



**Adobe**

# Overview

July 2006

**Adobe® LiveCycle™ Workflow**

Version 7.2

© 2006 Adobe Systems Incorporated. All rights reserved.

Adobe® LiveCycle™ Workflow 7.2 Overview for Microsoft® Windows®, Linux®, and UNIX®  
Edition 2.1, July 2006

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names and company logos in sample material or in the sample forms included in this software are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

All other trademarks are the property of their respective owners.

Powered by Celequest. Contains technology distributed under license from Celequest Corporation. Copyright 2005 Celequest Corporation. All rights reserved.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

This product includes software developed by the Jaxen Project (<http://www.jaxen.org/>).

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# Contents

---

<b>Preface</b> .....	<b>4</b>
What's in this guide?.....	4
Who should read this guide?.....	4
Related documentation .....	4
<b>1 About LiveCycle Workflow</b> .....	<b>5</b>
Automated processes .....	5
Process life cycle .....	7
Creating the electronic definition .....	7
Process initiation and execution.....	8
Process administration.....	8
Business activity monitoring.....	9
<b>2 LiveCycle Workflow Integration</b> .....	<b>10</b>
LiveCycle Workflow environment .....	10
Interaction with other software.....	11
Custom software integration .....	11
Server software integration.....	11
Integration with other LiveCycle products .....	12
<b>Glossary</b> .....	<b>13</b>

# Preface

---

Adobe® LiveCycle™ Workflow provides tools that you can use to automate business processes, including tools for creating electronic process definitions, server software that manages the life cycle of deployed processes during their execution, web-based administration tools, and web applications for creating and consuming dashboards that present real-time process metrics.

## What's in this guide?

This guide provides an overview of several key concepts of process automation, describes the components that LiveCycle Workflow includes, and describes how LiveCycle Workflow integrates into your business environment.

## Who should read this guide?

You should read this guide before using any of the product tools or reading the related documentation.

## Related documentation

You can refer to other product documentation to learn more about LiveCycle Workflow.

<b>For information about</b>	<b>See</b>
How to install, configure, and deploy LiveCycle Workflow.	<i>Installing and Configuring LiveCycle for JBoss</i> <i>Installing and Configuring LiveCycle for WebSphere</i> <i>Installing and Configuring LiveCycle for WebLogic</i>
How to use LiveCycle Workflow Designer.	<i>Creating Workflows</i> or <i>LiveCycle Workflow Designer Help</i>
How to track processes at run time.	<i>LiveCycle Workflow Designer Help</i>
How to set up reports based on real-time process data.	<i>Using LiveCycle Workflow Workbench</i>
How to use process reports and perform ad hoc queries on real-time process data.	<i>Using Business Activity Monitor Dashboard</i>
Last-minute changes that occur to the product.	<i>LiveCycle Workflow Readme</i>
Other LiveCycle products.	<a href="http://partners.adobe.com/public/developer/main.html">http://partners.adobe.com/public/developer/main.html</a>
Patch updates, technical notes, and additional information on this product version.	<a href="http://www.adobe.com/support/products/enterprise/index.html">www.adobe.com/support/products/enterprise/index.html</a>

# 1

## About LiveCycle Workflow

---

LiveCycle Workflow allows you to design, deploy, and manage automated business-process applications that connect systems and people. LiveCycle Workflow includes several tools for implementing your business automation plan.

**LiveCycle Workflow Designer:** A desktop application for creating the electronic definitions of business processes and deploying them to the server.

**LiveCycle Workflow Server:** A server application that executes the electronic definitions of business processes and coordinates process activity.

**LiveCycle administration:** A web-based application for configuring user authentication and authorization for all LiveCycle products, configuring run-time properties of LiveCycle Workflow Server, and managing the progress of processes as they are executed.

**LiveCycle Workflow Business Activity Monitor:** A set of web-based applications for creating and viewing dashboards that are based on real-time data from automated business processes.

### Automated processes

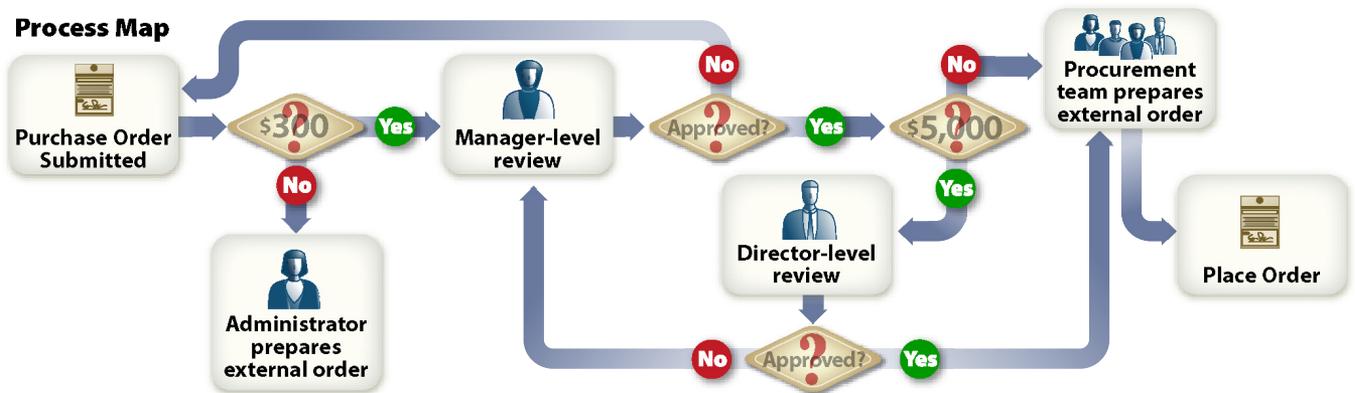
Automated processes are the electronic implementations of existing business processes. When a process is automated, LiveCycle Workflow Server is the central engine that manages the lifecycle of deployed process during their execution. The server manages the execution, state, and transaction behavior of each step that is defined in the automated process.

For example, an existing internal purchase order process uses a form that collects information about the requested purchase. The form is submitted to the appropriate person for approval and eventual ordering.

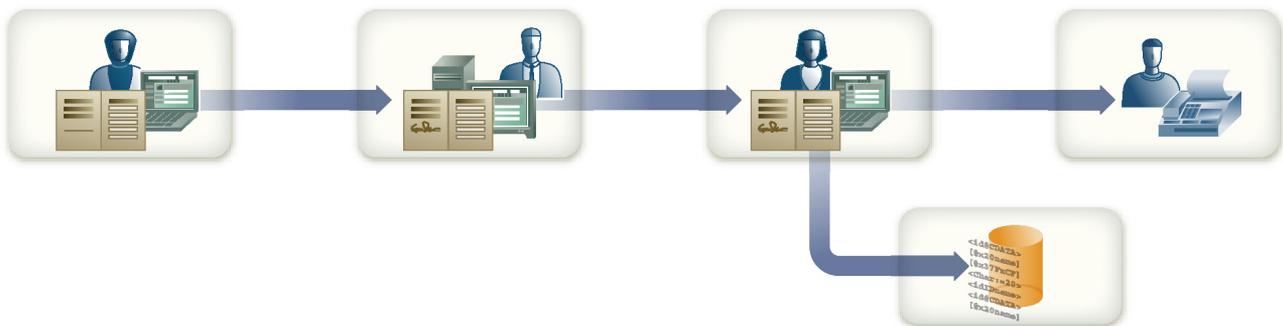
When the process is automated, a person fills and submits the initial form online.

LiveCycle Workflow Server receives the form data and uses it to determine who should next be involved in the process, and sends them data which is displayed in the appropriate form.

For each step of the process, the server uses form data and organizational data to make routing decisions. Information about the purchase is eventually stored for use by other back-end systems for accounting purposes. The server also initiates the placement of the order.



### Process Instance



### Business logic

Business logic provides the ability to make decisions during a process. Decisions involve determining how to proceed in a process, which people or systems are involved in the process, and when the next step should be executed.

LiveCycle Workflow applies rules to information to make decisions. Rules are created at design time and are applied to real-time data during process execution. Data used in rules can be gathered during the execution of the process through forms or information from corporate databases.

### Forms

Forms collect data and enable people to participate in automated processes. Forms can be submitted to LiveCycle Workflow Server to initiate processes, and LiveCycle Workflow Server can send forms to people to fill and submit to complete a step in a process.

Forms also let people work offline. When people receive a form, they can save it and fill it offline. When they are ready, they can email the form to the server to complete the step in the process.

## Process life cycle

The task of automating a business process typically involves a team of people who plan, design, implement, and maintain process automation and the process management system.

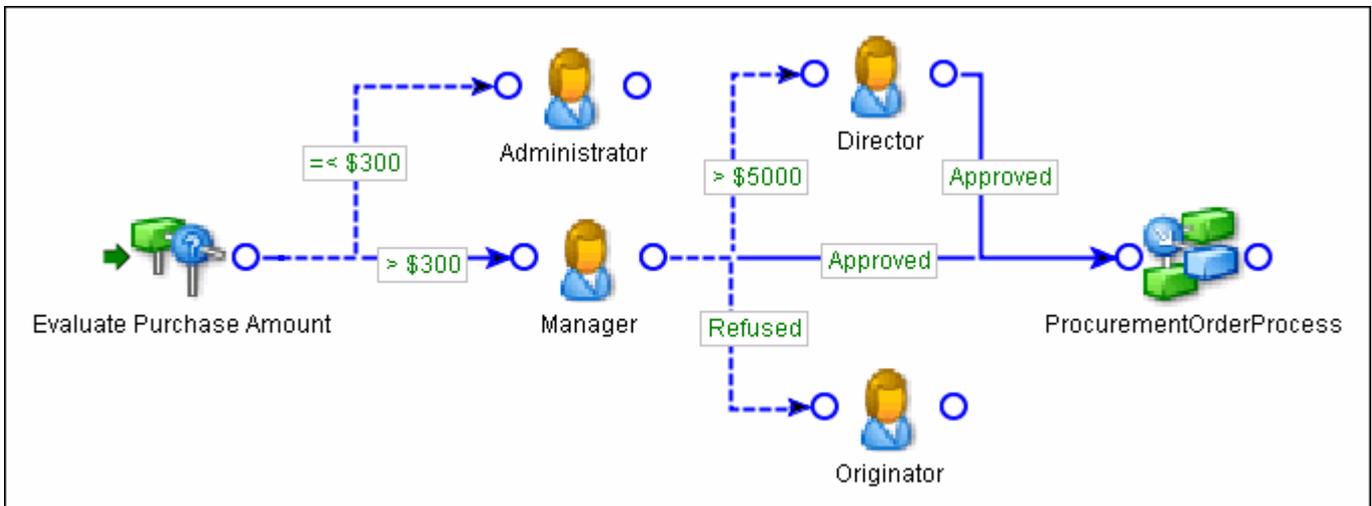
To automate business processes, requirements gathering first needs to be performed. Existing processes need to be mapped. The steps in the processes and how decisions are made during the process need to be well-understood, as well as which people or corporate business systems are involved in the process, and what information needs to be gathered and processed.

After the requirements are understood, the tools that LiveCycle Workflow provides can be used to create the electronic definition of the process and then deploy, manage, and monitor the automated process.

### Creating the electronic definition

You create the electronic definitions of processes, called workflows, with LiveCycle Workflow Designer.

LiveCycle Workflow Designer provides tools for creating a visual representation of the steps of the processes, adding the business logic required for making decisions automatically, and associating forms with the steps of the processes that involve people.



LiveCycle Workflow Designer includes several components that you use to create the visual representation. Each component represents a step in the process. After you add a component, you configure the properties as required for the current workflow.

Each type of component has a different use. For example, the User component assigns tasks to people and specifies forms to send to them. The Stall component purposely pauses the execution of a process when problem situations occur. Other components execute script, execute other workflows, and reference and update process data that is saved in variables.

You also use LiveCycle Workflow Designer to deploy workflows to LiveCycle Workflow Server. To make changes to deployed workflows, you can either modify the deployed workflow directly or create a new version and replace the existing workflow.

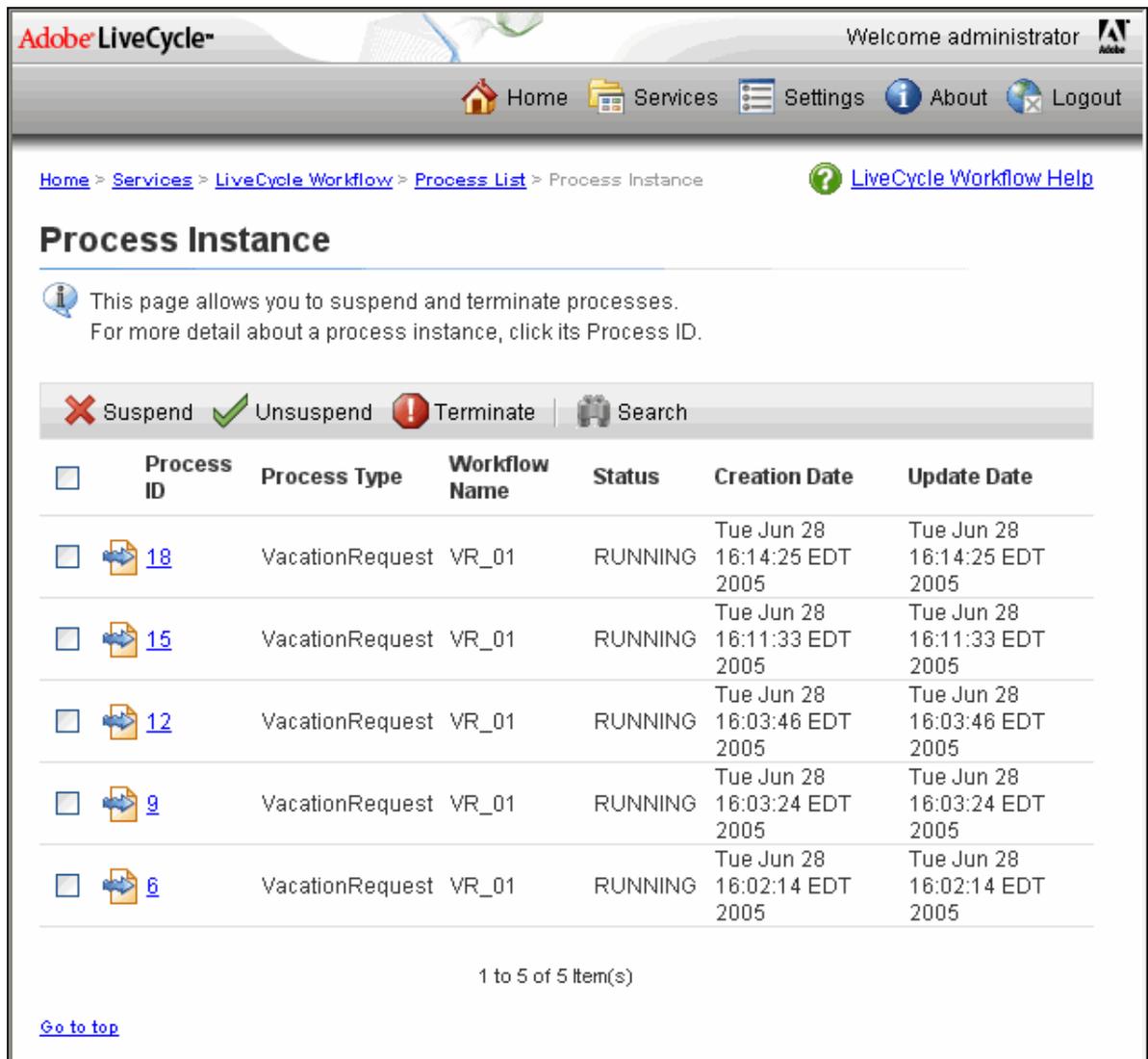
## Process initiation and execution

When workflows are deployed to LiveCycle Workflow Server, client software can initiate the process that the workflow represents. After the process is initiated, LiveCycle Workflow Server creates an instance of the process and executes the associated workflow. LiveCycle Workflow Server assigns a unique identification to each process instance so that they can be tracked.

There are multiple ways to initiate processes, such as through web services, email, Java APIs, and messaging queues, as well by using online forms. For each step in a process instance, either a task is assigned to a user or a service request is sent to a system, depending on the step. The process is completed when the final step in the process is executed.

## Process administration

The LiveCycle Workflow administration tool is a web-based application for managing processes and process instances. The administration web pages enable you to see information about process instances, such as which step of the process is currently being executed, who is assigned tasks, and whether the process instance is progressing as expected.



The screenshot shows the Adobe LiveCycle administration web interface. At the top, there is a navigation bar with links for Home, Services, Settings, About, and Logout. Below the navigation bar, a breadcrumb trail indicates the current location: Home > Services > LiveCycle Workflow > Process List > Process Instance. A "LiveCycle Workflow Help" link is also visible. The main heading is "Process Instance". Below the heading, an information icon and text state: "This page allows you to suspend and terminate processes. For more detail about a process instance, click its Process ID." Below this text is a toolbar with buttons for Suspend (red X), Unsuspend (green checkmark), Terminate (red exclamation mark), and Search (magnifying glass). The main content area contains a table with the following columns: Process ID, Process Type, Workflow Name, Status, Creation Date, and Update Date. The table lists five process instances, all of which are "VacationRequest" type and "VR\_01" workflow name, and are currently in a "RUNNING" status. The creation and update dates for all instances are from Tuesday, June 28, 2005. At the bottom of the table, it says "1 to 5 of 5 item(s)". A "Go to top" link is located at the bottom left of the page.

<input type="checkbox"/>	Process ID	Process Type	Workflow Name	Status	Creation Date	Update Date
<input type="checkbox"/>	<a href="#">18</a>	VacationRequest	VR_01	RUNNING	Tue Jun 28 16:14:25 EDT 2005	Tue Jun 28 16:14:25 EDT 2005
<input type="checkbox"/>	<a href="#">15</a>	VacationRequest	VR_01	RUNNING	Tue Jun 28 16:11:33 EDT 2005	Tue Jun 28 16:11:33 EDT 2005
<input type="checkbox"/>	<a href="#">12</a>	VacationRequest	VR_01	RUNNING	Tue Jun 28 16:03:46 EDT 2005	Tue Jun 28 16:03:46 EDT 2005
<input type="checkbox"/>	<a href="#">9</a>	VacationRequest	VR_01	RUNNING	Tue Jun 28 16:03:24 EDT 2005	Tue Jun 28 16:03:24 EDT 2005
<input type="checkbox"/>	<a href="#">6</a>	VacationRequest	VR_01	RUNNING	Tue Jun 28 16:02:14 EDT 2005	Tue Jun 28 16:02:14 EDT 2005

To investigate problems that occur during the execution of processes, you can see a list of process instances or tasks that have encountered problems, read error messages, and search for specific process instances or tasks that you want to troubleshoot.

To intervene in process instances, you can temporarily suspend their progress, force them to complete, and reassign tasks to different users. These troubleshooting tools are especially important for testing workflows during their development.

## Business activity monitoring

Business Activity Monitor enables you to monitor process activity based on process metrics.

Business Activity Monitor consists of two web-based tools.

**Business Activity Monitor (BAM) Workbench:** Enables administrators and developers to set up the collection of process data and the dashboards that provide a business view of the data.

**Business Activity Monitor (BAM) Dashboard:** Enables people to view dashboards and perform ad hoc queries on process data. Users can drill down on metrics, define business rules for alerts, and share dashboards with other users.

A dashboard is generated automatically for each automated process. These dashboards provide the following views of process metrics:

- Historical view
- Real-time view
- Workforce Management view
- System view

Another dashboard is provided that includes an overall view of the system, aggregating metrics across all business processes.

## 2

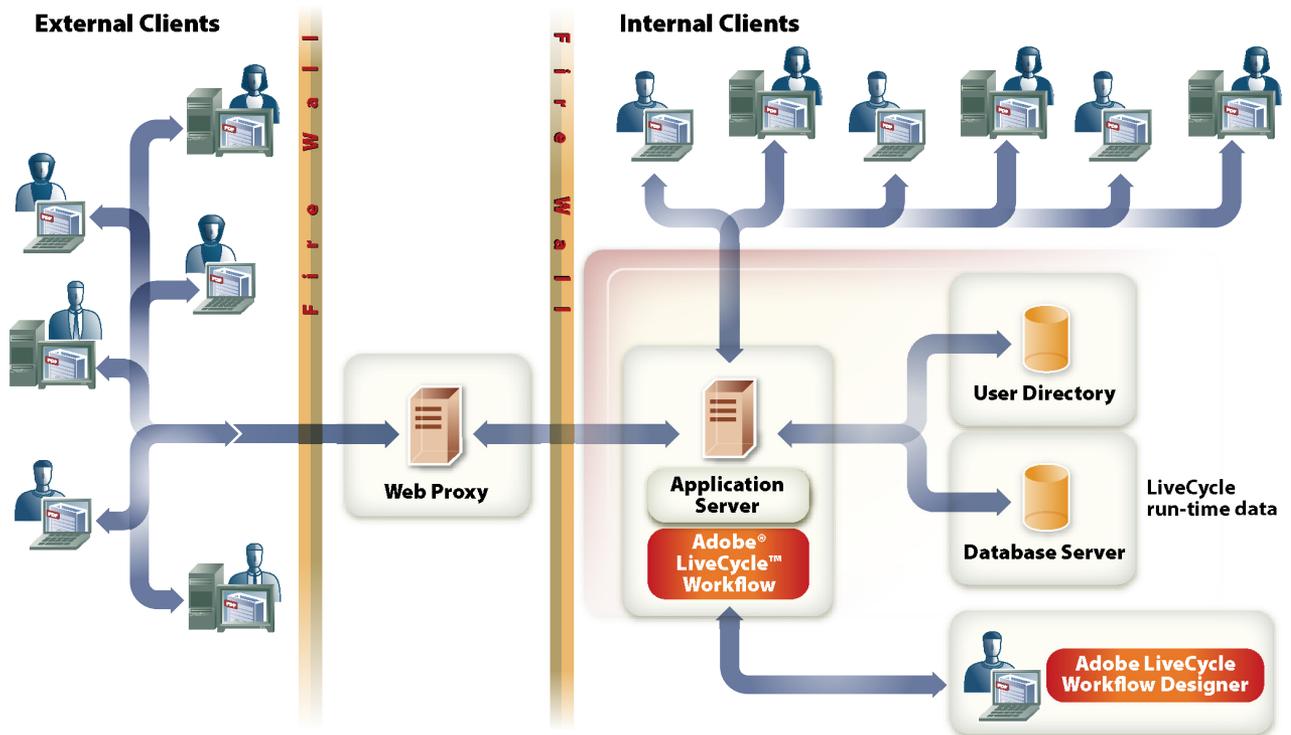
# LiveCycle Workflow Integration

This chapter provides information about how LiveCycle Workflow integrates into your production environment, how your custom client applications can interact with LiveCycle Workflow Server, and how LiveCycle Workflow Server interacts with other services, such as other LiveCycle products.

## LiveCycle Workflow environment

LiveCycle Workflow leverages several legacy systems that may be part of your production environment:

- A corporate LDAP-based directory for looking up user information for authentication purposes
- A relational database for storing LiveCycle Workflow configuration and run-time information
- A J2EE application server for hosting LiveCycle Workflow applications



For detailed information about the software and hardware required to run LiveCycle Workflow, see the *Installing and Configuring LiveCycle* guide for your application server.

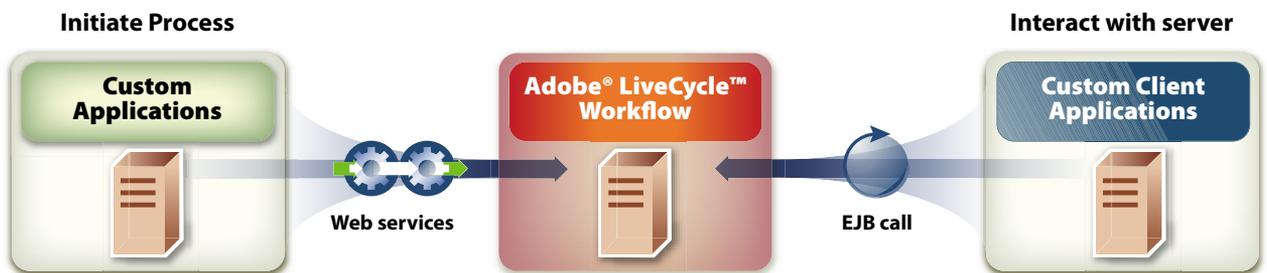
## Interaction with other software

LiveCycle Workflow integrates with the LiveCycle suite of products and core enterprise systems to streamline processes that involve documents and provide an end-to-end business process management solution.

### Custom software integration

There are two mechanisms that applications can use to interact with LiveCycle Workflow:

- Web services enable applications to initiate processes and to receive information about the process instance after it completes. LiveCycle Workflow Server provides a service for each deployed workflow that is appropriately configured. Each service has input and output parameters as defined in LiveCycle Workflow Designer. A deployed web service publishes an XML service definition document composed in Web Services Definition Language (WSDL), which provides service-binding information and information about available web service methods.
- LiveCycle Workflow Server provides JAVA interfaces that can be used to initiate process instances and perform process administration tasks. The LiveCycle Workflow SDK API provides Java packages that you can use to interact with LiveCycle Workflow Server. The LiveCycle Workflow SDK is available as a separate product and is not included with LiveCycle Workflow.



### Server software integration

LiveCycle Workflow uses Quick Process Action Components (QPACs) to integrate with other servers. QPACs are workflow components that enable workflows to make service requests to other servers. QPACs are deployed using LiveCycle Workflow Designer and are then available to use as components that you add to workflows.

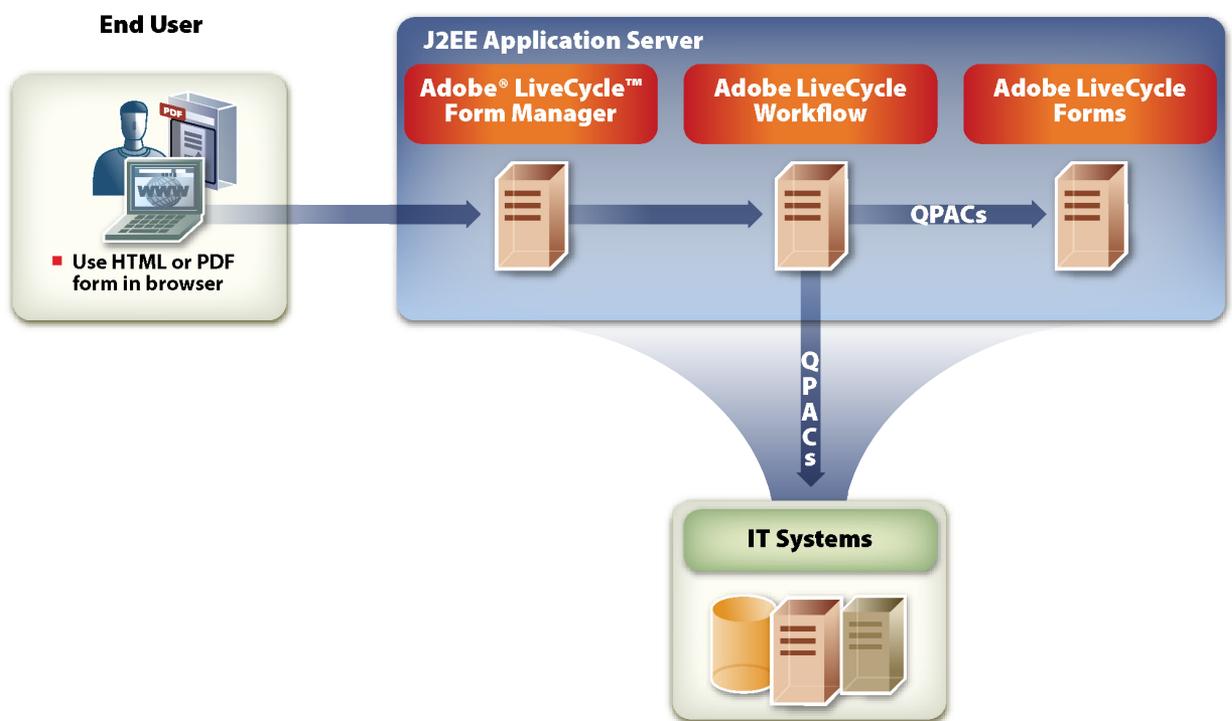
The LiveCycle Workflow SDK includes several QPACs that enable LiveCycle Workflow Server to send requests to several standards-based services, such as database servers, web services, FTP servers, and email servers. Other QPACs are provided that send requests to other LiveCycle products, such as LiveCycle Forms.

The SDK provides Java APIs for creating custom QPACs. You can create QPACs to leverage investments in existing enterprise or legacy business systems for which an API is available, such as enterprise resource planning (ERP), customer relationship management (CRM), and document management services. An integrated development environment for use with Eclipse is provided as an aid for developing custom QPACs.

## Integration with other LiveCycle products

LiveCycle Workflow works seamlessly with other LiveCycle products:

- LiveCycle Designer enables you to create XML or PDF forms that can be used with LiveCycle Workflow. You can also use PDF forms created with Adobe Acrobat® Professional. For information about how to configure your forms to support LiveCycle Workflow, see the *Creating Workflows* guide or *LiveCycle Workflow Designer Help*.
- Adobe LiveCycle Form Manager enables people to open and submit forms that initiate processes. It also provides folders that contain forms that LiveCycle Workflow Server sends to people, as well as a history of forms and tasks that have been completed as a result of participating in processes.
- QPACs enable LiveCycle Workflow to integrate with all LiveCycle products, such as LiveCycle Forms, to provide a complete end-to-end process management system. These QPACs are included in the LiveCycle Workflow SDK.



# Glossary

---

This glossary contains terminology definitions that are specific to documentation for the Adobe LiveCycle suite of products. These terms may have different meanings in other contexts, but they have restricted meanings in this documentation.

## A

### accessible forms

Forms that users with disabilities or vision impairment can view and fill using screen readers and other assistive technologies. See also *tagged Adobe PDF form*.

### Acrobat form

A PDF document, created in Acrobat, that contains one or more form fields. The PDF document may also contain non-form content.

### action

In a workflow, the representation of a step in a business process.

### Adobe certified document

A document that is signed with a specific Adobe root certificate. An Adobe certified document provides a strong guarantee as to the authenticity and immutability of the document. See also *certificate*.

### Adobe document services

Adobe document services extend the value of core enterprise systems to ensure more secure, reliable, and efficient use of business-critical information across the extended enterprise. Adobe document services include the Adobe LiveCycle suite of products and the Acrobat product line.

### application

A set of generally interdependent files that make up a self-contained application that Adobe LiveCycle products can run. Applications may include files such as form designs, Java Server Pages, HTML pages, servlets, and images.

## B

### branch

A branch contains a set of actions interconnected by routes, representing a sequential path taken by a process at execution. The branch always determines the behavior of the workflow.

## C

### certificate

An electronic file that establishes your identity, by binding your identity to your public key, when doing business or other transactions on the web. A *certificate* (or sometimes called a *digital certificate*) is issued by a certificate authority (CA). See also *Adobe certified document*.

### client

The requesting program in a client/server relationship. A web browser is an example of a client application.

### credential

The file that contains a private key. (The corresponding public key is contained in a certificate.) A private key is what one principal presents to another used to establish identity in decryption and signing operations. Credentials are issued by an authentication agent or a certification authority. See also *certificate*.

## D

### deadline

The time by which a person must complete a work item. Deadlines are properties of workflows.

### dynamic form

A form that can expand or contract to reflect the amount of incoming data. See also *interactive form*.

## E

### ebXML

Electronic Business using eXtensible Markup Language (ebXML). A modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. See also *registry*.

### encryption

The conversion of data into a format (called a ciphertext) that cannot be easily understood by unauthorized persons. The conversion is done using an encryption algorithm.

## F

### form

An electronic document that captures and delivers data. A person may add data to an interactive form, or a server process may merge a form design with data to produce a form.

### form authors

LiveCycle Designer users who are capable of creating fillable forms to be used in Acrobat or Adobe Reader, and simple non-interactive forms for deployment to LiveCycle Forms. See also *form developers*.

### FormCalc

A calculation language similar to that used in common spreadsheet software that facilitates form design without requiring a knowledge of traditional scripting techniques or languages.

### form design

The design-time version of a form that an author or developer creates in LiveCycle Designer.

### form developers

LiveCycle Designer users who are capable of creating complex form-based applications for use in different environments. See also *form author*.

### form object

A form element, such as a button or text field, that you can place on a form. An object has its own set of properties and events.

## I

### interactive form

A form that a person can interact with and complete electronically.

## N

### non-interactive form

A form that a person can view or print but cannot fill electronically. Non-interactive forms can be merged or prepopulated with data, but the data cannot be changed by a user. Non-interactive forms are designed for output.

## P

### PDF document

Portable Document Format. A file conforming to the PDF specification as published by Adobe Systems or a file conforming to the XDP specification, containing exactly one PDF packet and no more than one each XFA-Template, XFA-Configuration, XFA-SourceSet, and Annotations packets.

### PDF form

A form that users can access in Acrobat and Adobe Reader. PDF forms are either interactive or non-interactive.

### permissions

Security settings applied, for example, to restrict users from opening, editing, printing, or removing encryption from a PDF file. Permissions cannot be changed unless the user has the Permissions password. Permissions can be set in LiveCycle Designer, Acrobat, LiveCycle Document Security, and other products.

## policy

Defines a set of security permissions and users who can access a PDF document to which the policy is applied. Policies are created using LiveCycle Policy Server and can be applied to documents using LiveCycle Policy Server, LiveCycle Document Security, or Acrobat 7.0 or later.

## prepopulated form

A form that appears to the user with some or all fields automatically populated with data.

## Q

### QPAC

Quick Process Action Component. A JAR file that contains server-side code and client-side code for use with LiveCycle Workflow. In LiveCycle Workflow Designer, QPACs provide action components that can be added to workflows to represent a step in a process. LiveCycle Workflow Server interprets each action of the workflow and executes the server-side code of the corresponding QPACs. QPACs enable LiveCycle Workflow to interact with other Adobe LiveCycle products, such as LiveCycle Forms and LiveCycle Barcoded Forms.

## R

### reminder

A notification sent to people that reminds them to complete a work item. Reminders are properties of workflows.

### render

An action whereby LiveCycle Forms merges a form design, possibly with data, to display a form in PDF or HTML format in a browser.

### registry

An ebXML-compliant repository of shared information that provides services for the purpose of enabling business process integration between interested parties. See also *ebXML* and *repository*.

## repository

The underlying storage area within a registry. See also *registry*.

### restricted document

A PDF document with password security restrictions (permissions) that prevent the document from being opened, printed, or edited.

### rights-enabled document

A PDF document that includes security extensions that enable Adobe Reader users to fill forms, add comments, and sign documents.

### route

The path between actions on a workflow. Routes determine the order in which LiveCycle Workflow Server executes actions at run time.

### run time

For form rendering, the time when an application or server process retrieves a form design, possibly merges it with data, and presents it to a user for viewing or filling.

## S

### split

A segment in a workflow that contains one or more branches. The branches in a split are executed in parallel.

### static form

A form that remains exactly as it was designed. The layout does not change according to the amount of incoming data.

### subform

An object that can act as a container for form objects and other subforms. A subform helps to position form objects relative to each other and provide structure in dynamic form designs. A subform can also provide a reference point, when binding data to a form, by restricting the scope for a field so that it matches that of the corresponding data node.

## T

### **tagged Adobe PDF form**

Includes a logical structure and a set of defined relationships and dependencies among the various elements, plus additional information that permits reflow. See also *accessible forms*.

### **turnkey**

An installation option that automatically installs and configures the LiveCycle product files, JBoss application server, and MySQL database, and deploys the product files to JBoss. After you perform a turnkey installation, the LiveCycle product is ready to use.

## U

### **usage rights**

Rights that extend the functionality of Adobe Reader and enable users to save forms with data, add comments, and sign documents.

## W

### **workflow**

The electronic representation of a business process. Workflows are created using LiveCycle Workflow Designer.

## X

### **XDP file**

XML Data Package. LiveCycle Designer saves form designs as either XDP files or PDF files. LiveCycle Forms uses XDP files to render forms in PDF or HTML format.

### **XML Forms Architecture**

Represents the underlying technology beneath the Adobe XML forms solution. It enables the construction of robust and flexible form-based applications for use on either the client or the server.

### **XML form**

A PDF form that conforms to the Adobe PDF specification and the Adobe XML Forms Architecture. XML forms are typically created in LiveCycle Designer. XML forms can have the file name extension .xdp or .pdf.



# Creating Workflows

July 2006

**Adobe® LiveCycle™ Workflow**

Version 7.2

© 2006 Adobe Systems Incorporated. All rights reserved.

Adobe® LiveCycle™ Workflow 7.2 Creating Workflows for Microsoft® Windows®, Linux®, and UNIX®  
Edition 2.1, July 2006

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names and company logos in sample material or in the sample forms included in this software are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, LiveCycle, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

All other trademarks are the property of their respective owners.

Powered by Celequest. Contains technology distributed under license from Celequest Corporation. Copyright 2005 Celequest Corporation. All rights reserved.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

This product includes software developed by the Jaxen Project (<http://www.jaxen.org/>).

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# Contents

---

<b>Preface .....</b>	<b>9</b>
What's in this guide? .....	9
Who should read this guide? .....	9
Related documentation .....	9
<b>1 Getting Started .....</b>	<b>11</b>
Before you start.....	11
About the development process.....	11
Opening LiveCycle Workflow Designer.....	12
Navigating LiveCycle Workflow Designer .....	12
Customizing palette behavior .....	13
Using palettes in auto-hide mode.....	13
How processes are organized.....	14
About process categories .....	14
About process types .....	14
About workflows .....	15
Adding items to the Processes palette.....	15
Creating process categories.....	15
Creating process types.....	16
Creating workflows .....	17
Removing items from the Processes palette.....	18
Opening and closing workflows .....	18
Modifying the view .....	19
Zooming in and out .....	19
Hiding the workflow .....	19
Protecting workflows.....	19
Locking workflows.....	20
Unlocking workflows .....	20
<b>2 Drawing Visual Representations of Workflows.....</b>	<b>21</b>
About actions.....	21
About routes .....	21
About branches.....	22
About components .....	22
Drawing the visual representations of workflows.....	23
Adding and deleting actions .....	23
Component types .....	24
Adding and deleting routes .....	24
Modifying route labels .....	25
Adding text to the visual representation.....	26
Printing workflows.....	26

<b>3</b>	<b>Implementing Business Logic.....</b>	<b>27</b>
	Saving information in variables .....	27
	Variables as input and output parameters .....	28
	Creating variables .....	28
	About variable types.....	29
	Setting variable values at run time with Set Value actions.....	32
	Determining the order of execution .....	33
	Specifying the first action .....	33
	Executing one of many actions.....	33
	About rules.....	34
	Adding and modifying rules.....	35
	Prioritizing routes .....	35
	Decision Point actions.....	36
	Executing branches in parallel with splits.....	36
	Convergence after splits .....	37
	Adding splits.....	37
	Executing script with Script actions .....	38
	Workflow objects in script .....	38
	Completing the action .....	39
	Testing script .....	39
	About process data .....	39
	Running script.....	40
	Script keyboard shortcuts.....	40
	Controlling execution timing with Wait Point actions .....	41
	Linking processes with Chained Process actions.....	41
	Expressions .....	42
	About the XPathExpression Builder .....	42
	Building expressions.....	43
	Example XPath expressions and process data tree .....	44
	Accessing data in collections using XPath expressions .....	45
	Using XPath expressions as list indexes and map keys.....	46
<b>4</b>	<b>Configuring Human Interaction .....</b>	<b>47</b>
	Using forms and documents in workflows .....	47
	About initiating processes with forms .....	48
	Identifying forms that initiate processes.....	48
	Storing form data in form variables .....	49
	About XML form data in the process schema .....	51
	Making form fields appear in XPathExpression Builder.....	51
	Adding data nodes to form variables.....	52
	Referencing documents in document variables.....	53
	About forms and documents in User actions .....	54
	Configuring User actions to use forms and documents .....	55
	Following routes based on user decisions .....	55
	Automatically using route names as the choice list .....	57
	Making choice selection mandatory.....	57

## 4 Configuring Human Interaction (Continued)

Assigning tasks to users .....	57
Assigning tasks to specific people .....	58
Specifying users by user name .....	58
Specifying the process creator .....	58
Assigning tasks to previous participants.....	58
Assigning tasks to groups.....	59
Specifying users with expressions .....	59
Setting time constraints for users.....	60
Configuring escalations.....	61
Setting deadlines .....	62
Sending reminders .....	63
Providing instructions .....	64
Configuring attachments .....	64
Persisting attachments .....	65
Enabling users to see and add attachments .....	65
Adding attachments to tasks.....	65
Saving task attachments .....	66
Document attributes for attachments .....	66
Task attachments in email messages .....	67

## 5 Selecting Workflow and Branch Types ..... 68

Data persistence .....	68
Process complexity and system efficiency.....	68
Workflow and branch compatibility.....	69
Action and branch compatibility.....	69

## 6 Error Handling ..... 70

About errors .....	70
Branch behavior when errors occur .....	70
About transactions and branches.....	71
About transaction-aware actions.....	71
About long-lived actions.....	72
Catching situational errors with the Stall action.....	72

## 7 Managing Workflows ..... 73

Deploying workflows .....	73
Activating workflows .....	73
Determining workflow deployment state.....	74
Adding components to the Components palette.....	74
Creating component categories.....	74
Deploying components.....	75
Updating components.....	75
Moving components to different categories.....	75
Using custom component icons.....	76
Removing components.....	76
Working with workflow versions .....	77
Modifying workflows .....	77
Creating copies of workflows .....	77
Modifying deployed workflows.....	78

**7 Managing Workflows (Continued)**

Exporting and importing workflows ..... 78  
    Exporting workflows ..... 79  
    Importing workflows ..... 79  
Migrating workflows to production environments ..... 79

**8 Designing Forms for LiveCycle Workflow..... 81**

Form fields for use with workflows ..... 81  
    Fields for XML forms ..... 81  
    Fields for Acrobat forms ..... 82  
Configuring Submit buttons when handling documents ..... 83  
About field names in Acrobat forms ..... 84  
Initiating processes through email ..... 84  
Using multiple forms in workflows ..... 84  
    Use consistent field names ..... 85  
    Use common form architectures ..... 85  
Collecting date and time data in forms ..... 85

**A Component Properties ..... 86**

Viewing component properties in LiveCycle Workflow Designer ..... 86  
About component properties ..... 86

**B Function Reference ..... 88**

Boolean functions ..... 88  
    boolean ..... 88  
    false ..... 88  
    not ..... 88  
    true ..... 89  
Collection functions ..... 89  
    get-map-keys ..... 89  
    get-collection-size ..... 90  
Date-Time functions ..... 90  
    add-days-to-date ..... 90  
    add-days-to-dateTime ..... 90  
    add-hours-to-dateTime ..... 91  
    add-minutes-to-dateTime ..... 91  
    add-months-to-date ..... 92  
    add-months-to-dateTime ..... 92  
    add-seconds-to-dateTime ..... 93  
    add-years-to-date ..... 93  
    add-years-to-dateTime ..... 94  
    current-date ..... 94  
    current-dateTime ..... 95  
    current-time ..... 95  
    date-equal ..... 95  
    date-greater-than ..... 96  
    date-less-than ..... 96  
    dateTime-equal ..... 96  
    dateTime-greater-than ..... 97  
    dateTime-less-than ..... 97

## B Function Reference (Continued)

Date-Time functions (Continued)	
get-day-from-date .....	98
get-day-from-dateTime .....	98
get-days-from-date-difference .....	98
get-days-from-dateTime-difference .....	99
get-hours-from-dateTime .....	99
get-hours-from-dateTime-difference .....	100
get-hours-from-time .....	100
get-minutes-from-dateTime .....	100
get-minutes-from-dateTime-difference .....	101
get-minutes-from-time .....	101
get-month-from-dateTime .....	102
get-month-from-dateTime .....	102
get-months-from-date-difference .....	102
get-months-from-dateTime-difference .....	103
get-seconds-from-dateTime .....	103
get-seconds-from-dateTime-difference .....	104
get-seconds-from-time .....	104
get-timezone-from-date .....	104
get-timezone-from-dateTime .....	105
get-timezone-from-time .....	105
get-year-from-date .....	105
get-year-from-dateTime .....	106
get-years-from-dateTime-difference .....	106
get-years-from-date-difference .....	106
format-date .....	107
format-dateTime .....	107
parse-date .....	108
parse-dateTime .....	108
time-equal .....	109
time-greater-than .....	109
time-less-than .....	110
Document object functions .....	110
getDocLength .....	110
getDocAttribute .....	110
setDocAttribute .....	111
getDocContent .....	112
setDocContent .....	112
Miscellaneous .....	112
deserialize .....	112
is-null .....	113
serialize .....	113
Node set functions .....	113
count .....	113
last .....	114
position .....	114
sum .....	114

**B Function Reference (Continued)**

Number functions ..... 115  
    ceiling ..... 115  
    floor ..... 115  
    number ..... 115  
    round ..... 116  
String Functions ..... 116  
    concat ..... 116  
    contains ..... 116  
    lower-case ..... 117  
    normalize-space ..... 117  
    starts-with ..... 117  
    string ..... 118  
    substring-after ..... 118  
    substring-before ..... 118  
    substring ..... 119  
    string-length ..... 119  
    translate ..... 120  
    upper-case ..... 121  
Operators ..... 121  
Date and time parameters ..... 122

**C Migrating Workflows from Older Versions ..... 123**

**Glossary ..... 124**

**Index ..... 128**

# Preface

---

LiveCycle™ Workflow Designer is the tool that you use to create and deploy electronic definitions of business processes.

LiveCycle Workflow Designer provides tools for visually representing the flow of execution of a process and for implementing the business logic that enables decisions to be made automatically. After it is deployed, LiveCycle Workflow Server interprets the electronic definition and executes each step of the process accordingly.

## What's in this guide?

This guide includes conceptual and procedural information that you require to use LiveCycle Workflow Designer. You can use this guide to learn about the following aspects of using LiveCycle Workflow Designer:

- How to create new workflows and manage different workflow versions
- How to create the visual representation of the business process you are automating
- How to add business logic to workflows
- How to deploy workflows to LiveCycle Workflow Server
- The requirements of form designs so that they can be used with Adobe® LiveCycle Workflow

## Who should read this guide?

This guide is for people who are designing workflows in LiveCycle Workflow Designer. Before you use LiveCycle Workflow Designer, you should understand the business processes that you are automating. You or someone in your organization should have already mapped the business processes, installed LiveCycle Workflow Server and configured the connection to the database, and defined the forms that you will use for enabling people to participate in processes.

## Related documentation

You can refer to other product documentation to learn more about LiveCycle Workflow:

<b>For information about</b>	<b>See</b>
The LiveCycle Workflow environment and how to install and configure the product components	<i>Installing and Configuring LiveCycle for JBoss</i> <i>Installing and Configuring LiveCycle for WebSphere</i> <i>Installing and Configuring LiveCycle for WebLogic</i>
The components of LiveCycle Workflow and how LiveCycle Workflow integrates with other LiveCycle products	<i>LiveCycle Workflow Overview</i>
How to perform the tasks involved with tracking processes at run time	<i>LiveCycle Workflow Administration Help</i>

---

<b>For information about</b>	<b>See</b>
How to set up process activity reports based on real time data	<i>Using LiveCycle Workflow Workbench</i>
Last-minute changes that occur to the product	<i>LiveCycle Workflow Readme</i>
Other LiveCycle products	<a href="http://partners.adobe.com/public/developer/main.html">http://partners.adobe.com/public/developer/main.html</a>
Patch updates, technical notes, and additional information on this product version	<a href="http://www.adobe.com/support/products/enterprise/index.html">www.adobe.com/support/products/enterprise/index.html</a>

---

# 1

## Getting Started

LiveCycle Workflow Designer is a graphical design tool that you use to electronically define business processes. This chapter provides an introduction to the LiveCycle Workflow Designer interface, the concepts that you need to understand, and the tasks that you need to perform before developing workflows.

### Before you start

Before you use LiveCycle Workflow Designer, you should understand the business processes that you are automating. You or other key players in your organization should have already mapped the business processes, and defined the forms that you will use for enabling people to participate in processes. Understanding how data is captured and how data flows throughout the process is key to successfully developing workflows.

### About the development process

The following table outlines the tasks that you typically perform when developing workflows, and the typical order in which you perform them.

Task	Topic
Organize the processes that you are developing.	<a href="#">"Adding items to the Processes palette" on page 15</a>
Create the visual representation of the process you are defining.	<a href="#">"Drawing Visual Representations of Workflows" on page 21</a>
Add rules that incorporate business logic to enable decision making.	<a href="#">"Implementing Business Logic" on page 27</a>
Enable people to participate in automated processes through the use of forms and the automated routing of forms.	<a href="#">"Configuring Human Interaction" on page 47</a>
Specify options that maximize system efficiency depending on the characteristics of the process you are defining.	<a href="#">"Selecting Workflow and Branch Types" on page 68</a>
Incorporate safety features that handle unexpected behavior and enable administrative intervention.	<a href="#">"Error Handling" on page 70</a>
Deploy workflows to LiveCycle Workflow Server to make them available to clients to initiate.	<a href="#">"Deploying workflows" on page 73</a>
Work with different versions of processes to enable iterative modifications to the design while maintaining the availability of services.	<a href="#">"Working with workflow versions" on page 77</a>

## Opening LiveCycle Workflow Designer

When you open LiveCycle Workflow Designer, a connection is established with LiveCycle Workflow Server. To use LiveCycle Workflow Designer, you need to log in using a valid user name and password. Your LiveCycle Workflow administrator needs to configure your user account to let you use LiveCycle Workflow Designer.

**Note:** When LiveCycle Workflow Designer is installed on a different computer than where LiveCycle Workflow Server is deployed, the system clocks on both your computer and the server must be set within two hours of the actual time in the set time zone. If the system clocks are not accurate, LiveCycle Workflow Designer will not be able to connect to LiveCycle Workflow Server.

The server that LiveCycle Workflow Designer connects to is specified when LiveCycle Workflow Designer is installed. To connect to a different server, you need to uninstall and reinstall LiveCycle Workflow Designer. For information about installing, see the *Installing and Configuring LiveCycle* guide for your application server.

Several features that you see in LiveCycle Workflow Designer are provided by LiveCycle Workflow after you log in:

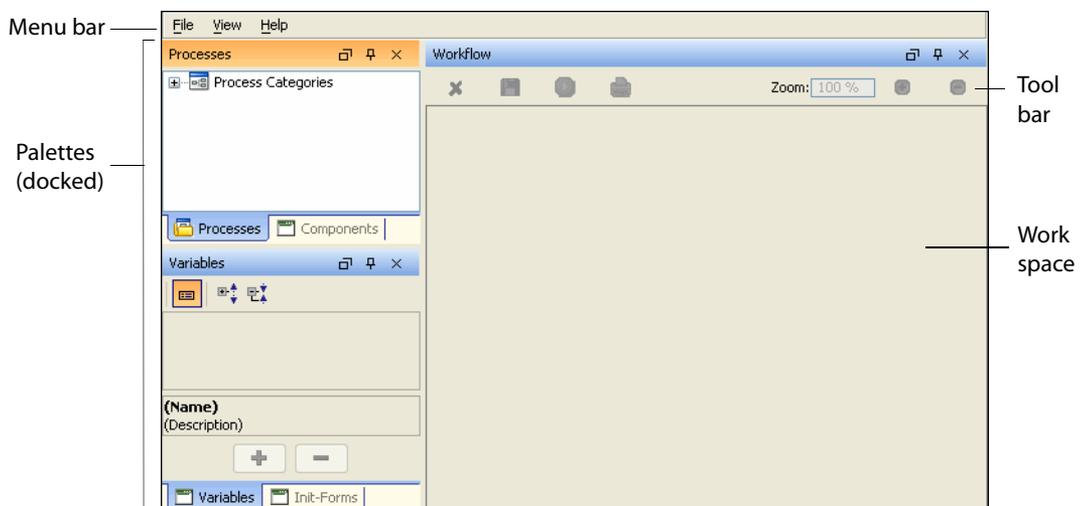
- Access to the electronic definitions of the processes, called workflows, that you work with in LiveCycle Workflow Designer
- Several workflow component and tool definitions
- *LiveCycle Workflow Administration Help*

**Tip:** You can run LiveCycle Workflow Designer from a command line so that the command window remains open. The command window provides error messages that are valuable when developing with LiveCycle Workflow Designer.

## Navigating LiveCycle Workflow Designer

LiveCycle Workflow Designer consists of several areas that you work with when developing workflows:

- The menu bar provides access to several commands.
- The tool bar provides quick access to commands that you frequently use when you are working on workflows.
- Palettes contain the tools that you use to develop electronic definitions of processes.
- The work space is the area where different windows and dialog boxes open when you use commands and tools.



## Customizing palette behavior

You can customize the way palettes behave to suit your preference. The following options are available for customizing palette behavior:

- Float palettes so that they appear in their own window
- Move docked palettes to a new location
- Automatically reveal palettes when you use them and hide them when not in use

By default, palettes are docked in the main LiveCycle Workflow Designer window and are always displayed. For information about how to use palettes when they are hidden, see [“Using palettes in auto-hide mode” on page 13](#).

► **To float or dock a palette:**

- On the title bar of the palette, click **Toggle Floating**. 

► **To move a docked palette:**

- Drag the title bar of the palette to a different area of the LiveCycle Workflow Designer window. As you drag the palette, the outline of the new location appears.

**Tip:** You can also use this method to float a palette.

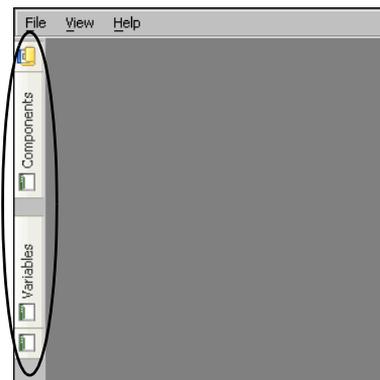
► **To automatically reveal and hide a palette:**

- On the title bar of the palette, click **Toggle Auto-Hide**.

## Using palettes in auto-hide mode

You can easily reveal palettes that are in auto-hide mode as you want to use them. After you use the palette, it automatically becomes hidden. You can also hide the palette again if you decide not to use it.

Palettes in auto-hide mode appear as tabs along the left side of the LiveCycle Workflow Designer window. For information about putting palettes in auto-hide mode, see [“Customizing palette behavior” on page 13](#).



► **To reveal a hidden palette:**

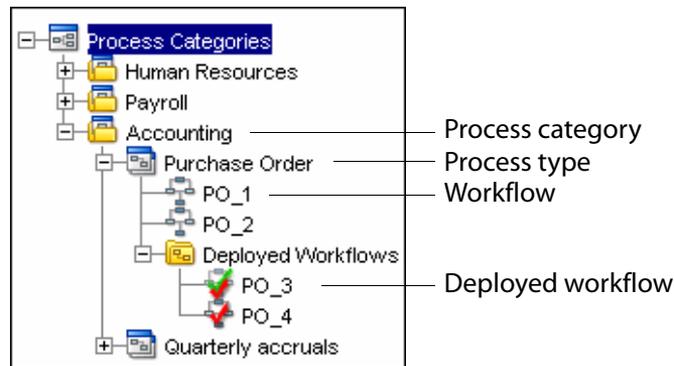
- Pause the pointer over the tab for the palette you want to use.

► **To hide a palette that you have revealed:**

- In the title bar of the palette, click **Hide active auto-hide window**  in the palette.

## How processes are organized

The Processes palette provides a listing of all processes for which electronic definitions have been created. The processes are organized in a tree structure that includes process categories, process types, workflows, and deployed workflow.



For information about deploying workflows, see [“Managing Workflows” on page 73](#).

### About process categories

Process categories enable you to organize process types into logical groups so that you can easily find the process type you want to work with. Process categories are useful when you have many process types. Process categories have the following properties:

- Name appears in the Processes palette.
- Description provides information for users who open the category properties.
- Status determines whether you should use this process category or whether the process category is to be deleted. If the status is Active, the contents of the process category are for active development and you can use it. If the status is Marked For Deletion, the process category and all the contents are going to be automatically deleted and you should not use them. For more information about deleting process categories, see [“Removing items from the Processes palette” on page 18](#).

For information about creating process categories and viewing the properties, see [“Creating process categories” on page 15](#).

### About process types

Process types represent the business processes that you are automating. One process type should exist for each business process. Process types have the following properties:

- Name appears in the Processes palette
- Category determines the category in which the process type appears
- Description provides information for users who open the process type properties

For information about creating process types and viewing the properties, see [“Creating process types” on page 16](#).

## About workflows

Workflows are the electronic definitions of business processes and are associated with the process types. Workflows include the visual representation of the process and the business logic that determines the order in which the steps of the process are performed.

You can create several workflows for the same process type. Several workflows enable you to modify the definition of a process without interrupting the workflow service. Each workflow is a different version of the process you are automating. For more information, see [“Working with workflow versions” on page 77](#).

Workflows have the following properties:

- Name identifies the process in the Processes palette.
- Description provides information about the workflow for users who open the workflow properties.
- Main Branch Type sets the type of branch that is used as the main workflow branch and determines workflow behavior at run time. For more information, see [“Selecting Workflow and Branch Types” on page 68](#).
- Transient determines whether the workflow type is Synchronous or Transient. For more information, see [“Selecting Workflow and Branch Types” on page 68](#).
- Web Service Access determines whether the workflow can be initiated through web services.
- Monitor Access determines whether the workflow is available for monitoring in LiveCycle Workflow Business Activity Monitor.

For information about creating new workflows, see [“Creating workflows” on page 17](#).

### Deployed workflows

Deployed workflows are workflows for which LiveCycle Workflow Server currently provides service to client applications. A process is available for initiation when a workflow for the process is deployed and activated. Only one workflow can be activated. For information about deploying workflows, see [“Deploying workflows” on page 73](#).

## Adding items to the Processes palette

Add items to the Processes palette to organize process types and workflows. You can add process categories, process types, and workflows to the Processes palette:

- To add a process type, the process category in which you want to include it must already exist.
- To create a workflow, the process type with which it is associated must already exist.

The items that are added to the Processes palette are available to other users who log into LiveCycle Workflow Designer.

If you want information about deleting items from the Processes palette, see [“Removing items from the Processes palette” on page 18](#).

## Creating process categories

Create a process category to establish a logical group of process types. When you create a category, you must provide a name. After you create a process category, you can change the name or delete it at any time. For information about deleting, see [“Removing items from the Processes palette” on page 18](#).

If you need more information about process categories, see [“About process categories” on page 14](#).

- ▶ **To create a process category:**
  1. Click **File > New > Process Categories**.
  2. In the **Name** box, type the name of the process category as you want it to appear on the palette.
  3. (Optional) In the **Description** box, type a description.
  4. In the **Status** list, ensure that **Active** is selected so that you can add process types to the category.
  5. Click **Save** and then click **OK** to close the notification.
  
- ▶ **To modify a process category:**
  1. Right-click the process category that you want to modify and click **Process Category Properties**.
  2. Change the name, status, or description as required.
  3. Click **Save** and then click **OK** to close the notification.

## Creating process types

Create a process type to represent a business process that you are automating. When you create a process type, you must specify a name. After you create a process type, you can modify the properties at any time. For information about deleting process types, see ["Removing items from the Processes palette" on page 18](#).

For more information about process types, see ["About process types" on page 14](#).

- ▶ **To create a new process type:**
  1. Click **File > New > Process Type**.
  2. In the **Name** box, type the name of the process type. The name can include letters, numbers, hyphens ("-"), underscores ("\_"), asterisks ("\*"), and periods ("").
  3. In the **Process Category** list, select the category in which the process type belongs.
  4. (Optional) In the **Description** box, type a description. The description appears in the process type properties as information for developers.
  5. Click **Save** and then click **OK** to close the notification.
  
- ▶ **To modify a process type:**
  1. Right-click the process type that you want to modify and click **Process Type Properties**.
  2. Change the name, category, or description as required.
  3. Click **Save** and then click **OK** to close the notification.

## Creating workflows

To develop the electronic definition of a process, you need to create a new workflow. When you create a workflow, you specify the workflow name, the process type to which it belongs, whether it can be initiated through web services, and whether it is monitored through Business Activity Monitor. After you create a workflow, you can change the properties at any time before deploying it.

When you create a workflow, it is automatically locked so that other developers cannot modify it. For more information, see [“Protecting workflows” on page 19](#).

If you want information about making copies of an existing workflow, see [“Creating copies of workflows” on page 77](#).

For information about deleting workflows, see [“Removing items from the Processes palette” on page 18](#).

For more information about workflows, see [“About workflows” on page 15](#).

### ► To create a new workflow:

1. Right-click the process type for which you want to create the workflow, and click **New Workflow**.
2. In the **Name** box, type a name for the workflow.
3. (Optional) In the **Description** box, type a description. The description appears in the workflow properties as information for developers.
4. In the **Type** list, select the type of workflow you want to create.

**Note:** If you are not sure what type to create, select **Asynchronous**. Asynchronous is the most general type of workflow. You can change the type at any time before you deploy the workflow. For more information, see [“Selecting Workflow and Branch Types” on page 68](#).

5. To provide access to the workflow through web services, select **Web Service Access**.
6. To monitor the workflow with Workflow Dashboard, select **Monitor Access**.
7. Click **Save** and click **OK** to close the notification. The workflow opens in the work space so you can edit it.

### ► To modify workflow properties:

1. Right-click the workflow that you want to modify and click **Workflow Properties**.
2. Change the **Name**, **Description**, **Main Branch Type**, **Transient**, **Web Service Access**, or **Monitor Access** properties, as required.
3. Click **Save** and then click **OK** to close the notification.

## Removing items from the Processes palette

To keep the Processes palette up to date and to manage space in the database, you can specify process categories that should be deleted. When process categories are deleted, the process types and workflows that they contain are also deleted.

You use LiveCycle Workflow Designer to indicate the items that you want to be deleted. Deletion does not occur immediately. You use the Purge feature of Object Definition Editor to delete items that are marked for deletion. Object Definition Editor is provided with the LiveCycle Workflow SDK.

**Note:** Until you use the Purge feature, items that are marked for deletion continue to function normally.

► **To delete a process category:**

1. In the Processes palette, right-click the process category and click **Process Category Properties**.
2. In the **Status** menu, select **Marked For Deletion**.
3. Click **Save**.
4. Open Object Definition Editor and use the Purge feature.

## Opening and closing workflows

Open a workflow when you want to view it or make changes to it. You can open any workflow that appears in the Processes palette. What you can do with the workflow depends on whether the workflow is locked:

- When you open a workflow for viewing, you can see the components of the workflow and view the properties, but you cannot change them. If a different user has locked the workflow, you can only open it for viewing.
- When you open a workflow for editing, you can see the workflow components and their properties, change their properties, and add or remove components. Also, the workflow is automatically locked so that other users cannot make changes to it.

**Note:** Workflows that include many actions and routes may not have the same layout as when they were previously opened. Although the layout may change, the workflows function the same as they did in their previously saved state.

When you are finished with the workflow, you can close it.

**Tip:** You can have only one workflow open at the same time. A workflow that is open will close if you open a different workflow.

For more information about locking workflows, see [“Protecting workflows” on page 19](#).

► **To open a workflow for viewing:**

- Right-click the workflow and click **View Workflow**.

► **To open a workflow for editing:**

- Right-click the workflow and click **Edit Workflow**.

► **To close a workflow:**

- Click **File > Close** or click the **Close Workflow** button.



## Modifying the view

There are several ways that you can modify the view of the workflow that appears in the work space:

- Increase or decrease the magnification of the view. See [“Zooming in and out” on page 19](#).
- Hide the workflow to use the work space for different purposes. See [“Hiding the workflow” on page 19](#).

### Zooming in and out

You can use the Zoom tool to increase or decrease the magnification of a workflow. Zoom is useful when you want to focus on a specific part of the visual representation.

If all of the components of the workflow do not appear in the view, you can use the Workflow Overview to see which part of the workflow is currently in view. The entire workflow appears in the Workflow Overview. A red border shows which part appears in the view.

➤ **To increase or decrease the magnification:**

- Click **Zoom In**  to magnify the view, and click **Zoom Out**  to decrease the magnification.

➤ **To see the Workflow Overview:**

- Click **View > Workflow Overview**.

### Hiding the workflow

You can temporarily hide the workflow when you want to use the work space for other purposes. After you hide the workflow, you can view it again. Hiding the workflow closes the view of the workflow but does not close the workflow file.

If you want information about closing the workflow file, see [“Opening and closing workflows” on page 18](#).

➤ **To close the view of the workflow:**

- Click the **Close** button. 

➤ **To see a hidden workflow:**

- Click **View > Workflow**.

## Protecting workflows

LiveCycle Workflow Designer and LiveCycle Workflow Server work together to provide a basic content management system for workflows. Workflow protection is important when more than one person is responsible for developing workflows:

- LiveCycle Workflow Designer lets you lock workflows to prevent other users from making changes to them. Locked workflows can be edited or unlocked only by the user who locked them.
- LiveCycle Workflow Server ensures that the lock is enforced for all other LiveCycle Workflow Designer users. Other users can open locked workflows for viewing only and cannot save any changes that they make to them.

LiveCycle Workflow administrators can unlock any workflow, regardless of who locked it. Also, only LiveCycle administrators can specify who is able to log in to LiveCycle Workflow Designer.

## Locking workflows

Lock workflows so that other users cannot open them for editing. When a workflow is locked, a red check mark appears over the workflow icon in the Processes palette. 

Workflows are automatically locked when you create them and when you open them for editing.

**Tip:** To see who has locked a workflow, open it for viewing. The user name of the person who locked the workflow appears at the top of the work space.

### ► To lock a workflow:

- Right-click the workflow and click **Lock Workflow**.

## Unlocking workflows

Unlock workflows to make them available to all LiveCycle Workflow Designer users for editing.

### ► To unlock a workflow:

- Right-click the workflow and click **Unlock Workflow**.

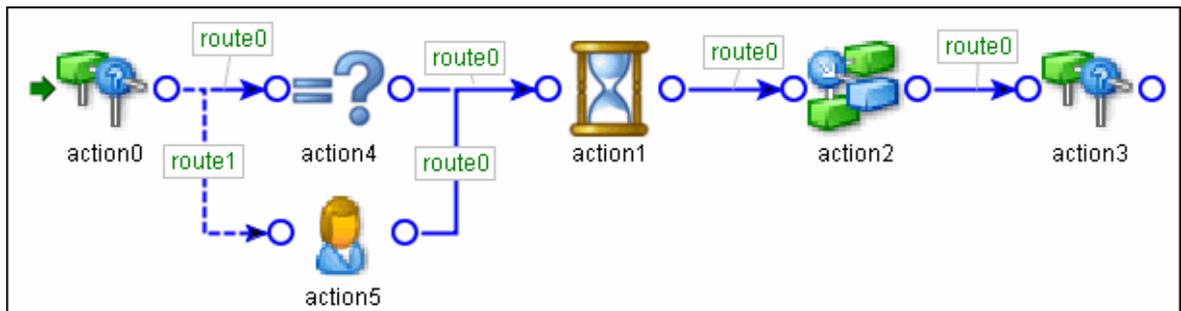
## 2

# Drawing Visual Representations of Workflows

Creating the visual representation of a process is the first step in developing a workflow. You use the visual representation to accomplish several tasks:

- Access the properties of the components of the workflow.
- Create a printed copy of the visual representation to use in presentations and reports.

In LiveCycle Workflow Designer, visual representations illustrate the steps in business processes and the order in which they occur. The visual representation of a process resembles a flowchart. Arrows indicate the direction of progress.



## About actions

Actions are the steps or activities that occur in the business process.

Workflows can include several types of actions. The type of action determines which activity that LiveCycle Workflow Server performs when the action is executed at run time.

For example, if a workflow includes a User action, when the process is run and the User action is executed, LiveCycle Workflow Server sends a form to a person to fill and submit.

Components are the templates from which actions are created. You use the Components palette to add actions to workflows. For more information, see [“About components” on page 22](#) and [“Adding and deleting actions” on page 23](#).

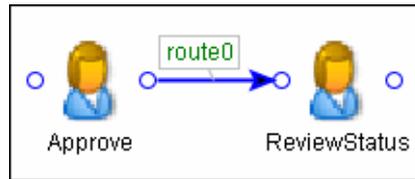
For more information about the different types of actions, see [“Component types” on page 24](#).

## About routes

Routes are used to represent the order of execution of actions in a workflow.

If you want information about adding routes, see [“Adding and deleting routes” on page 24](#).

A route joins two actions that are executed sequentially. Routes begin at one action and terminate at the action that is to be executed next. Arrowheads terminate routes and indicate the order of progression.



Routes can appear as solid lines or as dashed lines:

- Solid lines indicate that the action to which the routing line points is always executed.
- Dashed lines indicate that a rule determines whether the next action is executed. For information, see [“About rules” on page 34](#).

Each route has a label that you can customize to provide meaning. Typically, the label provides insight into the rule that is associated with the route. Route labels can also be exposed as form submission choices to process participants. For information, see [“Modifying route labels” on page 25](#) and [“Following routes based on user decisions” on page 55](#).

## About branches

Branches are segments of a workflow that include actions and routes. Branches allow you to implement different behaviors for different segments of the workflow. The type of branch influences the following behaviors:

- System stability and system efficiency
- The effect of a stalled task on previous tasks in the branch
- The information that appears in Adobe LiveCycle Form Manager when used to participate in processes

The branch type also dictates the type of action that can be added. For more information about the types of branches, see [“Selecting Workflow and Branch Types” on page 68](#).

Workflow definitions can include one or more branches. When you create a workflow and add components to it, the components belong to the default branch, called the main branch. The type of the main branch is specified when the workflow is created, and can be changed in the workflow properties. For information about creating workflows, see [“Creating workflows” on page 17](#).

To add branches to a workflow, you can add a split. For more information about splits, see [“Executing branches in parallel with splits” on page 36](#).

## About components

LiveCycle Workflow Designer provides several workflow components that you use to build workflows. Components appear in the Components palette and represent either tools or actions:

- Actions represent the steps in a process, as described in [“About actions” on page 21](#).
- Tools are added to the workflow to achieve various results. For example, you add a Split component so that you can add several branches that execute in parallel. You can also add a Text component to add labels that improve the presentation of the workflow.

Some components are included with LiveCycle Workflow Designer. Your development team can also use the QPAC Development Environment (QDE) and the Java API from the LiveCycle Workflow SDK to create custom components. To use components, you must deploy them. For information about deploying components, see [“Deploying components” on page 75](#).

## Drawing the visual representations of workflows

Each workflow must include a visual representation of the process that the workflow represents. To draw the visual representation, you add the actions that represent the steps in the process and the routes that show the order in which the actions are executed.

**Note:** Workflows that include many actions and routes may not preserve their exact layout after they are saved, closed, and opened again. Although the layout may change, the workflows function the same as they did in their previously saved state.

### Adding and deleting actions

Add actions to the visual representation of the workflow to represent steps in the process.

You may want to provide a meaningful name for the action as you add it. The name appears in the visual representation to distinguish the actions and to provide meaning to the workflow.

**Note:** An error message may appear when you attempt to add an action. The type of branch and workflow determines the type of action that you can add. For more information, see [“Selecting Workflow and Branch Types” on page 68](#).

**Tip:** When you add an action, the Process Action Type Properties dialog box for the action immediately opens. You can specify values for the properties as you add the action or at any time after you add the action. If you are adding a User action, you must specify the input variable or the form to use with the action before closing the dialog box. For more information, see [“Configuring User actions to use forms and documents” on page 55](#).

You can delete an action at any time after you add it. When you delete an action, any routes associated with the action are also deleted.

For information about the different types of actions that you can add, see [“Component types” on page 24](#).

#### ► To add an action:

1. Drag the action that you want to add from the Components palette to the workflow.
2. (Optional) In the Process Action Type Properties dialog box, specify values for the action properties.
3. Click **Save**.

#### ► To delete an action:

- Right-click the action you want to delete and click **Delete Action**.

## Component types

The following table describes the components that are provided with LiveCycle Workflow.

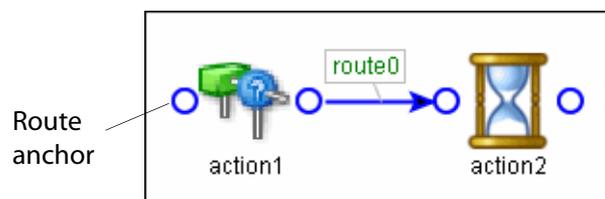
If you do not see some or all of these components in the Components palette, you must deploy them. Your administrator may have also deployed some custom components that do not appear in this table. For more information about deploying components, see [“Adding components to the Components palette” on page 74](#).

Icon	Name	Description	For more information
	Chained Process	Enables the current workflow to invoke another workflow.	<a href="#">“Linking processes with Chained Process actions” on page 41</a>
	Decision Point	Determines the next action to execute when more than one action is available.	<a href="#">“Decision Point actions” on page 36</a>
	Script	Runs a specified script.	<a href="#">“Executing script with Script actions” on page 38</a>
	Set Value	Sets the value of one or more variables.	<a href="#">“Setting variable values at run time with Set Value actions” on page 32</a>
	Stall	Stops the progress of a process instance and allows administrators to restart the process instance.	<a href="#">“Catching situational errors with the Stall action” on page 72</a>
	User	Assigns workflow tasks to users so that they can receive them in email messages or open them in LiveCycle Form Manager as work items.	<a href="#">“Assigning tasks to users” on page 57</a>
	Wait Point	Delays the execution of an action for a specified period of time.	<a href="#">“Controlling execution timing with Wait Point actions” on page 41</a>

**Note:** The icons that represent actions can be customized. The icon that appears in the table may not match the icon that you see in the Components palette.

## Adding and deleting routes

Add routes between actions to specify the order in which actions are executed. Routes originate and terminate at the route anchors of actions:



Routes begin at the route anchor on the right of the action. Routes terminate at the anchor on the left of the action. After you add a route, you can modify the route label. For information, see [“Modifying route](#)

labels” on page 25. You can also add rules that determine whether the route is followed. For information, see “About rules” on page 34.

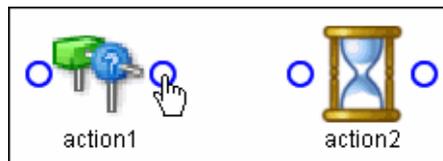
**Tip:** You can draw routes that loop back to the same action where they originated.

To draw a route between actions, both actions must already exist in the workflow.

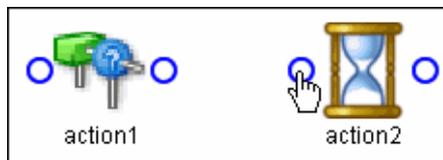
For more information about routes, see “About routes” on page 21.

► **To add a route:**

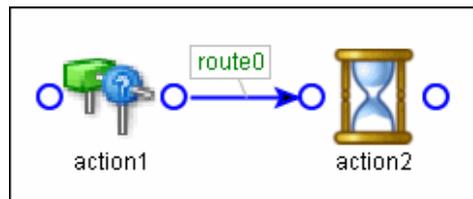
1. Pause the pointer over the route anchor of an action so that the pointer changes to a hand icon.  The handle that you use to begin drawing the route determines the direction of the route and the order in which the actions are executed:
  - If you want the route to originate at the action, pause the pointer over the anchor on the right of the action. For example, in the following graphic the route will originate at action1, which is to execute before action2.



- If you want the route to terminate at the action, pause the pointer over the anchor on the left of the action. For example, in the following graphic, the route will terminate at action2, which is to execute after action1.



2. Click and drag the pointer to the other action so that the route appears and is connecting the actions.



## Modifying route labels

Change the text of route labels to make the visual representation of the process more meaningful. When you add a route, LiveCycle Workflow Designer uses default text for the route.

► **To change a route label:**

1. Right-click the route and click **Route Properties**.
2. In the **Rule Name** box, delete the default name and type a new name for the route.
3. Click **Save**.

## Adding text to the visual representation

Add text to the visual representation to convey information about the workflow. For example, to easily identify the workflow when it is opened in the workspace, you can add a title to the visual representation.

When you add text, you can also specify the font properties. You can modify the text properties or delete the text at any time after you add the text.

➤ **To add a text label:**

- From the Tools area of the Components palette, drag the **Text** tool to the workflow.

➤ **To modify a text label:**

1. Right-click the text label and click **Text Properties**.
2. In the **Text** box, type the text you want to appear on the visual representation.
3. In the **Font** menu, select the font you want to use for the label. The fonts that are available depend on which fonts are installed on your computer.
4. In the **Size** menu, select the size of the text.

➤ **To delete a text label:**

- Right-click the text label you want to delete and click **Delete Text**.

## Printing workflows

You can print a copy of the visual representation of your workflow.

➤ **To print a workflow:**

- Click **File > Print**, and use the dialog box to specify the print properties and send the job to the printer.

# 3

## Implementing Business Logic

---

Business logic enables decisions to be made during the course of a process. You add business logic to your workflows to enable LiveCycle Workflow Server to automatically make decisions and execute actions based on the circumstances of the process instance:

- Create variables to store and retrieve critical information that is used to make decisions. See [“Saving information in variables” on page 27](#).
- Determine the next step in the process given the choice of several steps. See [“Determining the order of execution” on page 33](#).
- Execute script at any point in the workflow. See [“Executing script with Script actions” on page 38](#).
- Control the timing of when a step in the process is executed. See [“Controlling execution timing with Wait Point actions” on page 41](#).
- Link processes so that one process can start another process. See [“Linking processes with Chained Process actions” on page 41](#).

### Saving information in variables

Variables enable workflows to save and retrieve critical data at run time. You can create different types of variables for storing different types of data. You can also create variable collections for storing multiple instances of related, same-typed data.

Typically, you use a variable or a collection of variables when you need to make a decision based on the value that it holds, or to store information that you need later in a process.

For example, the value in a form field may be important for making several routing decisions in the workflow. You can save the value of the form field in a variable to easily access the value in routing rules.

You also use variables to save and retrieve data used to populate forms for use with User actions. For more information, see [“Using forms and documents in workflows” on page 47](#).

You can create variables, set variable values, and access variable values in rules and expressions. You can set the value of variables in two ways:

- Manually, so that the variable has a default value for each process instance. Only some variable types can have their values set manually. For information, see [“Creating variables” on page 28](#).
- Automatically, during the workflow, so that the value of the variable can be changed during the course of a process instance. For information, see [“Setting variable values at run time with Set Value actions” on page 32](#).

When you change the value of a variable automatically, the change affects only the process instance in which the change occurs.

## Variables as input and output parameters

You can specify that variables are used as input parameters when processes are initiated by using web services or the LiveCycle Workflow API. Similarly, variables can be used as parameter values that can be retrieved in the web service or API output when the process is complete. You can also make it mandatory to include a variable value in the call that initiates the process.

The following table describes the variable attributes that determine how variables are used with web services and in API calls.

Attribute	Description
in	Specifies whether the variable can be used as an input parameter in a web service or API call that initiates a process.
out	Specifies whether the variable value is available in the web service response message or returned Java object when the process is complete.
Required	Specifies whether providing the value of this variable is mandatory when initiating the process with a web service or API call.

These attribute descriptions apply to variables of any type.

For information about how to use these attributes for form and document type variables, see [“Using forms and documents in workflows” on page 47](#).

For information about how to use these attributes for variables used when passing data to chained processes, see [“Linking processes with Chained Process actions” on page 41](#).

For information about using web services and the LiveCycle Workflow API, you need to obtain the LiveCycle Workflow SDK. For information, go to <http://partners.adobe.com/public/developer/livecycle/workflow/devcenter.html>.

## Creating variables

Create a variable to save data that can be referenced in the workflow at run time. When you create a variable, you specify the variable name and data type. You can also specify options for use when invoking the workflow by using web services or the LiveCycle Workflow API.

Variable names have the following constraints:

- Variable names must be valid XML element names.
- Do not use database reserved words as variable names.
- Variable names use only lower-case letters. If you use upper-case letters in a variable name, LiveCycle Workflow Designer changes them to lower-case letters after you create the variable.

**Note:** You cannot change the variable name and data type after you create the variable. You can delete variables after you create them.

► **To create a variable:**

1. On the Variables palette, click the button that has the plus sign. 
2. In the **Name** box, type a name for the variable.
3. From the **Type** list, select the data type for the variable. For more information, see ["About variable types" on page 29](#).
4. If you are creating a `list` or `map` variable, in the second menu select the data type that you want to store in the list or map.
5. Select **In** to achieve either of the following:
  - To specify that the variable value can be used when the process is invoked by using web services or the LiveCycle Workflow API.
  - For variables of type Form or Document, to specify that the variable is populated by data from the form that was submitted in LiveCycle Form Manager to initiate the process. Document or Form Variables are populated depending on what value the Init-Form has for Variable Type Out.
6. To specify that the variable is a required value when invoking by using web services and the LiveCycle Workflow API, select **Required**.
7. To specify that the variable value is available in web service response messages and returned LiveCycle Workflow API objects, select **Out**.
8. Click **OK**.

## About variable types

You can create the following types of workflow variables. The table provides the format that you must use to express the literal values of the variables in XPath expressions.

**Note:** The properties of the `int`, `boolean`, `long`, `double`, `decimal`, `short`, and `float` types are the same as the properties of the Java primitive data types of the same name.

Variable type	Description	Variable literal value format in XPath expressions
<code>boolean</code>	Holds a Boolean value of <code>true</code> or <code>false</code>	The functions <code>true()</code> or <code>false()</code>
<code>date</code>	Holds a date value For more information, see <a href="#">"Date and time parameters" on page 122</a> .	<code>"yyyy-mm-dd"</code> For example, <code>"2006-05-24"</code>
<code>dateTime</code>	Holds a <code>dateTime</code> value For more information, see <a href="#">"Date and time parameters" on page 122</a> .	<code>"yyyy-mm-ddThh:mm:ssZ"</code> For example, <code>"2006-03-02T14:37:00Z"</code>

Variable type	Description	Variable literal value format in XPath expressions
decimal	<p>Holds a decimal value</p> <p>In the database, values are represented by an unscaled integer value and a scale that determines the location of the decimal.</p> <p>The <code>scale</code> attribute indicates the number of digits in the integer that are on the right side of the decimal. The default scale is 3.</p> <p>For example, an integer value of 12345 and a scale of 3 represents the decimal value of 12.345.</p> <p>The <code>length</code> attribute determines the maximum number of digits that can be stored. The default length is 10.</p> <p>You can change the values of the <code>scale</code> and <code>length</code> attributes on the Variables palette.</p>	<p>A decimal number</p> <p>For example, <code>123.4</code></p>
document	<p>Holds a <code>com.adobe.idp.Document</code> object, which is a deserialized version of a PDF document. The variable can hold a reference to a file on disk if it is too large to be contained in memory.</p> <p>For more information about using documents in workflows, see <a href="#">“About forms and documents in User actions” on page 54.</a></p>	<p>You cannot express the literal value of a document in XPath expressions.</p>
double	<p>Holds a double-precision floating point value (64-bit IEEE 754)</p>	<p>Any number between the values <code>4.94065645841246544e-324d</code> and <code>1.79769313486231570e+308d</code> (positive or negative)</p> <p>However, you cannot express double values using scientific notation, which XPath does not support.</p>
float	<p>Holds a single-precision floating point value (32-bit IEEE 754)</p>	<p>Any number between the values <code>1.40129846432481707e-45</code> and <code>3.40282346638528860e+38</code> (positive or negative)</p> <p>However, you cannot express float values using scientific notation, which XPath does not support.</p>

Variable type	Description	Variable literal value format in XPath expressions
form	<p>A complex data type that holds the following information:</p> <ul style="list-style-type: none"> <li>• Form data (XDP data)</li> <li>• Selected action</li> <li>• Action choices</li> <li>• Design-time URL to the form template</li> <li>• Run-time URL to the form template</li> </ul> <p>For more information about form data, see <a href="#">“Storing form data in form variables” on page 49</a> and <a href="#">“About XML form data in the process schema” on page 51</a>.</p>	You cannot express the literal value of a form in XPath expressions.
int	Holds an integer value (32-bit two's complement)	Any number between -2147483648 and 2147483647
list	<p>Holds an ordered collection of same-typed workflow variables</p> <p>Items in list variables are indexed. In XPath expressions, the index is one-based so that the first item in the list has an index of one. (See <a href="#">“Accessing data in collections using XPath expressions” on page 45</a>.)</p>	Determined by the type of workflow variable stored
long	Holds a long integer value (4-bit two's complement)	Any number between -9223372036854775808 and +9223372036854775807
map	<p>Holds a keyed collection of same-typed workflow variables</p> <p>Items in map variables are key-based. Specific data items can be accessed using XPath expressions that use the key associated with the data item. (See <a href="#">“Accessing data in collections using XPath expressions” on page 45</a>.)</p> <p><b>Note:</b> The order of data items in map variables can change.</p>	Determined by the type of workflow variable stored
short	Holds a short integer value (16-bit two's complement)	Any number between -32768 and 32767
string	<p>Holds a string value</p> <p>The <code>length</code> attribute determines the maximum number of characters that the <code>string</code> variable can hold. You can change the default length of 100 on the Variables palette.</p>	Any text enclosed in quotation marks

Variable type	Description	Variable literal value format in XPath expressions
xml	<p>Holds an XML document or a partial XML document</p> <p>You can use the <code>xsd</code> reference attribute to specify a schema that constrains the XML format.</p> <p>At design time, the schema enables you to see XML data represented in the process data tree of the XPathExpression Builder. (See <a href="#">“About the XPathExpression Builder” on page 42.</a>)</p>	Valid XML enclosed in quotation marks

## Setting variable values at run time with Set Value actions

Use the Set Value action to set the value of one or more variables at a step in the process.

**Note:** Variable data is persisted and available through the entire workflow. You should avoid setting the value of the same variable from different branches in a split. If branches in a split execute simultaneously and write data to the same variable, one branch may overwrite important data that the other branch has saved.

### ► To use the Set Value action:

1. From the Components palette, drag the **Set Value** action onto your workflow.
2. In the **Name** box, type a name for the action.
3. Click the icon that has the plus sign  to add a definition for setting a variable value.
4. For the definition you added, click the **Location** field and then click the ellipsis button to open the XPathExpression Builder.
5. In the **Conditions** box, navigate the conditions tree to locate the variable that you want to set, click the variable so that the expression that represents the variable appears in the expression work space, and click **OK**.
6. Click the corresponding **Expression** field and then click the ellipsis button to open the XPathExpression Builder.
7. Use the XPathExpression Builder to build the expression that defines the value for the variable, and click **OK**.

For information about building expressions, see [“Expressions” on page 42.](#)

**Note:** If you are using Adobe Acrobat® forms and are setting the value of a field in a form variable, the expression must use a fully-qualified path to identify the field. Also, the path must include the node name `fields` as the form schema root node. For more information, see [“Storing form data in form variables” on page 49](#) and [“About XML form data in the process schema” on page 51.](#)

8. In the Set Value action properties, click **Save**.

## Determining the order of execution

Routes determine the sequence in which actions can be executed. Although routes establish many possible sequences of actions, you may need to integrate business logic so that LiveCycle Workflow Server can determine the correct sequence of actions depending on the circumstances of the process instance.

Depending on the complexity of a business process, there can be many ways that an instance of the process can progress. For example, for an internal purchase order requisition, the cost of a purchase may determine which approval process is required.

To determine the order in which actions are executed during a process instance, you need to perform one or more of the following tasks, depending on the requirements of your workflow:

- Specify which action is the first action that is executed when a process is initiated. You must specify the first action. For information, see [“Specifying the first action” on page 33](#).
- Provide rules to determine which action of many to execute. For information, see [“Executing one of many actions” on page 33](#).
- Execute branches in parallel so that actions can be executed simultaneously. For information, see [“Executing branches in parallel with splits” on page 36](#).

### Specifying the first action

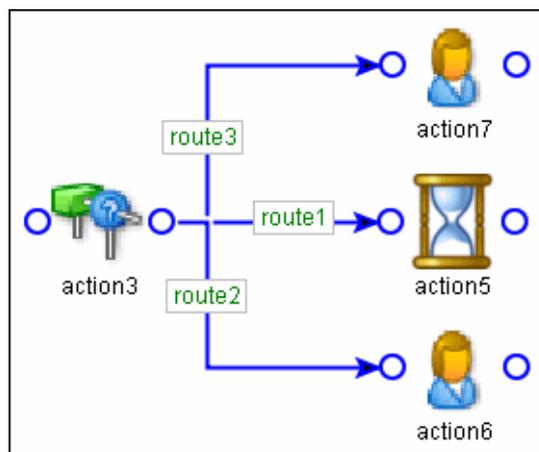
For all workflows, you must specify the action that is executed first when the process is initiated. On the workflow, an arrow on the action icon indicates that it is the start action.

► **To specify the start action:**

- Right-click the action that you want to specify as the start action, and click **Set Start Action**.

### Executing one of many actions

Often, processes can proceed following one of several routes. The route that is followed depends on the circumstances of the process instance. On a workflow, this situation appears as several routes that originate at a single action, and each route terminates at a different action.



When several routes originate at a single action, LiveCycle Workflow Server follows the first valid route that it finds. You need to configure the routes and the actions so that LiveCycle Workflow Server can determine how to proceed:

- You use rules to determine if a route is valid. For information, see [“About rules” on page 34](#).
- You configure actions to specify the order in which the routes are evaluated. For information, see [“Prioritizing routes” on page 35](#).

**Note:** Multiple routes can originate at any type of action. However, there are situations when there is no single step in the process that precipitates the need to evaluate multiple routes. In this situation, you can use the Decision Point action as the origin of several routes. For information, see [“Decision Point actions” on page 36](#).

If you want information about executing multiple actions simultaneously, see [“Executing branches in parallel with splits” on page 36](#).

## About rules

Rules can be used to determine whether a route is valid. Each route can have one or more rules associated with it. The rule for a route is evaluated when the action at the start of the route is completed. If the rule evaluates to true, the route is valid and the next action is executed.

**Note:** If a route does not have an associated rule, the route is always valid.

Rules consist of three parts, in the following format:

```
expression1 operator expression2
```

Expressions either consist of values from the process schema or form data or are derived from data values by using functions. Operators define a relationship between two expressions. For example, if you want to follow a route only if the value in a form field is greater than 5000, the rule would be:

```
PurchaseAmt > 5000
```

For more information about expressions, see [“Expressions” on page 42](#).

### Route selection

When multiple routes originate at an action, you use rules to select which route to follow. You may want to use one route as the default route that is followed when no other routes are valid:

- If no routes are found to be valid, the process instance is completed.
- The default route has no rule associated with it, so it is always valid.
- The default route is evaluated last. See [“Prioritizing routes” on page 35](#).

### Using multiple rules for a route

A route can have more than one rule associated with it. For multiple rules, you can specify that the route is valid when all of the rules evaluate to true or when only one or more of the rules evaluates to true.

The join condition (AND or OR) of the rules determines how the rules are evaluated:

- The AND condition causes the route to be valid when all the rules evaluate to true.
- The OR condition causes the route to be valid when one or more of the rules evaluates to true.

You cannot specify the join condition until multiple rules are created for a route. The default join condition is AND.

## Adding and modifying rules

Create rules to specify conditions that must be met for a rule to be valid. When you create a rule, you associate it with a route. When a route has a rule associated with it, it appears as a dotted line on the workflow.

To add a rule, the route must already exist on the workflow. You can modify routing rules any time after you create them. You can also modify rules after you have deployed the workflow.

### ► To add a rule:

1. Right-click the route to which you want to associate the rule, and click **Route Properties**.
2. In the first **Expression** box, add an expression.  
For information about creating expressions, see [“Expressions” on page 42](#).
3. From the **Operator** list, select an operator to define the relationship between the two expressions.
4. In the second **Expression** box, add an expression.
5. Click **Add** to add the rule.
6. Repeat steps 2 to 5 for each rule you want to add to the route.
7. To specify the join condition, at the bottom of the dialog box, select either **Use OR join for conditions** or **Use AND join for conditions**.

### ► To modify a rule:

1. Right-click the route and click **Route Properties**.
2. Change the rule expressions and operators as required.
3. Click **Save**.

## Prioritizing routes

When several routes originate from a single action, you need to specify the order in which routes are evaluated to prioritize the routes. LiveCycle Workflow Server follows the first valid route that it encounters. Prioritize the routes when multiple routes can potentially evaluate to true, but one route should be taken instead of the others.

Setting the evaluation order is similar to using `IF . . . ELSE` statements in script. If the first route is valid, that route is followed. Otherwise, the next route is evaluated.

**Note:** You cannot prioritize routes that originate at Split actions. To prioritize routes following a Split, place a Decision Point action after the split. Use the Decision Point as the origin of the multiple routes and configure the Rule Evaluation Order property of the Decision Point.

### ► To prioritize routes:

1. Right-click the action from which the routes originate and click **Rule Evaluation Order**.  
The routes appear in a list in order of priority, from first to last. The route that is evaluated first appears at the top of the list.
2. To change the priority of a route, click the route and use the arrow buttons to move the route in the list.

## Decision Point actions

A Decision Point action represents a point in the process where a decision is made that affects how the process progresses.

Use a Decision Point action when you need multiple routes to originate at an action but there is no single step in the process that precipitates the need to evaluate multiple routes. The Decision Point action acts as a node in the workflow that serves as the origin of many routes, but has no executable function itself.

Several situations require the use of the Decision Point action:

- The LiveCycle Workflow Server must evaluate several routes to determine the first action to execute in a workflow.

This situation occurs when a process is initiated when a person submits a form, and form data determines the first action to execute in the workflow. For example, a customer can fill an invoice dispute form through your corporate web site. The dollar amount of the invoice determines whether the form is routed to a first-level manager for approval or to a credit representative for processing.

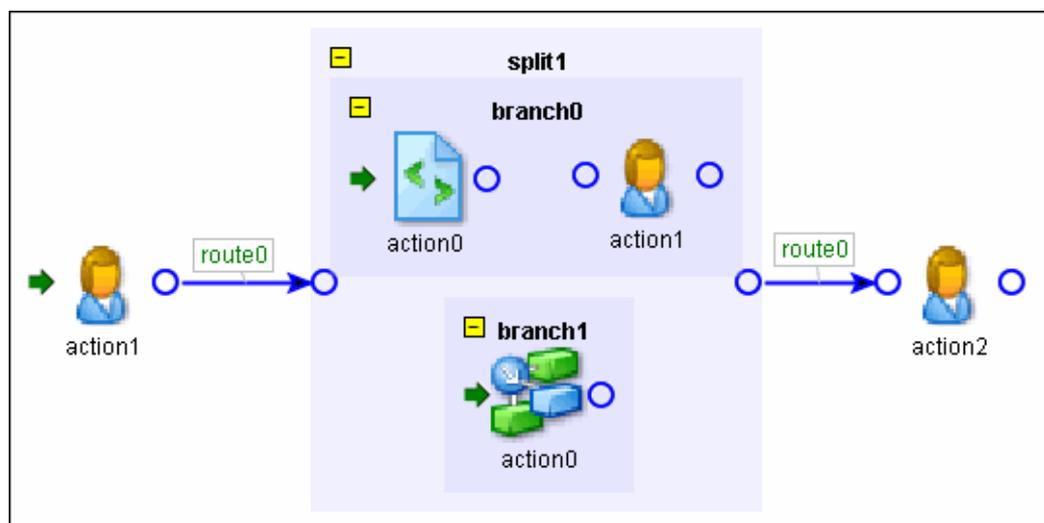
- Several different routes in a process converge at a point where a set of rules are evaluated.

This situation can occur when the process loops to a step where a set of rules are reevaluated. For example, in a quality assurance process, an issue may have to go through a retesting process until it is fixed and the process can proceed.

This situation can also occur if several branches converge after running in parallel. For example, in a process for hiring new employees, when an applicant is hired, several subprocesses are initiated as part of the hiring process. When each of the subprocesses completes, multiple rules based on the data of each subprocess is evaluated to determine the next step.

## Executing branches in parallel with splits

You add a split to your workflow when you need to execute actions in separate branches simultaneously. After you add a split, you can add one or more branches to it. All branches in a split are executed simultaneously. Including a branch in a split is similar to including a different subprocess in your workflow.



When you add a split, the split contains one branch and no actions. You can add actions and additional branches as required.

**Note:** You cannot add splits to transactional branches.

## Convergence after splits

After a split, the main workflow can either wait for one or more branches in the split to complete before proceeding, or it can proceed immediately after initiating the branches. The behavior of the main workflow is determined by the join operator that you specify in the split properties. The join operator can have a value of AND-WAIT, OR-WAIT, or NO-WAIT:

- A value of AND-WAIT causes the main workflow to proceed after all of the branches in the split are complete.
- A value of OR-WAIT causes the main workflow to proceed after only one of the branches in the split are complete.
- A value of NO-WAIT causes the main workflow to proceed immediately after the branches in the split are executed. The main workflow does not wait for the branches to complete.

## Adding splits

Add a split to your workflow when you want to execute branches in parallel. For more information about splits, see [“Executing branches in parallel with splits” on page 36](#).

### ► To add a split:

1. From the Components palette, drag the **Split** action to the workflow work space.
2. Right-click the split and click **Split Properties**.
3. To distinguish the split from other splits in the workflow, in the **Name** box, type a name for the split.
4. In the **Join Type** list, specify the way the main workflow behaves after the split is executed, and click **OK**:
  - To specify that the main branch proceeds only after all of the split branches complete, select **AND-WAIT**.
  - To specify that the main branch should proceed after only one of the split branches completes, select **OR-WAIT**.
  - To specify that the main branch should proceed immediately after the split is executed, select **NO-WAIT**.
5. In the split, right-click the default branch named **branch0** and click **Branch Properties**.
6. In the **Name** box, type a name for the branch.
7. In the **Branch Type** list, specify the branch type and click **OK**. For information about branch types, see [“Selecting Workflow and Branch Types” on page 68](#).
8. To add actions to the split branch, drag the action from the **Components** palette to the branch part of the split.
9. To add branches to the split, drag the **Branch** tool from the **Components** palette to the split. You can configure the new branch and add actions to it as required.

## Executing script with Script actions

You can use the Script action to execute script that you provide. Java is the supported scripting language for which BeanShell 1.3.0 is used as the implementation. For information about writing Java code for BeanShell, go to [www.beanshell.org](http://www.beanshell.org).

The Script action provides a script editor where you can type your Java code. The Script action also provides tools for testing your script.

You can specify a Script action to be interactive or not interactive:

- Interactive actions require an external action to complete the action after the script is executed. For Script actions that are interactive, if your script behaves as a User action and sends a form to a user's worklist in LiveCycle Form Manager, the process continues after the user completes their work item. Or, the script can send a message to LiveCycle Workflow Server to complete the action.
- Non-Interactive Script actions are completed immediately after the script is executed.

## Workflow objects in script

The following table includes the implicit objects that are available to the Script action script.

Object	Description
patServiceContext	Provides access to the object manager, process manager, deployment properties, JNDI initial context, and JNDI application context.
patServiceResult	Used to store information gathered as a result of executing the Script action, and is used to transfer data back to LiveCycle Workflow Server.
patExecContext	Provides all context data for use during execution of the Script action.

The following example demonstrates how to use the patServiceContext object to instantiate the object manager:

```
import com.adobe.workflow.client.QLCSessionFactory;
import com.adobe.pof.omapi.POFObjectManager;
import com.adobe.pof.GenericObject;
import com.adobe.pof.odapi.*;
import com.adobe.idp.Context;
import com.adobe.pof.schema.POFAttribute;

POFObjectManager m_om = patServiceContext.getObjectManager();
GenericObject _testobject = m_om.newObject("workflow", "testobject");
testobject.setStringValue("ID", "test"+System.currentTimeMillis());
testobject.setStringValue("description", "insert thru script QPAC");
GenericObject _object = m_om.writeObject(_testobject);
```

The following script sets the value of a workflow variable named *status* to the value of “approved”. The script also prints messages for testing purposes. The messages appear in the Output tab of the Script action.

```
import com.adobe.pof.omapi.POFObjectManager;
import com.adobe.workflow.manager.ProcessManager;

POFObjectManager _om = patServiceContext.getObjectManager();
ProcessManager _pm = patServiceContext.getProcessManager();

patExecContext.setProcessDataStringValue("/process_data/status",
"approved");

String str1 =
patExecContext.getProcessDataStringValue("/process_data/status");
System.out.println("status is now " + str1);
```

For more information about coding with the implicit objects, see the documentation for the LiveCycle Workflow SDK.

## Completing the action

When a Script action is interactive, to complete the action at run time you must include script that completes the action after the script has executed.

To complete the action, you need to create an object that represents the process manager and then use the process manager to change the state of the current action to complete:

- To create an instance of the process manager, you use the `patServiceContext` variable.
- To retrieve the identification of the current action, you use the `patActionInstance` variable.
- To complete the action, you use the `completeAction` method of the process manager.

```
import com.adobe.workflow.manager.*;
import com.adobe.workflow.boi.*;
ProcessManager _pm = patServiceContext.getProcessManager();
long actionID = patExecContext.getActionID();
pm.completeAction(actionID);
```

For more information about using the API, see the LiveCycle Workflow SDK documentation.

## Testing script

The Script action includes tools for testing script without deploying the workflow. To test a script you can perform the following tasks:

- Provide process data that can be references in your script
- Run script and view the results in a console so that you can verify and debug your script

## About process data

The test tool simulates a process environment in which it runs a new process instance. You can specify properties for the process instance and test your script against those properties.

The following table describes the process properties that you can specify.

Property	Description
id	The identifier of the process type. id is of type <code>long</code> .
create_time	The date and time when the process type was created. create_time is of type <code>string</code> in the format "yyyy-mm-ddThh:mm:ssZ".
update_time	The date and time when the process type was last saved. update_time is of type <code>string</code> in the format "yyyy-mm-ddThh:mm:ssZ".
creator_id	The identifier of the user who created the process type. creator_id is of type <code>string</code> .
status	The status of the process type, which can be either active or inactive. status is of type <code>string</code> .

For more information about the format of date and time values, see ["Date and time parameters" on page 122](#).

## Running script

Run the script of a Script action when you want to test the script.

**Caution:** Do not test script in a production environment. Use the test tool in a development environment only. The test tool is convenient for immediately running script, however the script affects the database in the same way that running the workflow does.

### ► To run script:

1. Click the **Test** tab in the Script action properties.
2. (Optional) In the **Process Data** tab, type the properties that you want to use for testing.
3. Click **Run**.
4. Click the **Output** tab to see the results of the script.
5. (Optional) Click **Clear** to remove the results in the Output tab before running the script again.

## Script keyboard shortcuts

The script editor supports several keyboard shortcuts.

Command	Keyboard shortcut
Copy	Ctrl+C
Paste	Ctrl+V
Cut	Ctrl+X
Select all	Ctrl+A

## Controlling execution timing with Wait Point actions

Add a Wait Point action to delay the execution of an action. The Wait Point action completes after a specified amount of time, and then the next action in the workflow is executed.

You specify the amount of time to wait by providing values for days, hours, minutes, and seconds. The total time is the cumulation of each value.

### ► To control execution timing:

1. Add a Wait Point action to your workflow, or right-click an existing Wait Point action and click **Component Properties**.
2. In the **Days** box, type the number of days to wait before completing the action, or click the ellipsis button  to specify the number of days by using an expression.  
For more information about building expressions, see [“Expressions” on page 42](#).
3. In the **Hours** box, type the number of hours to wait, or click the ellipsis button to specify the number of hours by using an expression.
4. In the **Minutes** box, type the number of minutes to wait, or click the ellipsis button to specify the number of minutes by using an expression.
5. In the **Seconds** box, type the number of seconds to wait, or click the ellipsis button to specify the number of seconds by using an expression.
6. Click **Save**.

## Linking processes with Chained Process actions

Use a Chained Process action to initiate a different process type from within your workflow. You can specify whether the current process waits for the subprocess to complete before continuing or it continues immediately after initiating the subprocess.

You can also pass data to the subprocess and retrieve data from the subprocess after it is complete. For each variable you defined in your workflow, you can specify a corresponding variable in the subprocess to pass the data to. The variable in the subprocess must have the In option selected. You can retrieve data from each variable in the subprocess that has the Out option selected.

For information about variables, see [“Saving information in variables” on page 27](#).

### ► To link processes:

1. Add a Chained Process action to your workflow, or right-click an existing Chained Process action and click **Component Properties**.
2. In the **Process Type to Chain** list, select the process type to initiate.
3. If you want the current process to wait for the subprocess to complete before continuing, select **Wait for Completion**.
4. (Optional) To specify which data to pass to the subprocess and to receive from the subprocess, click **Data Mappings**.

5. In the Input Mappings area, for one or more variables in your workflow, select a corresponding variable from the subprocess.
6. In the Output Mappings area, for one or more output variables of the subprocess, select a corresponding variable from your workflow to store the data when the subprocess completes.
7. Click **OK**, and then click **OK** to save the changes to the action.

## Expressions

XPath expressions are used to identify data items from the process schema, data items from form schemas, and workflow variables. Expressions can also include functions and operators:

- Simple expressions contain a single data item
- Complex expressions use functions or operators on one or more data items

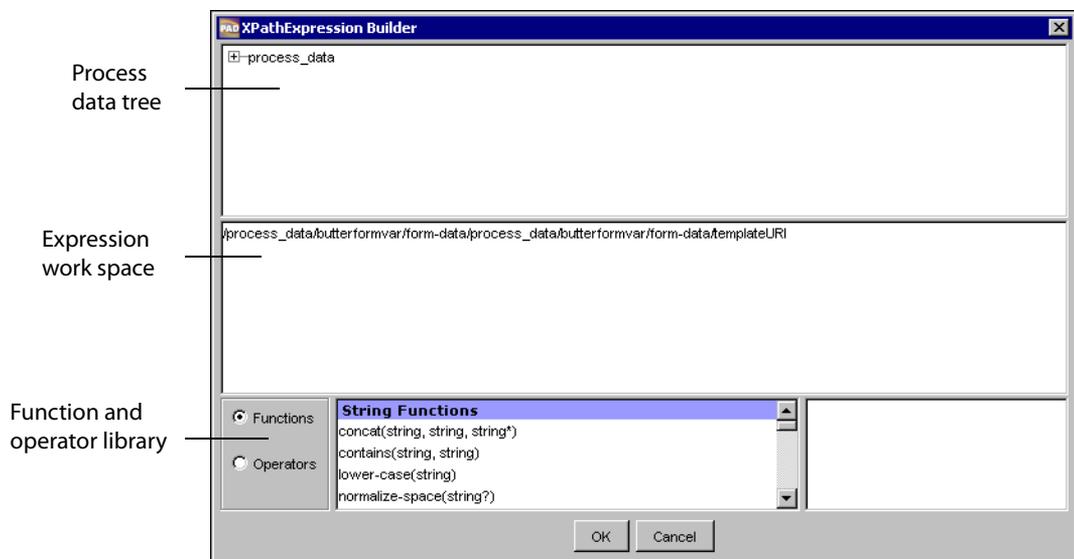
For information about making form fields available in expressions, see [“Making form fields appear in XPathExpression Builder” on page 51](#).

XPath expressions are used to specify values for properties of several workflow components, such as routing rules and Set Value actions. Expressions are written in XML Path (XPath). You can use the XPathExpression Builder to create expressions, or you can create them manually.

For information about XPath, go to [www.w3.org/TR/xpath](http://www.w3.org/TR/xpath) in your web browser. For information about XPath functions that LiveCycle Workflow supports, see [“Function Reference” on page 88](#).

## About the XPathExpression Builder

To build expressions you use the XPathExpression Builder. You can open the XPathExpression Builder from several component property dialog boxes.



The XPathExpression builder includes a data tree, an expression work space, a function library, and an operator library:

- The process data tree shows the structure of the process data in a tree structure. The data tree provides access to the information that is stored for the process, such as variable values and values in form data.
- The expression work space is a text editor where you author XPath expressions. When you click items in the data tree or in the function or operator library, the items appear in the expression work space.
- The function and operator libraries provide XPath functions and operators that you can apply to nodes in the data tree.

## Building expressions

The method you use to build an expression depends on whether you want a simple expression or a complex expression. The procedures in this topic use XPathExpression Builder to create expressions. If you know the XPath language, you can optionally create expressions manually.

### ► To build a simple expression:

1. In the XPathExpression Builder, click the expression work space where you want the expression to appear.
2. Navigate the data tree to locate the item you want to include in the expression, and click the item.

### ► To build a complex expression by using functions:

1. In the XPathExpression Builder, ensure that **Function** is selected.
2. From the function library, click the function you want to use.
3. Specify data items to use for each of the parameters of the function you added:
  - In the expression work space, select a parameter for the function.
  - From the data tree, click the data item you want to use as the parameter.

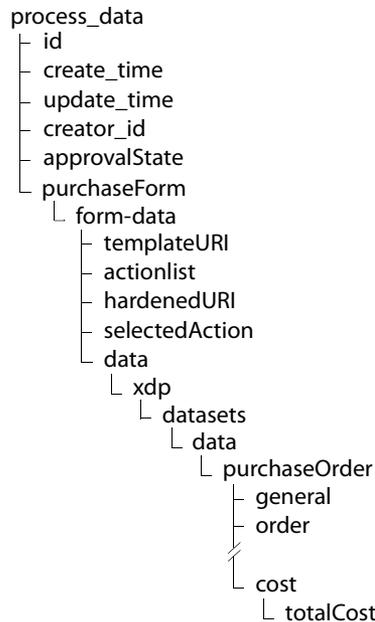
### ► To build a complex expression by using operators:

1. In the condition tree of the XPathExpression Builder, click the data item you want to appear on the left side of the operator.
2. Ensure that **Operator** is selected.
3. From the operator library, click the operator you want to use in the expression.
4. From the data tree, click the data item you want to use on the right side of the operator.

## Example XPath expressions and process data tree

This topic demonstrates how to apply XPath expressions to data from an example process data tree. The workflow implements a purchase order process. Routing decisions are made based on the amount of the purchase. A form is used to collect data about the purchase and is routed to several people to approve the purchase and place the order.

The workflow includes a string variable named *approvalState* that tracks whether the purchase request has been approved. Form data is saved in a form variable named *purchaseForm*. The root node of the form schema is named *purchaseOrder*. The process schema is illustrated in the following diagram.



For information about the nodes that represent form data, see [“About XML form data in the process schema” on page 51](#).

The nodes below the purchaseOrder node represent the form schema. The totalCost node is bound to a field on the form that stores the total cost of the requested purchase.

The business process dictates that purchases of a total cost less than \$300 do not require approval by a manager. Those purchases can be routed to an administrator to be ordered immediately. The routing rule that decides whether to route to an administrator or a manager uses an expression to access the value of the totalCost field on the form:

```
/process_data/purchaseForm/form-data/data/xdp/datasets/data/purchaseOrder/cost/totalCost
```

If the process required that the purchase amounts be specified in a different currency, you could use a Set Value action to change the values in the form fields. Changing the value of the totalCost field by an exchange rate of 0.82 would require the following for the Value Expression in the Set Value action:

```
number (/process_data/PurchaseForm/form-data/data/xdp/datasets/data/purchaseOrder/cost/totalCost) * 0.82)
```

**Note:** When extracting the date from a form field and processing it using a Date or Date-Time function in an XPath expression, the extracted date must be in the format that is valid for the function. For information about valid date and time formats, see [“Date and time parameters” on page 122](#).

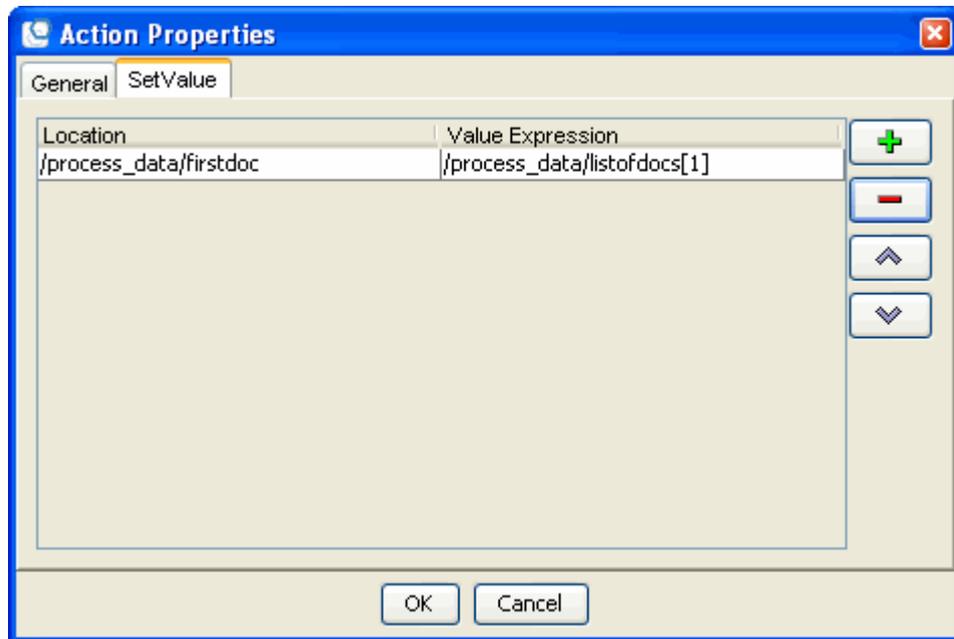
## Accessing data in collections using XPath expressions

Workflow `list` and `map` variables are referenced in XPath expressions by name in the same way that other variables are referenced. However, to access specific data items in `list` and `map` variables, you also need to indicate which data item in a `list` or `map` variable you want to set or access.

### list variables

To reference an entire `list` variable, you use the name of the `list` variable. For example, to copy all of the data from one `list` variable named `original_list` to another `list` variable named `copy_list`, the XPath expression that you use to reference `original_list` is `/process_data/original_list`.

To indicate the data item in a `list` variable, you need to specify the index position of the item. For example, a workflow stores document data in a `list` variable named `listofdocs`. You use a Set Value action to retrieve the first document in the list and save it in a `document` variable named `firstdoc`. The XPath expression that returns the first document in the list is `/process_data/listofdocs[1]`.

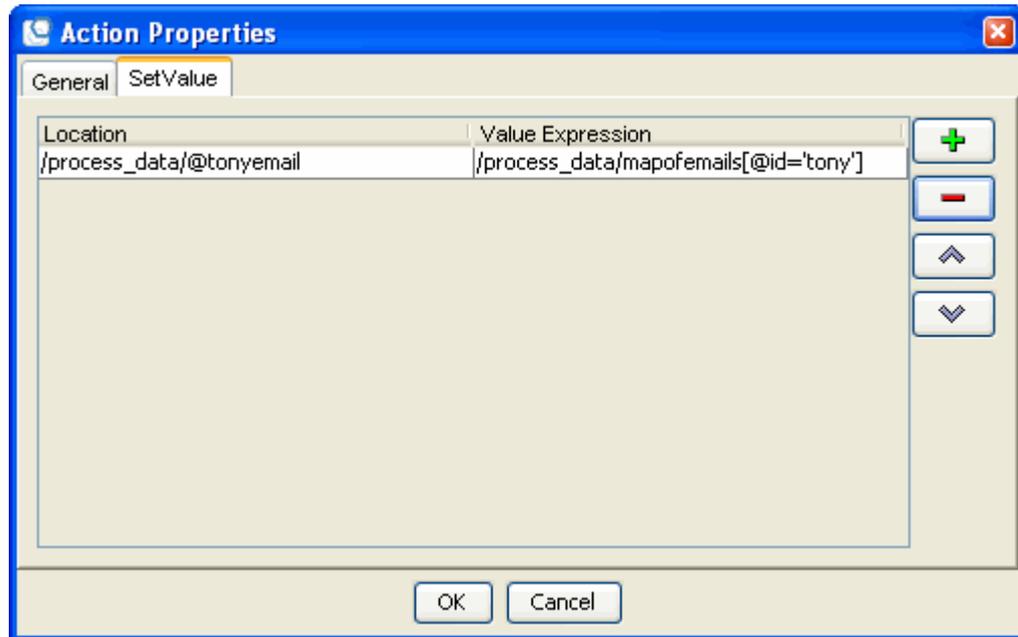


### map variables

To reference an entire `map` variable, you use the name of the `map` variable. For example, to copy all of the data from one `map` variable named `original_map` to another `map` variable named `copy_map`, the XPath expression that you use to reference `original_map` is `/process_data/original_map`.

To indicate the data item in a `map` variable, you need to specify the key for the key-value pair. For example, a workflow stores names and email addresses in a `map` variable named `varmapofemails`. Names are used as the key and the email addresses are the corresponding values. You use a Set Value action to retrieve Tony's email address and store it in a `string` variable named `tonyemail`. The XPath expression

that returns the value in the map variable corresponding to the key of `tony` is `/process_data/mapofemails[@id='tony']`.



**Note:** Unlike `list` variables, the order in which data items are stored in `map` variables can be different each time you use the `map` variable. When referencing data items in `map` variables, you cannot make any assumptions about the order of the data items.

## Using XPath expressions as list indexes and map keys

When referencing workflow `list` and `map` variables in XPath expressions, you can use other workflow variables for the list's `index` value or the map's `key` value.

For example, a workflow `list` variable named `listvar` holds a collection of customer names. An `integer` variable named `intvar` holds the index of the data item in `listvar` that you want to retrieve. The following XPath expression returns the data item from `listvar`:

```
\process_data\listvar[\process_data\@intvar]
```

A workflow `map` variable named `mapvar` holds customer addresses and uses the customer name as the key. The customer name for which you want the address is stored in a `string` variable named `stringvar`. The following XPath expression retrieves the address from `mapvar`, which has, as its key, the customer name stored in `stringvar`:

```
\process_data\mapvar[@id='\process_data\@stringvar']
```

The User action lets users interact with automated processes through LiveCycle Form Manager. You can use the User action to configure most of the aspects of human interaction:

- Specify the forms to associate with workflow tasks. See [“Using forms and documents in workflows” on page 47](#).
- Configure workflows to follow routes based on choices made when submitting forms. See [“Following routes based on user decisions” on page 55](#).
- Specify users who are assigned tasks that are generated from User actions. See [“Assigning tasks to users” on page 57](#).
- Specify time constraints for users to complete tasks. See [“Setting time constraints for users” on page 60](#).
- Provide instructions that describe what to do with tasks in LiveCycle Form Manager. See [“Providing instructions” on page 64](#).
- Configure how task attachments are handled in the workflow. See [“Configuring attachments” on page 64](#).

### Using forms and documents in workflows

LiveCycle Workflow lets people participate in automated processes through the use of forms. Users participate with forms in several ways, such as submitting forms from LiveCycle Form Manager to initiate processes and filling forms that they receive as participants in processes.

Also, LiveCycle Workflow Server can send forms to users in email to work on while offline. Users can email forms back to LiveCycle Workflow Server to complete tasks.

- Before you configure your workflows to use forms and User actions, you should upload the forms to the LiveCycle Form Manager repository. Uploading forms lets you browse for their location when specifying their URL. For information about uploading forms, see *LiveCycle Form Manager Administration Help*.
- Note:** To ensure that workflows function properly, when you upload a form, you need to include it in a form category.
- To enable processes to be initiated when forms are submitted to LiveCycle Workflow Server, you need to identify the form in the Init-Form palette. See [“About initiating processes with forms” on page 48](#).
  - Typically, form data is passed from person to person in a process. To pass form data from User action to User action in a workflow, you need to create form variables that store the form data. See [“Storing form data in form variables” on page 49](#).
  - Some situations require that forms and form data are saved as PDF documents before they are passed from person to person in a process. To use PDF documents, you use document variables. See [“Referencing documents in document variables” on page 53](#).
  - You need to configure User actions to specify the document or the form data to associate with tasks. See [“About forms and documents in User actions” on page 54](#) and [“Configuring User actions to use forms and documents” on page 55](#).

## About initiating processes with forms

Any process can be initiated when a form is submitted to LiveCycle Workflow Server. For example, a person can open, fill, and submit a form in LiveCycle Form Manager. When the form is submitted, the process that is associated with the form is initiated.

To enable processes to be initiated by a form, you need to identify the form in the workflow using the Init-Form palette. For information, see [“Identifying forms that initiate processes” on page 48](#). Before you use the Init-Form palette, you need to store the form in the LiveCycle Form Manager repository and add the form to a category. You also need to ensure that the form is configured properly. For more information, see [“Designing Forms for LiveCycle Workflow” on page 81](#).

To store the data that was entered on the initiation form, you need to create a form or document variable that has the in attribute selected. See [“Storing form data in form variables” on page 49](#) and [“Referencing documents in document variables” on page 53](#).

The event of submitting a form to initiate a process does not appear on the visual representation of the workflow. Submitting an initiation form causes LiveCycle Workflow Server to execute the start action of the workflow.

If you want to use the form data to determine which action to execute, you need to use a Decision Point action as the start action. For more information about the Decision Point action, see [“Decision Point actions” on page 36](#).

**Note:** LiveCycle Workflow does not support the initiation of processes using forms through web services.

## Identifying forms that initiate processes

Identify initiation forms in your workflow so that when the form is submitted in LiveCycle Form Manager, LiveCycle Workflow Server initiates the process. When you identify the initiation form, you need to specify whether the form should be subsequently handled as a form or as a PDF document.

**Note:** Many forms can initiate the same process type. However, a form can be used to initiate only one process type. If you deploy a workflow that uses an initiation form that is already used for a different process type, an error message appears to notify you and you must correct the situation.

### ► To identify an initiation form:

1. In the Init-Form palette, click the plus sign  to open the Init-Form dialog box.
2. In the **Name** box, type a descriptive name for the Init form. Use a different name for each init-form that you create for a workflow.
3. Specify the URL that describes the location of the form using one of the following methods:
  - In the **Template-URL** box, type the URL. The URL is comprised of the directory names and the file name as they are identified in LiveCycle Form Manager

**Note:** Use the proper case when you specify the URL. If you do not use the directory and file name exactly as they are identified in LiveCycle Form Manager, the form cannot be retrieved when the process is initiated, causing the action instance to stall.

- Click **Browse** and select the template from its location in the LiveCycle Form Manager repository.

4. (Optional) In the **Choice-list** box, type the submit choices that you want to appear on the form. Separate the choices with a comma. The choices appear in a pop-up menu on the initiation form, next to the Submit button.

**Note:** Do not include a blank choice. If users open the initiation form in LiveCycle Form Manager and select a blank choice from the list, they will not be able to submit the form.

5. In the **Data Type Out** menu, select either **Form** or **Document**:
  - Select **Form** if the first User action in the workflow uses an XML form.
  - Select **Document** if the first User action in the workflow uses a PDF document that is to be unchanged after it was submitted, or if you use an Acrobat form as an initiation form.

**Note:** You must create a variable of the type you specified that has the in option selected.

6. Click **OK**.

The details about the initiation form appear in the Init-Form palette. The details include an attribute named schema rootnode, which shows the name of root node of the form schema for information purposes only. The value is taken from the form that is specified by the form URL.

## Storing form data in form variables

Variables of type form are primarily used to store data from forms that are submitted to LiveCycle Workflow Server so that the data can be accessed at run time. Form variables are typically used in the following situations:

- To store form data submitted from initiation forms.
- To provide form data for a task that a User action assigns to a LiveCycle Form Manager user.
- To store form data that a user submitted to complete a task.
- To store the location of the form that was used to collect the data.

Like any other variable types, when you create a form variable, it appears in the process schema in XPathExpression Builder.

If you save the form schema with the form and specify the URL of the form in the variable attributes, the form schema is available in expressions. For more information about saving the form schema with the form, see [“Making form fields appear in XPathExpression Builder” on page 51](#).

Form variables handle only form data but not the form itself. When using form variables, the variable data is extracted or merged with a form template when forms are submitted or sent to users. If you need to store the form data and the form together in a PDF file so that it remains unchanged as it is referenced, use a variable of type document. For information, see [“Referencing documents in document variables” on page 53](#).

The following table describes the attributes of form variables and how they are typically used.

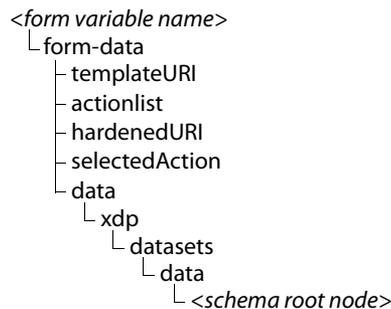
Attribute	Description
in	<p>When this option is selected, the variable holds the data from the initiation form that was used to initiate the process. The form data in this variable can then be used as an input variable for a User action.</p> <p>Form data copied from the initiate form includes the following information:</p> <ul style="list-style-type: none"> <li>• The form URL.</li> <li>• The choice list that appears on the form for submitting the form.</li> <li>• The choice that was selected when the initiation form was submitted.</li> <li>• The form data that was collected in the form fields.</li> </ul> <p>For more information about form data, see <a href="#">“About XML form data in the process schema” on page 51</a>.</p>
out	<p>Specifies whether the data stored in this variable is made available at the completion of the process. This option is useful in workflows that are initiated as subprocesses by using the Chained Process action. Form variables with this option selected enable form data to be passed back to the parent process on completion.</p>
Required	<p>Specifies whether providing the value of this variable is mandatory when initiating the process.</p>
form-url	<p>Stores the URL of the form used to collect the form data that the variable holds:</p> <ul style="list-style-type: none"> <li>• At design time, LiveCycle Workflow Designer uses this URL to reference the form and populate the XPathExpression Builder with the form schema. The URL is comprised of the directory names and the file name as they are identified in LiveCycle Form Manager.</li> </ul> <p><b>Note:</b> Use the proper case when you specify the URL. If you do not use the directory and file name exactly as they are identified in LiveCycle Form Manager, LiveCycle Workflow Designer cannot locate the form.</p> <ul style="list-style-type: none"> <li>• At run time, the value of this attribute is overwritten with the URL of the form that was used to collect the form data.</li> </ul>
data	<p>Holds form data in XML format according to the form schema:</p> <ul style="list-style-type: none"> <li>• Stores the form data from the initiation form if the form variable has the in option selected.</li> <li>• Stores form data from a submitted form if the form variable is used as output data for the User action.</li> <li>• The stored form data populates the form of a task if the form variable is used as the input data for the User action.</li> </ul> <p>You can specify form data in this attribute to populate the form for a task.</p>
schema rootnode	<p>Shows the root node of the form schema. The value of this attribute is filled automatically when the form URL is provided. You do not need to provide a value for schema rootnode.</p> <p>If no form URL is provided, the default value is <code>fields</code>.</p>

**Note:** The form data of each task that is generated for a User action is stored separately in the database. Storing form data in form variables for use at run time, or manipulating form data in variables, does not alter the information in the database.

## About XML form data in the process schema

A typical task that expressions accomplish is retrieving and setting data that is stored in form variables.

To access the information in form variables, you need to understand how the data in form variables is represented internally. The following graphic illustrates how XML form data is represented internally as a tree structure.



The following table describes each node in the tree.

Node	Description
templateURI	The location of the form template that was entered in the form-url attribute of the form variable at design time.
actionlist	Stores the choice list that was provided on the form at run time.
hardenedURI	The location of the form template that was used to collect the form data at run time.
selectedAction	Stores the choice that was selected when the form was submitted to complete the task.
data/xdp/datasets/data	The root node of the data that was collected on the form. The structure of the data below this node represents the form schema and depends on the form design.

To see the data structure for an XML form in the XPathExpression Builder, you need to perform the following tasks:

- Embed the form schema with the form and deploy the form to the LiveCycle Form Manager repository. See [“Making form fields appear in XPathExpression Builder” on page 51](#).
- Define a variable of type form and specify the URL of the associated form. See [“Storing form data in form variables” on page 49](#).

## Making form fields appear in XPathExpression Builder

You can make the fields in XML forms appear in the data tree of the XPathExpression Builder. To make the fields appear, you need to expose to LiveCycle Workflow Designer the data structure that the form uses. To expose the data structure, you can embed the XML schema in the form or include a data file in the form variable.

Fields in Acrobat forms are not available in XPathExpression Builder. You can reference the fields in expressions, but you need to create the XPATH expression manually, and you need to understand the data model of the form.

### Embedding the XML schema

If you have the XML schema that describes the data that the form collects, you can embed the schema in the form to make the data structure appear in XPathExpression Builder. To make the fields in XML forms appear, you need to ensure the following tasks have been performed:

- In LiveCycle Designer, with the form open, you connect to a schema data source and select the option for embedding the schema in the form. For more information, see *LiveCycle Designer Help*.
- For the form variable that is associated with the form, for the template-url attribute, you need to indicate the location of the form in the LiveCycle Form Manager repository. For more information about form variables, see [“Storing form data in form variables” on page 49](#).

### Including a data file

You can create a data file in XDP format from the XML form and include the data in the associated form variable so that the data structure appears in the XPathExpression Builder. To make the fields in XML forms appear, you need to ensure the following tasks have been performed.

#### ► To see form data using a data file:

1. In LiveCycle Designer, save the XML form as a dynamic PDF file. For more information, see *LiveCycle Designer Help*.
2. Open the PDF file in Adobe Acrobat® Professional and export the form data as an XDP file. For more information, see *Acrobat Help*.
3. In LiveCycle Workflow Designer, add the XDP file to the data attribute of the form variable that is associated with the form. For more information, see [“Storing form data in form variables” on page 49](#).
4. For the form variable that is associated with the form, for the template-url attribute, you need to indicate the location of the form in the LiveCycle Form Manager repository.

For information about XPathExpression Builder, see [“Expressions” on page 42](#).

## Adding data nodes to form variables

You can add nodes to the data tree of form variables and use them to store information.

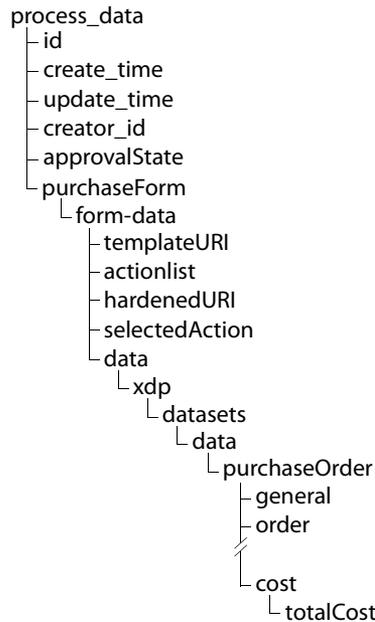
You add nodes using XPath expressions. When you use an expression that searches for a node that does not exist, LiveCycle Workflow Server creates the node when it evaluates the expression. After the node is created, you can use it as you would any other node in the tree.

The following rules apply when adding nodes:

- You can add nodes only below the root node of the form schema.
- You must use the fully qualified path to the node.

When you add a node, it does not appear in the process tree in XPathExpression Builder.

For example, the following data tree is for a process that includes a form variable named *purchaseForm*. The root element of the form schema is named *purchaseOrder*.



The following expression adds a node named *newNode* to the schema:

```
/process_data/purchaseForm/form-data/data/xdp/datasets/data/purchaseOrder/newNode
```

## Referencing documents in document variables

Variables of type document are used for storing the location of saved PDF documents. Typically, these documents have been converted to PDF from form data and forms that users submit to LiveCycle Workflow Server. You use document variables when you want to handle documents without altering them by extracting or merging data, as is done with form data in form variables.

For example, a process may require a person to fill a form and digitally sign it. The process requires that the form is unchanged after it is submitted. When they submit the form in LiveCycle Form Manager, the form is saved as a PDF document and the location of the form is saved in a document variable. The next User action in the workflow generates a task that includes the signed document intact.

The way document variables are used in the workflow is very similar to the way form variables are used. The main difference is that document variables provide access to a document but form variables provide form data and the location of the form used to collect the data. For more information about form variables, see [“Storing form data in form variables” on page 49](#).

The following table describes the attributes of document variables and how they are typically used.

Attribute	Description
in	When this option is selected, the variable holds a reference to the document that was created from the initiation form that was used to initiate the process.
out	Specifies whether the document address stored in this variable is made available at the completion of the process. This option is useful in workflows that are initiated as subprocesses by using the Chained Process action. Document variables with this option selected enable documents to be passed back to the parent process on completion.
Required	Specifies whether providing the value of this variable is mandatory when initiating the process.

**Note:** Referencing documents in variables for use at run time does not alter the document.

## About forms and documents in User actions

When User actions are executed, a task is assigned to a LiveCycle Form Manager user. The input settings and the output settings of User actions determine how form data and documents are handled in the workflow.

The input settings of User actions determine whether a form and form data are included in new tasks or whether PDF documents are included. Input data can be taken from a form variable or from a document variable:

- When a form variable is the input data, you can use either the form that is specified in the variable or a different form with the data. If you specify a different form, it must be compatible with the form data.
- When a document variable is the input data, a copy of the referenced document is used in the task. You can specify the form that was used to create the document so that LiveCycle Form Manager can associate form properties with the document, such as the form name and description. If you do not specify the form, no information about the form appears in the worklist.

**Note:** When the input data is a document variable, the output for the User action must also be stored in a document variable.

When forms are submitted, the output settings of User actions determine which task data is saved in variables:

- Form data from the task can be stored in a form variable, or the form can be saved as a document and referenced with a document variable.
- The identification of the user that completed the task can be stored in a variable of type string. This is useful for assigning subsequent tasks to previous users. See [“Assigning tasks to previous participants” on page 58](#).
- The identification of the task can be stored in a variable of type long.

**Note:** If the output settings specify that the form is stored in a document variable, the form that is used as input must use a Submit button that is configured appropriately. See [“Configuring Submit buttons when handling documents” on page 83](#).

## Configuring User actions to use forms and documents

For User actions, you need to specify the input and output settings that determine the data and forms used for tasks and how the task data is saved when they are completed.

**Note:** Before you configure the User action, you need to create the required variables and deploy any required forms:

For more information about providing forms for User actions, see [“About forms and documents in User actions” on page 54.](#)

### ► To specify the input and output forms or documents for a user action:

1. Right-click the User action and click **Component Properties**.
2. Click the **Mappings** tab.
3. To specify data to associate with the task, in the **Input Variable** menu, select either a form variable or a document variable. If you do not specify a variable, the fields of the form that is associated with the task will be empty.
  - If you specified a form variable to use as input, specify the form to use with the form data:
    - To use the form URL that is defined in the form variable, ensure that **Use Form URL defined by form input variable** is selected.
    - To specify a different form to use, select **Change the form template url to**, and then click **Browse** to locate the form in the LiveCycle Form Manager repository.
  - (Optional) If you specified a document variable to use as input, to specify the form that was used to create the document, select **Change the form template url to**, and click **Browse** to locate the form in the LiveCycle Form Manager repository.
4. If you want to save the form data that was submitted to complete the task, in the **Output Variable** menu, select either a form variable or a document variable.
5. To save the identification of the user who completed the task, in the **Save task completed by user ID in** menu, select a string variable.
6. To save the identification of the completed task, in the **Save task ID in** menu, select a long variable.
7. Click **OK**.

**Note:** If the form variable you specify does not have the form URL specified or you do not specify a form URL for the **Change the form template url to** option, process instances will stall at this action.

## Following routes based on user decisions

You can use the selected choice from the list on forms to determine the route to follow in the workflow. The way the choice list is populated determines which method you can use to determine the next route.

**Note:** You cannot use this feature if you use document variables as the Input Variable or as the Output Variable of the associated User action.

### Use route names as choices

You can use the names of the routes that originate at the User action as the items in the choice list. When the user selects a choice and submits the form, the selected choice determines the route to follow. For more information, see [“Automatically using route names as the choice list” on page 57](#).

When using route names as choices, the choice list includes a blank choice that the user can select if they do not want to use any of the choices when submitting. When a user does not select a choice, routing rules and rule evaluation order determines the route to follow.

You can configure User actions so that a choice must be made before the form is submitted. Making choice selection mandatory ensures that a route is followed based on the user’s decision. You should make it mandatory to select a choice only if you are using route names as choices. For more information, see [“Making choice selection mandatory” on page 57](#).

**Note:** If the choice list is hard coded on the form, you can still make choice selection mandatory. The items in the choice list must match the route names on the workflow or errors will occur.

### Use the value of selectedAction in routing rules

The information that form variables store when forms are submitted to LiveCycle Workflow Server includes the choice that was selected when the form was submitted. The choice that was selected is stored in the selectedAction node of the process schema:

```
/process_data/<form variable name>/form-data/selectedAction
```

You can use the value of this node in routing rules to make routing decisions.

For example, two routes originate at a user action, and the form for the User action includes Accepted and Rejected in the choice list. You use the value saved in the selectedAction node to determine the route to follow:

- The route that is followed when Rejected is selected uses the following rule:

```
/process_data/<form variable name>/form-data/selectedAction = "Rejected"
```

- The route that is followed when Accepted is selected uses the following rule:

```
/process_data/<form variable name>/form-data/selectedAction = "Accepted"
```

For more information about the selectedAction node, see [“About XML form data in the process schema” on page 51](#) for more information.

### Use expressions with a custom pop-up menu

You can use Xpath expressions with form objects to provide choice lists and to store selected choices.

For example, you can place a pop-up list on the form and use it to hold a choice-list. The selected choice can populate a hidden text box on the form using client-side script:

- Use a Set Value action to populate the pop-up list with choices. The Set Value action would set the value of the node in the process schema that corresponds with the pop-up list on the form. On the workflow, the Set Value action would be placed before the User action.
- The routes that originate at the User action can use rules that reference the hidden text field that stores the selected choice.

## Automatically using route names as the choice list

Configure a User action so that the names of the routes that originate at the action are the items in the choice list that appears on the form. The choice that people select when they submit the form determines the route to the next action in the workflow.

You cannot use route names as the choice list for user actions under the following circumstances:

- The User action is the final action in a split. For these actions, the route names do not appear in the choice list.
- The User action uses a document variable as the Input Variable or the Output Variable. The choice list is saved in the form data. Form data is not accessible when saved in PDF documents.

For information about other methods you can use to determine the route to the next action based on submit options, see [“Following routes based on user decisions” on page 55](#).

► **To use route names as submit options:**

1. Right-click the User action and click **Component Properties**.
2. Click the **Routes and Attachments** tab.
3. Select **Include route names in form data** and click **Save**.

## Making choice selection mandatory

Configure a User action so that users need to select an item from the choice list to submit the form.

**Note:** You should make choice selection mandatory only if you use route names as items in the choice list.

For more information about providing choices to users when submitting forms, see [“Following routes based on user decisions” on page 55](#).

► **To make choice selection mandatory:**

1. Right-click the User action and click **Component Properties**.
2. Click the **Routes and Attachments** tab.
3. Select **User has to select a route name to complete this task** and click **OK**.

## Assigning tasks to users

The User action allows you to specify users who are assigned tasks at a step in a process. There are several ways you can assign tasks:

- Specify a specific person. For information, see [“Assigning tasks to specific people” on page 58](#).
- Specify a previous process participant. For information, see [“Assigning tasks to previous participants” on page 58](#).
- Specify an organizational group. For information, see [“Assigning tasks to groups” on page 59](#).
- Specify a person by using an expression. For information, see [“Specifying users with expressions” on page 59](#).

## Assigning tasks to specific people

Configure the User action properties to specify a user to assign the task. To select a user, you identify their LDAP user account or indicate that the process creator should be assigned the task.

This method of assigning tasks to users is not very flexible because you typically want to assign tasks to the person who plays a specific role in a process. If the person who plays that role changes you need to change the workflow accordingly. This method is useful for testing workflows in a development environment and is not typically used in production.

### Specifying users by user name

Select the user from a list of user records. The information in the user records is taken from your corporate LDAP server that is used for user authentication.

You can apply a search filter to shorten the list and make it easier to find the user account.

#### ► To specify a user by their user name:

1. Right-click the User action and click **Component Properties**.
2. On the **Initial User** tab, ensure that **Specify User** is selected, and click **Select User**.
3. To filter the list of users, in the **Filter users on** box, type part or all of the user identification, given name, last name, or email address of the person you want to find, and click **Apply Filter**.
4. Click the record of the user you want to assign tasks to, and click **OK**.
5. In the User action properties, click **OK**.

### Specifying the process creator

You can specify that tasks are assigned to the person who initiated the process without actually specifying their user name.

**Note:** When a process is initiated using a Chained Process action, the system account is used to initiate it.

#### ► To specify the process creator:

1. Right-click the User action and click **Component Properties**.
2. In the **Initial User** tab, select **Process Creator**, and click **OK**.

## Assigning tasks to previous participants

The User action lets you save the user identification of the person who completed the task that the action generated. The user identification is saved in a variable of type string. In subsequent User actions, to assign tasks to the saved user, you specify the user in an expression that resolves to the variable.

For information about specifying users in expressions, see [“Specifying users with expressions” on page 59](#).

► **To save the user identification:**

1. Create a variable of type string. For information, see [“Creating variables” on page 28](#).
2. Right-click the User action and click **Component Properties**.
3. On the **Mappings** tab, in the **Save Task Completed By User ID in** menu, select the string variable that you created.
4. Click **OK**.

## Assigning tasks to groups

You can assign tasks to groups that are defined in your corporate LDAP server. When you assign tasks to groups, LiveCycle Workflow Server selects a specific user in the group to assign the task to, depending on how you configure the User action:

- Assign tasks to a group worklist so that users can retrieve the task and manually assign it to an individual.
- Randomly select a user from the group and assign that person the task.

You can apply a filter to shorten the list and make it easier to find the user account.

If you want information about assigning tasks to a person, see [“Assigning tasks to users” on page 57](#).

► **To assign tasks to a group:**

1. Right-click the User action and click **Component Properties**.
2. On the **Initial User** tab, select **Group**, and click **Select Group**.
3. To filter the list of groups, in the **Filter users on** box, type part or all of the group name or email address, and click **Apply Filter**.
4. Click the record of the group you want to assign tasks to, and click **OK**.
5. Specify how you want LiveCycle Workflow Server to select the member of the group:
  - To specify a shared worklist, select **Add Task to Group Queue**.
  - To specify a random member, select **Select Random User in Group**.
6. In the User action properties, click **Save**.

## Specifying users with expressions

You can use expressions to specify a user to assign tasks. Expressions are useful if you want to specify a user based on information from the process instance.

For example, a workflow can use a Set Value action to extract a user name from a form that was submitted to complete a User action. A custom action, created with the LiveCycle Workflow SDK, queries the corporate LDAP to determine the manager of the user and saves that user name in a different variable. The names of both user accounts can be retrieved using expressions.

► **To specify a user with an expression:**

1. Right-click the User action and click **Component Properties**.
2. On the **Initial User** tab, select **Use Path Expression**, and click the ellipsis button.
3. Use the XPathExpression Builder dialog box to build an expression that evaluates to the user you want to assign tasks to, and click **OK**.
4. In the User action properties, click **Save**.

## Setting time constraints for users

You can set constraints on the amount of time that a person has to complete tasks. There are three ways to set time constraints:

- Escalate tasks and send them to a different user after a specified period of time.
- Set deadlines and proceed along a specified route when the deadlines occur.
- Send reminders to the user to prompt them to complete tasks.

### Escalating tasks

Assign a task to a different user if the task is not completed after a specified period of time has passed after it was initially assigned.

For example, your workflow may assign tasks to a group and assigns the tasks to a random person in the group. This distribution method can be inefficient if a person is working on one or more tasks that take an unusual amount of time to complete. You can escalate the task and send it to a different person to ensure that the work is completed in a timely manner.

To escalate, you need to perform two tasks:

- Specify the period of time after the task is initially assigned that the escalation occurs.
- Specify the user who receives the escalated task.

If a critical task must be completed by the person who was originally assigned the task, you should use a reminder to encourage completion.

For information about how to escalate tasks, see [“Configuring escalations” on page 61](#).

### Deadlines

Set a deadline on a User action if you want the process to proceed after a specified period of time. When the deadline occurs and the task is not yet completed, the process proceeds along a specified route. Deadlines guarantee that the process proceeds after a period of time has elapsed. Deadlines also enable you to control how the process proceeds.

Deadline notifications can be sent in email messages, in LiveCycle Form Manager, or both:

- Email messages are configured in the LiveCycle Workflow administration web pages. See *LiveCycle Workflow Administration Help*.
- Deadline notifications that appear in LiveCycle Form Manager are configured in User action properties. Work items that deadline appear with a red background.

For information about how set deadlines, see [“Setting deadlines” on page 62](#).

## Reminders

Configure reminders on a User action so that users are prompted to complete the task. Reminders help to ensure that tasks are completed in a timely manner by the person who is assigned the task.

Reminders can be sent in email messages, in LiveCycle Form Manager, or both:

- Email messages are configured in the LiveCycle Workflow administration web pages. See *LiveCycle Workflow Administration Help*.
- Reminders that appear in LiveCycle Form Manager are configured in User action properties. Work items with reminders appear with a blue background.

The timing for reminders is set in User action properties. For information about setting reminders, see [“Sending reminders” on page 63](#).

**Note:** Deadlines and reminders are queued sequentially for processing by LiveCycle Workflow Server; deadlines and reminders are processed only after previous ones are processed. If LiveCycle Workflow Server is overloaded with tasks, it may not be able to process deadlines and reminders when they are configured to occur. If this situation occurs, you may need to increase hardware resources to meet the requirements of your environment.

## Configuring escalations

Set the escalation period to specify when tasks are escalated. The escalation period is specified in days, hours, and minutes. You can also choose to escalate only tasks that have not yet been worked on.

You also need to specify the person who is assigned the escalated task.

For more information about setting time constraints for tasks, see [“Setting time constraints for users” on page 60](#).

### ► To configure escalation:

1. Right-click the User action and click **Component Properties**.
2. Click the **Escalation User** tab.
3. Specify the amount of time the user has to complete the task after it is initially assigned to them:
  - In the **Days** box, type the number of days.
  - In the **Hours** box, type the number of hours in addition to the days you specified.
  - In the **Minutes** box, type the number of minutes in addition to the days and hours you specified.

**Note:** Escalation does not occur if you do not specify a time period.

4. Specify whether you want to escalate all tasks or only those that have not been worked on:
  - To escalate all tasks, select **Escalate tasks which have been previously saved or reassigned**.
  - To escalate only tasks that have not been worked on or reassigned to another person, do not select **Escalate tasks which have been previously saved or reassigned**.

5. Specify the user who is assigned the escalated task:
  - For information about specifying a specific user, see [“Assigning tasks to specific people” on page 58.](#)
  - For information about specifying a group, see [“Assigning tasks to groups” on page 59.](#)
  - For information about using expressions to specify a person, see [“Specifying users with expressions” on page 59.](#)
6. Click **OK**.

## Setting deadlines

Set deadlines on a User action when you want the process to proceed when tasks are not completed within a specific period of time. You can specify a specific route to follow, as well as provide a deadline message in LiveCycle Form Manager. Work items that have deadlined appear red in LiveCycle Form Manager.

**Note:** Email messages that are sent to users when deadlines occur are configured in the LiveCycle Workflow administration web pages.

For more information about setting time constraints for tasks, see [“Setting time constraints for users” on page 60.](#)

### ► To set a deadline:

1. Right-click the User action and click **Component Properties**.
2. Click the **Deadline** tab.
3. To enable the deadline, select **Deadline occurs after**.
4. Specify the amount of time the user has to complete the task after it is initially assigned to them:
  - In the **Days** box, type the number of days.
  - In the **Hours** box, type the number of hours in addition to the days you specified.
  - In the **Minutes** box, type the number of minutes in addition to the days and hours you specified.
5. To proceed along a specific route when the deadline occurs, select **Use Deadline Action**. If you do not select this option, normal route validation occurs according to rules and rule evaluation order.
6. If you selected **Use Deadline Action**, in the associated list select the route to follow when the deadline occurs.
7. To change the task instructions that appear in LiveCycle Form Manager work items to inform the user that the deadline occurred, select **Change task instructions on deadline**.
8. If you are changing the task instructions, in the **Deadline Instructions** box type the new instructions.
9. Click **OK**.

**Tip:** The **Deadline Instructions** box can include XPath expressions.

## Sending reminders

Set reminders on a User action to prompt users to complete the associated tasks. You can send an initial reminder with the option of repeating the reminder at regular intervals. You can also change the task instructions that appear in LiveCycle Form Manager when the reminder occurs. In LiveCycle Form Manager, tasks (work items) appear blue when a reminder occurs for them.

Reminders remain configured for User actions even if they have been escalated to another user. If you are configuring both reminders and escalations, ensure that the timing for them is appropriate.

**Note:** Email messages that are sent to users when reminders occur are configured in the LiveCycle Workflow administration web pages.

For more information about setting time constraints for tasks, see [“Setting time constraints for users” on page 60](#).

► **To set a reminder:**

1. Right-click the User action and click **Component Properties**.
2. Click the **Reminders** tab.
3. To send an initial reminder, select **First Reminder After**, and specify the amount of time after the task is initially assigned that the reminder occurs:
  - In the **Days** box, type the number of days.
  - In the **Hours** box, type the number of hours in addition to the days you specified.
  - In the **Minutes** box, type the number of minutes in addition to the days and hours you specified.
4. To specify that the reminder occurs at regular intervals, select **Repeat Reminders**, and specify the interval at which the reminder is repeated:
  - In the **Days** box, type the number of days.
  - In the **Hours** box, type the number of hours in addition to the days you specified.
  - In the **Minutes** box, type the number of minutes in addition to the days and hours you specified.
5. To change the task instructions that appear in LiveCycle Form Manager work items to inform the user that the reminder occurred, select **Change task instructions on reminder**.
6. If you are changing the task instructions, in the **Reminder Instructions** box type the new instructions.
7. Click **OK**.

**Tip:** The **Reminder Instructions** box can include XPath expressions.

## Providing instructions

Specify instructions for the User action that explain to people what they need to do to complete tasks that are generated at run time. Instructions appear in the worklist in LiveCycle Form Manager.

**Note:** If you want instructions to span multiple lines, you need to end each line with the HTML `<br>` element.

For more information about how to configure the content of tasks, see [“Configuring Human Interaction” on page 47](#).

► **To provide instructions:**

1. Right-click the User action and click **Component Properties**.
2. In the **Instructions** box, type the instructions that you want to provide with the task.
3. Click **Save**.

**Tip:** The **Instructions** box can include XPath expressions.

## Configuring attachments

Tasks can have file attachments associated with them. Attachments are useful for providing people with supplemental information that is required to make decisions and complete tasks.

Attachments are made available to User actions by copying the attachments from action to action. For every process instance, a copy of the attachment is created and stored in the database for each action in the workflow:

- Modifications that people make to attachments are persisted to the next action.
- Attachments can require a considerable amount of storage space in the database.

Attachments are persisted only within a branch. For example, for a branch in a split, attachments are persisted to the User actions in the branch, but are not persisted to the User actions after the split.

When you use attachments in a workflow, you should consider how important they are and how much storage space they will require. For each user action, you can configure how attachments are handled:

- You can specify whether attachments are copied to user actions from the previous action. (See [“Persisting attachments” on page 65](#).)
- You can specify whether people can add attachments when they complete tasks. (See [“Enabling users to see and add attachments” on page 65](#).)
- You can add documents that are referenced by document variables to tasks as attachments. (See [“Adding attachments to tasks” on page 65](#).)
- You can save task attachments in list variables. (See [“Saving task attachments” on page 66](#).)

Also, attachments have special attributes that you can access using XPath expressions. (See [“Document attributes for attachments” on page 66](#).)

For more information about how to configure the content of tasks, see [“Configuring Human Interaction” on page 47](#).

## Persisting attachments

You can persist attachments from action to action in a workflow branch. Copy attachments from the previous action to the next when people need to use the attachments to complete their work. When attachments are not copied to a user action, then no subsequent actions in the workflow can access the attachment.

Attachments are persisted only within the same branch:

- Attachments are not persisted to processes that are initiated with a Chained Process action.
- Attachments are not persisted from the main workflow branch to the branches in a split, or from the branches in a split to the main workflow branch.

For more information about attachments, see [“Configuring attachments” on page 64](#).

### ► To persist attachments to a user action:

1. Right-click the User action and click **Component Properties**.
2. Click the **Routes and Attachments** tab.
3. Select **Copy all the attachments from the previous task**, and click **OK**.

## Enabling users to see and add attachments

You can enable the attachments window in LiveCycle Form Manager so that users can use task attachments. When a user adds an attachment, they specify the access properties for the attachment. For example, they can specify whether subsequent users can read, write, or delete the attachment.

**Note:** When users receive task forms in email, they cannot delete task attachments.

For more information about attachments, see [“Configuring attachments” on page 64](#).

### ► To enable users to add attachments:

1. Right-click the User action and click **Component Properties**.
2. Click the **Routes and Attachments** tab.
3. Select **Show Attachment Window for this task**, and click **OK**.

## Adding attachments to tasks

You can configure User actions so that documents are added to tasks as attachments. To attach documents to tasks, the documents need to be stored as workflow document data in a `list` variable.

Before you can perform the following procedure, you need to create the `list` variable.

► **To add attachments to a task:**

1. Right-click the User action that you want to configure and click **Component Properties**.
2. Click the **Routes and Attachments** tab.
3. Select **Copy attachments from a List of Documents**.
4. In the **Input List of Documents** menu, select the `list` variable that stores the documents you want to attach to tasks.
5. Click **OK**.

## Saving task attachments

You can configure User actions so that documents that have been added to tasks as attachments are saved in a `list` variable. For example, if you need to store a PDF file that has been digitally signed and attached to a task, you can configure the corresponding User action to store the PDF file in a `list` variable.

You store task attachments in `list` variables that store data of type `document`. Before you can configure the User action to store attachments, you need to create the `list` variable in which to store the attached documents.

► **To store attachments from a task:**

1. Right-click the User action that you want to configure and click **Component Properties**.
2. Click the **Routes and Attachments** tab.
3. Select **Map out Attachments into a List of Document Variables**.
4. In the **Output List of Documents** menu, select the `list` variable that you want to use to store the attachments from tasks.
5. Click **OK**.

## Document attributes for attachments

Document variables can have several attributes that are useful when documents are used as task attachments. The following table lists the attributes.

Attribute name	Description	Default value
<code>wsfilename</code>	The file name of the document	<code>Attachnumber</code> where <i>number</i> is the index that indicates the location of the document in the <code>list</code> variable that stores the attachment (see <a href="#">"Saving task attachments" on page 66</a> )

Attribute name	Description	Default value
<code>wspermission</code>	<p>A bitmask number that represents the access permissions for the document</p> <p>Access permissions determine how the document can be used when it is a task attachment. The value of the number is the sum of the numbers representing each access permission provided for the document:</p> <ul style="list-style-type: none"> <li>• Read access: 1</li> <li>• Write access: 2</li> <li>• Delete access: 4</li> </ul> <p>Valid values are 1, 3, 5, and 7.</p> <p>For example, the value of <code>wspermission</code> for a document with read and write access is 3.</p>	<p>7</p> <p>(Read, write, and delete access)</p>
<code>wscreatorid</code>	The unique identifier of the user who attached the document to the task	Workflow user identification
<code>wsdescription</code>	A description of the attachment	<i>Empty string</i>

To set attachment attributes, you use XPath expressions. For more information, see [“Expressions” on page 42](#).

## Task attachments in email messages

The following rules apply when users participate in processes using email and tasks have file attachments:

- The attachment permissions applied in LiveCycle Form Manager do not apply when the task form is delivered in an email. For example, if a user adds an attachment to a work item in LiveCycle Form Manager and applies the read-only permission, but the next user in the process receives the task form in an email message, that user will be able to modify the attachment.
- Users can add attachments to forms that they submit to LiveCycle Workflow Server in email messages. The attachment is persisted to the task for the next step of the process.
- If users modify attachments that came with task forms in an email, they need to manually attach the modified file to the email before sending it to LiveCycle Workflow Server. At the next step in the process, the task includes the original attachment as well as the updated attachment.

# 5

## Selecting Workflow and Branch Types

This chapter provides information about workflow types and branch types so that you can make decisions about the types that you should use. You should consider the following factors when deciding which workflow type and branch type to use:

- The amount of information about process instances you want to store. See [“Data persistence” on page 68](#).
- The complexity of the workflow. See [“Process complexity and system efficiency” on page 68](#).
- Whether the branch type is compatible with the type of workflow you are developing. See [“Workflow and branch compatibility” on page 69](#).
- Whether the actions you want to use are compatible with the branch type. See [“Action and branch compatibility” on page 69](#).

**Note:** If you want to roll back transactions for a branch when computational errors occur you need to use a Transactional branch. See [“Branch behavior when errors occur” on page 70](#).

### Data persistence

The type of workflow determines the workflow data that is saved in the database. The level of data persistence may be a factor for determining the type of workflow to use.

You may decide that it is important to keep a record of everything that happened during the process instance. You may decide that saving storage space is more important than keeping detailed records:

- If you want a complete record of the process instance, use an Asynchronous workflow.
- If you want no record of the process instance, use a Transient workflow.

Workflow type	Information stored
Asynchronous	All process information is stored: <ul style="list-style-type: none"><li>● Process instance data</li><li>● Branch instance data</li><li>● Action instance data</li></ul>
Transient	No process information is stored.

### Process complexity and system efficiency

The complexity of the workflow you are designing is a factor that you should use to decide which type of workflow to use. Some types of workflows are better for use with complicated processes. However, workflow types that can support complicated process can be less efficient than those that support simple workflows.

The following table describes guidelines to use for deciding the type of workflow to use based on the complexity of the process:

## Workflow and branch compatibility

Not all branch types are compatible with all workflow types. Workflow and branch compatibility may help you to decide which type of workflow and branch to use.

Workflow type	Branch types that can be used
Asynchronous	Asynchronous, Synchronous, and Transactional branches.
Transient	Transactional branches.

## Action and branch compatibility

Not all types of actions are compatible with all branch types. Action and branch compatibility may help you to decide the type of branch that you should use in your workflow.

Branch type	Compatible actions
Asynchronous	All actions
Synchronous	All actions <b>Note:</b> The Chained Process, User, Wait Point, and Script actions and the Split tool cause Synchronous branches to behave as Asynchronous branches.
Transactional	Chained Process, Set Value, Stall, and Script actions. You can only use non-interactive script with Script actions.

Workflow type	Supported complexity	Relative efficiency
Asynchronous	Complex workflows	Least efficient
Transient	Simple workflows	Most efficient

Simple workflows include a single branch. Complex workflows include several branches.

**Note:** You use splits to add additional branches to a workflow. LiveCycle Workflow Designer does not allow you to add a Split to Synchronous and Transient workflows.

This chapter provides information about the types of errors that can occur during the execution of a workflow and how to mitigate the effects of the errors:

- For information about the types of errors that can occur, see [“About errors” on page 70](#).
- For information about the behavior of different types of branches when errors occur, see [“Branch behavior when errors occur” on page 70](#).
- For information about using the Stall action to deliberately stall a branch, see [“Catching situational errors with the Stall action” on page 72](#).

## About errors

The two types of errors that can occur during the execution of a process instance are computational errors and situational errors. Different methods are available for handling both types of errors.

### Computational errors

Computational errors occur when the execution of a command results in an error. For example, an error can occur in a routing rule when the function in an expression uses a parameter of the wrong data type.

When a computational error occurs, the branch or action involved is stalled until the LiveCycle Workflow administrator intervenes. The type of branch that your workflow uses can mitigate the effects of computational errors. For more information, see [“Branch behavior when errors occur” on page 70](#).

### Situational errors

Situational errors occur when an event that should not occur does occur during a process, even though the workflow behaved as designed.

For example, a workflow assigns tasks to a random person of a specific user group. If that user does not complete the task by a certain time, it is randomly assigned to a user in the group again. A situational error occurs if the task was reassigned to the same person. To avoid situational errors, you can cause the branch to stall before the situation occurs. The LiveCycle Workflow administrator can then intervene and attempt to correct the situation.

For more information, see [“Catching situational errors with the Stall action” on page 72](#).

## Branch behavior when errors occur

The behavior of a branch when a stalled branch or a stalled action is retried is a factor you should consider when deciding which type of branch to use in your workflow.

For all branch types, when a branch exception occurs the branch is stalled, and when an action exception occurs, the action for which the exception occurred is stalled.

The following table describes how the different branch types continue execution after the LiveCycle Workflow administrator retries the stalled branch or action.

Branch type	Behavior when a stalled branch or action is retried
Asynchronous	The process continues from the point in the workflow where the exception occurred.
Synchronous	The process continues from the point in the workflow where the exception occurred.
Transactional	The branch is rolled-back and the process continues from the first action in the branch.

For more information about Transactional branches, see [“About transactions and branches” on page 71](#).

## About transactions and branches

In Transactional branches, the actions in a branch are executed in a single transaction. In an Asynchronous or Synchronous branch, each action is executed in separate transactions.

A transaction is a context in which one or more commands are executed. Generally, the commands that are executed within a transaction are not committed until all of the commands are executed successfully. If one command in a transaction fails to execute successfully, the transaction is rolled back and the effects of any commands that were already executed in that transaction are reversed.

Transactional branches are useful when a common goal is dependent on the successful execution of a series of actions. If an action in a Transactional branch executes with errors, the branch is stalled. When the branch is retried, the transaction for the branch is rolled back and LiveCycle Workflow Server executes the first action in the branch again.

For information about the behavior of actions in Transactional branches, see [“About transaction-aware actions” on page 71](#).

Asynchronous and Synchronous branches are useful when the failure of one action in the branch does not devalue the successful execution of previous actions in the branch. If an action in an Asynchronous or Synchronous branch executes with errors, the branch is stalled. When the branch is retried, only the failed action is rolled back. LiveCycle Workflow Server executes the failed action again.

## About transaction-aware actions

Actions that are transaction-aware will participate in a transaction. When a Transactional branch is rolled back, the actions in the branch that are transaction-aware are returned to their original state before that branch was executed.

If an action is not transaction-aware and it is used in a Transactional branch that stalls, the effects of the action are not rolled back when the branch is retried. The tasks are executed a second time. For example, if you use the Email action in a transactional branch, each time the branch stalls and is retried another email is generated. The Email Quick Process Action Component (QPAC) is available in the LiveCycle Workflow SDK.

The only transaction-aware action that LiveCycle Workflow provides is the Set Value action. However, your development team may create transaction-aware actions using the LiveCycle Workflow 7.0 SDK.

## About long-lived actions

Long-lived actions execute independently of the branch to which they belong:

- When a Transactional branch is stalled and is retried, any long-lived actions in the branch continue to run and are not rolled back.
- If a computational error occurs during the execution of a long-lived action, the action is stalled, but the branch is not stalled even if the action belongs to a Transactional branch.

None of the actions that LiveCycle Workflow provides are long-lived. However, your development team may create long-lived actions using the LiveCycle Workflow 7.0 SDK.

To see if an action is long-lived, open the action properties. For more information, see [“Viewing component properties in LiveCycle Workflow Designer” on page 86](#).

## Catching situational errors with the Stall action

When a stall action is executed, the branch to which it belongs is stalled. Stall actions are useful for preventing situational errors that you anticipate may occur.

If you are aware of a possible situational error in your workflow, you can add a route that leads to a Stall action. You can add a rule to the route that checks if the situational error has occurred.

For example, workflows can use data that is provided from an external resource, such as a partner's database. A Script action can be used to verify that the data is valid. If the data is not valid, a Stall action can be executed that stalls the process instance while the data in the database is corrected.

For more information about situational errors, see [“About errors” on page 70](#).

This chapter provides information that you need to manage the workflows that you create:

- After you have developed a workflow, you deploy it to the production environment and activate it so that client applications can use the workflow to initiate new process instances. For information, see [“Deploying workflows” on page 73](#) and [“Activating workflows” on page 73](#).
- Process types typically require the creation and development of several workflows. You can replace active workflows with updated versions. For information, see [“Working with workflow versions” on page 77](#).
- After you deploy and activate workflows, you can make limited changes to them. For information, see [“Modifying workflows” on page 77](#).
- You can export workflows to a file to make them more transportable. For information, see [“Exporting and importing workflows” on page 78](#).

## Deploying workflows

Deploy workflows to make them available to LiveCycle Workflow Server. For information about making changes to deployed workflows, see [“Modifying deployed workflows” on page 78](#).

For information about seeing if a workflow is deployed, see [“Determining workflow deployment state” on page 74](#).

**Note:** You cannot undeploy workflows.

### ► To deploy a workflow:

1. Open the workflow for editing.
2. Click the **Deploy** button. 

## Activating workflows

Activate deployed workflows to make them available to client applications for initiating new process instances. You can activate any workflow that is deployed.

**Note:** A process type can have only one active workflow. If a workflow is currently active, it will be made inactive when you activate a different one.

For information about seeing if a workflow is deployed, see [“Determining workflow deployment state” on page 74](#).

### ► To activate a workflow:

- Right-click the workflow you want to activate and click **Activate Workflow**.

## Determining workflow deployment state

The state of a workflow determines how LiveCycle Workflow Server can use the workflow. The three states of a workflow are summarized in the following table.

Icon	State	Description
	Not deployed	The workflow is not available to LiveCycle Workflow Server. Workflows that are not deployed appear directly under the process type in the Processes tree.
	Deployed	The workflow is available to LiveCycle Workflow Server. Deployed workflows appear under Deployed Workflows in the Processes palette.
	Active	LiveCycle Workflow Server runs this workflow when client applications make a request to initiate a new process instance.

## Adding components to the Components palette

Components are organized in categories so that LiveCycle Workflow Designer users can easily find them. Components are the client-side part of QPAC. You deploy QPACs to add them to the Components palette.

When you deploy QPACs, you create the category in which they appear. For information, see [“Creating component categories” on page 74](#).

After you deploy a QPAC, you can specify different icons that are used to represent the component in the Components palette or on the workflow. For information, see [“Using custom component icons” on page 76](#). You can also delete the component from the Components palette. For information, see [“Removing components” on page 76](#).

QPACs are provided on the LiveCycle Workflow installation media, in the directory `AdobeLiveCycleWorkflow7.0/QPACS`.

## Creating component categories

Create component categories so that you can organize the QPACs that you deploy.

- **To create a component category:**
  1. Click **File > New > Component Category**.
  2. In the **Name** box, type a name for the category.
  3. Click **Save**.

## Deploying components

To make QPACs available in LiveCycle Workflow Designer, you need to deploy them. When you deploy a QPAC, you specify the category in which you want it to appear. You may also need to specify other deployment properties, depending on the QPAC.

► **To deploy a component:**

1. Click **File > New > Component**.
2. Select one or more QPAC files to deploy and click **Select**. QPAC files have the JAR file name extension. A New Component dialog box appears where you specify values for the component properties.
3. In the **Category** list, select the category in which you want to include the component.
4. In the **Deployment Properties** area, specify any properties required for the component to function properly.
5. Click **OK**.

If you selected more than one QPAC file to deploy, a New Component dialog box appears for each QPAC, for which you need to perform steps 3 to 5.

## Updating components

You can replace components with updated QPAC files any time after you add them to the Components palette. To update a component, you redeploy the QPAC file. When you update a component, the updates affect all workflows that use the components, including deployed and active workflows.

For example, you are testing a workflow that uses a custom component that your development team has created. A developer changes the QPAC to fix the behavior of a feature. You redeploy the QPAC and the workflow behaves according to the updated feature.

**Tip:** To update a component without affecting workflows that currently use the component, deploy the QPAC using a different component name. The older version and the updated version can both be added to the Components palette if they use different names.

► **To update a component:**

1. On the Components palette, right-click the component you want to update and click **Component Properties**.
2. Click **Redeploy Component**.
3. Specify values for the component properties and click **OK**.
4. Click **Save** and then click **OK** to close the notification.

## Moving components to different categories

You can move components to different categories after you deploy them. Moving components to different categories does not affect the actions that you have added to your workflow.

- **To move an action to a different category:**
  1. In the Component palette, right-click the action and click **Component Properties**.
  2. In the **Category** list, select the category in which you want the action to appear in the Component palette.
  3. Click **Save** and click **OK** to close the notification.

## Using custom component icons

You can provide your own icons for components. The icons appear in the Components palette to represent components, and on the workflow to represent actions:

- The icon that you specify to represent the component in the Components palette should have dimensions 16x16 pixels.
- The icon that you specify to represent actions in workflows should have dimensions 100x100 pixels.
- Graphics must be in GIF or JPG format.

When you change the icon that appears in workflows, the new icon appears for the actions that are already in the workflows.

For more information about components, see [“About components” on page 22](#).

- **To specify custom component icons:**
  1. On the Component palette, right-click the action and click **Component Properties**.
  2. To change the icon that appears on the Component palette, click **Load Toolbar Icon**, browse for the graphic file to use as the icon, and click **Select**.
  3. To change the icon that appears on the workflow, click **Load Graph Icon**, browse for the graphic file to use as the icon, and click **Select**.
  4. Click **Save** and click **OK** to close the notification.

## Removing components

You can specify components that are to be removed from the Components palette.

You use LiveCycle Workflow Designer to indicate the items that you want to be deleted. Deletion does not occur immediately. You use the Purge feature of Object Definition Editor to delete items that are marked for deletion. Object Definition Editor is provided with the LiveCycle Workflow SDK.

- **To have a component removed:**
  1. In the Components palette, right-click the component and click **Component Properties**.
  2. In the **Status** menu, select **Marked For Deletion**.
  3. Click **Save** and click **OK** to close the notification.
  4. Open Object Definition Editor and use the Purge feature.

## Working with workflow versions

You can create multiple workflows for the same process type. Multiple workflows let you change the design of the process while keeping the previous design intact. For example, you may develop a workflow and then want to improve on the original design. Changes to business processes may also require that you modify the design of the original workflow.

When the new workflow version is complete, you can deploy and activate it to replace the currently active workflow:

- LiveCycle Workflow Server runs the currently active workflow to create new process instances.
- Process instances only use the workflow with which they were originally created. Activating a new version of a workflow has no effect on process instances that already existed.
- Workflows that were previously active remain deployed so that LiveCycle Workflow Server can continue to run them to complete the process instances that are associated with them.

**Note:** LiveCycle Workflow Designer does not provide visual cues that inform you of workflow versions. You should use meaningful names or descriptions of workflows to track version information.

For information about making copies of existing workflows and changing the design, see [“Creating copies of workflows” on page 77](#).

## Modifying workflows

There are two ways that you can modify the behavior of a process type:

- Create a copy of the workflow and use it to replace the currently active workflow. You can change the copies of workflows in any way.
- Modify deployed workflows directly so that changes affect existing process instances. You can make limited changes to workflows that are deployed.

## Creating copies of workflows

Create a copy of a workflow when you want to use the existing workflow as the basis for further development. You can make copies of workflows only within the same process type.

If you want information about moving a workflow to a different process type, see [“Exporting and importing workflows” on page 78](#).

**Tip:** LiveCycle Workflow Designer does not provide visual cues that inform you of workflow versions. You can use meaningful names or descriptions of workflows to track version information.

### ► To copy a workflow version:

1. Right-click the process type for which you want to create the workflow and click **New Workflow**.
2. In the **Name** box, type a name for the workflow.
3. Click **Copy**, select the workflow you want to copy, and click **OK**.
4. Click **Save**, and then click **OK** to close the notification.

## Modifying deployed workflows

You can make changes to workflows that are deployed and have process instances associated with them.

When you make changes to deployed workflows, the changes are implemented immediately after you save the changes. Existing and new process instances associated with the workflow will progress according to the latest changes.

You must use caution when changing deployed workflows. Some changes can cause existing process instances to stall. For example, if you remove variables that are referenced in XPath expressions, the expressions will cause the process instance to stall.

## Exporting and importing workflows

Exporting a workflow creates an XML file that contains the definition of the workflow. After you have the workflow in XML file format, you can save it to your hard drive or move it to another computer. You can also import the workflow to any process type that you have created in LiveCycle Workflow Designer. After you import a file, you can use the workflow definition as is, or make modifications as necessary.

If you want information about copying workflows within a process type, see [“Creating copies of workflows” on page 77](#).

LiveCycle Workflow Server lets you export and import workflows to create backup copies of workflows, move workflows to the production environment, and repurpose existing work. The environment where you import workflows should be very similar to the environment where you exported from so that the workflow properties remain valid. When you import a workflow, you should ensure the environment can support the workflow:

- Ensure that the components that the workflow uses are deployed, and that the deployed components have the same ID property as those used in the workflow.
- Ensure that expressions used in action properties reference valid data elements.
- Ensure that child processes specified in Chained Process actions are deployed and are located in the original categories.
- If the workflow includes User actions that assign tasks to users or groups, ensure that the domain name of your current environment is the same as the domain name of the environment where the workflow was created. If the domain names are different, the users and groups do not appear in the User action. Domain names are case-sensitive.

**Note:** If you import a workflow that uses components that are not deployed or have a different ID, you will not be able to open the workflow. To make the workflow usable, deploy the required components and specify ID values that are the same as those used to create the workflow.

## Exporting workflows

Export a workflow to a file to save it to your hard drive as an XML file.

**Caution:** If any component or action properties include passwords, the password is saved in the exported XML file in plain text.

► **To export a workflow:**

1. Right-click on the workflow and click **Export**.
2. Specify the file name and location and click **Export**.
3. Click **OK** to close the notification.

## Importing workflows

Import a workflow to move it into a process type. The workflow that you want to import must have already been exported to an XML file.

► **To import a workflow:**

1. Right-click the process type in which you want to import, and click **New Workflow**.
2. In the **Name** box, type a name for the workflow.
3. Click **Import** and select the workflow that you want to import.
4. Click **Save**, and click **OK** to close the notification.

## Migrating workflows to production environments

You migrate the workflows that you have created and tested in a development environment to a production environment to make them available to users and systems.

When migrating to production, you move the workflow and its collateral, such as the QPACs and the forms that it uses, to the production environment. You also need to ensure that any action properties are modified, if necessary. For example, if a User action assigns tasks to a specific user, you may need to update the settings on the Initial User tab of the User Action Properties dialog box. The user account that you specify in production may be different than the account used in development for testing purposes.

Before migrating to production, you need to install another instance of LiveCycle Workflow Designer that connects to the computer that hosts LiveCycle Workflow Server in the production environment. For more information, see the *Installing and Configuring LiveCycle* guide for your application server.

**Note:** If you are migrating workflows that are invoked by other workflows using the Chained Processes action, you must migrate the invoked workflow before you migrate the workflow that invokes it.

► **To migrate a workflow to production:**

1. In the development environment, export the workflow to an XML file. (See [“Exporting workflows” on page 79.](#))
2. If the workflow uses forms, deploy the forms and any referenced images or script files to LiveCycle Form Manager in the production environment.

**Note:** To avoid having to change any form URLs in the workflow variables or User action properties, in the production environment, use the same directory structure in LiveCycle Form Manager that you used in the development environment. For example, if the form1.xdp file was deployed to the Forms/Workflow directory in the development environment, you must deploy the file to the Forms/Workflow directory in the production environment.

3. In LiveCycle Form Manager, configure the form properties, such as access rights and category assignments, so that they have the same properties as they did in the development environment.
4. In the production environment, use LiveCycle Workflow Designer to deploy all of the QPACs that the workflow uses as actions:
  - You need to specify the same Name property for each QPAC that was used in the development environment. Workflows identify the components that it uses by the Name property.
  - If a QPAC requires you to specify deployment properties, use properties that are specific to the production environment. For example, if the QPAC requires the URL of an FTP server, specify the URL of the server that is used in production.

For information about deploying QPACs, see [“Deploying components” on page 75.](#)

5. In the production environment, use LiveCycle Workflow Designer to create the required process categories and process types. (See [“Adding components to the Components palette” on page 74.](#))
6. In the production environment, use LiveCycle Workflow Designer to create a new workflow by importing the workflow XML file that you exported in step 1. (See [“Importing workflows” on page 79.](#))
7. Ensure that the Transient, Web Service Access, and Monitor Access workflow properties are configured as they were in the development environment. These settings are not persisted when importing workflows. (See [“Creating workflows” on page 17.](#))
8. Open the workflow in LiveCycle Workflow Designer and update the action properties as required to reflect any difference in the production environment from the development environment.
9. Deploy the workflow.
10. If you have developed applications that use the workflow web service, update the applications for use in the production environment. For example, the URL of the workflow web service is now different than it was in the development environment.
11. If you have developed Java applications that interact with LiveCycle Workflow, update the applications so that they connect with the instance of LiveCycle Workflow Server in the production environment.

# 8

## Designing Forms for LiveCycle Workflow

Forms have several design requirements so that they are compatible with LiveCycle Workflow. LiveCycle Workflow supports Acrobat forms and XML forms:

- Acrobat forms are PDF forms that are created in Acrobat Professional.
- XML forms are form designs created in LiveCycle Designer that are saved either as XDP files or as PDF files.

The type of form you are using and the way you use the forms determine what you need to do to the form so that it works with your workflows:

- You need to add workflow-related fields to your forms. See [“Form fields for use with workflows” on page 81](#).
- When submitting forms that are saved in document variables, you need to configure the Submit button on the form appropriately. See [“Configuring Submit buttons when handling documents” on page 83](#).
- For Acrobat forms, you need to follow specific rules when naming fields. See [“Fields for Acrobat forms” on page 82](#).
- You need to configure forms to let people initiate processes by sending forms in email. See [“Initiating processes through email” on page 84](#).
- If your workflow uses more than one form, you need to design the forms appropriately. See [“Using multiple forms in workflows” on page 84](#).

You can also embed the form schema in forms so that the field data is available in the XPathExpression Editor. For information, see [“Making form fields appear in XPathExpression Builder” on page 51](#).

### Form fields for use with workflows

You need to add fields to your forms to enable them to work with LiveCycle Workflow. The form fields that you add depend on the type of form you are using.

#### Fields for XML forms

To use XML forms with LiveCycle Workflow, the form must include several workflow-related fields. To add the fields, you use LiveCycle Designer and add the custom library named Adobe LiveCycle Workflow Fields.

The custom library is available with the LiveCycle Workflow SDK. The library includes the following fields.

Field name	Type	Description
AWS_SUBMIT	Button	Submits the form to LiveCycle Workflow Server.
AWS_ACTION	Pop-up menu	Lists the choices available for completing the form. If you do not want to provide choices, you do not need to include this field.

Field name	Type	Description
AWS_ASSIGNED_ID	Text field (invisible)	A hidden text field that stores the name of the participant that submits the form.
AWS_STATUS	Text field (invisible)	A hidden text field that is automatically populated with workflow data.
AWS_TASKID	Text field (invisible)	A hidden text field that is automatically populated with workflow data.
AWS_MAILTO	Text field (invisible)	A hidden text field that contains the email address that LiveCycle Workflow Server uses.  You need to enter a value in this field if you want people to initiate processes by emailing the form to LiveCycle Workflow Server. See <a href="#">“Initiating processes through email” on page 84.</a>
AWS_CHOICE	Text field (invisible)	A hidden text field that is automatically populated with workflow data.
AWS_PROCESSTYPE	Text field (invisible)	A hidden text field that contains the process type with which the form is associated.  You need to enter a value in this field only if you want people to initiate processes by emailing the form to LiveCycle Workflow Server. See <a href="#">“Initiating processes through email” on page 84.</a>
FSSUBMIT_	Text field (invisible)	hidden button that is automatically populated with information for sending the form to Adobe LiveCycle Forms.

**Note:** You must not group the workflow fields. If you group the fields, the form will not work with LiveCycle Workflow.

## Fields for Acrobat forms

To use Acrobat forms with LiveCycle Workflow, the form must include the following workflow-related fields.

Field name	Type	Description
AWS_CHOICE	Text (hidden)	A hidden text field that is automatically populated with workflow data.
FSAPPLICATIONDATA_	Text (hidden)	A hidden text field that is automatically populated with workflow data.
FSTARGETURL_	Text (hidden)	A hidden text field that is automatically populated with workflow data.

Field name	Type	Description
AWS_ACTION	Combo box	<p>Lists the submit options available for completing the form. This field also includes a calculation script that initializes the form.</p> <p>The default settings for this field are for use with XDP forms and with dynamic interactive PDF forms (created with LiveCycle Designer). You can modify the settings so that it displays choices in static interactive PDF forms. You use LiveCycle Designer to manipulate the field:</p> <ul style="list-style-type: none"> <li>• For the AWS_ACTION drop-down list, move the JavaScript code from the form:ready event to the preOpen event.</li> <li>• Ensure the Language property for the preOpen event is set to JavaScript, and the Run At property is set to Client.</li> </ul> <p>If you do not want to provide choices, you can hide this field.</p>
AWS_TASKID	Text (hidden)	A hidden text field that is automatically populated with the identification of the task to which the form is associated.
Submit	Button	<p>Submits the form to LiveCycle Workflow Server.</p> <p>If your PDF form already has a submit button, modify the existing button to include the script from this button. This new script should be processed last.</p>

You can copy the fields from a PDF file that is included with the LiveCycle Workflow SDK. You must use Acrobat Professional, version 6.0 or later, to copy the fields onto your PDF form.

**Note:** When you deploy Acrobat forms to LiveCycle Form Manager, you must select the **Merge data into form** option in the form properties.

## Configuring Submit buttons when handling documents

When you want to convert forms and form data to a PDF document and save the reference in a document variable, you need to configure the Submit button on the form. The Submit button must use a submit format of PDF. To configure the Submit button, you use LiveCycle Designer.

For more information about converting forms to PDF documents and using document variables, see [“Referencing documents in document variables” on page 53](#).

**Note:** If you do not configure the Submit button appropriately, the process does not progress past the point where the PDF document is created.

### ► To configure a Submit button:

1. Open the XML form in LiveCycle Designer.
2. Click the **Submit** button.
3. On the Object palette, in the Control Type area of the **Field** tab, select **Submit**.
4. Click the **Submit** tab, and in the **Submit Format** menu select **PDF**.
5. Save the form.

## About field names in Acrobat forms

To ensure that your Acrobat forms function properly with LiveCycle Workflow, your forms must conform to the following rules:

- Field names must not contain Double Byte Character Set (DBCS) characters. The Adobe PDF standard does not permit the use of DBCS characters in field names.
- Field names should include only characters that are valid in XML. When form data is converted to XML, the field names correspond with XML element names. Your field names must be valid XML element names.

## Initiating processes through email

LiveCycle Workflow Server can initiate a process type when it receives a start form in an email message. If you want to let people initiate a process type through email, you need to use an XML form and you need to configure the form appropriately.

In addition to adding the required workflow fields and creating the corresponding init-form variable in LiveCycle Workflow Designer, the form requires values in the `AWS_PROCESSTYPE` field and the `AWS_MAILTO` field.

Field name	Required value
<code>AWS_PROCESSTYPE</code>	The name of the process type. LiveCycle Workflow Server uses the value of this field to execute the active workflow that is associated with the process type named in this field.
<code>AWS_MAILTO</code>	The email address that LiveCycle Workflow Server is configured to use. When the Submit button is pressed on the form, the form is automatically sent to the email address in this field.

For information about adding the appropriate form fields to the form, see [“Form fields for use with workflows” on page 81](#).

To submit the form in email, the form must be opened in an appropriate client application such as LiveCycle Designer or Acrobat. LiveCycle Workflow Server processes the form only if the email address used to send the email is associated with a LiveCycle user.

When the Submit button is pressed, the client application sends the form in an email to the address specified in `AWS_MAILTO`. If no address or process type is specified in the fields, when the user presses the Submit button, they are prompted for the email address and the process type.

## Using multiple forms in workflows

To use different forms for different User actions in a workflow, the forms should have similar architectures. If the forms used for separate User actions are not similar, the form data may not properly appear in the form.

**Tip:** If required, you can use a Set Value action to manipulate the data in the form variable to make the data compatible with a form architecture.

Form data is persisted from one User action to the next throughout the entire process. LiveCycle Workflow Server uses field names and the layout of the form to decide which items of data populate which fields. If you want form data to populate form fields appropriately, you must design your forms appropriately:

- Use the same field names in all your forms for the fields that you want populated with the same data. See [“Use consistent field names” on page 85](#).
- The fields must appear in the same order in all your forms. See [“Use common form architectures” on page 85](#).

**Note:** Using different field names does not cause a loss of data. The data remains associated with the task, but does not appear on the form.

## Use consistent field names

Use field names for forms consistently if you use more than one form in the same workflow. When you use the same names for fields, the same data automatically populates each of the different forms.

Workflow Server manages data and forms separately. In this way, the same data can appear on different forms.

When a person first enters data on a form, each item of data is associated with the name of the field that it populates. When the form data is used with subsequent User actions, each item of data populates the fields according to the names of the form fields.

If you want data to appear correctly on different forms in subsequent User actions:

- Ensure that the forms use the same names for fields.
- Ensure that the fields that have the same name serve the same purpose from form to form.

**Note:** All form data is persisted from action to action regardless of the form used. If no form field corresponds with an item of data, the data is persisted but it does not appear on the form.

## Use common form architectures

If subsequent User actions use different input forms, use similar form architectures for each form. If you use a common form architecture, form data that is persisted from action to action correctly populates the forms.

The form architecture determines the structure of the form data. If the structure of the form data does not match the structure of the form, the data does not properly populate the form.

## Collecting date and time data in forms

When using forms that collect date and time values, you must ensure that the data in the form fields is formatted appropriately. When extracting the date from a form field and processing it using an XPath Date or Date-Time function in an XPath expression, the extracted date must be in the format that is valid for the function.

For information about the valid formats for Date or Date-Time functions, see [“Date and time parameters” on page 122](#).

# A

## Component Properties

---

Each workflow component that appears in the Components palette includes static properties that provide information about how it behaves and how it can be used as an action. This appendix provides a reference of properties for all components that are included with LiveCycle Workflow Designer.

- You can view component properties in LiveCycle Workflow Designer. For information, see [“Viewing component properties in LiveCycle Workflow Designer” on page 86](#).
- For an explanation of the properties, see [“About component properties” on page 86](#).

## Viewing component properties in LiveCycle Workflow Designer

You can view the properties of components in LiveCycle Workflow Designer.

► **To view component properties:**

1. In the Components palette, right-click the component and click **Component Properties**.
2. Click the **Info** tab.

## About component properties

Each component has its own properties to indicate the component type, the version, and the branch type with which it is compatible. The following table includes descriptions of the component properties.

Property	Description
Component ID	A string that uniquely identifies the component.
Version	The version of the component that is deployed.
Interactive	Indicates whether the component is interactive. True indicates the component is interactive and false indicates the component is not interactive.  Interactive components require external intervention to signal to LiveCycle Workflow Server that actions are complete. For example, to be completed, User actions require that a person submits the form that they receive when they are assigned a task.

<b>Property</b>	<b>Description</b>
Long-lived	<p>Indicates whether the component is long-lived. True indicates the component is long-lived, and false indicates the component is not long-lived.</p> <p>The execution of long-lived components occurs independently of the branch to which they belong. When a transactional branch is stalled and the transaction is retried, any long-lived actions in the branch continue to run and are not rolled back.</p> <p>If an exception occurs during the execution of a long-lived component, the action is stalled.</p>
Message Transaction Type	<p>Indicates whether the message transaction type is container- or action-managed. Action indicates action-managed and container indicates container-managed.</p>
Execution Transaction Type	<p>Indicates whether the execute transaction type is container- or action-managed. Action indicates action-managed and container indicates container-managed.</p>
Specification Version	<p>The version of the API used to develop this component.</p>
Supports Asynchronous Branch	<p>Indicates whether the component can be used in Asynchronous branches. True indicates the component can be used in Asynchronous branches and false indicates it cannot.</p>
Supports Synchronous Branch	<p>Indicates whether the component can be used in Synchronous branches. True indicates the component can be used in Synchronous branches and false indicates it cannot.</p>
Supports Transactional Branch	<p>Indicates whether the component can be used in Transactional branches. True indicates the component can be used in Transactional branches and false indicates it cannot.</p>

# B

## Function Reference

---

This appendix provides information about the functions and operators that are available for use in XPath expressions in LiveCycle Workflow Designer. The functions and operators are based on the XML Path Language version 1.0 W3C Recommendation. The recommendation is available at [www.w3c.org/TR/xpath](http://www.w3c.org/TR/xpath).

The LiveCycle Workflow implementations may not function as they are described in the W3C recommendation.

### Boolean functions

This section describes the Boolean functions that are available in expressions.

#### boolean

Converts a given value to an equivalent Boolean value.

**Syntax**

```
boolean(Expression)
```

**Parameters**

Expression

**Returns**

A Boolean value.

#### false

Returns `false`.

**Syntax**

```
false()
```

**Parameters**

None.

**Returns**

A Boolean value of `false`.

#### not

The inverse value of a given Boolean value.

### Syntax

```
not (boolean)
```

### Parameters

A boolean value.

### Returns

A Boolean value of true if the parameter `boolean` is false, otherwise false.

## true

Returns true.

### Syntax

```
true()
```

### Parameters

None.

### Returns

A Boolean value of true.

## Collection functions

This section describes the collection functions that are available for use on `list` and `map` variables in expressions.

### get-map-keys

Retrieves the keys from a `map` variable.

### Syntax

```
get-map-keys (Expression)
```

### Parameters

`Expression` is an XPath expression that resolves to a workflow `map` variable.

### Returns

A workflow `list` object that contains `string` objects that represent `map` keys.

### Example

A `map` variable named `varmap` includes the key-value pairs (`username`, `admin`) and (`password`, `1234`). The following example returns a `list` object that includes the strings `username` and `password`:

```
get-map-keys (/process_data/varmap)
```

## get-collection-size

Retrieves the number of items in a workflow data collection.

### Syntax

```
get-collection-size (node-set)
```

### Parameters

`node-set` is an XPath expression that resolves to the data node that represents a `list` or `map` variable.

### Returns

An integer that contains the number of data items in the `node-set`.

### Example

A `list` variable named `varlist` contains 6 data items. The following example returns 6:

```
get-collection-size (/process_data/varlist)
```

## Date-Time functions

This section describes the date-time functions that are available in expressions.

### add-days-to-date

Adds the specified number of days to a given date.

### Syntax

```
add-days-to-date (strDate1, days)
```

### Parameters

`strDate1` is a `string` that represents the date and is specified in the format `yyyy-mm-dd`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

`days` is a `number` that holds the number of days to add to the given date. A negative number for `days` subtracts days from the given date.

### Returns

A `string` object in the format `yyyy-mm-dd`.

### Example

The following expression returns a `string` of value `2001-10-30`:

```
add-days-to-date ("1999-11-28", 337)
```

### add-days-to-dateTime

Adds the specified number of days to a given date and time.

### Syntax

```
add-days-to-dateTime(strDateTime1, days)
```

### Parameters

`strDateTime1` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

`days` is a number that holds the number of days to add to the given date and time. A negative number for `days` subtracts days from the given date and time.

### Returns

A string object in the format `yyyy-mm-ddThh:mm:ssZ`.

### Example

The following expression returns a `dateTime` object of value `2000-10-30T11:12:00Z`:

```
add-days-to-dateTime("1999-11-28T11:12:00Z", 337)
```

## add-hours-to-dateTime

Adds the specified number of hours to a given date and time.

### Syntax

```
add-hours-to-dateTime(strDateTime1, hours)
```

### Parameters

`strDateTime1` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

`hours` is a number that holds the number of hours to add to the given date and time. A negative number for `hours` subtracts hours from the given date and time.

### Returns

A string object in the format `yyyy-mm-ddThh:mm:ssZ`.

### Example

The following expression returns a string object of value `1999-11-28T03:12:00Z`:

```
add-hours-to-dateTime("1999-11-28T11:12:00Z", -8)
```

## add-minutes-to-dateTime

Adds the specified number of minutes to a given date and time.

### Syntax

```
add-minutes-to-dateTime(strDateTime1, minutes)
```

### Parameters

`strDateTime1` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

`minutes` is a number that holds the number of minutes to add to the given date and time. A negative number for `minutes` subtracts minutes from the given date and time.

### Returns

A string object in the format `yyyy-mm-ddThh:mm:ssZ`.

### Example

The following expression returns a string object of value `1999-11-28T11:22:00Z`:

```
add-minutes-to-dateTime("1999-11-28T11:12:00Z",10)
```

## add-months-to-date

Adds the specified number of months to a given date.

The `addmonthstodate` function returns the date computed by adding months to the normalized value of `strDate1`. If `months` is negative, the date returned precedes `strDate1`.

### Syntax

```
add-months-to-date(strDate1, months)
```

### Parameters

`strDate1` is a string that represents the date and is specified in the format `yyyy-mm-dd`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

`months` is a number that holds the number of months to add to the given date. A negative number for `months` subtracts months from the given date.

### Returns

A string object in the format `yyyy-mm-dd`.

### Example

The following expression returns a string object of value `2000-10-28`:

```
add-months-to-date('1999-11-28',11)
```

## add-months-to-dateTime

Adds the specified number of months to a given date and time.

### Syntax

```
add-months-to-dateTime(strDateTime1, months)
```

### Parameters

`strDateTime1` is a string that represents the date and time, and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

`months` is a number that holds the number of months to add to the given date and time. A negative number for `months` subtracts months from the given date and time.

### Returns

A string object in the format `yyyy-mm-ddThh:mm:ssZ`.

### Example

The following expression returns a string object of value `2000-10-28T11:12:00Z`:

```
add-months-to-dateTime('1999-11-28T11:12:00Z', 11)
```

## add-seconds-to-dateTime

Adds the specified number of seconds to a given date and time.

### Syntax

```
add-seconds-to-dateTime(strDateTime1, seconds)
```

### Parameters

`strDateTime1` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

`seconds` is a number that holds the number of seconds to add to the given date and time. A negative number for `seconds` subtracts seconds from the given date and time.

### Returns

A string object in the format `yyyy-mm-ddThh:mm:ssZ`.

### Example

The following expression returns a string object of value `1999-11-28T11:11:30Z`:

```
add-seconds-to-dateTime('1999-11-28T11:12:00Z', -30)
```

## add-years-to-date

Adds the specified number of years to a given date.

### Syntax

```
add-years-to-date(strDate1, years)
```

### Parameters

`strDate1` is a string that represents the date and is specified in the format `yyyy-mm-dd`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

`years` is a number that holds the number of years to add to the given date and time. A negative number for `years` subtracts years from the given date.

### Returns

A string object in the format `yyyy-mm-ddThh:mm:ssZ`.

### Example

The following expression returns a string object of value `2001-11-28`:

```
add-years-to-date ("1999-11-28", 2)
```

## add-years-to-dateTime

Adds the specified number of years to a given date and time.

### Syntax

```
add-years-to-dateTime(strDateTime1, years)
```

### Parameters

`strDateTime1` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

`years` is a number that holds the number of years to add to the given date and time. A negative number for `years` subtracts years from the given date and time.

### Returns

A string object in the format `yyyy-mm-ddThh:mm:ssZ`.

### Example

The following expression returns a string object of value `2001-11-28T11:12:00Z`:

```
add-years-to-dateTime ("1999-11-28T11:12:00Z", 2)
```

## current-date

Returns the current date.

### Syntax

```
current-date()
```

### Parameters

None.

### Returns

A string object in the format `yyyy-mm-dd`.

## current-dateTime

Returns the current date and time.

### Syntax

```
current-dateTime()
```

### Parameters

None.

### Returns

A `string` object in the format `yyyy-mm-ddThh:mm:ssZ`.

## current-time

Returns the current time.

### Syntax

```
current-time()
```

### Parameters

None.

### Returns

A `string` object in the format `hh:mm:ss`.

## date-equal

Compares two given dates to see if they are equal.

### Syntax

```
date-equal(strDate1, strDate2)
```

### Parameters

`strDate1` is a `string` that represents one date to compare and is specified in the format `yyyy-mm-dd`.

`strDate2` is a `string` that represents the other date to compare and is specified in the format `yyyy-mm-dd`.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A Boolean value of `true` if the date that `strDate1` represents is equal to the date that `strDate2` represents, otherwise `false`.

## date-greater-than

Compares two given dates to see if one is greater than the other.

### Syntax

```
date-greater-than(strDate1, strDate2)
```

### Parameters

`strDate1` is a `string` that represents one date to compare and is specified in the format `yyyy-mm-dd`.

`strDate2` is a `string` that represents the other date to compare and is specified in the format `yyyy-mm-dd`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A Boolean value of `true` if the date that `strDate1` represents is greater than the date that `strDate2` represents, otherwise `false`.

## date-less-than

Compares two given dates to see if one is less than the other.

### Syntax

```
date-less-than(strDate1, strDate2)
```

### Parameters

`strDate1` is a `string` that represents one date to compare and is specified in the format `yyyy-mm-dd`.

`strDate2` is a `string` that represents the other date to compare and is specified in the format `yyyy-mm-dd`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A Boolean value of `true` if the date that `strDate1` represents is less than the date that `strDate2` represents, otherwise `false`.

## dateTime-equal

Checks whether two given dates and times are equal.

### Syntax

```
dateTime-equal(strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a string that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a string that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A Boolean value of `true` if the date and time that `strDateTime1` represents is equal to the date and time that `strDateTime2` represents, otherwise `false`.

## dateTime-greater-than

Checks whether a given date and time is greater than another given date and time.

### Syntax

```
dateTime-greater-than(strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a string that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a string that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A Boolean value of `true` if the date and time that `strDateTime1` represents is greater than the date and time that `strDateTime2` represents, otherwise `false`.

## dateTime-less-than

Checks whether a given date and time is less than another given date and time.

### Syntax

```
dateTime-less-than(strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a string that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a string that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A Boolean value of `true` if the date and time that `strDateTime1` represents is less than the date and time that `strDateTime2` represents, otherwise `false`.

## get-day-from-date

Returns the day part of a given date.

### Syntax

```
get-day-from-date (strDate)
```

### Parameters

`strDate` is a string that represents the date for which the day is returned and is specified in the format `yyyy-mm-dd`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the day part of the given date.

## get-day-from-dateTime

Returns the day part of a given date and time.

### Syntax

```
get-day-from-dateTime (strDateTime)
```

### Parameters

`strDateTime` is a string that represents the date for which the day is returned and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the day part of the given date and time.

## get-days-from-date-difference

Calculates the difference between the day part of two given dates and returns the difference.

### Syntax

```
get-days-from-date-difference (strDate1, strDate2)
```

### Parameters

`strDate1` is a string that represents one date to compare and is specified in the format `yyyy-mm-dd`.

`strDate2` is a string that represents the other date to compare and is specified in the format `yyyy-mm-dd`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A `number` that holds the difference between the day part of the given dates.

### Example

The following expression returns a `number` of value 337:

```
get-days-from-date-difference('2000-10-30','1999-11-28')
```

## get-days-from-dateTime-difference

Calculates the difference between the day part of two given date and time values and returns the difference.

### Syntax

```
get-days-from-dateTime-difference(strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a `string` that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a `string` that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A `number` that holds the difference between the day part of the given date and time values.

### Example

The following expression returns a `number` of value 337:

```
get-days-from-date-Time-difference('2000-10-30T11:12:20Z',  
                                   '1999-11-28T09:00:05Z')
```

## get-hours-from-dateTime

Returns the hours part of a given date and time.

### Syntax

```
get-hours-from-dateTime(strDateTime)
```

### Parameters

`strDateTime` is a `string` that represents the date and time for which the hours is returned, and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A `number` that holds the hours part of the given date and time.

## get-hours-from-dateTime-difference

Calculates the difference between the hour part of two given date and time values and returns the difference.

### Syntax

```
get-hours-from-dateTime-difference (strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a *string* that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a *string* that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A *number* that holds the difference between the hour part of the given date and time values.

### Example

The following expression returns a number of value 2:

```
get-hours-from-dateTime-difference ('2000-10-30T11:12:20Z',  
                                     '1999-11-28T09:00:05Z')
```

## get-hours-from-time

Returns the hours part of a given time.

### Syntax

```
get-hours-from-time (strTime)
```

### Parameters

`strTime` is a *string* that represents the time and is specified in the format `hh:mm:ss`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A *number* that holds the hour part of the given time.

## get-minutes-from-dateTime

Returns the minutes part of a given date and time value.

### Syntax

```
get-minutes-from-dateTime (strDateTime)
```

### Parameters

`strDateTime` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the minutes part of the given date and time value.

## get-minutes-from-dateTime-difference

Calculates the difference between the minutes part of two given date and time values and returns the difference.

### Syntax

```
get-minutes-from-dateTime-difference(strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a string that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a string that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the difference between the minutes part of the given date and time values.

### Example

The following expression returns a number of value 12:

```
get-minutes-from-dateTime-difference('2000-10-30T11:12:20Z',  
                                     '1999-11-28T09:00:05Z')
```

## get-minutes-from-time

Returns the minutes part of a given time.

### Syntax

```
get-minutes-from-time(strTime)
```

### Parameters

`strTime` is a string that represents the time and is specified in the format `hh:mm:ss`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the minutes part of the given time.

## get-month-from-dateTime

Returns the month part of a given date.

### Syntax

```
get-month-from-dateTime (strDate)
```

### Parameters

`strDateTime` is a string that represents the date and is specified in the format `yyyy-mm-dd`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the months part of the given date.

## get-month-from-dateTime

Returns the month part of a given date and time value.

### Syntax

```
get-month-from-dateTime (strDateTime)
```

### Parameters

`strDateTime` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the months part of the given date and time value.

## get-months-from-date-difference

Calculates the difference between the months part of two given date and time values and returns the difference.

The `getmonthsfromdatedifference` function returns the months component in the difference between two dates as an `yearMonthDuration`.

### Syntax

```
get-months-from-date-difference (strDate1, strDate2)
```

### Parameters

`strDate1` is a string that represents one date to compare and is specified in the format `yyyy-mm-dd`.

`strDate2` is a string that represents the other date to compare and is specified in the format `yyyy-mm-dd`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A `number` that holds the difference between the months part of the given date and time values.

### Example

The following expression returns a `number` of value 11:

```
get-months-from-date-difference('2000-10-30','1999-11-28')
```

## get-months-from-dateTime-difference

Calculates the difference between the months part of two given date and time values and returns the difference.

### Syntax

```
get-months-from-dateTime-difference(strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a `string` that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a `string` that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A `number` that holds the difference between the months part of the given date and time values.

### Example

The following expression returns a `number` of value 11:

```
get-months-from-dateTime-difference('2000-10-30T11:12:20Z',  
                                     '1999-11-28T09:00:05Z')
```

## get-seconds-from-dateTime

Returns the seconds part of a given date and time value.

### Syntax

```
get-seconds-from-dateTime(strDateTime)
```

### Parameters

`strDateTime` is a `string` that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A `number` that holds the seconds part of the given date and time value.

## get-seconds-from-dateTime-difference

Calculates the difference between the seconds part of two given date and time values and returns the difference.

### Syntax

```
get-seconds-from-dateTime-difference (strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a *string* that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a *string* that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A *number* that holds the difference between the seconds part of the given date and time values.

### Example

The following expression returns a number of value 15:

```
get-seconds-from-dateTime-difference ('2000-10-30T11:12:20Z',  
                                     '1999-11-28T09:00:05Z')
```

## get-seconds-from-time

Returns the seconds part of a given time.

### Syntax

```
get-seconds-from-time (strTime)
```

### Parameters

`strTime` is a *string* that represents the time, and is specified in the format `hh:mm:ss`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A *number* that holds the seconds part of the given time.

## get-timezone-from-date

Returns the time zone part of a given date.

### Syntax

```
get-timezone-from-date (strDate)
```

### Parameters

`strDate` is a *string* that represents the date, and is specified in the format `yyyy-mm-dd`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the time zone part of the given date.

## get-timezone-from-dateTime

Returns the time zone part of a given date and time value.

### Syntax

```
get-timezone-from-dateTime (strDateTime)
```

### Parameters

`strDateTime` is a string that represents the date and time, and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the time zone part of the given date and time value.

## get-timezone-from-time

Returns the time zone part of a given time.

### Syntax

```
get-timezone-from-time (strTime)
```

### Parameters

`strTime` is a string that represents the time and is specified in the format `hh:mm:ss`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the time zone part of the given time.

## get-year-from-date

Returns the year part of a given date.

### Syntax

```
get-year-from-date (strDate)
```

### Parameters

`strDate` is a string that represents the date and is specified in the format `yyyy-mm-dd`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the year part of the given date.

## get-year-from-dateTime

Returns the year part of a given date and time value.

### Syntax

```
get-year-from-dateTime (strDateTime)
```

### Parameters

`strDateTime` is a string that represents the date and time and is specified in the format `yyyy-mm-ddThh:mm:ssZ`. For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the year part of the given date and time value.

## get-years-from-dateTime-difference

Calculates the difference between the year part of two given date and time values and returns the difference.

### Syntax

```
get-years-from-dateTime-difference (strDateTime1, strDateTime2)
```

### Parameters

`strDateTime1` is a string that represents one date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

`strDateTime2` is a string that represents the other date and time to compare and is specified in the format `yyyy-mm-ddThh:mm:ssZ`.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A number that holds the difference between the year part of the given date and time values.

### Example

The following expression returns a number of value 0:

```
get-years-from-dateTime-difference ('2000-10-30T11:12:20Z',  
                                     '1999-11-28T09:00:05Z')
```

## get-years-from-date-difference

Calculates the difference between the year part of two given dates and returns the difference.

The `getyearsfromdatedifference` function returns the years component in the difference between two dates as an `yearMonthDuration`.

### Syntax

```
get-years-from-date-difference (strDate1, strDate2)
```

### Parameters

`strDate1` is a `string` that represents one date to compare and is specified in the format `yyyy-mm-dd`.

`strDate2` is a `string` that represents the other date to compare and is specified in the format `yyyy-mm-dd`.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A `number` that holds the difference between the year part of the given dates.

### Example

The following expression returns a number of value 3:

```
get-years-from-date-difference('2003-10-30','1999-11-28')
```

## format-date

Returns the date for a given date and locale.

### Syntax

```
format-date(strDate, (language, country, variant, timezone)?)
```

### Parameters

`strDate` is a `string` that represents the date in the format `yyyy-mm-dd`.

You can also specify zero or one set of parameters that identifies the locale of the equivalent date:

- `language` is a `string` that identifies the language of the locale.
- `country` is a `string` that identifies the country of the locale.
- `variant` identifies a vendor or web browser.
- `timezone` identifies the time zone of the locale.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A `string` that represents the equivalent date in the format `yyyy-mm-dd`.

## format-dateTime

Returns a `string` that represents the date and time in the format that is specific to a given locale and `timezone`.

### Syntax

```
format-dateTime(strDateTime, (language, country, variant, timezone)?)
```

### Parameters

`strDateTime` is a string that represents the date and time in the format `yyyy-mm-ddThh:mm:ssZ`.

You can also specify zero or one set of parameters that identifies the locale of the equivalent date:

- `language` is a string that identifies the language of the locale.
- `country` is a string that identifies the country of the locale.
- `variant` identifies a vendor or web browser.
- `timezone` identifies the time zone of the locale.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A string that represents the localized date and time in the format `yyyy-mm-ddThh:mm:ssZ`.

## parse-date

Translates a given date to a given locale and time zone.

### Syntax

```
parse-date(strDate, (language, country, variant, timezone)?)
```

### Parameters

`strDate` is a string that represents the date in the format `yyyy-mm-dd`.

You can also specify zero or one set of parameters that identifies the locale of the equivalent date:

- `language` is a string that identifies the language of the locale.
- `country` is a string that identifies the country of the locale.
- `variant` identifies a vendor or web browser.
- `timezone` identifies the time zone of the locale.

For more information about date and time parameters, see ["Date and time parameters" on page 122](#).

### Returns

A string that represents the equivalent date in the format `yyyy-mm-dd`.

## parse-dateTime

Translates a given date and time to a given locale and timezone.

### Syntax

```
parse-dateTime(strDateTime, (language, country, variant, timezone)?)
```

### Parameters

`strDateTime` is a string that represents the date and time in the format `yyyy-mm-ddThh:mm:ssZ`.

You can also specify zero or one set of parameters that identifies the locale of the equivalent date:

- `language` is a string that identifies the language of the locale.
- `country` is a string that identifies the country of the locale.
- `variant` identifies a vendor or web browser.
- `timezone` identifies the time zone of the locale.

For more information about date and time parameters, see [“Date and time parameters” on page 122](#).

### Returns

A string that represents the translated date and time in the format `yyyy-mm-ddThh:mm:ssZ`.

## time-equal

Compares two given time values to see if they are equal.

### Syntax

```
time-equal(strTime1, strTime2)
```

### Parameters

`strTime1` is a string that represents one time to compare and is specified in the format `hh:mm:ss`.

`strTime2` is a string that represents the other time to compare and is specified in the format `hh:mm:ss`.

### Returns

A Boolean value of `true` if the given time values are equal, otherwise `false`.

## time-greater-than

Compares two given time values to see if one is greater than the other.

### Syntax

```
time-greater-than(strTime1, strTime2)
```

### Parameters

`strTime1` is a string that represents one time to compare and is specified in the format `hh:mm:ss`.

`strTime2` is a string that represents the other time to compare and is specified in the format `hh:mm:ss`.

### Returns

A Boolean value of `true` if the time that `strTime1` represents is greater than the time that `strDate2` represents, otherwise `false`.

## time-less-than

Compares two given time values to see if one is less than the other.

### Syntax

```
time-less-than(strTime1, strTime2)
```

### Parameters

`strTime1` is a `string` that represents one time to compare and is specified in the format `hh:mm:ss`.

`strTime2` is a `string` that represents the other time to compare and is specified in the format `hh:mm:ss`.

### Returns

A Boolean value of `true` if the time that `strTime1` represents is less than the time that `strDate2` represents, otherwise `false`.

## Document object functions

This section describes functions that operate on document objects that are available in expressions.

### getDocLength

Retrieves the file size of a document.

### Syntax

```
getDocLength(com.adobe.idp.Document)
```

### Parameters

`com.adobe.idp.Document` is a reference to a serialized document.

### Returns

A `Long` that contains the file size of the document in bytes. Returns 0 if the parameter does not evaluate to a document.

### Example

A workflow variable named `vardocument` references a PDF file that was serialized after being submitted as a work item form. The following example returns 455343, the size of the referenced document in bytes:

```
getDocLength(/process_data/@vardocument)
```

### getDocAttribute

Retrieves the value of the specified document attribute.

### Syntax

```
getDocAttribute(com.adobe.idp.Document, String)
```

### Parameters

`com.adobe.idp.Document` is a reference to a serialized document. `String` is a string that holds the name of the document attribute to retrieve. For information about available attributes, see [“Document attributes for attachments” on page 66](#).

### Returns

The value of the attribute specified by `String`. The attribute determines the type of object returned. Returns an empty string if the document variable parameter does not evaluate to a `com.adobe.idp.Document` object. Returns `null` if `String` does not hold a valid attribute name.

### Example

A document variable named `varattachment` references a PDF file that was attached to a task. The following example returns an `integer` that represents the access permissions of the attachment:

```
getDocAttribute(/process_data/@varattachment, "wspermission")
```

## setDocAttribute

Sets the value of a document attribute.

### Syntax

```
setDocAttribute(com.adobe.idp.Document, String, String)
```

### Parameters

`com.adobe.idp.Document` is a reference to a serialized document. The first `String` parameter is a string that holds the name of the document attribute to set. The second `String` parameter is a string that holds the value to set the document attribute to. For information about available attributes, see [“Document attributes for attachments” on page 66](#).

### Returns

A reference to the input `com.adobe.idp.Document` object with the attribute specified by the first `String` parameter set to the value specified by the second `String` parameter.

Returns a reference to an empty `com.adobe.idp.Document` object with the specified attribute set to the specified value if the first parameter does not evaluate to a document.

Returns a reference to the unchanged `com.adobe.idp.Document` object if the second or third parameters do not evaluate to a `String`.

### Example

The following example returns the description of a document that the document variable named `vardoc1` references:

```
setDocAttribute(/process_data/@vardoc1, "wsdescription", "An example  
description.")
```

## getDocContent

Retrieves the content type of a document. The content type identifies the type of the document, similar to the `mime-type` property of documents that are sent over the web. For example, a PDF document has a document type of `application/pdf`.

The document type of a document can have no value or can have any value that is set with the `setDocContent` function, when the document was attached to a task. For more information, see [“setDocContent” on page 112](#).

### Syntax

```
getDocContent (com.adobe.idp.Document)
```

### Parameters

`com.adobe.idp.Document` is a serialized document that is referenced by a document variable.

### Returns

A `String` that contains the content type. Returns an empty `String` if the parameter does not evaluate to a document.

## setDocContent

Sets the content type of a document.

### Syntax

```
setDocContent (com.adobe.idp.Document, String)
```

### Parameters

`com.adobe.idp.Document` is a serialized document that is referenced by a document variable. `String` is a string that contains the content type that you want to set for the document.

### Returns

A reference to a copy of the `com.adobe.idp.Document` object that was used as a parameter. Returns a new, empty document with the specified content type if the first parameter does not evaluate to a `com.adobe.idp.Document`.

## Miscellaneous

This section describes miscellaneous functions that are available in expressions.

### deserialize

Converts the text of a given node into an XML document.

#### Syntax

```
deserialize (node)
```

### Parameters

`node`.

### Returns

An `org.w3c.dom.Document` object that represents the XML document.

## is-null

Checks whether the value for a workflow variable is stored as Null in the database. In XPath, Null values are represented as empty string values.

### Syntax

```
is-null (node)
```

### Parameters

The `node` that represents the POF attribute.

### Returns

A `boolean` value of `true` if the attribute is Null, otherwise `false`.

## serialize

Returns the string XML representation of a given node. You can specify whether the string includes the XML declaration as the first line.

### Syntax

```
serialize (node, bOmitXML)
```

### Parameters

A `node` and an optional `boolean` `bOmitXML`. The `node` represents the node to represent as a string. If a value of `true` is provided for `bOmitXML`, the first line in the string representation includes the XML declaration. If `bOmitXML` is `false`, the XML declaration is not included. If `bOmitXML` is not specified, the XML declaration is provided.

### Returns

A `string` that represents the contents of the given node.

## Node set functions

This section describes the functions that operate on schema nodes that are available in expressions.

### count

Returns the number of nodes in a given node set.

### Syntax

```
count (node-set)
```

### Parameters

A `node-set` object.

### Returns

A `number` that holds the number of nodes.

## last

Returns a number equal to the context size from the expression evaluation context.

### Syntax

```
last()
```

### Parameters

None.

### Returns

`number`.

## position

Returns a number equal to the context position from the expression evaluation context.

### Syntax

```
position()
```

### Parameters

None.

### Returns

`number`.

## sum

Returns the sum, for each node in a given `node-set`, of the result of converting the string-values of the node to a number.

### Syntax

```
sum(node-set)
```

### Parameters

`node-set`.

### Returns

`number`.

## Number functions

This section describes the number functions that are available in expressions.

### ceiling

Returns the smallest integer value that is not less than a given numeric value. Values are smaller in the direction of negative infinity.

The ceiling function returns the smallest (closest to negative infinity) number that is not less than the argument and that is an integer.

#### Syntax

```
ceiling(Expression)
```

#### Parameters

`Expression` is an expression that returns a numeric value.

#### Returns

An integer value.

### floor

Returns the largest integer value that is not greater than a given numeric value. Values are larger in the direction of positive infinity.

#### Syntax

```
floor(number)
```

#### Parameters

`number` is a numeric value.

#### Returns

An integer value.

### number

The number function converts a given numeric value to a number.

#### Syntax

```
number(object?)
```

#### Parameters

Zero or one object.

#### Returns

A number value.

## round

Returns a whole number that is closest to a given numeric value.

### Syntax

```
round(number)
```

### Parameters

`number` is any numeric value:

- If `number` is NaN, NaN is returned.
- If `number` is positive infinity, positive infinity is returned.
- If `number` is negative infinity, negative infinity is returned.
- If `number` is positive zero, positive zero is returned.
- If `number` is negative zero, negative zero is returned.
- If `number` is less than zero and greater than or equal to -0.5, negative zero is returned.
- If `number` is an equal distance from two different numbers, the number that is closest to positive infinity is returned.

### Returns

An integer value.

## String Functions

This section describes the string functions that are available in expressions.

### concat

Concatenates two or more given strings.

#### Syntax

```
concat(string, string, string*)
```

#### Parameters

Two or more string values, separated by commas.

#### Returns

String.

### contains

Checks whether a given string contains a second given string.

#### Syntax

```
contains(string, string)
```

### Parameters

Two string values separated by commas. `contains` checks if the first string contains the second string.

### Returns

A Boolean value of `True` if the first parameter contains the second parameter, otherwise `False`.

## lower-case

Converts all uppercase characters in a given string to the lowercase equivalent, if one exists.

### Syntax

```
lower-case(string)
```

### Parameters

A string value.

### Returns

String.

## normalize-space

Removes leading and trailing whitespace characters from the given string and also replaces consecutive whitespace characters with a single space character.

### Syntax

```
normalize-space(string?)
```

### Parameters

Zero or one string to normalize.

### Returns

String.

## starts-with

Checks that the first characters of a given string match the characters of another given string.

### Syntax

```
starts-with(string, string)
```

### Parameters

Two string values separated by a comma. `starts-with` checks that the first parameter starts with the second parameter.

### Returns

A Boolean value of `True` if the first parameter starts with the second parameter, otherwise `False`.

## string

Converts an object to a string.

### Syntax

```
string(object?)
```

### Parameters

Zero or one objects.

### Returns

String.

## substring-after

Checks that a given string contains the characters of a second given string and returns the part of the first string that follows the occurrence of the second string.

### Syntax

```
substring-after(string, string)
```

### Parameters

Two string values separated by a comma. `substring-after` checks that the first string contains the second string.

### Returns

A string that contains the characters from the first parameter that follow the occurrence of the second parameter in the first parameter. If the first parameter does not include the characters from the second parameter, `substring-after` returns an empty string.

### Example

The following expression returns 04/01:

```
substring-after("2005/04/01", "/")
```

## substring-before

Checks that a given string contains the characters of a second given string and returns the part of the first string that precedes the occurrence of the second string.

### Syntax

```
substring-before(string, string)
```

### Parameters

Two string values separated by a comma. `substring-before` checks that the first string contains the second string.

### Returns

A string that contains the characters from the first parameter that precede the occurrence of the second parameter in the first parameter. If the first parameter does not include the characters from the second parameter, `substring-before` returns an empty string.

### Example

The following expression returns 2005:

```
substring-before("2005/04/01", "/" )
```

## substring

Returns a substring of a given string. The substring is specified by the index of the given string where the substring begins and the length of the substring. The length can be specified either explicitly or as the remaining characters after the beginning of the substring.

### Syntax

```
substring(string, number, number?)
```

### Parameters

A string and one or more numbers separated by commas. The string is the parameter from which the substring is extracted. The first number is the index of the string parameter where the substring begins. The second number, if provided, is the length of the substring. If the third parameter is not provided, the substring includes the characters from the index to the end of the given string.

### Returns

String.

### Example

For example, the first of the following two expressions returns 234, and the second expression returns 6789:

```
substring("123456789", 2, 3)  
substring("123456789", 6)
```

## string-length

Calculates the number of characters in the given string.

### Syntax

```
string-length(string?)
```

### Parameters

Zero or one strings.

### Returns

Number.

## translate

In a given string, replaces the occurrence of any of the characters of a second given string with the corresponding characters of a third given string or removes the occurrence of characters of a second given string if no corresponding character exists in the third given string.

### Syntax

```
translate(string, string, string)
```

### Parameters

Three `string` values separated by commas. The first string is the string in which characters are replaced. The second string contains the characters whose occurrences in the first string are replaced. The third string contains the replacement characters that correspond with the characters in the second string.

The characters in the second string correspond with the characters in the third string that have the same index:

- If the second string is longer than the third string, the characters in the second string that have an index that is greater than the length of the third string have no corresponding characters.
- If the third string is longer than the second string, the characters in the third string that have an index that is greater than the length of the second string are ignored.
- If a character occurs more than once in the second string, the first occurrence of the character determines how that character is replaced in the first string.

### Returns

A string that contains the first parameter with the replaced characters.

### Example

The following expression includes a second parameter that includes characters that all correspond with a character in the third parameter. The expression returns `BAr`:

```
translate("bar", "abc", "ABC")
```

The following expression includes a second parameter that is longer than the third parameter. The expression returns `AAA`:

```
translate("--aaa--", "abc-", "ABC")
```

The following expression includes a second parameter that includes multiple occurrences of a character. The expression returns `BAr`:

```
translate("bar", "aba", "ABC")
```

## upper-case

Converts all lowercase characters in a given string to the uppercase equivalent, if one exists.

### Syntax

```
upper-case (string)
```

### Parameters

One string.

### Returns

String.

## Operators

The following table provides the meaning of each operator that is available in expressions.

Operator	Meaning
and	logical AND
or	logical OR
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
!=	does not equal
=	equals
div	division
mod	modulus
*	multiply
+	add
-	subtract

## Date and time parameters

The following table describes the values you can specify for parameters that are related to expressing a date, a time, or information about a locale.

Parameter	Description
<code>strDate</code>	<p>A string that represents a calendar date in the format <code>yyyy-mm-dd</code>:</p> <ul style="list-style-type: none"> <li>• <code>yyyy</code> is the year.</li> <li>• <code>mm</code> is the number that represents the month.</li> <li>• <code>dd</code> is the calendar day of the month.</li> </ul>
<code>strTime</code>	<p>A string that represents the time in the format <code>hh:mm:ssZ</code>:</p> <ul style="list-style-type: none"> <li>• <code>hh</code> is the hour, using the 24-hour format.</li> <li>• <code>mm</code> is the minutes</li> <li>• <code>ss</code> is the seconds</li> <li>• <code>Z</code> is the time zone. See the <code>time zone</code> parameter in this table for more information.</li> </ul>
<code>strDateTime</code>	<p>A string that represents the date and time in the format <code>yyyy-mm-ddThh:mm:ss</code>.</p>
<code>country</code>	<p>A two-letter code that identifies a country, as specified by ISO 3166. For example, <code>FR</code> identifies France, and <code>US</code> identifies the United States of America.</p>
<code>language</code>	<p>A two-letter code that identifies a language, as specified by ISO 639. For example <code>en</code> identifies the English language and <code>fr</code> identifies the French language.</p>
<code>time zone</code>	<p>A sequence of characters that identifies the time zone relative to Coordinated Universal Time (UTC), as specified in ISO 8601. <code>time zone</code> is of the form <code>+hhmm</code>, <code>-hhmm</code>, or <code>Z</code>:</p> <ul style="list-style-type: none"> <li>• Specify <code>+hhmm</code> if your locale is <code>hh</code> hours and <code>mm</code> minutes ahead of UTC.</li> <li>• Specify <code>-hhmm</code> if your locale is <code>hh</code> hours and <code>mm</code> minutes behind UTC.</li> <li>• Specify <code>Z</code> if your locale is in the UTC time zone.</li> </ul> <p>For example, if your locale is in a time zone that is five hours ahead of UTC, then <code>time zone</code> is <code>+500</code></p>
<code>variant</code>	<p>A code that specifies a vendor or web browser. For example, <code>WIN</code> specifies Microsoft® Windows® and <code>MAC</code> specifies Apple Macintosh. To specify two variants, separate them with an underscore and put the most important variant first.</p>

# C

## Migrating Workflows from Older Versions

This appendix provides information about using workflows created with earlier versions of LiveCycle Workflow Designer with the latest version of LiveCycle Workflow.

Generally, workflows created with earlier versions of LiveCycle Workflow Designer function correctly when deployed to later versions of LiveCycle Workflow. However, to use the features that are introduced in new versions of LiveCycle Workflow, you may need to make changes to the older workflows.

### Migrating workflows from LiveCycle Workflow 7.0.1

The User QPAC that is provided with LiveCycle Workflow 7.2 includes new features for adding attachments to tasks and for storing task attachments in `list` variables. (See [“Adding attachments to tasks” on page 65](#) and [“Saving task attachments” on page 66](#).) You should update the older User component with the User QPAC that is provided with LiveCycle Workflow 7.2. (See [“Updating components” on page 75](#).) After updating to the User QPAC, you can still use your older workflows and, if you do not use the new features, the workflows behave the same as before when executed on LiveCycle Workflow Server.

If you want to change older workflows to use the new features provided with the User QPAC, you must create a copy of the older workflow, make the changes in a the new version of the workflow, and leave the older version unchanged:

- Process instances continue to execute against the version of the workflow that was activated when the process instances were initiated.
- Process instances that were initiated when the older version of the workflow was activated must continue to execute against the older version of the workflow.
- If you change the older version of the workflow to use the new attachment features, errors can occur in the existing process instances that can cause them to stall.

After you update to the User QPAC, you can open the Action Properties dialog box of the User actions of older workflows to see the following aspects of the action properties:

- All of the settings that you specified for the User action properties remain the same.
- The Routes and Attachments tab includes the new features for adding and retrieving task attachments.

#### ► To use the new attachment features in older workflows:

1. Update the User component with the User QPAC provided in LiveCycle Workflow 7.2. (See [“Updating components” on page 75](#).)
2. Create a copy of the older workflow to create a new version of it. (See [“Creating copies of workflows” on page 77](#).)
3. Open the new version of the workflow and make the necessary changes to the User action properties.
4. Deploy and activate the new version of the workflow. (See [“Deploying workflows” on page 73](#) and [“Activating workflows” on page 73](#).)

# Glossary

---

This glossary contains terminology definitions that are specific to documentation for the Adobe LiveCycle suite of products. These terms may have different meanings in other contexts, but they have restricted meanings in this documentation.

## A

### accessible forms

Forms that users with disabilities or vision impairment can view and fill using screen readers and other assistive technologies. See also *tagged Adobe PDF form*.

### Acrobat form

A PDF document, created in Acrobat, that contains one or more form fields. The PDF document may also contain non-form content.

### action

In a workflow, the representation of a step in a business process.

### Adobe certified document

A document that is signed with a specific Adobe root certificate. An Adobe certified document provides a strong guarantee as to the authenticity and immutability of the document. See also *certificate*.

### Adobe document services

Adobe document services extend the value of core enterprise systems to ensure more secure, reliable, and efficient use of business-critical information across the extended enterprise. Adobe document services include the Adobe LiveCycle suite of products and the Acrobat product line.

### application

A set of generally interdependent files that make up a self-contained application that Adobe LiveCycle products can run. Applications may include files such as form designs, Java Server Pages, HTML pages, servlets, and images.

## B

### branch

A branch contains a set of actions interconnected by routes, representing a sequential path taken by a process at execution. The branch always determines the behavior of the workflow.

## C

### certificate

An electronic file that establishes your identity, by binding your identity to your public key, when doing business or other transactions on the web. A *certificate* (or sometimes called a *digital certificate*) is issued by a certificate authority (CA). See also *Adobe certified document*.

### client

The requesting program in a client/server relationship. A web browser is an example of a client application.

### credential

The file that contains a private key. (The corresponding public key is contained in a certificate.) A private key is what one principal presents to another used to establish identity in decryption and signing operations. Credentials are issued by an authentication agent or a certification authority. See also *certificate*.

## D

### deadline

The time by which a person must complete a work item. Deadlines are properties of workflows.

### dynamic form

A form that can expand or contract to reflect the amount of incoming data. See also *interactive form*.

## E

### ebXML

Electronic Business using eXtensible Markup Language (ebXML). A modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. See also *registry*.

### encryption

The conversion of data into a format (called a ciphertext) that cannot be easily understood by unauthorized persons. The conversion is done using an encryption algorithm.

## F

### form

An electronic document that captures and delivers data. A person may add data to an interactive form, or a server process may merge a form design with data to produce a form.

### form authors

LiveCycle Designer users who are capable of creating fillable forms to be used in Acrobat or Adobe Reader®, and simple non-interactive forms for deployment to LiveCycle Forms. See also *form developers*.

### FormCalc

A calculation language similar to that used in common spreadsheet software that facilitates form design without requiring a knowledge of traditional scripting techniques or languages.

### form design

The design-time version of a form that an author or developer creates in LiveCycle Designer.

### form developers

LiveCycle Designer users who are capable of creating complex form-based applications for use in different environments. See also *form author*.

### form object

A form element, such as a button or text field, that you can place on a form. An object has its own set of properties and events.

## I

### interactive form

A form that a person can interact with and complete electronically.

## N

### non-interactive form

A form that a person can view or print but cannot fill electronically. Non-interactive forms can be merged or prepopulated with data, but the data cannot be changed by a user. Non-interactive forms are designed for output.

## P

### PDF document

Portable Document Format. A file conforming to the PDF specification as published by Adobe Systems or a file conforming to the XDP specification, containing exactly one PDF packet and no more than one each XFA-Template, XFA-Configuration, XFA-SourceSet, and Annotations packets.

### PDF form

A form that users can access in Acrobat and Adobe Reader. PDF forms are either interactive or non-interactive.

### permissions

Security settings applied, for example, to restrict users from opening, editing, printing, or removing encryption from a PDF file. Permissions cannot be changed unless the user has the Permissions password. Permissions can be set in LiveCycle Designer, Acrobat, LiveCycle Document Security, and other products.

## **policy**

Defines a set of security permissions and users who can access a PDF document to which the policy is applied. Policies are created using LiveCycle Policy Server and can be applied to documents using LiveCycle Policy Server, LiveCycle Document Security, or Acrobat 7.0 or later.

## **prepopulated form**

A form that appears to the user with some or all fields automatically populated with data.

## **Q**

### **QPAC**

Quick Process Action Component. A JAR file that contains server-side code and client-side code for use with LiveCycle Workflow. In LiveCycle Workflow Designer, QPACs provide action components that can be added to workflows to represent a step in a process. LiveCycle Workflow Server interprets each action of the workflow and executes the server-side code of the corresponding QPACs. QPACs enable LiveCycle Workflow to interact with other Adobe LiveCycle products, such as LiveCycle Forms and LiveCycle Barcoded Forms.

## **R**

### **reminder**

A notification sent to people that reminds them to complete a work item. Reminders are properties of workflows.

### **render**

An action whereby LiveCycle Forms merges a form design, possibly with data, to display a form in PDF or HTML format in a browser.

### **registry**

An ebXML-compliant repository of shared information that provides services for the purpose of enabling business process integration between interested parties. See also *ebXML* and *repository*.

## **repository**

The underlying storage area within a registry. See also *registry*.

### **restricted document**

A PDF document with password security restrictions (permissions) that prevent the document from being opened, printed, or edited.

### **rights-enabled document**

A PDF document that includes security extensions that enable Adobe Reader users to fill forms, add comments, and sign documents.

### **route**

The path between actions on a workflow. Routes determine the order in which LiveCycle Workflow Server executes actions at run time.

### **run time**

For form rendering, the time when an application or server process retrieves a form design, possibly merges it with data, and presents it to a user for viewing or filling.

## **S**

### **split**

A segment in a workflow that contains one or more branches. The branches in a split are executed in parallel.

### **static form**

A form that remains exactly as it was designed. The layout does not change according to the amount of incoming data.

### **subform**

An object that can act as a container for form objects and other subforms. A subform helps to position form objects relative to each other and provide structure in dynamic form designs. A subform can also provide a reference point, when binding data to a form, by restricting the scope for a field so that it matches that of the corresponding data node.

## T

### **tagged Adobe PDF form**

Includes a logical structure and a set of defined relationships and dependencies among the various elements, plus additional information that permits reflow. See also *accessible forms*.

### **turnkey**

An installation option that automatically installs and configures the LiveCycle product files, JBoss application server, and MySQL database, and deploys the product files to JBoss. After you perform a turnkey installation, the LiveCycle product is ready to use.

## U

### **usage rights**

Rights that extend the functionality of Adobe Reader and enable users to save forms with data, add comments, and sign documents.

## W

### **workflow**

The electronic representation of a business process. Workflows are created using LiveCycle Workflow Designer.

## X

### **XDP file**

XML Data Package. LiveCycle Designer saves form designs as either XDP files or PDF files. LiveCycle Forms uses XDP files to render forms in PDF or HTML format.

### **XML Forms Architecture**

Represents the underlying technology beneath the Adobe XML forms solution. It enables the construction of robust and flexible form-based applications for use on either the client or the server.

### **XML form**

A PDF form that conforms to the Adobe PDF specification and the Adobe XML Forms Architecture. XML forms are typically created in LiveCycle Designer. XML forms can have the file name extension .xdp or .pdf.

# Index

## A

Acrobat forms  
  description 81  
  field names 84  
  required fields in forms 82  
action properties, descriptions 86  
actions  
  adding and deleting 23  
  adding to the Components palette 75  
  arranging in the Components palette 75  
  branch compatibility 69  
  Chained Process action 41  
  Decision Point 36  
  description 21  
  long-lived 72  
  one of many 33, 36  
  rule evaluation order 35  
  Script 38  
  Set Value 32  
  specifying icons 76  
  Stall action 72  
  stalled 70  
  start action 33  
  transaction-aware 71  
  transactions 71  
  types 24  
  User action 57  
  viewing properties 86  
  Wait Point action 41  
activating  
  replacing active workflows 77  
  workflows 73  
add-days-to-date function 90  
add-days-to-dateTime function 90  
add-hours-to-dateTime function 91  
adding  
  actions 23  
  attachments 65  
  components 74  
  process categories 15  
  process types 16  
  routes 24  
  rules 35  
  splits 37  
  to the Processes palette 15  
  variables 28  
  workflows 17  
adding nodes, form data 52  
add-minutes-to-dateTime function 91  
add-months-to-date function 92  
add-months-to-dateTime function 92  
add-seconds-to-dateTime function 93  
add-years-to-date function 93

add-years-to-dateTime function 94  
AND-WAIT join operator 37  
architecture of forms 85  
assigning tasks  
  expressions 59  
  to users 58  
Asynchronous branches  
  action compatibility 69  
  stalled 70  
Asynchronous workflows 68  
  about 68  
  branch compatibility 69  
attachments  
  adding at run time 65  
  description 64  
  document attributes 66  
  in email 67  
  passing between User actions 65  
  saving 66  
  user-added 65  
attributes, document 66  
auto-hide mode, palettes 13

## B

backing up workflows 78  
binary variable 29  
boolean function 88  
boolean variable 29  
branch types  
  action compatibility 69  
  stalled branches 70  
branches  
  adding 37  
  description 22  
  executing in parallel 36  
  stalled 70  
  transactions 71  
  workflow compatibility 69  
business activity monitor, enabling for workflows 15, 17

## C

categories  
  arranging components 75  
  components 74  
  processes 14  
ceiling function 115  
Chained Process action 41  
changing  
  route labels 25  
choice list  
  description 55  
  matching with route names 57  
  route names 57

- closing workflows 18
- compatibility
  - actions and branches 69
  - workflows and branches 69
- completing interactive Script actions 39
- component categories 74
- Component ID 86
- components
  - Chained Process 41
  - Decision Point 36
  - deploying 75
  - description 22, 74
  - icons 76
  - moving to different categories 75
  - redeploying 75
  - removing 76
  - Script 38
  - Set Value 32
  - Split and Branch 36
  - types 24
  - Wait Point 41
- Components palette
  - arranging components 75
  - custom icons 76
  - description 22
- computational errors
  - description 70
  - stalled branches and actions 70
- concat function 116
- conditions, multiple rules 34
- connecting to LiveCycle Workflow Server 12
- consistent field names, forms 85
- contains function 116
- content management, workflows 19
- copying workflows 77
- count function 113
- creating expressions 43
- current-date function 94
- current-dateTime function 95
- current-time function 95

**D**

- dashed routing lines 22
- data attribute, form variables 49, 51
- data collections, XPath expressions 45, 46
- Data Mappings 41
- data tree
  - about 42
  - adding nodes 52
  - form data 51
- date and time data
  - form fields 85
- date variable 29
- date-equal function 95
- date-greater-than function 96
- date-less-than function 96
- dateTime variable 29
- dateTime-equal function 96
- dateTime-greater-than function 97

- dateTime-less-than function 97
- deadlines
  - description 60
  - setting 62
- decimal variable 29
- Decision Point action
  - using with initiation forms 48
- Decision Point action, using 36
- delaying action execution 41
- deleting actions 23
- deleting process categories 18
- deployed workflows 15
- deploying
  - multiple workflows 77
  - QPACs 74, 75
  - workflows 73
- deployment state, workflows 74
- deserialize function 112
- designing forms 81
- docked palettes 13
- document variable 29, 53
- documents
  - attributes 66
  - in workflows 47
- double variable 29

## E

- editing workflows 18
- email
  - initiating process instances 84
  - sending reminders 63
  - task attachments 67
- errors, computational and situational 70
- escalating tasks
  - description 60
  - setting 61
- evaluation order, rules 35
- executing actions, order 33
- Execution Transaction type 86
- exporting workflows 78
- expressions
  - creating 43
  - description 42
  - examples 44
  - form fields 51
  - in rules 34
  - operators reference 121
  - specifying users 59
  - XPathExpression Builder 42

## F

- false function 88
- fields
  - Acrobat forms 82
  - layout 85
  - naming 85
  - XML forms 81
- float variable 29
- floating palettes 13

- floor function 115
- form data 51
  - adding nodes 52
- form data, storing 49
- form fields
  - date and time values 85
  - in expressions 51
  - naming 85
  - naming, Acrobat 84
  - required for Acrobat forms 82
  - rules for names on Acrobat forms 84
- form properties
  - Acrobat forms 82
- form schemas 51
- form variables
  - input and output 55
  - storing 49
  - types 29
- format-date function 107
- format-dateTime function 107
- forms
  - Acrobat forms in LiveCycle Form Manager 82
  - adding 54
  - architecture 85
  - design requirements 81
  - fields for XML forms 81
  - initiating processes 48
  - several 84
  - starting processes 84
  - User actions 55
- forms, in workflows 47
- form-url, form variables 49
- function
  - add-days-to-date 90
  - add-days-to-dateTime 90
  - add-hours-to-dateTime 91
  - add-minutes-to-dateTime 91
  - add-months-to-date 92
  - add-months-to-dateTime 92
  - add-seconds-to-dateTime 93
  - add-years-to-date 93
  - add-years-to-dateTime 94
  - boolean 88
  - ceiling 115
  - concat 116
  - contains 116
  - count 113
  - current-date 94
  - current-dateTime 95
  - current-time 95
  - date-equal 95
  - date-greater-than 96
  - date-less-than 96
  - dateTime-equal 96
  - dateTime-greater-than 97
  - dateTime-less-than 97
  - deserialize 112
  - false 88
  - floor 115
  - function (Continued)
    - format-date 107
    - format-dateTime 107
    - get-collection-size 90
    - get-day-from-date 98
    - get-days-from-date-difference 98
    - get-days-from-dateTime-difference 99
    - getDocAttribute 110
    - getDocContent 112
    - getDocLength 110
    - get-hours-from-dateTime 99
    - get-hours-from-dateTime-difference 100
    - get-hours-from-time 100
    - get-map-keys 89
    - get-minutes-from-dateTime 100
    - get-minutes-from-dateTime-difference 101
    - get-minutes-from-time 101
    - get-month-from-dateTime 102
    - get-months-from-date-difference 102
    - get-months-from-dateTime-difference 103
    - get-seconds-from-dateTime 103
    - get-seconds-from-dateTime-difference 104
    - get-seconds-from-time 104
    - get-timezone-from-date 104
    - get-timezone-from-dateTime 105
    - get-timezone-from-time 105
    - get-year-from-date 105
    - get-year-from-dateTime 106
    - get-years-from-date-difference 106
    - get-years-from-dateTime-difference 106
  - is-null 113
  - last 114
  - lower-case 117
  - normalize-space 117
  - not 88
  - number 115
  - parse-date 108
  - parse-dateTime 108
  - position 114
  - round 116
  - serialize 113
  - setDocAttribute 111
  - setDocContent 112
  - starts-with 117
  - string 118
  - string-length 119
  - substring 119
  - substring-after 118
  - substring-before 118
  - sum 114
  - time-equal 109
  - time-greater-than 109
  - time-less-than 110
  - translate 120
  - true 89
  - upper-case 121
- function library 42
- functions, expressions 43

## G

- get-collection-size function 90
- get-day-from-date function 98
- get-days-from-date-difference function 98
- get-days-from-dateTime-difference function 99
- getDocAttribute function 110
- getDocContent function 112
- getDocLength function 110
- get-hours-from-dateTime function 99
- get-hours-from-dateTime-difference function 100
- get-hours-from-time function 100
- get-map-keys function 89
- get-minutes-from-dateTime function 100
- get-minutes-from-dateTime-difference function 101
- get-minutes-from-time function 101
- get-month-from-dateTime function 102
- get-months-from-date-difference function 102
- get-months-from-dateTime-difference function 103
- get-seconds-from-dateTime function 103
- get-seconds-from-dateTime-difference function 104
- get-seconds-from-time function 104
- get-timezone-from-date function 104
- get-timezone-from-dateTime function 105
- get-timezone-from-time function 105
- get-year-from-date function 105
- get-year-from-dateTime function 106
- get-years-from-date-difference function 106
- get-years-from-dateTime-difference function 106
- groups, assigning tasks 59

## H

- handling documents 53
- hidden palettes 13
- hiding workflows 19

## I

- icons, for actions 76
- importing workflows 78
- in attribute
  - Chained Process 41
  - form variables 49
- in attribute, variables 28
- Init-Form palette 48
- init-forms 48
- initiating
  - processes with forms 48
  - processes, email 84
  - subprocesses 41
  - workflows, activating 73
- input form variable 55
- int variable 29
- Interactive, action property 86
- is-null function 113

## J

- join conditions 34

## K

- keyboard shortcuts, script editor 40

## L

- labels
  - for workflows 26
  - routes 25
- last function 114
- LDAP groups, assigning tasks 59
- list variables
  - accessing data items 45
  - expressions as indexes 46
  - properties 29
- locking workflows 20
- logging in 12
- long variable 29
- long-lived actions 72
- Long-lived, action property 86
- lower-case function 117

## M

- main branch type 15
- map variables
  - accessing data items 45
  - expressions as keys 46
  - properties 29
- mapping routes to submit choices 57
- marked for deletion
  - process categories 18
- marked for deletion, components 76
- menu bar 12
- Merge data into form, Acrobat forms 82
- Message Transaction Type 86
- migrating workflows 79, 123
- modifying
  - deployed workflows 78
  - process categories 15
  - process types 16
  - route labels 25
  - rules 35
  - workflow properties 17
- monitor access 15
- moving palettes 13
- multiple routes 33, 55

## N

- naming
  - fields 84
- new
  - actions 23
  - expressions 43
  - process categories 15
  - process types 16
  - routes 24
  - rules 35
  - splits 37
  - variables 28
  - workflows 17

normalize-space function 117  
not function 88  
notifications  
    reminders 63  
NO-WAIT join operator 37  
number function 115

## O

objects, in script 38  
opening workflows 18  
operator library 42  
operators, expressions 43, 121  
order of execution, actions 33  
organizing process types 14  
OR-WAIT join operator 37  
out attribute  
    Chained Process 41  
    form variables 49  
    variables 28  
output form variable 55

## P

palettes  
    about 12  
    Components palette 22  
    Components, adding components 74  
    customizing 13  
    hiding 13  
    Init-Form 48  
    Processes palette 14  
parallel execution, branches 36  
parse-date function 108  
parse-dateTime function 108  
password 12  
PDF documents, handling 53  
position function 114  
printing workflows 26  
procedure, creating workflows 11  
process categories 14  
    new 15  
    removing 18  
process complexity and workflow type 68  
process creator, assigning tasks 58  
process data  
    by workflow type 68  
    passing to subprocesses 41  
    saving in variables 27  
    testing script 39  
process data tree example 44  
process participants, assigning tasks 58  
process properties, testing script 39  
process steps  
    See actions  
process types 14  
    new 16  
    organizing 14  
    removing 18

Processes palette  
    about 14  
    adding items 15  
processes, invoking with email 84  
production environments, migrating workflows 79  
properties, viewing for actions 86  
protecting workflows 19

## Q

QPACs  
    categories 74  
    deploying 74, 75  
    redeploying 75  
Quick Process Action Components  
    See QPACs

## R

redeploying components 75  
reminders  
    description 60  
    setting 63  
replacing active workflows 77  
required attribute, variables 28  
revealing hidden palettes 13  
rolling back actions 71  
round function 116  
route names  
    matching with choice lists 57  
    route selection 57  
route selection 34  
routes  
    adding 24  
    description 21  
    evaluation order 35  
    labels 25  
    user-driven 55  
Rule Evaluation Order 35  
rules  
    adding 35  
    description 34  
    evaluation order 35  
    join conditions 34  
running script 40

## S

saving  
    attachments 66  
    process data 32  
    workflows, in XML files 78  
schema rootnode attribute, form variables 49  
script  
    implicit objects 38  
    interactive 39  
    running 40  
    testing 39

- Script action
  - about 38
  - interactive 39
- selectedAction 55
- selecting routes, description 33
- selecting routes, user decisions 55
- selecting workflow type 69
- serialize function 113
- Set Start Action 33
- Set Value component 32
- setDocAttribute function 111
- setDocContent function 112
- setting variable values 32
- several forms, using 84
- short variable 29
- situational errors
  - catching 72
  - description 70
- solid routing lines 22
- Specification version 86
- specifying users, User actions 58
- splits
  - adding 37
  - description 36
  - join operator 37
- Stall action 72
- stalled branches and actions 70
- start actions 33
- starts-with function 117
- string function 118
- string variable 29
- string-length function 119
- submit options using route names 57
- subprocesses 41
- substring function 119
- substring-after function 118
- substring-before function 118
- sum function 114
- Supports Asynchronous Branch, action property 86
- Supports Synchronous Branch, action property 86
- Supports Transactional Branch, action property 86
- Synchronous branches
  - action compatibility 69
  - stalled 70
- Synchronous workflows 68
  - about 68
  - branch compatibility 69

## T

- tasks
  - adding attachments 65
  - assigning to previous participants 58
  - assigning to process creator 58
  - assigning to users 58
  - attachments in email 67
  - forms 54
  - instructions 64
  - saving attachments 66

- testing script
  - description 39
  - process data 39
  - running 40
- text, adding to workflows 26
- time-equal function 109
- time-greater-than function 109
- time-less-than function 110
- Transactional branches
  - action compatibility 69
  - stalled 70
- transaction-aware actions 71
- transactions 71
- Transient workflows 68
  - about 68
  - branch compatibility 69
- transient, workflow property 15
- translate function 120
- true function 89
- types of components 24
- types of variables 29

## U

- undeploying workflows 73
- unlocking workflows 20
- upper-case function 121
- User action
  - attachments 64
  - attachments and users 65
  - description 57
  - documents 53
  - escalations 61
  - expressions 59
  - forms 55
  - forms and documents 54
  - groups 59
  - instructions 64
  - migrating from LiveCycle Workflow 7.0.1 123
  - reminders 63
  - setting deadlines 62
  - specifying users 58
  - time constraints 60
- user selection, in groups 59
- users, assigning task methods 58
- using several forms 84

## V

- valid routes
  - based on submit choices 55
  - rules 34
- variables
  - adding 28
  - description 27
  - document 53
  - form type 49
  - setting values 32
  - types 29
  - web services 28

Version, action property 86  
versions  
    copying workflows 77  
    workflows 77  
viewing  
    workflows 18  
viewing workflows 19

## W

Wait Point action 41  
web services  
    enabling 15, 17  
    variables 28  
workflow fields  
    Acrobat forms 82  
    XML forms 81  
workflow objects, script 38  
workflow overview 19  
workflow properties, modifying 17  
workflow type  
    data saved 68  
    process complexity 68  
workflows  
    about 15  
    activating 73  
    branch compatibility 69  
    copying 77  
    deployed 15  
    deploying 73  
    deployment state 74  
    design procedure 11  
    exporting and importing 78  
    forms and documents 47  
    hiding workflows 19  
    init-form 48

workflows (Continued)  
    locking 20  
    migrating from LiveCycle Workflow 7.0.1 123  
    migrating to production 79  
    modifying deployed 78  
    new 17  
    opening 18  
    printing 26  
    protecting 19  
    removing 18  
    text labels 26  
    unlocking 20  
    using multiple forms 84  
    versions 77  
    zooming 19  
workspace 12

## X

XML forms  
    description 81  
    form data  
        about 51  
    workflow fields 81  
xml variable 29  
XPath expressions  
    date and time values 85  
XPath expressions, data collections 45, 46  
XPathExpression Builder  
    description 42  
    form fields 51  
    using 43

## Z

zooming 19



# Adobe® LiveCycle™ Workflow

Version 7.2

- What's New
- Overview
- Installing and Configuring LiveCycle for JBoss
- Installing and Configuring LiveCycle for WebSphere
- Installing and Configuring LiveCycle for WebLogic
- Creating Workflows
- Adobe LiveCycle Workflow Administration and User Management Help
- Using Business Activity Monitor Dashboard
- Using LiveCycle Workflow Workbench
- Business Activity Monitor Server Reference
- Adobe LiveCycle Workflow Readme

July 2006

# Installing and Configuring LiveCycle

The Installing and Configuring LiveCycle™ documentation provides information about installing, configuring, and deploying LiveCycle products. To ensure that customers have access to the most recent and updated information, the documentation is located at the following website:

[www.adobe.com/support/documentation/en/livecycle/](http://www.adobe.com/support/documentation/en/livecycle/)

## Installing and Configuring guides

The Installing and Configuring LiveCycle documentation is provided in a set of six guides based on the LiveCycle product and application servers.

The *Installing and Configuring LiveCycle for JBoss*, *Installing and Configuring LiveCycle for WebLogic*, and *Installing and Configuring LiveCycle for WebSphere* guides describe how to install and configure the following LiveCycle products and deploy the products to the specific application server:

- Adobe® LiveCycle Assembler 7.2.1
- Adobe LiveCycle Forms 7.2
- Adobe LiveCycle Form Manager 7.2
- Adobe LiveCycle PDF Generator 7.2 Professional, LiveCycle PDF Generator 7.2 Elements, and LiveCycle PDF Generator 7.2 for PostScript®
- Adobe LiveCycle Print 7.2
- Adobe LiveCycle Workflow 7.2.1
- Watched Folder

The *Installing and Configuring LiveCycle Security Products for JBoss*, *Installing and Configuring LiveCycle Security Products for WebLogic*, and *Installing and Configuring LiveCycle Security Products for WebSphere* guides describe how to install and configure the following LiveCycle products and deploy the products to the specific application server:

- Adobe LiveCycle Document Security 7.2
- Adobe LiveCycle Policy Server 7.2
- Adobe LiveCycle Reader® Extensions 7.2

## Updated product information

A Knowledge Center article has also been posted to communicate any updated LiveCycle product information. You can access the article from the following website:

[www.adobe.com/support/products/enterprise/knowledgecenter/c4811.pdf](http://www.adobe.com/support/products/enterprise/knowledgecenter/c4811.pdf)



**Adobe**

## What's New

# Adobe® LiveCycle™ Workflow

Version 7.2

- Platform Support [More ...](#)
- Improved Installation and Configuration Documentation [More ...](#)
- Improved Configuration Manager [More ...](#)

July 2006

## Platform Support

Adobe® LiveCycle™ Workflow is now supported on a standard set of operating systems, application servers, and databases.

For a complete list of the supported application servers, operating systems, and databases, see the *Installing and Configuring LiveCycle for JBoss*, *Installing and Configuring LiveCycle for WebSphere*, or *Installing and Configuring LiveCycle for WebLogic* guide.

[Back to top](#)

## Improved Installation and Configuration Documentation

To improve the interoperability experience of LiveCycle products, the installation and configuration instructions for LiveCycle Workflow have been combined with the instructions for Adobe LiveCycle Assembler, Adobe LiveCycle Form Manager, Adobe LiveCycle Forms, Adobe LiveCycle PDF Generator, and Adobe LiveCycle Print.

The installation and configuration guides are specific to each application server:

- *Installing and Configuring LiveCycle for JBoss*
- *Installing and Configuring LiveCycle for WebSphere*
- *Installing and Configuring LiveCycle for WebLogic*

The installation and configuration guides also include instructions on how to install Watched Folder.

[Back to top](#)

## Improved Configuration Manager

LiveCycle Workflow now includes an enhanced Adobe Configuration Manager. Using Configuration Manager, you can perform the following tasks:

- Configure and assemble LiveCycle products for deployment
- Configure your IBM® WebSphere® application server or BEA WebLogic Server® for LiveCycle products
- Validate application server settings for LiveCycle products
- Automatically deploy LiveCycle products to an application server
- Initialize database schemas for deployed LiveCycle products
- Verify deployed LiveCycle products are available and operational

[Back to top](#)

Adobe, the Adobe logo, and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

BEA WebLogic Server is a registered trademark of BEA Systems, Inc.

IBM and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries, or both.

All other trademarks are the property of their respective owners.