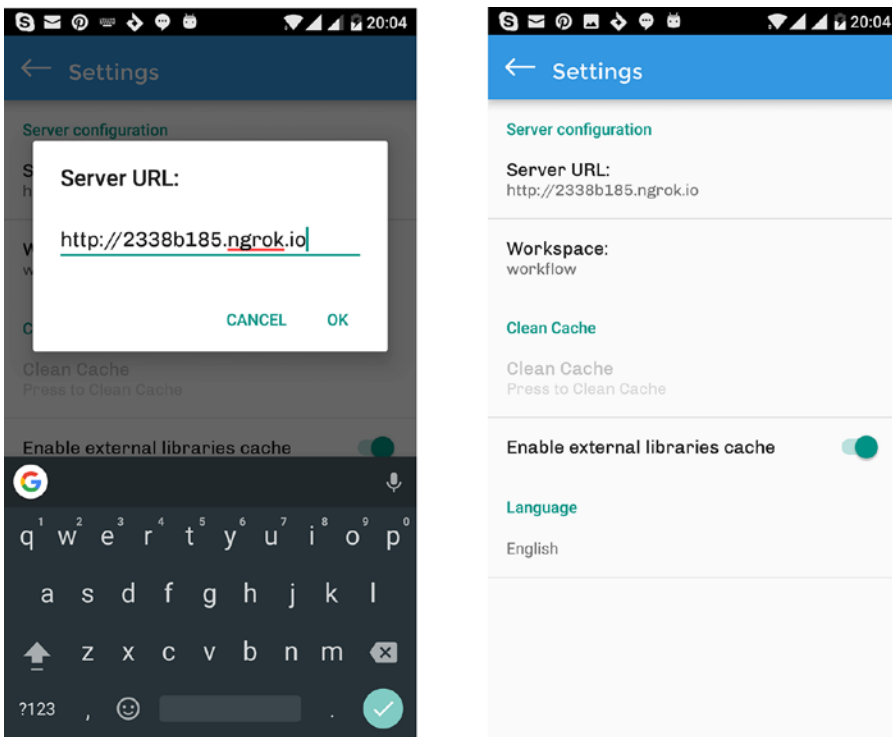
```

Once launched, ngrok provides us with a forwarding address that we can use to access the web server. In the screen shot, the assigned URL is <https://26ecca99.ngrok.io>, and we can see that it can be accessed on both HTTP and HTTPS protocols.

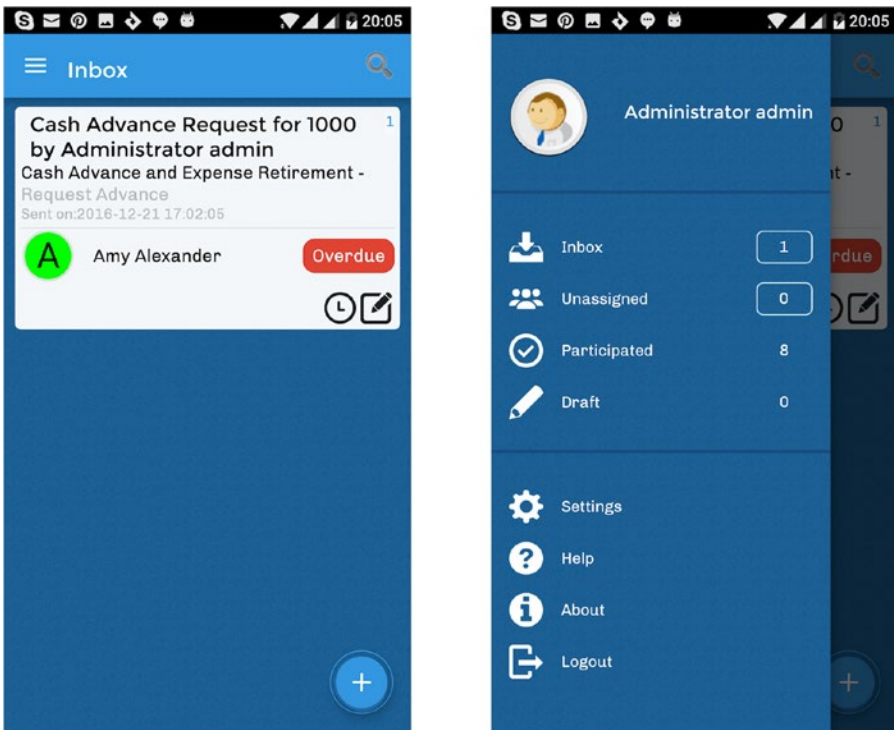
Configure Mobile App Settings

Now that our ProcessMaker instance is available over the Internet, the next step is to configure the settings of the mobile app to use the new URL.

Open the mobile app on your device and, on the login screen, click the Settings button to display the Settings page as shown here. Change the Server URL to the ngrok-assigned URL and save the settings.



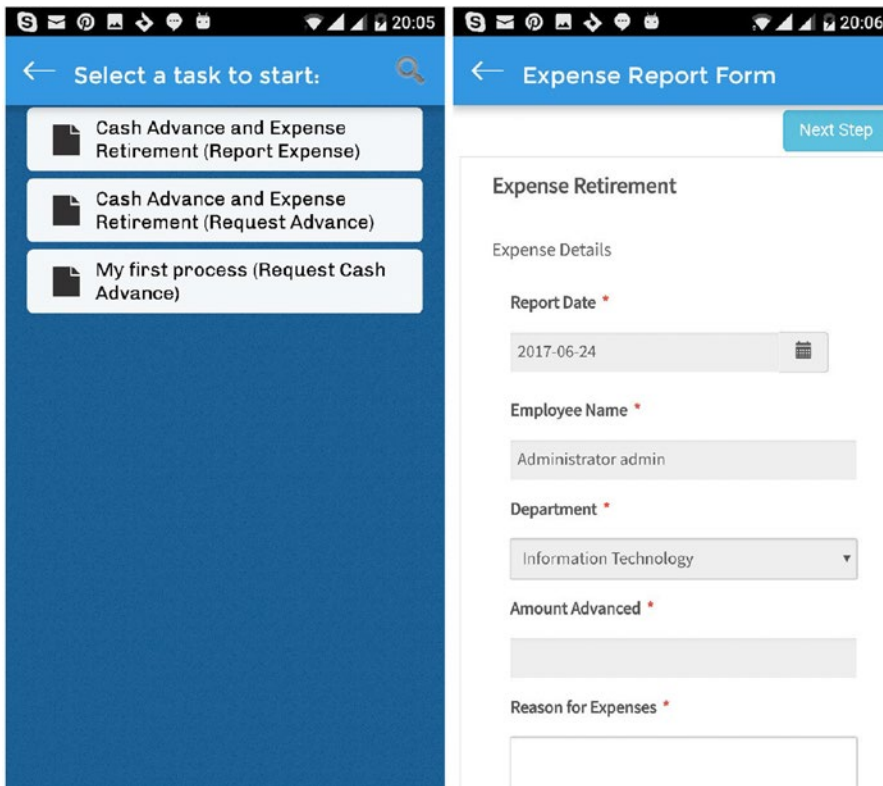
With the settings saved, return to the login screen and enter the username and password of the Admin user or any other user in ProcessMaker. On successful authentication, the inbox is displayed as shown next. Click the navigation icon to display the menu options. We are now mobile!



Go ahead and explore the other menu options.

Create a Case

You can try creating a new case by tapping the Create (plus) icon in the bottom-right corner of the inbox screen. This displays the available list of starting tasks. Tap one of the displayed tasks and start a new case. The dynaform is displayed, showing the fields similar to the mobile responsive preview in the Dynaform Designer. Go ahead and submit the case.



Deploying to Production

You might be thinking, “This is cool, but I definitely don’t want everyone connecting to my system to run their cases. What if I shut down my system or I lose network connectivity?” I agree with you 100%. What we have so far is good for developing and testing that the process we have designed works, but to make it available to others, we need to deploy it to a server that can be accessed by everyone in the organization.

Depending on your organization’s policies, you might want to deploy it to a cloud server so that it can be accessed over the Internet or you might want to deploy it to an on-premise server in your local network and have it accessible only from devices on the network.

Irrespective of your choice, the next chapter explores the options for installing ProcessMaker on a cloud server, and most of the steps we will cover are applicable to both cloud and on-premise deployments.

Another option for deployment is to sign up for the ProcessMaker cloud. This requires no installation on your part, and you can dive right into creating your users, importing your processes from your local PC and start using it straight away. It comes with a 30 day free trial (<https://www.processmaker.com/etrialreg>) and you can sign up at no cost.

CHAPTER 19

Installing ProcessMaker on a Cloud Server

For the purpose of our demonstration, we will be deploying to Digital Ocean, where we will set up a VPS (Virtual Private Server) and install ProcessMaker. We will then be able to access our ProcessMaker installation over the Internet from anywhere and make it available to our colleagues at MSB Corp. Let's get started.



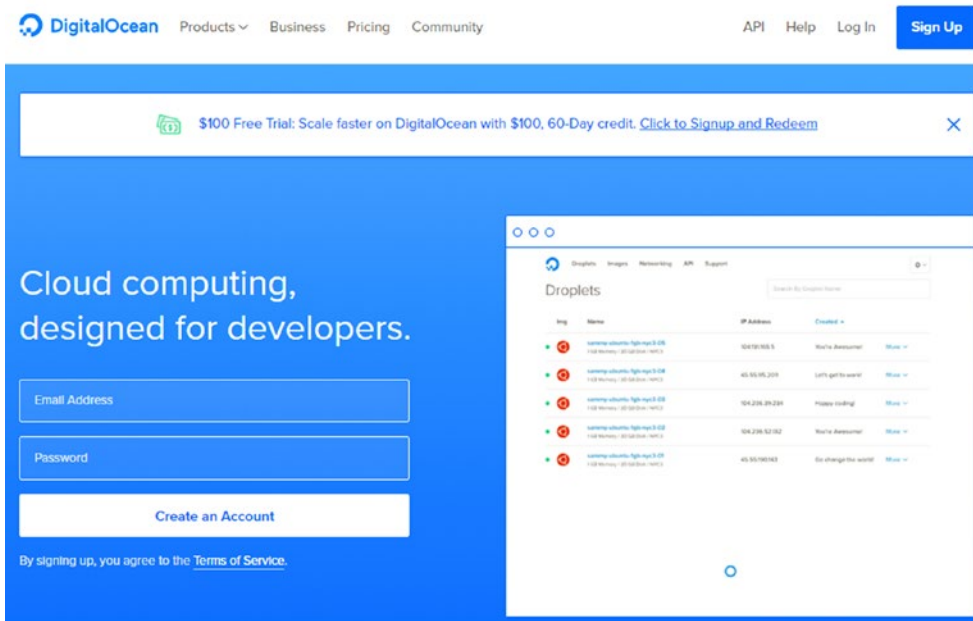
What is DigitalOcean? DigitalOcean is a simple and robust cloud computing platform, designed for developers. It allows developers easily spin up a Virtual Private Server (called a *droplet*) within a minute and add other features for easily managing the servers.

Getting a DigitalOcean Account

First we need to create an account. You can sign up on the DigitalOcean website, <https://www.digitalocean.com>.

Register for Your Account

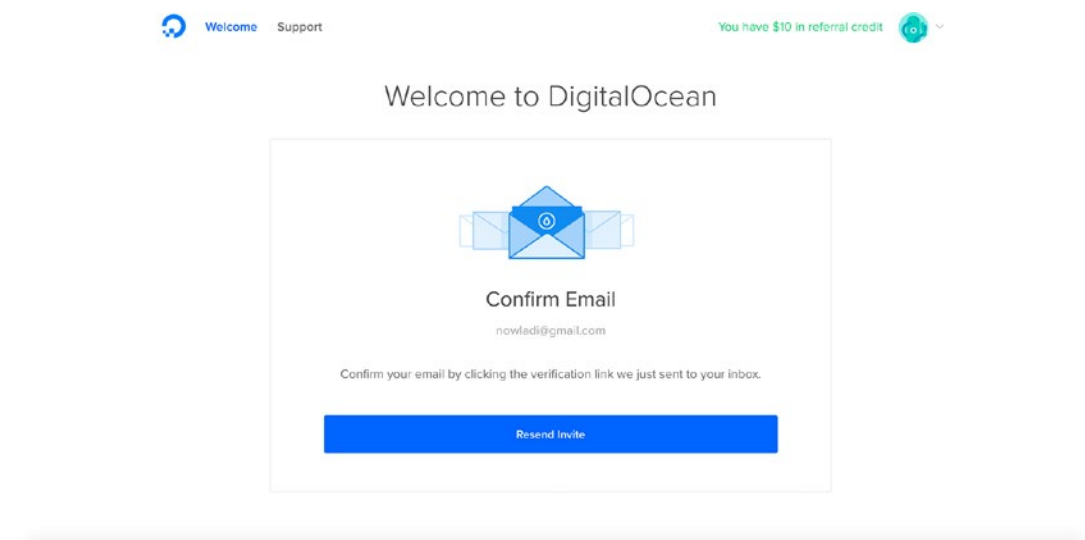
The landing page for Digital Ocean is displayed as shown next. Enter your email address and a password and click the Create an Account button.



DigitalOcean landing page

Confirm Your Email Address

The next step is to confirm your email address by clicking the link sent to the email address used to register.



DigitalOcean Email Confirmation page

Account Verification

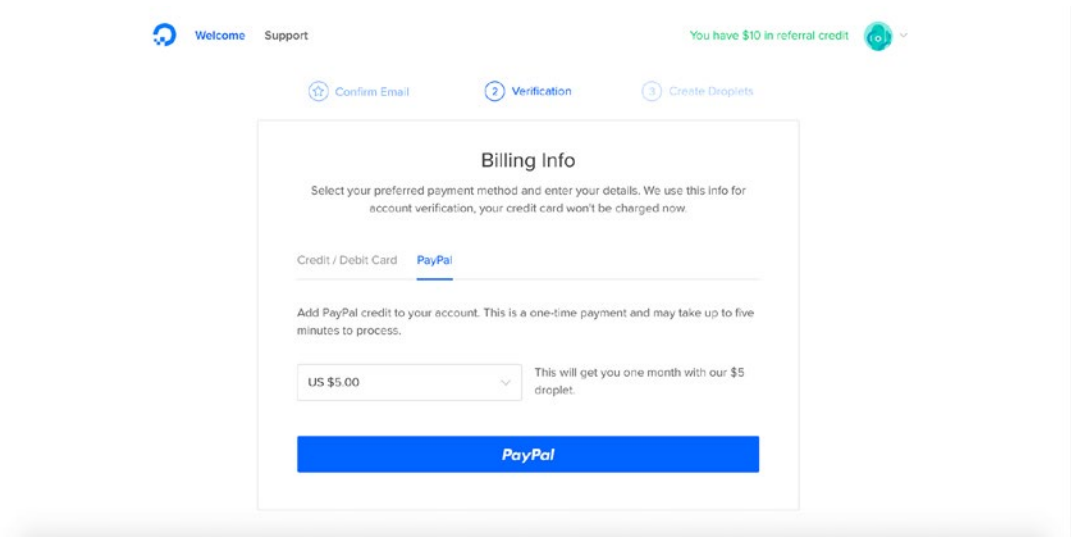
Once you confirm your email, the next step is to verify your account. DigitalOcean requires you to provide your billing info as a means of verification as shown next. If you have a credit card, enter the details and click the Save Card button.

The screenshot shows the DigitalOcean account verification page. At the top, there is a navigation bar with 'Welcome' and 'Support' links, and a notification that says 'You have \$10 in referral credit'. Below this is a progress indicator with three steps: '1 Confirm Email', '2 Verification' (the current step), and '3 Create Droplets'. The main content area is titled 'Billing Info' and contains the following text: 'Select your preferred payment method and enter your details. We use this info for account verification, your credit card won't be charged now.' There are two tabs: 'Credit / Debit Card' (selected) and 'PayPal'. Under 'ENTER CARD DETAILS', there is a card number field with '4242 4242 4242 4242' and a 'MM / YY CVC' field. Under 'BILLING ADDRESS', there are fields for 'First Name', 'Last Name', and 'Street Address'.

DigitalOcean verification page

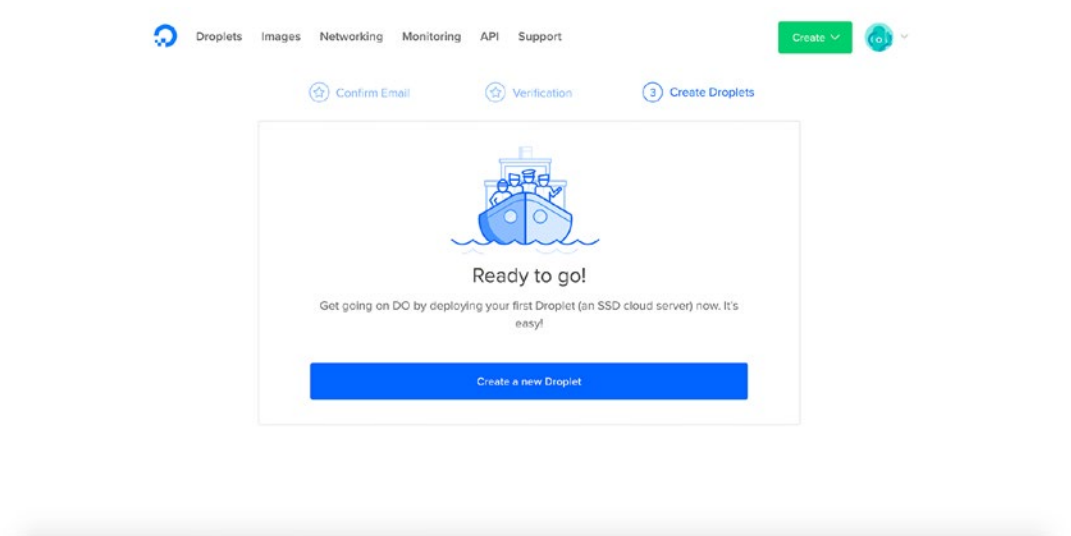
⚠ Debit and Pre-Paid Cards Are Not Accepted DigitalOcean does not currently accept debit or pre-paid cards for billing info verification. If you do not have a credit card, you can use the PayPal option. This will, however, require you to make a deposit of at least \$5.

Alternatively, you can make a deposit of a minimum of \$5 via PayPal to verify your account, as shown here.



DigitalOcean PayPal option

Once verification is done, you are now ready to create a droplet as shown in the following screen. Click the Create a New Droplet button to get started.









Creating Your Droplet (Virtual Private Server)

The Create Droplets page is displayed, and we can see a variety of Linux distribution images to choose from. For this guide, we will be using CentOS 7 as it is one of the officially supported ProcessMaker stacks (http://wiki.processmaker.com/3.1/Supported_Stacks). Select CentOS as shown here.

Choose an image ?

[Distributions](#) [One-click apps](#)

 Ubuntu Select version ▾	 FreeBSD Select version ▾	 Fedora Select version ▾	 Debian Select version ▾	 CoreOS Select version ▾	 CentOS 7.3.1611 x64 ▾
---	--	---	---	---	---

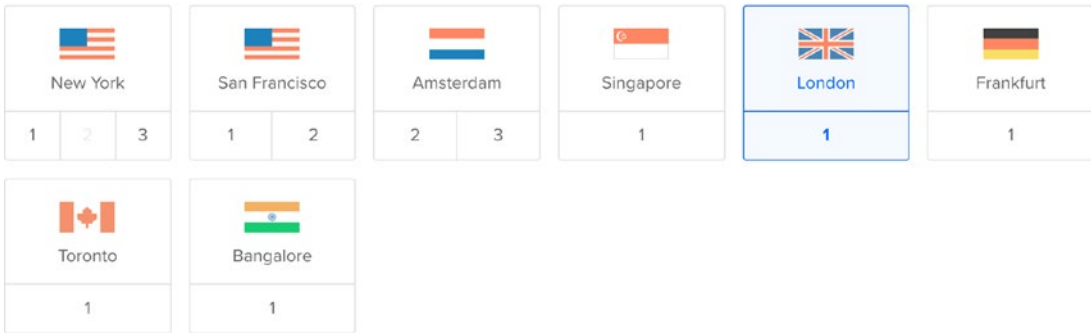
Next, we choose a size for our droplet. The droplets are billed per hour, let's choose the \$10 per month droplet.

Choose a size

\$5/mo \$0.007/hour 512 MB / 1 CPU 20 GB SSD disk 1000 GB transfer	\$10/mo \$0.015/hour 1 GB / 1 CPU 30 GB SSD disk 2 TB transfer	\$20/mo \$0.030/hour 2 GB / 2 CPUs 40 GB SSD disk 3 TB transfer	\$40/mo \$0.060/hour 4 GB / 2 CPUs 60 GB SSD disk 4 TB transfer	\$80/mo \$0.119/hour 8 GB / 4 CPUs 80 GB SSD disk 5 TB transfer	\$160/mo \$0.238/hour 16 GB / 8 CPUs 160 GB SSD disk 6 TB transfer
\$320/mo \$0.476/hour 32 GB / 12 CPUs 320 GB SSD disk 7 TB transfer	\$480/mo \$0.714/hour 48 GB / 16 CPUs 480 GB SSD disk 8 TB transfer	\$640/mo \$0.952/hour 64 GB / 20 CPUs 640 GB SSD disk 9 TB transfer			

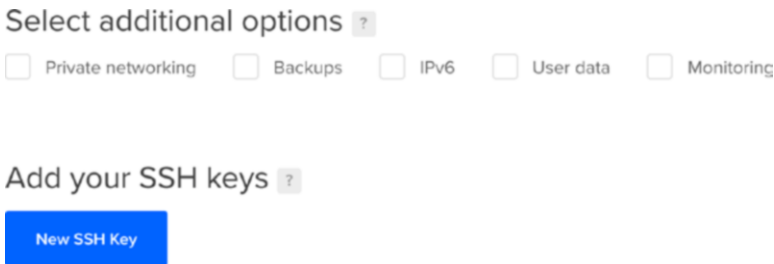
Our next line of action is to choose the data center where our droplet should be located. Depending on your location, you will want to select a data center nearest to you or your potential users. I will be using the London data center as shown here.

Choose a datacenter region



The next section presents a set of options such as Private Networking, Backups, IPv6, User Data, and Monitoring. You can mouse over each option to learn more about it. For example, Private Networking allows you to set up a private network between your droplets hosted in the same data center. In a larger deployment, we would have a separate droplet or droplets (if using a cluster) for the database and another droplet or droplets (if using a web farm) for the application.

For the purpose of this book, we will be keeping things simple and will only need one droplet. We can choose the Monitoring option to keep a tab on resource usage on our droplet.



In addition to the options, we can also add SSH Keys, which allow us to log in to the droplet without a username and password. We will be sticking with the traditional username and password approach, but you can learn more about securing your droplets with SSH keys here: <https://www.digitalocean.com/community/tutorials/how-to-use-ssh-keys-with-digitalocean-droplets>.



SSH Keys An SSH key is an access credential in the SSH protocol. Its function is similar to that of usernames and passwords, but the keys are primarily used for automated processes and for implementing single sign-on by system administrators and power users. Source: <https://www.ssh.com/ssh/key/>.

Finalize and create

How many Droplets?

Deploy multiple Droplets with the same [configuration](#).

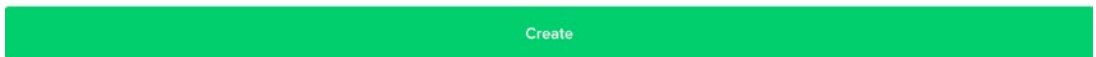
— 1 Droplet +

Choose a hostname

Give your Droplets an identifying name you will remember them by. Your Droplet name can only contain alphanumeric characters, dashes, and periods.

processmaker

[Add Tags](#)




Finally, specify a hostname for the droplet (I am using **processmaker**) and click the Create button. The droplet creation is initialized as shown next and should take less than a minute.




Droplets

Search by Droplet name

[Droplets](#) [Volumes](#)

Name	IP Address	Created	Tags
 processmaker 1 GB / 30 GB Disk / LON1 - CentOS 7.3.1611 x64			

Once ready, the IP address of the droplet is displayed as shown in this screen. You will also get an email containing the IP address of your new droplet, along with the username and password.

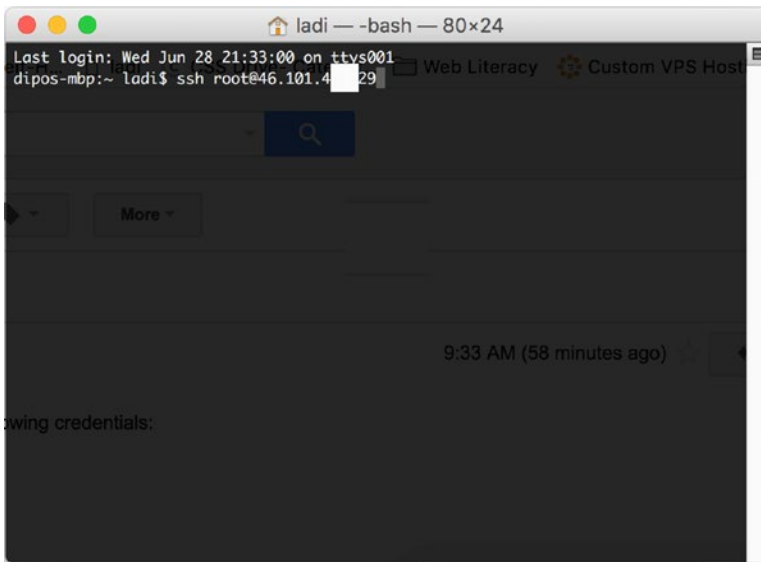
Name	IP Address	Created	Tags
 processmaker 1 GB / 30 GB Disk / LON1 - CentOS 7.3.1611 x64	46.101.4[REDACTED]29	Good to go!	More

Connecting to the Droplet

We now have a server in the cloud on which we can install ProcessMaker, but in order to do so, we have to connect to it first.

Using Mac or Linux

If using a Mac or Linux system, simply open a terminal window and enter the following command as shown in the screen: `ssh root@ip_address`, where `ip_address` is the IP address of the newly created droplet, and press Enter.



You will be prompted to confirm the authenticity of the host. Type **yes** and press Enter. Next you are prompted for the root password. Enter the password you received via email after creating the droplet. Since this is the first time you are logging in, you will be prompted to change the password. If you enter a weak password, the droplet will display a BAD PASSWORD message as shown in the following image.


```

Last login: Wed Jun 28 21:33:00 on ttys001
[dipos-mbp:~ ladi$ ssh root@46.101.4...29
The authenticity of host '46.101.4...29 (46.101.48.229)' can't be established.
ECDSA key fingerprint is SHA256:C3gDTy85303xM7FoMzFeokMRK2JaXSXks7LJ1l+evn4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '46.101.4...29' (ECDSA) to the list of known hosts.
[root@46.101.4...29's password:
Connection closed by 46.101.4...29
[dipos-mbp:~ ladi$ ssh root@46.101.4...29
[root@46.101.4...29's password:
You are required to change your password immediately (root enforced)
Last failed login: Sat Jul 1 08:43:28 UTC 2017 from [redacted] on ssh:notty
There were 12 failed login attempts since the last successful login.
Changing password for root.
30 June 2017, 10:56 PM
(Current) UNIX password:
New password: stuff
BAD PASSWORD: The password fails the dictionary check - it is too simplistic/systematic
New password: Maxani - Kilodetofem@13
[Retype new password:
[root@processmaker:~]# dipodiamond/Kilodetofem@13
go daddy - Kilodetofem@13
Th1nk0ut$1d3TheB0*
Th1nk0ut$1d3TheB0*

```

Enter a strong and secure password and retype it to confirm it. Make sure it is a password you can remember. Once accepted, you are logged in to the droplet. Type the clear command and press Enter to clear the screen.

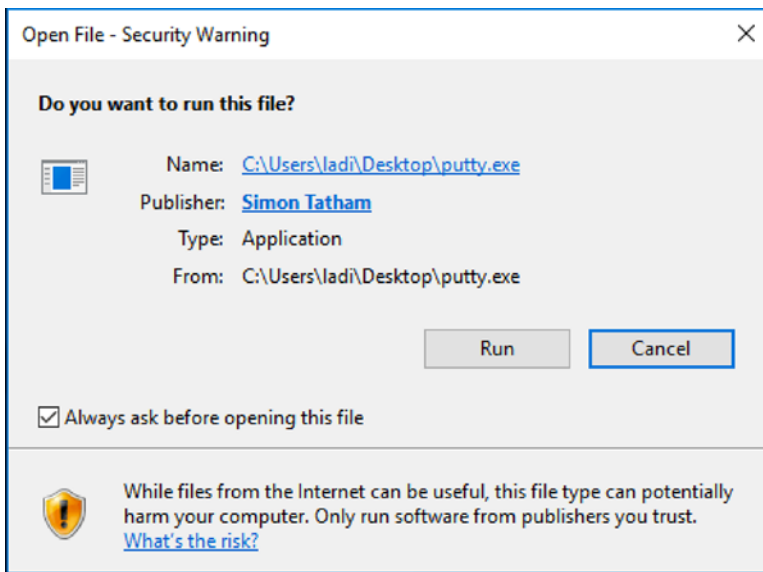
Using Windows

For Windows users, you will need to install PuTTY, an open source SSH and Telnet client for Windows that allows you securely connect to remote Linux servers from a local Windows computer.

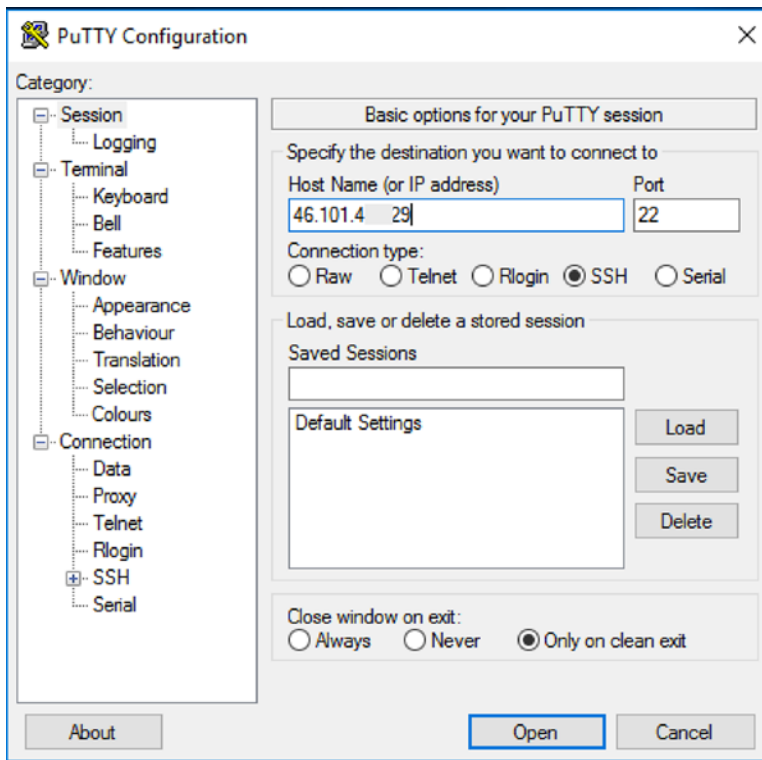
Go to the PuTTY download page (<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>). Go to the Alternative Binary section as shown next and download the putty.exe version that matches the architecture of your computer.



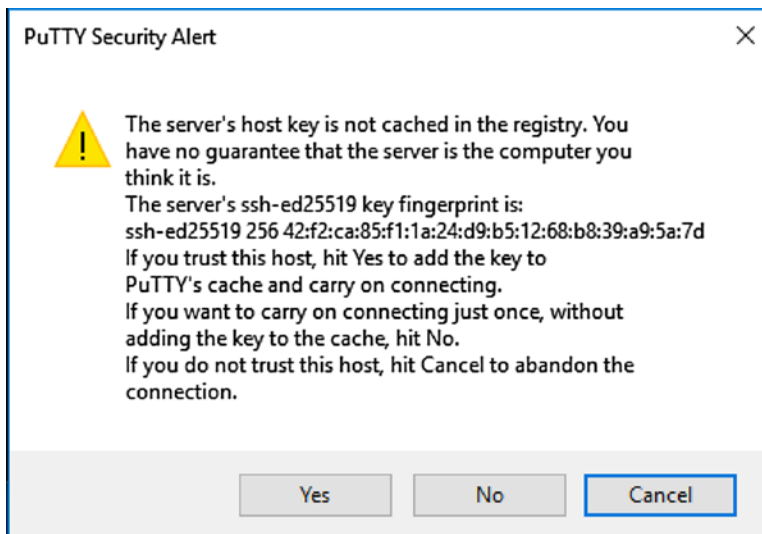
Once it is downloaded, double-click the executable to launch the PuTTY client. If prompted with a security warning like the following, click Run.



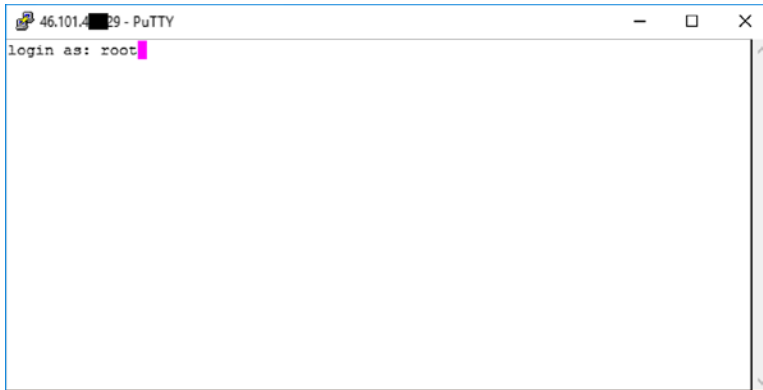
This launches the PuTTY client as shown next. Enter the IP address of your droplet and ensure that the port is set to 22. You can click the Save button to store the settings. Click the Open button to connect to the droplet.



This prompts a security alert to confirm the authenticity of the host, as shown next. Click the Yes button to trust the host and connect.



This launches the PuTTY terminal, as shown next. Enter the username root and press Enter. You should be prompted for the root user password. Enter the password sent to you via email when the droplet was created. Since this is your first login, you will be prompted to change the password. Enter a strong and secure password. Once that is accepted, you should be logged in to the server.



🔑 Securing your Droplet To secure your droplet, you can check out these additional guides on DigitalOcean: “Initial Server Set up with CentOS 7” (<https://www.digitalocean.com/community/tutorials/initial-server-set-up-with-centos-7>), or “Additional Recommended Steps for New CentOS 7 Servers” (<https://www.digitalocean.com/community/tutorials/additional-recommended-steps-for-new-centos-7-servers>).

Installing ProcessMaker

Now that we are connected to our server, the next thing to do is install the Apache, PHP, and MySQL stack required by ProcessMaker. The steps we will follow are from the ProcessMaker wiki (http://wiki.processmaker.com/3.1/ProcessMaker_CentOS_RHEL_Installation), with a few tweaks. Let’s get started.

Remove MariaDB

MariaDB is a distribution of MySQL installed by default on CentOS and can conflict with the MySQL database we will install later. To remove it, type the following command in the terminal connected to your droplet and press Enter:

```
yum -y remove mariadb*
```

```
[[root@processmaker ~]# yum -y remove mariadb*
```

This removes the MariaDB packages from the server and when it's done, a Complete! message is shown.

Install Apache

Next we install the Apache web server with `mod_ssl`. As before, in the terminal, type the following command and press Enter:

```
yum -y install httpd mod_ssl
```

When it's completed, we start the Apache service with the following command:

```
service httpd start
```

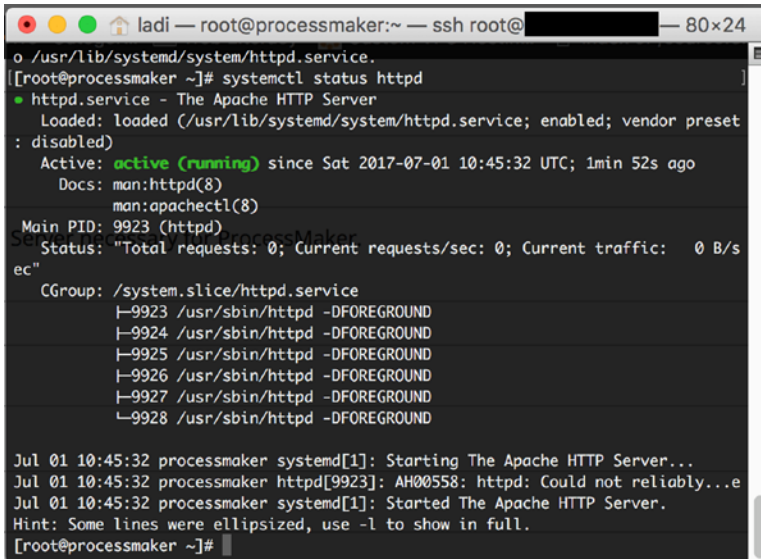
Next we also configure the service to start automatically after a server reboot with the following command.

```
chkconfig httpd on
```

Finally, let us check that the Apache (`httpd`) service is running. Enter the following command to check the status of the service.

```
systemctl status httpd
```

The service status should be displayed as active (running) as shown here.



```
o /usr/lib/systemd/system/httpd.service.
[root@processmaker ~]# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset
: disabled)
   Active: active (running) since Sat 2017-07-01 10:45:32 UTC; 1min 52s ago
     Docs: man:httd(8)
           man:apachectl(8)
   Main PID: 9923 (httd)
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic:  0 B/s
ec"
   CGroup: /system.slice/httpd.service
           └─9923 /usr/sbin/httd -DFOREGROUND
           └─9924 /usr/sbin/httd -DFOREGROUND
           └─9925 /usr/sbin/httd -DFOREGROUND
           └─9926 /usr/sbin/httd -DFOREGROUND
           └─9927 /usr/sbin/httd -DFOREGROUND
           └─9928 /usr/sbin/httd -DFOREGROUND

Jul 01 10:45:32 processmaker systemd[1]: Starting The Apache HTTP Server...
Jul 01 10:45:32 processmaker httd[9923]: AH00558: httd: Could not reliably...
Jul 01 10:45:32 processmaker systemd[1]: Started The Apache HTTP Server.
Hint: Some lines were ellipsized, use -l to show in full.
[root@processmaker ~]#
```

Install PHP 5.6

Centos 7 uses the EPEL repository, which comes with PHP 5.4 by default. ProcessMaker however requires PHP 5.6. To get PHP 5.6, we add the Webtatic Yum Repository (<https://webtatic.com/projects/yum-repository/>) using the following commands:

```
rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm
```

With our repos updated, we then proceed to install PHP 5.6 and the extensions required by ProcessMaker using these commands:

```
yum -y install php56w
yum -y install php56w-mysqlnd php56w-gd php56w-soap php56w-ldap php56w-xml
php56w-mbstring php56w-cli php56w-curl php56w-mcrypt php56w-devel
php56w-pecl-apcu
```

Once installation is complete, you can verify that the PHP version installed with the command `php -v` and also check that the required PHP modules were installed with `rpm -qa |grep php` as shown next.

```
[root@processmaker ~]# php -v
PHP 5.6.30 (cli) (built: Jan 19 2017 22:31:39)
Copyright (c) 1997-2016 The PHP Group
Zend Engine v2.6.0, Copyright (c) 1998-2016 Zend Technologies
[root@processmaker ~]# rpm -qa |grep php
php56w-common-5.6.30-1.w7.x86_64
php56w-5.6.30-1.w7.x86_64
php56w-process-5.6.30-1.w7.x86_64 php56w-cli php56w-curl php56w-mcrypt
php56w-pdo-5.6.30-1.w7.x86_64
php56w-devel-5.6.30-1.w7.x86_64
php56w-mcrypt-5.6.30-1.w7.x86_64
php56w-mbstring-5.6.30-1.w7.x86_64
php56w-soap-5.6.30-1.w7.x86_64
php56w-cli-5.6.30-1.w7.x86_64
php56w-xml-5.6.30-1.w7.x86_64
php56w-pear-1.10.4-1.w7.noarch
php56w-mysqldb-5.6.30-1.w7.x86_64
php56w-gd-5.6.30-1.w7.x86_64
php56w-pecl-apcu-4.0.11-1.w7.x86_64
php56w-ldap-5.6.30-1.w7.x86_64
[root@processmaker ~]#
```

Install MySQL 5.5.X

Our next step is to download and install the MySQL database, and the recommended version is 5.5.X.

⚠ Unsupported MySQL STRICT Mode ProcessMaker is not compatible with MySQL STRICT mode, which is turned on by default in MySQL 5.7. This version of MySQL is NOT part of any official ProcessMaker stack.

To download the MySQL installer, go to the MySQL Downloads page (<https://dev.mysql.com/downloads/mysql/5.5>). For the Operating System, select Red Hat Enterprise Linux /Oracle Linux and set the Select OS Version to Red Hat Enterprise Linux 7/Oracle Linux 7 (x86, 64-bit). Click the Download button displayed as shown here.

RPM Bundle	5.5.56	164.9M	Download
<small>(MySQL-5.5.56-1.el7.x86_64.rpm-bundle.tar)</small>		<small>MD5: 92f064843715110c3d5d974116126eb1 Signature</small>	

This redirects to the download page as shown next. There are options to log in or sign up for an Oracle web account. At the bottom, there’s a direct download link (**No thanks, just start my download**). We will skip the login/signup option and use the direct link.

Begin Your Download

MySQL-5.5.56-1.el7.x86_64.rpm-bundle.tar

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system
- Comment in the MySQL Documentation

Direct Download Link

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

No thanks, just start my download.

To download the link directly to the server, right-click the link and in the context menu, select Copy Link Address. The address should be something like https://dev.mysql.com/get/Downloads/MySQL-5.5/MySQL-5.5.56-1.el7.x86_64.rpm-bundle.tar.

Now in the terminal, enter the command `wget download_url`, replacing `download_url` with the copied link address as in the following:

```
wget https://dev.mysql.com/get/Downloads/MySQL-5.5/MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar
```

```
[root@processmaker ~]# wget https://dev.mysql.com/get/Downloads/MySQL-5.5/MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar
--2017-07-01 12:16:16-- https://dev.mysql.com/get/Downloads/MySQL-5.5/MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar
Resolving dev.mysql.com (dev.mysql.com)... 137.254.60.11
Connecting to dev.mysql.com (dev.mysql.com)|137.254.60.11|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://cdn.mysql.com//Downloads/MySQL-5.5/MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar [following]
--2017-07-01 12:16:17-- https://cdn.mysql.com//Downloads/MySQL-5.5/MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar
Resolving cdn.mysql.com (cdn.mysql.com)... 23.40.97.175
Connecting to cdn.mysql.com (cdn.mysql.com)|23.40.97.175|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 172933120 (165M) [application/x-tar]
Saving to: 'MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar'

100%[=====>] 172,933,120 24.1MB/s in 6.0s

2017-07-01 12:16:23 (27.6 MB/s) - 'MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar' saved [172933120/172933120]

[root@processmaker ~]# █
```

This downloads the MySQL installer archive to the server as shown in the screen capture. To view the downloaded file, enter the command `ll` or `ls -lrt`. This displays the installation file we just downloaded. Next we extract the installer using the command `tar -xvf installer_archive`, replacing `installer_archive` with the name of the downloaded file as shown here:

```
tar -xvf MySQL-5.5.56-1.e17.x86_64.rpm-bundle.tar
```

This extracts a number of RPM files to the server. We will install the MySQL Server and MySQL Client RPM files. First we install a dependency required by the installer, using the following command:

```
yum install libaio
```

Once the dependency is installed, enter the next commands to install the MySQL Server and MySQL client, ensuring that the file name matches the MySQL version you have downloaded.

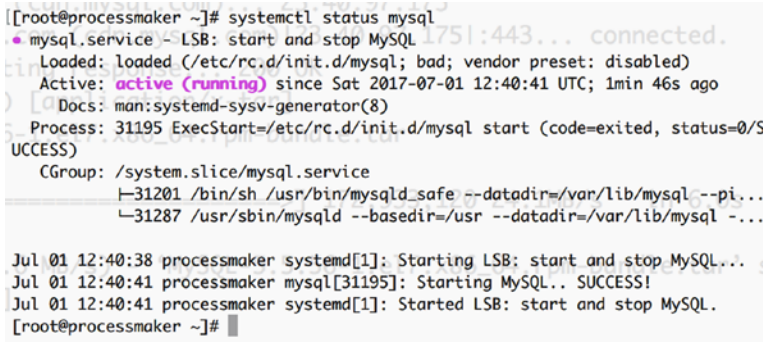
```
rpm -ivh MySQL-server-5.5.56-1.el7.x86_64.rpm  
rpm -ivh MySQL-client-5.5.56-1.el7.x86_64.rpm
```

Once the installation is complete, use the next commands to start the MySQL service and configure it to start automatically after a reboot:

```
systemctl start mysql  
chkconfig mysql on
```

To confirm that the MySQL service is running, use this command:

```
systemctl status mysql
```



Secure the MySQL Installation

Before using the installed MySQL, it is recommended to secure the installation. We do this by running the `mysql_secure_installation` command from the terminal. The resulting installer prompts with a series of questions that help set a password for the root user and other security settings. Go ahead and enter the command in the terminal and answer the prompted questions as shown here.

1. **Enter current password for root (enter for none):** Press Enter, since we do not have a root password yet.
2. **Set root password? [Y/n]:** Choose Y.

3. **New password:** Enter a strong and secure password you will remember. *Please note that ProcessMaker does NOT support special characters (such as: @ # \$ % ^ & (/) in the root password.*
4. **Re-enter new password:** Confirm the password just entered.
5. **Remove anonymous users? [Y/n]:** Choose Y.
6. **Disallow root login remotely? [Y/n]:** Choose Y.
7. **Remove test database and access to it? [Y/n]:** Choose Y.
8. **Reload privilege tables now? [Y/n]:** Choose Y.

Disable SELINUX

Next we disable SELINUX using the following commands:

```
echo "SELINUX=disabled" > /etc/selinux/config
echo "SELINUXTYPE=targeted" >> /etc/selinux/config
```

Enable Firewall and Open ProcessMaker ports

Our next line of action is to enable the firewall. To do so, enter these commands to start the firewall service and set it to start automatically after a reboot:

```
systemctl start firewalld chkconfig firewalld on
```

To confirm that the service is running, enter the following command, and you should see the status displayed as active similar to the image.

```
systemctl status firewalld
```

```

[[root@processmaker ~]# systemctl start firewalld
[[root@processmaker ~]# chkconfig firewalld on
Note: Forwarding request to 'systemctl enable firewalld.service'.
Created symlink from /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service to /usr/lib/systemd/system/firewalld.service.
Created symlink from /etc/systemd/system/basic.target.wants/firewalld.service to /usr/lib/systemd/system/firewalld.service.
[[root@processmaker ~]# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2017-07-01 13:00:11 UTC; 32s ago
     Docs: man:firewalld(1)
  Main PID: 31550 (firewalld)
    CGroup: /system.slice/firewalld.service
            └─31550 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid

Jul 01 13:00:10 processmaker systemd[1]: Starting firewalld - dynamic firewa...
Jul 01 13:00:11 processmaker systemd[1]: Started firewalld - dynamic firewa...
Hint: Some lines were ellipsized, use -l to show in full.
[[root@processmaker ~]#

```

With our firewall in place, we need to permit the ports used by ProcessMaker so that it can be accessible. The ProcessMaker web application will use port 80 by default and we will also add port 443 for SSL support. Enter the following commands to permit the ports and reload the firewall service.

```

firewall-cmd --zone=public --add-port=80/tcp --permanent
firewall-cmd --zone=public --add-port=443/tcp --permanent
firewall-cmd --reload

```

Download and Extract ProcessMaker Installer

So far we have been getting our server environment ready for ProcessMaker. At this point we need to download the ProcessMaker installer to our server. Let us head over to the ProcessMaker Sourceforge page (<https://sourceforge.net/projects/processmaker/files/ProcessMaker/>) to download ProcessMaker.



Working with Newer Versions The steps described next should work for newer versions of ProcessMaker, but we will be using version 3.1.x for this guide.

Summary **Files** Reviews Support News Wiki Mailing Lists

Looking for the latest version? [Download bitnami-processmakercommunity-3.2-0-osx-x86_64-installer.dmg \(123.2 MB\)](#)

Home / ProcessMaker

Name ↕	Modified ↕	Size ↕	Downloads / Week ↕
↑ Parent folder			
3.2	2017-05-16		786
3.1.3	2016-12-12		15
3.1.2	2016-10-25		4

The page displays a list of folders for the different ProcessMaker versions, starting with the most recent release. Click the folder for the most recent release of version 3.1 to display the page listing the installers as shown here.

Home / ProcessMaker / 3.1.3

Name ↕	Modified ↕	Size ↕	Downloads / Week ↕
↑ Parent folder			
README.txt	2016-12-12	5.8 kB	0
bitnami-processmakercommunity-3...	2016-12-09	147.9 MB	1
bitnami-processmakercommunity 3...	2016-12-09	144.8 MB	1
bitnami-processmakercommunity-3...	2016-12-09	120.5 MB	0
bitnami-processmakercommunity-3...	2016-12-09	125.8 MB	0
processmaker-3.1.3-community.tar.gz	2016-12-09	58.7 MB	13
Totals: 6 Items		597.8 MB	15

Right-click and copy link address

Look for the TAR archive installer as shown in the image, right-click it, and select Copy Link Address. Open a text editor and paste the address copied. It should look similar to the following address:

```
https://sourceforge.net/projects/processmaker/files/ProcessMaker/3.1/
processmaker-3.1.3-community.tar.gz/download
```

Remove the /download at the end of the URL so that the link ends with the tar.gz portion. Next use the wget command in your terminal to download the file:

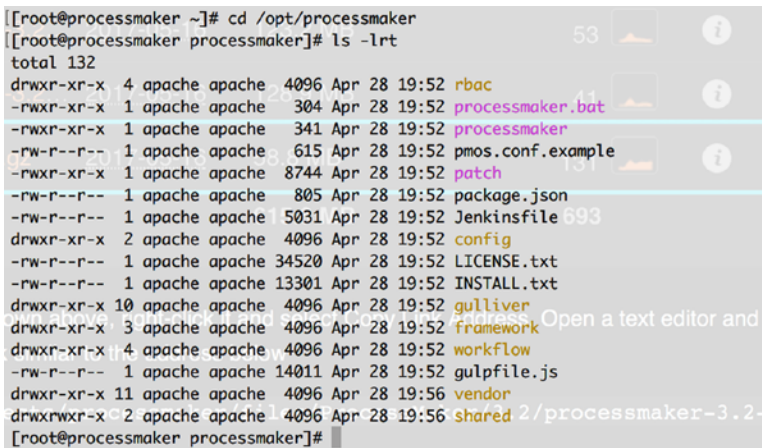
```
wget https://sourceforge.net/projects/processmaker/files/ProcessMaker/3.1/
processmaker-3.1.3-community.tar.gz
```

This downloads the ProcessMaker installer, and if you enter ll or ls -lrt in the terminal you should now see a file named processmaker-3.1.3-community.tar.gz.

Next extract the downloaded archive to the /opt folder using the command

```
tar -C /opt -xzf processmaker-3.1.3-community.tar.gz
```

Once the archive is extracted, change directory to the extracted folder using the command cd /opt/processmaker and list the contents of the directory with ll or ls -lrt. The list of extracted folders should be displayed as in the next screen.



Next, we set the permissions on the files and folders so that they can be accessed by ProcessMaker.

Enter the following commands:

```
chmod -R 770 shared workflow/public_html
chmod -R 770 gulliver/js gulliver/thirdparty/html2ps_pdf/cache

cd workflow/engine/
chmod -R 770 config content/languages plugins xmlform js/labels
```

Finally, we change the owner of the files to the apache user, which is the user account that will be used to run the Apache web server service and requires read and write permissions to the ProcessMaker application files. To do so, enter the following command in the terminal:

```
chown -R apache:apache /opt/processmaker
```

Configure Apache Web Server

We now have the ProcessMaker files on our server, but the Apache web server is unaware of the ProcessMaker installation files. We need to configure the Apache web server to point to the ProcessMaker files. First we remove the default website config file.

Go to the Apache config folder using the command `cd /etc/httpd/conf.d/`. Display the list of config files using the `ll` or `ls -l` command as shown here.

```
[[root@processmaker engine]# cd /etc/httpd/conf.d/
[[root@processmaker conf.d]# ll js gulliver/thirdparty/html2ps_pdf/c
total 32
-rw-r--r-- 1 root root 2926 Apr 12 21:03 autoindex.conf
-rw-r--r-- 1 root root 625 Jan 19 22:39 php.conf
-rw-r--r-- 1 root root 366 Apr 12 21:04 README
-rw-r--r-- 1 root root 9438 Apr 12 13:50 ssl.conf
-rw-r--r-- 1 root root 1252 Apr 12 13:50 userdir.conf
-rw-r--r-- 1 root root 824 Apr 12 13:50 welcome.conf
[[root@processmaker conf.d]#
```


Remove the `welcome.conf` file with the command

```
rm -rf welcome.conf
```

Next we copy the ProcessMaker config file from the ProcessMaker folder we extracted earlier to the Apache config folder

```
cp /opt/processmaker/pmos.conf.example /etc/httpd/conf.d/pmos.conf
```


Using a text editor like vim or nano, edit the `pmos.conf` file to match the code shown next, replacing `your_ip_address` with the IP address of the droplet and save your changes. We will set the `ServerName` to the IP address for now and when we get a domain name later on, we will update it to the domain name.

 **Using Text Editors** You can install the vim text editor on the server using the command `yum install -y vim`. To edit a file, use the command `vim path_to_filename` for example `vim /etc/httpd/conf.d/pmos.conf`.

You can grasp the basic concepts with this quick guide “[Vim 101: A Quick-and-Dirty Guide to Our Favorite Free File Editor](https://www.engadget.com/2012/07/10/vim-how-to/)” (<https://www.engadget.com/2012/07/10/vim-how-to/>), or “[Getting Started with Vim: An Interactive Guide](https://scotch.io/tutorials/getting-started-with-vim-an-interactive-guide)” (<https://scotch.io/tutorials/getting-started-with-vim-an-interactive-guide>).

```
#processmaker virtual host
<VirtualHost your_ip_address>
    ServerName "your_ip_address"

    DocumentRoot /opt/processmaker/workflow/public_html
    DirectoryIndex index.html index.php

    <Directory /opt/processmaker/workflow/public_html>
        Options Indexes FollowSymLinks MultiViews
        AddDefaultCharset UTF-8

        AllowOverride All

        Require all granted
        ExpiresActive On

    <IfModule mod_rewrite.c>
        RewriteEngine On

        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteRule ^.*/(.*)$ app.php [QSA,L,NC]
```



```
</IfModule>
```

```
#Deflate filter is optional.
```

```
#It reduces download size, but adds slightly more CPU processing:
AddOutputFilterByType DEFLATE text/html
```

```
</Directory>
```

```
</VirtualHost>
```

Next we verify that the following Apache modules required by ProcessMaker have been enabled: expires, filter, rewrite, deflate and vhost_alias. To do so, enter the command `httpd -M`, which lists all the enabled modules, and verify that those modules are included.



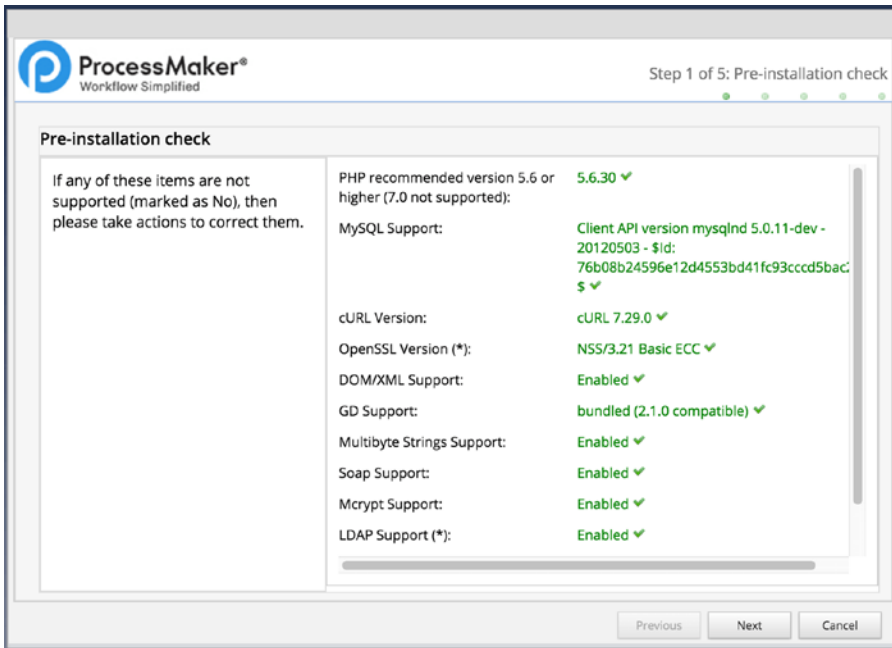
Enabling Apache Modules If any of the modules are not enabled, you can enable them by uncommenting them in the module config file.

Open the file with the command `vim /etc/httpd/conf.modules.d/00-base.conf` and remove the # in front of the module name.

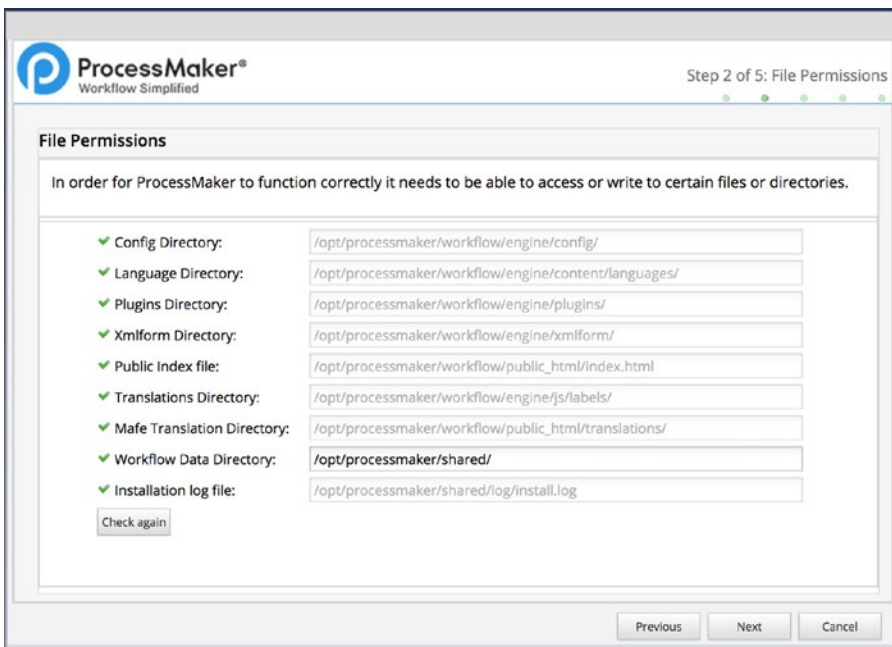
Finally restart the Apache web server with the command `systemctl restart httpd`.

Complete the Installation

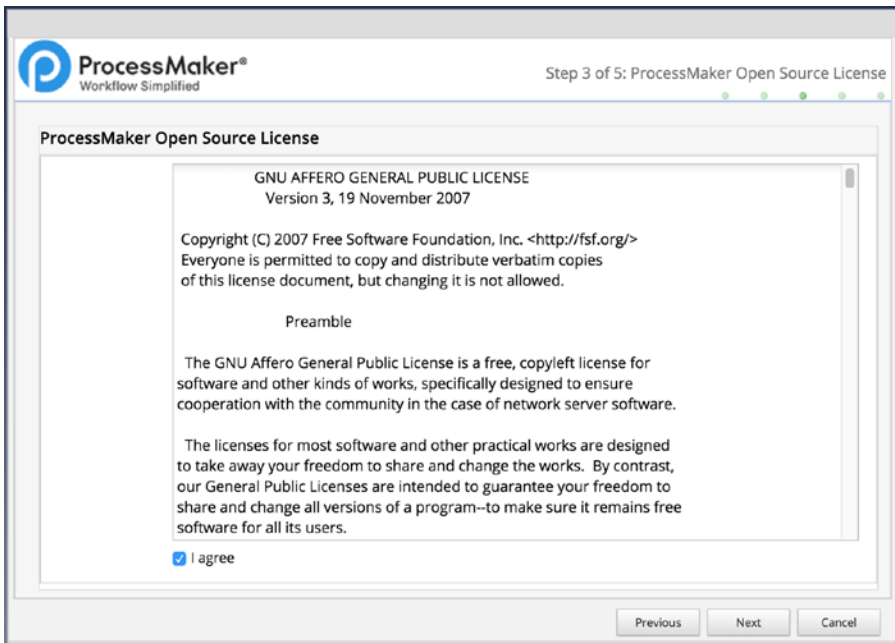
All is now set, and we can complete the installation by opening a browser and entering the IP address of the droplet as the URL; that is, `http://00.00.00.00`, replacing 00.00.00.00 with the IP address of your droplet. If all is well configured, you should be redirected to the ProcessMaker Pre-Installation Check window shown here.



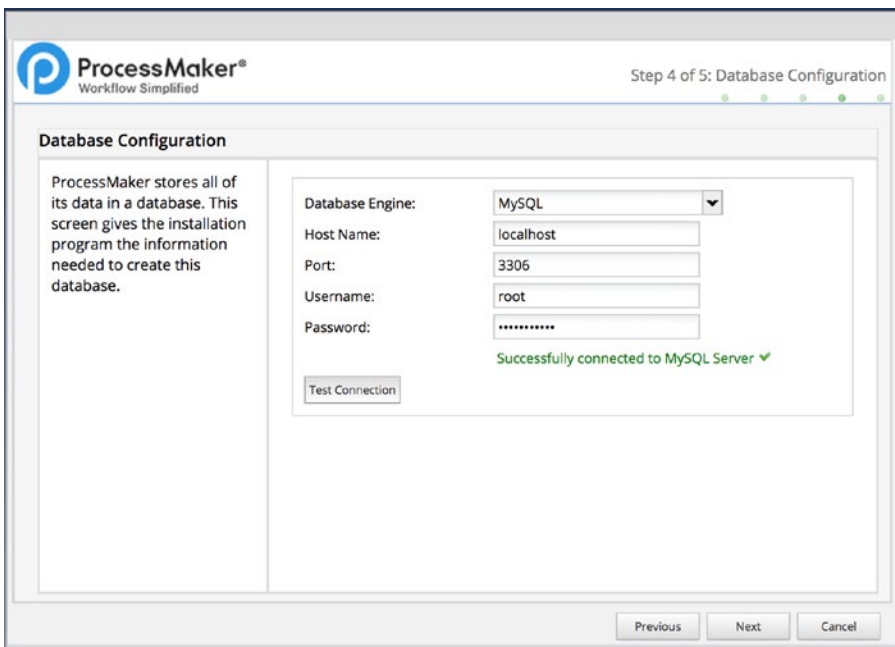
Ensure that all items in the checklist are checked and click the Next button. If the file permissions show an access denied error, try restarting the server.



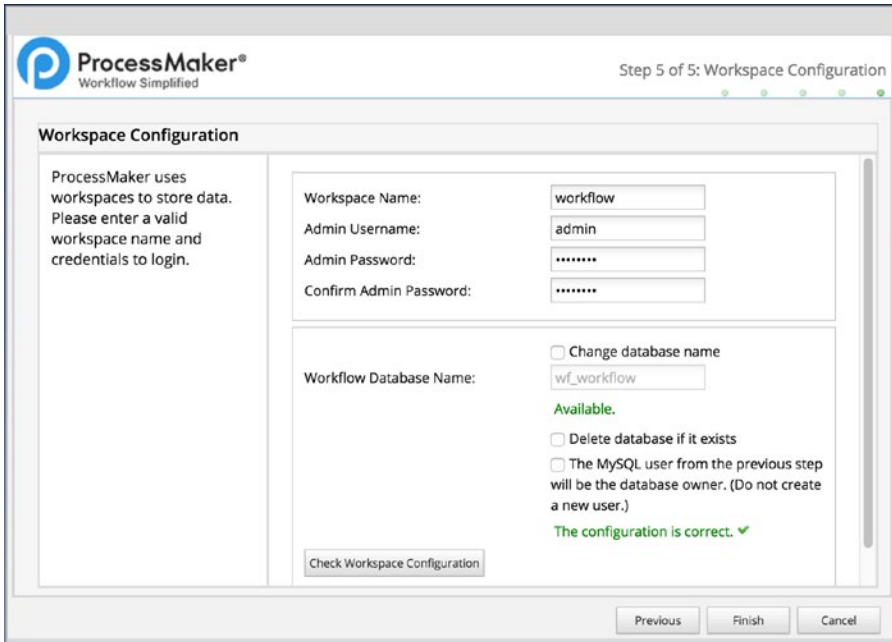
Ensure that ProcessMaker can read and write the required files and folders as shown and click the Next button.



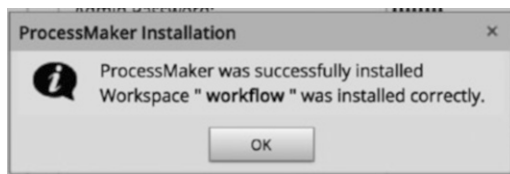
Accept the ProcessMaker Open Source License by checking the I Agree box and click the Next button.



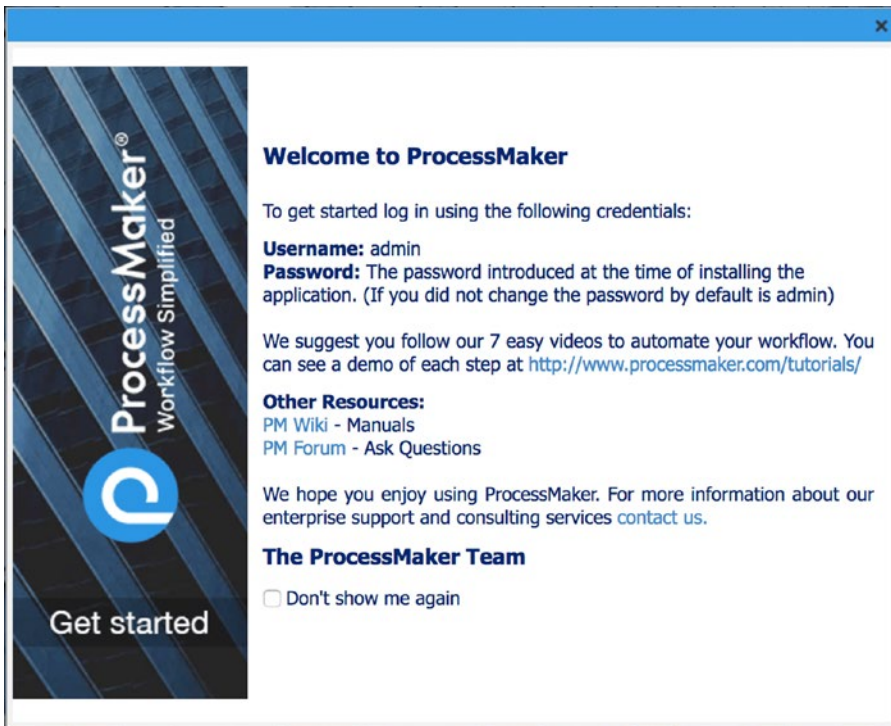
Enter the MySQL database root user password that was set during the Secure MySQL Installation step earlier and click the Test Connection button. When successfully connected, click Next.



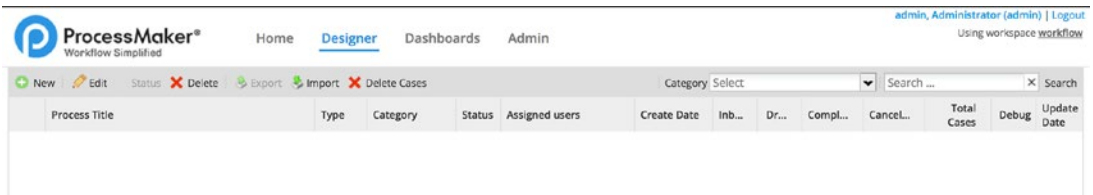
Enter a strong password that you will remember in the Admin Password field and confirm the password. Click the Check Workspace Configuration button. When the “configuration is correct” message is displayed, click the Finish button.



The ProcessMaker installation is completed and the success message is displayed. Click the OK button to display the Welcome screen as shown here.



Check the “Don’t show me again” checkbox and log in with the admin username and password. This displays the Designer menu.



Congratulations. You now have ProcessMaker set up and running in the cloud that can be accessed from anywhere with an Internet access. All that is left for us to do is configure the settings, add our users, and import the processes from our local instance to production.

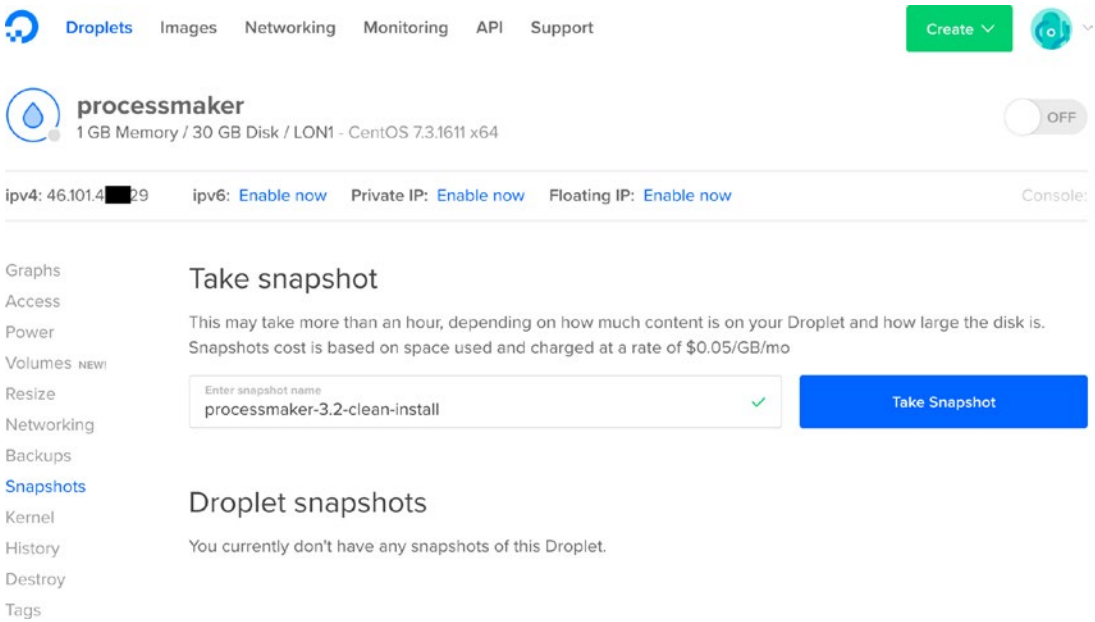
Take a Snapshot

You will agree that getting this all set up took considerable effort and it would be great to save a copy of this blank and clean install such that if you need to deploy another ProcessMaker server, you could just use that copy.

Luckily for us, DigitalOcean allows us to take a snapshot. “Snapshots copy an image of your entire VPS and store it on the DigitalOcean servers.” Snapshots can also help you save costs such that you can take a snapshot of your droplet and destroy the droplet when you are done using it.


When next you need to work with the server, you just create a new droplet from the snapshot and resume from where you left off. To take a snapshot of your droplet, you must shut it down first. In the terminal, enter the shutdown command.

Once the server is shut down, log in to DigitalOcean and click the droplet. Click Snapshots in the left sidebar navigation and the Snapshots page is displayed.



Enter a meaningful name for the snapshot and click the Take Snapshot button. When it's done, the snapshot is displayed as follows, showing the size and the data center where it is located.

Droplet snapshots

Name	Size	Regions	Created ▲
 processmaker-3.2-clean-install <small>Created from processmaker</small>	2.36 GB	LON1	2 minutes ago More ▼

With our snapshot done, we can power on the droplet by clicking Power in the sidebar navigation and clicking the Power On button.

Graphs

Access

Power

Volumes NEW!

Resize

Networking

Backups

Snapshots

Kernel

History

Destroy

Tags

Power On

Your Droplet is currently powered down. This action will boot your Droplet.

Do you want to proceed?

Power On



We have covered a lot in this chapter and learned how to install ProcessMaker in a cloud deployment, and in the next and final chapter we will deploy our processes from the local instance to our production instance after configuring it and securing communication with SSL.

CHAPTER 20

Deploying to Production

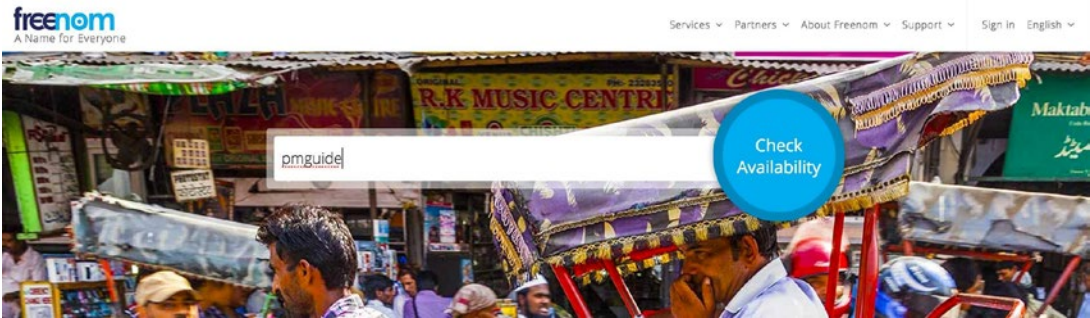
We now have ProcessMaker running in the cloud and we are ready to share our new process with others. However, you might have noticed that we access the server using the IP address, which might not always be easy to remember. We will want to give our server a domain name that can be easily remembered. Also, it is recommended to encrypt data between public web servers and the clients accessing them using SSL.

In the next sections, we will get a free domain name for our server from Freenom and a free SSL certificate from Let's Encrypt. Once we have that all set up, we will import our Cash Advance and Retirement Process from our desktop, deploying it to the production server, and we'll configure our mobile app to access it. Let's get started.

There are other methods of obtaining domain names and SSL certificates, but we will be using Freenom and Let's Encrypt for tutorial purposes only.

Get a Free Domain Name

To get our free domain name, head over to the Freenom web site (<http://www.freenom.com>), and on the landing page enter a name you would like to use. I am using pmguide as shown next. Click the Check Availability oval button to see if your preferred name is available.



If the domain name you have entered is available, you will see a list of available free TLDs (Top Level Domains) you can use. If the domain you have entered is not available or is a paid one, you can search for another name.

Get one of these domains. They are free!

pmguide .tk	• FREE	0. ⁰⁰	Get it now!
pmguide .ml	• FREE	0. ⁰⁰	Get it now!
pmguide .ga	• FREE	0. ⁰⁰	Get it now!

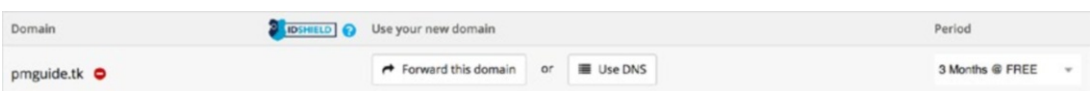
Click the Get It Now! button beside the domain name you would like to use to select it as shown here. Once you've selected, click the green Checkout button.

1 domain in cart [Checkout](#)

Get one of these domains. They are free!

pmguide .tk	• FREE	0. ⁰⁰	✓ Selected
----------------	--------	------------------	------------

On the order review page shown next, click the Use DNS button.



This displays an option to use either Freenom DNS or our own DNS. We will use our own DNS because we want to manage the DNS on DigitalOcean. Select the Use Your Own DNS tab and in the Nameserver fields, enter the following values as shown in the image:

- ns1.digitalocean.com
- ns2.digitalocean.com

You can choose to extend the period from 3 months if desired. Once you are done, click the Continue button.

Domain: pmguide.tk

Use your new domain

Period: 3 Months @ FREE

Forward this domain or Use DNS

Use Freenom DNS Service Use your own DNS

Enter your A record here

Nameserver	ns1.digitalocean.com	IP address	<input type="text"/>
Nameserver	ns2.digitalocean.com	IP address	<input type="text"/>

This displays the checkout page as shown here. Complete the checkout process by registering and verifying your email address.

Description	Price
Domain Registration - pmguide.tk	\$0.00USD
Subtotal:	\$0.00USD
Total Due Today:	\$0.00USD

Please enter your email address and click verify to continue to the next step

Enter Your Email Address

Verify My Email Address

Already Registered? Click here to login

Use social sign in



Once verification is done, you will be presented with an Order Confirmation page similar to the following.

Order Confirmation

Thank you for your order. You will receive a confirmation email shortly.

Your Order Number is: 8012049774

If you have any questions about your order, please open a support ticket from your client area and quote your order number.

[Click here to go to your Client Area](#)


We now have a free domain name ready to be used on DigitalOcean. You can log in to the Client Area and from the menu select Services ► My Domains to view the newly registered domain.

Set Up DNS

Our next step is to set up DNS hosting for our new domain on DigitalOcean. Log in to your DigitalOcean account and select Networking from the top menu. In the Domains tab, enter the name of your newly registered domain and click the Add Domain button.

Networking

[Domains](#) [Floating IPs](#) [Load Balancers](#) [Firewalls](#) [PTR records](#)



Looks like there are no domains here.

You can add one easily, though. Enter a domain below and start managing your DNS with DigitalOcean.

This displays the domain page, listing the DNS records for the domain. You will notice at the top of the page a “What’s Next” message, stating “*You need to update your nameservers with your domain registrar for the records below to take effect.*” We have already done this when we set the nameservers to DigitalOcean nameservers at the point of registering the domain. This saves us the hassle of waiting for the domain name to propagate if changed later.

Next, we need to add a DNS record pointing the domain name to our server. In the Create New Record section of the page, enter @ for the hostname and select your droplet as the server to redirect to, as shown next. Click the Create Record button.

Create new record

A AAAA CNAME MX TXT NS SRV

Use @ to create the record at the root of the domain or enter a hostname to create it elsewhere. A records are for IPv4 addresses only and tell a request where your domain should direct to.

HOSTNAME	WILL DIRECT TO	TTL (SECONDS)	
<input type="text" value="Enter # or hostname"/> @ ✓	<input type="text" value="processmaker"/> LON1 / 46.101.4...29 ✖	<input type="text" value="Enter TTL"/> 3600 ✓	<input type="button" value="Create Record"/>

pmguide.tk

We might also want users to be able to access the server using `www.{domainname}`, for example `www.pmguide.tk`. To allow that, we can also add a `www` record as shown next. Please note that this can be any name that suits your purpose.

HOSTNAME	WILL DIRECT TO	TTL (SECONDS)	
<input type="text" value="Enter # or hostname"/> www ✓	<input type="text" value="processmaker"/> LON1 / 46.101.4...29 ✖	<input type="text" value="Enter TTL"/> 3600 ✓	<input type="button" value="Create Record"/>

www.pmguide.tk

With our DNS set up, open a browser and enter your new domain name. This should open up your ProcessMaker instance. It looks like we are all set. Not quite. If you look at the address bar in your browser, you will notice it is marked as insecure.

🔒 Not Secure www.pmguide.tk/sys/en/neoclassic/login/login ☆

Chrome address bar



Mozilla address bar

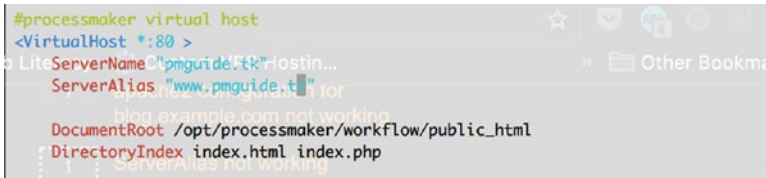
To make our users more comfortable and secure when using the application, we will get an SSL certificate to make the site secure.

One last thing we need to do is modify our web server configuration to use the new domain name. You will recall that in the previous chapter we set this to the IP address, since we did not have a domain name then. Open the `pmos.conf` file using the following command:

```
vim /etc/httpd/conf.d/pmos.conf
```

Once it is opened, find the line with `ServerName "your_ip_address"` and change it to `ServerName "your_domain_name"` as in the following image, using the domain name you registered. If you also included a `www` or other subdomain, add it as an alias using the `ServerAlias "your_subdomain_name"` directive.

Because we also now have a domain name, we can change the virtual host to match any IP address; this allows us to restore our droplet to a new server without having to change the IP address. Your configuration should be similar to the following screen when done, the only difference being the domain name.



Modifying ServerName to domain name

When you are done, save the file and restart the Apache web server using the command

```
systemctl restart httpd
```

Install SSL Certificate

The folks at DigitalOcean have put together a detailed guide called “How to Secure Apache with Let’s Encrypt on CentOS 7” (<https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-centos-7>), which we will adapt for our needs.

Create a non-Root Super User

First we begin by creating a non-root super user account as follows. SSH to the server and log in as root and create a new user using the following command, replacing *username* with the name of the user as in the next image.

```
adduser username
```

Next set a password for the user using the command

```
passwd username
```

Finally, grant the user super privileges by adding the user to the Wheel group using this command:

```
gpasswd -a username wheel
```

```
[root@processmaker ~]# adduser pmuser
[root@processmaker ~]# passwd pmuser
Changing password for user pmuser.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@processmaker ~]# gpasswd -a pmuser wheel
Adding user pmuser to group wheel
[root@processmaker ~]#
```

Install the Required Software

We will use Certbot to request a certificate from Let’s Encrypt. “Certbot is an easy-to-use automatic client that fetches and deploys SSL/TLS certificates for your webserver.”

In a new terminal or puTTY session, SSH to the server using the new username and password created.

Once you are logged in, enter the following command to install the Certbot executable:

```
sudo yum -y install python-certbot-apache
```

When prompted, enter the password for the newly created user and wait for the Certbot installation to complete.

Request a Certificate from Let's Encrypt

With Certbot in place, we will request a certificate for our domain name. In my case, I am using `pmguide.tk`. You should replace this with the name you registered for your server. Since I also added a `www` subdomain, I will include it in the request. If you used a different subdomain, modify it as appropriate in the command.

Still logged in as the new super user, enter the following command, remembering to use your correct domain name:

```
sudo certbot --apache -d pmguide.tk -d www.pmguide.tk
```

```
[pmuser@processmaker ~]$ sudo certbot --apache -d pmguide.tk -d www.pmguide.tk ]
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices) (Enter 'c' to
[cancel]): @gmail.com ]
Starting new HTTPS connection (1): acme-v01.api.letsencrypt.org

-----

Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.1.1-August-1-2016.pdf. You must agree
in order to register with the ACME server at
https://acme-v01.api.letsencrypt.org/directory

-----

(A)gree/(C)ancel: A ]

-----

Would you be willing to share your email address with the Electronic Frontier
Foundation, a founding partner of the Let's Encrypt project and the non-profit
organization that develops Certbot? We'd like to send you email about EFF and
our work to encrypt the web, protect its users and defend digital rights.

-----

(Y)es/(N)o: Y ]
Starting new HTTPS connection (1): supporters.eff.org
Obtaining a new certificate
Performing the following challenges:
tls-sni-01 challenge for pmguide.tk
tls-sni-01 challenge for www.pmguide.tk
Waiting for verification...
Cleaning up challenges
Created an SSL vhost at /etc/httpd/conf.d/pmos-le-ssl.conf
Deploying Certificate for pmguide.tk to VirtualHost /etc/httpd/conf.d/pmos-le-ss
l.conf
Deploying Certificate for www.pmguide.tk to VirtualHost /etc/httpd/conf.d/pmos-l
e-ssl.conf

Please choose whether HTTPS access is required or optional.

-----

1: Easy - Allow both HTTP and HTTPS access to these sites
2: Secure - Make all requests redirect to secure HTTPS access

-----

[Select the appropriate number [1-2] then [enter] (press 'c' to cancel): 2 ]
Added an HTTP->HTTPS rewrite in addition to other RewriteRules; you may wish to
check for overall consistency.
Redirecting vhost in /etc/httpd/conf.d/pmos.conf to ssl vhost in /etc/httpd/conf
.d/pmos-le-ssl.conf

-----

Congratulations! You have successfully enabled https://pmguide.tk and
https://www.pmguide.tk
```


This generates an interactive dialog as shown in the above image. Enter your email and agree to the terms. You can optionally choose to subscribe to the EFF mailing list.

Certbot generates a certificate for each specified domain and prompts whether to make HTTPS access compulsory. I am making it compulsory by choosing the second option, Secure.

Once the certificate is installed, restart the web server using the `sudo systemctl restart httpd` command. In a browser, navigate to your domain name with `https://` prefixed to the URL, and the site should load with secure encryption as shown here.



Chrome address bar



Mozilla address bar

However, if you're browsing to the site without HTTPS, it does not redirect as expected. To fix this, open the web server config file

```
sudo vim /etc/httpd/conf.d/pmos.conf
```

Enter the password if prompted. In the config, scroll to the bottom just before the closing `</VirtualHost>` tag. You will notice the rewrite rules added by the Certbot as shown here.

```
</Directory>  
RewriteEngine On  
RewriteCond %{SERVER_NAME} =pmguide.tk [OR]  
RewriteCond %{SERVER_NAME} =www.pmguide.tk  
RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]  
</VirtualHost>
```

Edit the config and add the `RewriteEngine On` line highlighted in the image to your config file. Save the file and restart the web server with the command

```
sudo systemctl restart httpd
```

Now try browsing to the server without HTTPS, and you should be automatically redirected.

Finally, let us set up a cron job (scheduled task) to renew the certificate automatically when it expires.

Enter the following command

```
sudo crontab -e
```

This opens the crontab editor. Edit the file by pasting the following on one line:

```
0 2 * * * /usr/bin/certbot renew >> /var/log/le-renew.log
```

Save your changes. This sets the cron job to execute the Certbot renew command at 2:30 every day.

```
[pmuser@processmaker ~]$ sudo certbot renew
Saving debug log to /var/log/letsencrypt/letsencrypt.log

-----
Processing /etc/letsencrypt/renewal/pmguide.tk.conf
-----
Cert not yet due for renewal

The following certs are not due for renewal yet:
/etc/letsencrypt/live/pmguide.tk/fullchain.pem (skipped)
No renewals were attempted.
[pmuser@processmaker ~]$ sudo crontab -e
no crontab for root - using an empty one
crontab: installing new crontab
[pmuser@processmaker ~]$
```

With our production environment all set and secured, we can now deploy our process to make it available to our users.



Additional Steps Before the production environment can be used by others, you will need to add them as users and configure the departments, roles, and authentication sources as required, as we did in the chapter on user administration.

The groups, however, will be imported along with processes that use them, but you will still have to add users to the groups.

You will also need to set up the email settings for notifications.



Exercise As an exercise, use the knowledge gained so far to set up the Production environment with users, groups, roles, authentication sources, departments, and email notifications matching the local instance.

Deploying the Process

To deploy our process, launch your local ProcessMaker Bitnami instance and log in as an administrator. Then take the following steps:

1. Export the Cash Advance and Expense Retirement Process.
2. Export the Schema and Data from the EMPLOYEE_TYPE PM Table.
3. Export the Schema for the EXPENSE_REPORT Report Table.
4. Next, log in to the cloud instance as the administrator.
5. Import the Cash Advance and Expense Retirement Process. When prompted, keep the imported Process UID.
6. Import the EMPLOYEE_TYPE and EXPENSE_REPORT tables.
7. Create the process category and edit the imported process to use the category.
8. Add the admin user to the Employees group.

Ensure that you have created the other users as required in the earlier exercise and assigned them to the appropriate groups.



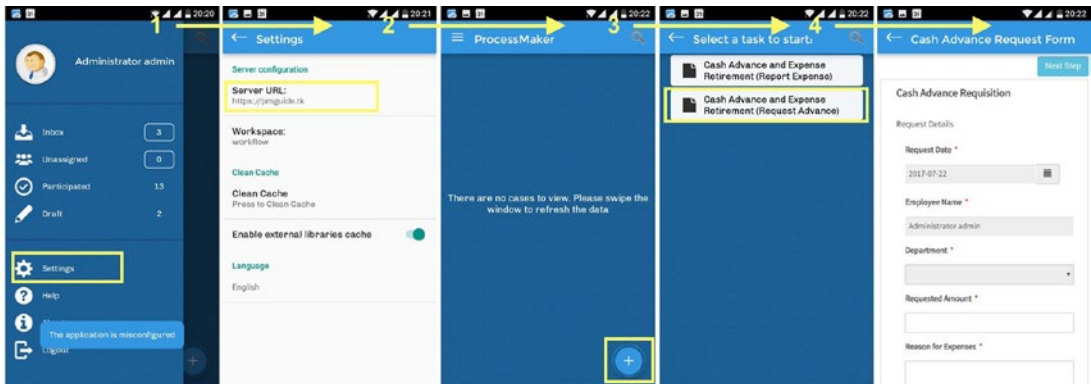
Alternative Deployment Method It is also possible to copy a local installation to another server and restore the database.

Configuring the Mobile App

With our process deployed, we can now give users a more permanent URL for the mobile app settings unlike the temporary `ngrok.io` tunnels we used in an earlier chapter. On your mobile device, launch the ProcessMaker app. You might get an error message stating that the app is misconfigured. This is because the app can no longer reach the `ngrok.io` URL.

Follow these steps to update the configuration:

1. Open the Settings tab from the menu.
2. Enter the URL with your new domain and HTTPS as shown in the next image in the Server URL. Save the settings.
3. Log in as the admin user and try to create a case.
4. Select the Cash Advance and Expense Retirement (Request Advance) task.
5. The dynaform should load as displayed next.



That brings us to the end of this guide. It has been a learning experience for me, and I do hope your curiosity flame has been ignited and you will go on to learn more about ProcessMaker and how you can use it to automate business processes.

What Next?

Now that you have the basics (and a little more) covered, what comes next? Try thinking of some processes you can automate on your own. You can ask colleagues or your boss for ideas about what processes to try your hands on. If you get stuck, here are some resources you can consult:

- The ProcessMaker forum: <https://forum.processmaker.com/>.
- The ProcessMaker wiki: <http://wiki.processmaker.com/>.
- The ProcessMaker bug tracker: <http://bugs.processmaker.com/>. If you run into a bug, you can submit a bug report here. It can also be a good place to check if others have had similar issues and if there is a known workaround or resolution.
- The ProcessMaker webinars page: <https://www.processmaker.com/webinar>. I also recommend visiting here to learn more about other features and register for upcoming webinars showcasing new features in ProcessMaker.

You are welcome to also visit the companion blog for the book, Learning BPM (<https://www.learningbpm.com>), for articles showing tips and tricks on ProcessMaker.

Index

A

Accounting system, [81](#), [87](#)
Ad-hoc users, [230](#)
Admin
 calendar, [387](#), [389](#)
 Cases List, [395](#)
 clear cache, [396](#)
 Dashboards, [402](#), [404](#)
 email servers, [387](#)
 environment, [395](#)
 language, [391](#)–[393](#)
 login, [401](#)–[402](#)
 logo, [386](#)–[387](#)
 logs, [406](#)
 plugins, [404](#)–[406](#)
 PM table, [398](#)–[401](#)
 Process Categories, [389](#)–[390](#)
 report table, [396](#)–[398](#)
 skins, [394](#)
 system, [404](#)
Apache, [431](#)
Apache, MySQL, and PHP (AMP), [8](#)
Apache web server, [441](#)–[443](#)
Array variable, [98](#)

B

Bitnami Application Manager, [32](#)
 Manage Servers screen, [33](#)–[34](#)
 Welcome screen, [32](#)–[33](#)

Bitnami installations, [8](#)
Bitnami ProcessMaker Installer page, [11](#)
 for Mac OS X users, [11](#), [19](#)
 Configure SMTP Settings, [15](#)–[16](#)
 Create Admin Account screen, [14](#)
 Deploy ProcessMaker, [16](#)
 Installation Folder, [13](#)
 Installation Information, [18](#)
 PhpMyAdmin, [13](#)
 Ready to Install, [17](#)
 setup wizard, [12](#)
 Web Server Port, [14](#)
 for Windows users, [20](#)
 Configure SMTP Settings, [25](#)–[26](#)
 Create Admin Account, [22](#)
 Deploy ProcessMaker, [27](#)
 firewall warning, [28](#)
 Gmail configuration, [26](#)
 Installation Folder, [22](#)
 Installation Information, [29](#)
 PhpMyAdmin, [21](#)
 Ready to Install, [27](#)
 setup wizard, [21](#)
 User Account Control dialog, [20](#), [30](#)
 Web Server Port, [23](#)–[24](#)
 Welcome to ProcessMaker
 modal, [31](#)
Bitnami Stack, [8](#)
Boolean variable, [98](#)
Bootstrap's grid system, [110](#)

INDEX

BPMN 2.0, [39](#) *see also* Shapes
Toolbox, BPMN

Business process, [5](#)

Business process management
(BPM), [5-7](#)

Business Process Model and Notation
(BPMN) *see* BPMN 2.0

C

Camel case, [97](#)

Case, [36](#)

Case assignment method, [230](#)

Case labels, [333](#)

Case permissions and notes

creation, [350](#)

group/user, [352](#)

origin task, [352](#)

participation required, [352](#)

permission, [352](#)

status Case, [351](#)

target task, [351](#)

type, [352](#)

Cash Advance and Expense Retirement

process, [51](#), [80](#), [83](#), [231](#), [232](#)

approve/reject report, [308-309](#)

assignment rules, [312-313](#)

Case Notes, [358-360](#)

Comments, [323](#)

computeReimburseRefund, [349-350](#)

display mode property, [347](#)

Edit Variable, [344](#)

EscalateFinanceUnclaimedCase,
[354-356](#)

feedback, [331-332](#)

HTML Editor, [315-318](#)

imported form, [303-307](#)

JavaScript, [348-349](#)

mode property, [308](#)

NotifyCashAdvanceApproval, [357](#)

NotifyCashAdvanceRejection, [357](#)

properties, [314](#)

receipts, [325-326](#)

receipt uploads, [313](#)

Reimbursement/Refund Details, [310](#)

Request Advance, [356-357](#)

routing rule, [312](#)

SetCashAdvanceInitialData, [345-346](#)

SetCurrentUser, [322](#)

SetExpenseReportInitialData, [346](#)

Signoff/Approval, [323-324](#)

task and select, [311](#)

total amount, [324](#)

transaction, [326](#), [329](#)

UpdateCashAdvanceApprover, [319](#), [321](#)

UpdateCashAdvanceDisburser, [319](#)

UpdateCashAdvanceRequestor, [319](#)

UpdateExpenseReportApprover, [320](#)

UpdateExpenseReportPreparer, [320](#)

UpdateExpenseReportProcessor, [320](#)

Cash Advance Requisition

add, comments, [176-179](#)

approval functionality

approval with code, [184-189](#)

approval without code, [184](#)

Assigned Elements, [192-197](#)

assignment and routing, [191-192](#)

Create/Select Variable, [169-171](#)

export, [183-184](#)

JavaScript, [181-182](#)

Request Details, [168](#)

Signoff/Approval, [174-175](#)

tablet/mobile, [171-173](#)

Colosa Inc., [7](#)

Conditions, [37](#)

Custom triggers, [262](#), [267-269](#)

Cyclical assignment method, [232-235](#)

D

Datetime Control, [347](#)

Datetime variable, [98](#)

Default flow, [383](#)

DigitalOcean

account register, [419](#)

account verification, [421](#)

definition, [419](#)

email confirmation, [420](#)

PayPal option, [422](#)

DNS

Check Availability, [451](#)

Checkout, [452](#)

mobile app, [463](#)

networking, [454](#)

ProcessMaker Bitnami, [462](#)

record, [455](#)

registering and verifying, [453](#)

SSL certificate (*see* SSL)

Droplet

additional options, [424](#)

data center, [423](#)

hostname, [425](#)

ProcessMaker

Mac/Linux system, [426–427](#)

Windows, [427–428, 430](#)

size selection, [423](#)

snapshots, [449](#)

SSH Keys, [424](#)

Dynaforms, [37](#)

Dynaform Designer, [103](#)

container, [107](#)

form control properties, [108–109](#)

row control properties, [109–110](#)

control and properties panel, [106](#)

history to use, [107](#)

properties, [107](#)

web controls, [106–107](#)

menu, [104](#)

close, [106](#)

export, [104](#)

import, [105](#)

Language option, [105](#)

preview, [105](#)

save, [104](#)

to clear, [105](#)

title, [104](#)

Dynaforms (dynamic forms), [95, 101–102](#)

adding input documents, [280–281](#)

Dynaform web controls

Cash Advance and Expense Retirement
(*see* Cash Advance Requisition)

checkbox, [125–126](#)

checkgroup, [127–128](#)

datetime

Clear button, [134](#)

datepicker view, [133](#)

default date, [133](#)

format, [131](#)

max date, [132](#)

min date, [131–132](#)

selection date, [132–133](#)

dividing, row, [137–139](#)

dropdown, [122–124](#)

file, [143–146](#)

grid (*see* Grid layout)

hidden, [139–140](#)

image, [142–143](#)

label, [141](#)

link, [141](#)

multiple file uploader, [146–147](#)

panel, [159–163](#)

radio, [128–129](#)

reposition rows, [135–136](#)

subform, [163, 165](#)

INDEX

Dynaform web controls (*cont.*)

submit and button, [147-149](#)

suggestions, [136-137](#)

textarea, [121-122](#)

textbox

display modes, [119-120](#)

Dynaform Designer, [119](#)

properties, [114-115](#), [117-118](#)

title and subtitle, [140](#)

variables, [112-113](#)

E

Email notifications

creating, template, [336-337](#)

NotifyCashAdvanceApproval, [342](#)

NotifyCashAdvanceRejection, [342](#)

PMFSendMessage, [338-342](#)

SMTP, [342](#)

task notification property, [334-335](#)

EscalateFinanceUnclaimedCase, [354-356](#)

Exclusive (XOR) gateway, [365](#)

F

Feedback

employees, [332](#)

finance officers, [331](#)

supervisors, [332](#)

File variable, [98](#)

Float variable, [98](#)

G, H

generateCode function, [268](#)

Graphical user interface (GUI), [13](#)

Grid layout

adding and deleting rows, [153](#)

controls, [151-153](#)

mathematical functions, [157](#), [159](#)

modifying, [155-157](#)

page size property, [154](#)

validation error, [157](#)

Grid variable, [98](#)

Group artifact, [82](#)

I, J, K

Inclusive (OR) gateway

conditions, [380-381](#)

Edit Process, [379](#)

HR Officer, [379-380](#)

import, [378-379](#)

testing, [381-382](#)

Input documents, [277](#)

creating, [278-280](#)

Document Management System,
viewing in, [286-287](#)

to dynaforms, [280-281](#)

as a step, [281-285](#)

Integer variable, [98](#)

L

Lightweight Directory Access

Protocol (LDAP), [220](#)

M, N

MacOS (OS X) users, [11](#), [19](#)

Configure SMTP Settings, [15-16](#)

Create Admin Account screen, [14](#)

Deploy ProcessMaker, [16](#)

Installation Folder, [13](#)

Installation Information, [18](#)

PhpMyAdmin, [13](#)

Ready to Install, [17](#)

- setup wizard, 12
 - Web Server Port, 14
 - Manual assignment
 - method, 235–236
 - MariaDB, 431
 - Master process, 55
 - Mobile apps
 - configuration, 414–415
 - deployment, 416–417
 - installation, 407–410
 - ngrok
 - download, 411
 - Mac OS X, 411–412
 - Windows, 413–414
 - Modeling a process, 73, 86–91
 - adding tasks
 - Approve Advance task, 75
 - Disburse Advance, 76
 - dragging and dropping, 75
 - Request Advance, 75
 - connecting tasks, 76–78
 - creating new process, 74–75
 - tasks, 73
 - Multiple file variable, 98
 - MySQL 5.5.X, 433–436
- O**
- Object Management Group (OMG), 39
 - OS X, 8
 - Output documents, 287
 - creating, 287, 297–298, 300
 - description, 288
 - destination path, 289
 - enable versioning, 289
 - filename generated, 288
 - to generate, 289
 - generated file link, 290
 - grids, 294
 - HTML editor, 293
 - margin, 289
 - media, 289
 - orientation, 289
 - PDF security, 289
 - prefix, 296
 - report generator, 288
 - Submit Form button, 295
 - tags, 290
 - title, 288
 - Upload File button, 291–292
 - values, 290
- P, Q**
- Parallel (AND) gateway
 - Administration, 371
 - assign forms, 373
 - assign users/groups, 373–374
 - Employee Onboarding, 369–370
 - Employee Supervisor, 371–372
 - Human Resources, 371
 - Information Technology, 371
 - Process Map, 365–366
 - testing, 375–378
 - UpdateAdminOfficer, 372
 - UpdateHROfficer, 372
 - UpdateITOfficer, 372
 - UpdateSupervisor, 372
 - variables, 367–368
 - Pascal case, 97
 - PHP 5.6, 432–433
 - PMFSendMessage
 - CashAdvanceNotifyFinance, 338
 - CashAdvanceNotifyRejection, 338
 - NotifyCashAdvanceApproval, 340–342
 - NotifyCashAdvanceRejection, 339–340

INDEX

Predefined triggers, 262–266

Process, 36

Process Designer, 47

Process Map, 50

Process Map area, 50–51

Shapes Toolbox, 51

End events, 70–72

Events elements, 60

Exclusive (XOR) gateway, 58

Gateway element, 57

Inclusive (OR) gateway, 59

Intermediate Catching

events, 68–70

Intermediate Throwing

events, 66–68

Parallel (AND) gateway, 58–59

Start events, 60–63, 65

Sub-process element, 55–57

Task element, 51–55

Top Toolbar, 47

Close option, 50

Export Diagram option, 48

Export option, 48

Full Screen option, 48

Help option, 50

Save and Save As option, 49

Undo and Redo options, 48

Zoom option, 48

Process List

actions

Category filter and Search

box, 44

debug, 44

Delete and Delete Cases

option, 43

Edit option, 42

Export option, 42

Import option, 43

New option, 40

Status option, 42

columns

canceled cases, 46

category, 46

completed cases, 46

create date, 46

debug status, 46

draft, 46

inbox, 46

status, 46

title, 45

total cases, 46

type of process, 45

update date, 46

user owner, 46

ProcessMaker, 7–9

Apache, 431

Apache web

server, 441–443

authentication sources

importing users, 222–224

LDAP, 220

setting up, 220–222

concepts, 36–38

deletion, 215

departments

add, 211–212

assigning users, 212–213

Set Manager and No Set

Manager, 213–214

firewall enable, 437–438

groups

assigning groups, 209–210

assigning users, 208–209

creation, 206–207

deletion, 207

editing, 207

- installer, [438–440](#)
- interface, [35–36](#)
- list (*see* Process List)
- MariaDB, [431](#)
- MySQL database, [446](#)
- MySQL 5.5.X, [433–436](#)
- mysql_secure_installation, [436](#)
- objects, [44](#)
- password confirmation, [446–447](#)
- PHP 5.6, [432–433](#)
- Pre-Installation Check, [443, 445](#)
- roles
 - assigning users, [218, 220](#)
 - default, [215–216](#)
 - viewing and editing, [216, 218](#)
- SELINUX disable, [437](#)
- user experience, [225–227](#)
- users
 - delete, [206](#)
 - disable, [204–205](#)
 - editing, [204](#)
 - new user, [201, 203](#)
 - summary, group and authentication, [206](#)
- Process Map, [81, 89](#)
- Process model *see* Modeling a process

R

- Request Advance task, [75](#)

S

- Self-service assignment
 - method, [247–251](#)
- SetCurrentUser, [321](#)
- Shapes Toolbox, [79](#)
 - artifacts, [82](#)

- Group, [82](#)
 - Text Annotation, [82](#)
- data elements, [79](#)
 - Data Object element, [80](#)
 - Data Store element, [81](#)
- Lasso, [83–86](#)
- pools and lanes, [81–82](#)
- Shapes Toolbox, BPMN, [51](#)
 - End events, [70](#)
 - email message, [71](#)
 - empty, [71](#)
 - error, [72](#)
 - message, [72](#)
 - signal, [72](#)
 - terminate, [72](#)
 - Events elements, [60](#)
 - Exclusive (XOR) gateway, [58](#)
 - Gateway element, [57](#)
 - Inclusive (OR) gateway, [59](#)
 - Intermediate Catching events, [68](#)
 - conditional, [70](#)
 - receive a message, [69](#)
 - signal, [70](#)
 - timer, [69](#)
 - Intermediate Throwing events, [66](#)
 - email message, [66](#)
 - send message, [67](#)
 - signal, [68](#)
 - Parallel (AND) gateway, [58–59](#)
 - Start events, [60–61, 63](#)
 - Assigned Users list, [62](#)
 - Assignment Rules, [61](#)
 - Cash Advance and Expense Retirement Process, [63](#)
 - delete, [65](#)
 - edit label, [65](#)
 - properties, [65](#)
 - types, [63](#)

INDEX

Shapes Toolbox, BPMN (*cont.*)

- Web Entry feature, 65

Sub-process element

- asynchronous process, 56

- Cancel button, 57

- context menu, 56

- IT System Request process, 56

- master process, 55

- synchronous process, 56

Task element

- assignment rules, 55

- Cash Advance process, 51

- delete, 55

- edit label, 55

- marker type, 54

- properties, 55

- quick toolbar, 52

- steps, 55

- type of task, 53

Snake case, 97

Snapshots, 448–449

SSH key, 425

SSL

- Certbot, 458, 460–461

- non-root super user, 457

- puTTY, 457

String variable, 97

T

Task, 37

Text Annotation artifact, 82

Triggers, 37, 257

- case variables

 - prefixes, 260

 - system variables, 260–261

 - variable selector, 262

- creating, 262

 - copying triggers, 269

 - custom triggers, 267–269

 - predefined triggers, 263–266

- debugging, 272

 - enabling and disabling Debug

 - mode, 273–274

 - identifying errors, 276

 - ProcessMaker Debugger

 - window, 274–276

- testing, 270–272

- timing, 258

 - before assignment, 258

 - after a step, 258

 - after routing, 259

 - before routing, 259

 - before a step, 258

U

Unique Identifier (UID), 44

Users, tasks

- Ad-hoc users, 230

- case assignment

 - method, 230

- cyclical assignment, 232–235

- cyclical, manual and value-based

 - assignment, comparing, 237–241

- and groups, 230–231

- manual assignment, 235–236

- reports, 245–247

- self-service assignment, 247–251

- self-service value-based assignment,

 - 251–255

- value-based assignment, 241–245

V

Value-based assignment, [241–245](#)

Variables

accepted variable values, [99](#)

creating, [100–101](#)

database connection, [99](#)

name, [96–97](#)

plus (+) icon, [96](#)

SQL, [99](#)

to store data, [94–95](#)

System, [94](#)

type, [97](#)

array, [98](#)

Boolean, [98](#)

Datetime, [98](#)

file, [98](#)

float, [98](#)

grid, [98](#)

integer, [98](#)

multiple file, [98](#)

string, [97](#)

W, X, Y, Z

Web controls, [106–107](#)

Windows users, [20](#)

Configure SMTP Settings, [25–26](#)

Create Admin Account, [22](#)

Deploy ProcessMaker, [27](#)

firewall warning, [28](#)

Gmail configuration, [26](#)

Installation Folder, [22](#)

Installation Information, [29](#)

PhpMyAdmin, [21](#)

Ready to Install, [27](#)

setup wizard, [21](#)

User Account Control dialog, [20, 30](#)

Web Server Port, [23–24](#)

Welcome to ProcessMaker modal, [31](#)

Workflow, [1](#)

sample cash advance requisition

form, [4](#)

sample expense retirement form, [3](#)

steps, [2](#)