Redbooks

ibm.com/redbooks

Deliver Modern UI for IBM BPM with the Coach Framework and Other Approaches

Rackley Boren

Eric Ducos

Ge Gao

Thalia Hooker

Matthew Oatts

Paul Pacholski

Dennis Parrott

Claudio Tagliabue

Cloud

WebSphere

IBM

Redbooks

International Technical Support Organization

Deliver Modern UI for IBM BPM with the Coach Framework and Other Approaches

October 2016

SG24-8355-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (October 2016)

This edition applies to Version 8, Release 5, Modification 7 of IBM Business Process Manager.

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided “AS IS”, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Bluemix®
developerWorks®
FileNet®
IBM®
IBM API Connect™
IBM MobileFirst™
PureApplication®
Redbooks®
Redbooks (logo)  ®
WebSphere®

The following terms are trademarks of other companies:

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Find and read thousands of IBM Redbooks publications

- Search, bookmark, save and organize favorites
- Get personalized notifications of new content
- Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App

iOS **Download Now** Android

Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!

ibm.com/Redbooks
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

IBM® Coach Framework is a key component of the IBM Business Process Manager (BPM) platform that enables custom user interfaces to be easily embedded within business process solutions. Developer tools enable process authors to rapidly create a compelling user experience (UI) that can be delivered to desktop and mobile devices. IBM Process Portal, used by business operations to access, execute, and manage tasks, is entirely coach-based and can easily be configured and styled. A corporate look and feel can be defined using a graphical theme editor and applied consistently across all process applications. The process federation capability enables business users to access and execute all their tasks using a single UI without being aware of the implementation or origin. Using Coach Framework, you can embed coach-based UI in other web applications, develop BPM UI using alternative UI technology, and create mobile applications for off-line working.

This IBM Redbooks® publication explains how to fully benefit from the power of the Coach Framework. It focuses on the capabilities that Coach Framework delivers with IBM BPM version 8.5.7. The content of this document, though, is also pertinent to future versions of the application.

Authors

A team of specialists from around the world working at the International Technical Support Organization, Raleigh Center, produced this book. Information about the authors follows:

Rackley Boren started his career at IBM and joined Salient Process Inc. a Premier IBM BPM Business Partner, in August 2015. He has been working with the BPM platform since 2011. Working various delivery roles from analysis to development, he is now a Competency Lead providing guidance throughout the Salient practice. He is passionate about process and helping customers get the most out of their resources.

Eric Ducos is a technology executive, trusted principal, and active partner in the IBM Smarter Process initiative. He currently serves as Chief Innovator for Salient Process Inc., a Premier IBM Business Partner. Prior to joining Salient Process, Eric also led the Research & Development Program and Innovation Center as Chief Technology Officer at EmeriCon, another Premier IBM Business Partner.

Eric has been a relentless innovation driver for the last 20 years in the business process and decision management field. He is a visionary, author, architect, and co-implementer of numerous frameworks, libraries, solutions, and methodologies for many medium and large organizations worldwide and across industries.

The IBM BPM Toolkits covered in this book under the SPARK brand are examples of such innovations.

Ge Gao is a solution architect working in the IBM Boeblingen Lab in Germany, where core components of IBM BPM are developed. Ge focuses on delivering solutions based on IBM Smarter Process products to worldwide customers. He helps numerous customers solve critical situations.

As a solution architect, Ge has experienced the changes in technology and business needs in the BPM world over the past decade. Based on his rich experience, Ge creates toolkits, presentations, architectural patterns, and other assets to enable BPM solution delivery to colleagues and customers. He is the technical lead of BPM UI-related topics in the IBM Cloud software services organization. He is often invited to speak at conferences such as IBM WebSphere® Technical Academy and InterConnect. Currently, Ge is building a complex BPM solution as lead architect for one of the largest automobile companies in the world.

Thalia Hooker, Ph.D, has over 20 years of experience in IT, including the last 14 years at IBM. She is currently a Hybrid Cloud Process Transformation Worldwide Tech Seller specializing in Mobile Smarter Process. In a pre-sale capacity, she has worked with customers on various engagements, including customer briefings, executing complex proof-of-concepts, architecting, developing and delivering proof-of-technologies (PoTs) and demonstrations, conducting design and discovery workshops, and so on.

Matthew Oatts started his career with IBM BPM through the Lombardi acquisition in 2010. He has spent most of the last six years in BPM delivery roles providing analysis, development, and architecture expertise. He joined Salient Process (an IBM BPM Business Partner) in April 2015 as a Solution Architect and now works as the BPM Client Services Leader.

Paul Pacholski has been with IBM Canada Development Lab for 33 years, initially working as a Senior Developer on several IBM software offerings. For the last 18 years he has been the BPM Technical Sales Leader, responsible for technical enablement within IBM and influencing BPM product directions. Paul's other responsibilities include working with customers to help select the right BPM technology, presenting at technical conferences, publishing technical papers, and filing BPM-related patents. In his most recent role as a Lead Architect, Paul heads a team that develops SAP capabilities for IBM Smarter Process.

Dennis Parrott is a Senior Offering Manager for IBM BPM within the Smarter Process group. Dennis is passionate about BPM, which has been his primary focus throughout his 25-year plus career. Starting with a development background, Dennis turned his talents to consulting, which led to a position with Lombardi Software in 2007. He was one of the first Lombardi consultants in Europe, helping early adopters to successfully build and deploy BPM solutions.

Dennis joined IBM in 2010 when Lombardi was acquired. In 2013 he returned to his development roots as an offering manager. In addition to managing each new IBM BPM release, Dennis is responsible for new capabilities for improving the business-user experience. In recent releases, Dennis helped to define the requirements for the Coach Framework. He was also responsible for mobile enablement, process federation, the responsive IBM Process Portal, and developer tools for building modern coach-based UIs.

Claudio Tagliabue is a certified IBM technical specialist, focusing on Software as a Service (SaaS) and Business Transformation. He is excited by business change in the 21st Century and the consequent demands on technology. Transforming the "complicated" into "business-as-usual" has been Tag's focus for the past five years.

Through his work with clients as a domain expert for IBM's groundbreaking SaaS and BPM technologies, Tag has gained significant insight into today's senior IT stakeholder motivations. Tag has presented his work at several conferences around Europe. His experience extends beyond the IBM Software world, having worked on innovative projects involving open source technologies like OpenStack and Docker.

This project was led by Margaret Ticknor, an IBM Technical Content Services Project Leader in the Raleigh Center. She primarily leads projects about WebSphere products and the IBM PureApplication® System. Before joining the IBM Technical Content Services group, Margaret worked as an IT specialist in Endicott, NY. Margaret attended the Computer Science program at State University of New York at Binghamton.

Thanks to the following people for their contributions to this project:

Victor Chan
STSM, BPM Tools, User Interface and Repository, IBM Cloud

Sebastian Carbajales
IBM Business Process Management, IBM Cloud

Kim Clark
BPM and Integration software, IBM Cloud

Ryan Cox
WebSphere Technical Sales Specialist, North America SWAT, IBM Cloud

Evan Kettlewood
The Royal Bank of Scotland

Ian King
The Royal Bank of Scotland

Scott Johnson
Blockchain development, solutions and systems of record integration, IBM Cloud

Vlad Klicnik
Business Process Management, Architecture, and Design, IBM Cloud

Dieter Koenig
Business Process Manager Architect, Case Management, and ECM Integration, IBM Cloud

Varadarajan Ramamoorthy
BPM and Business Monitor End User UI Lead, IBM Cloud

Suzette Samoojh
Architect: IBM BlueworksLive, IBM Cloud

George Spanogiannopoulos
Software Developer: Business Process Management, IBM Cloud

Francesca Sutton
Process Transformation Technical Sales, Hybrid Cloud, IBM Cloud

Grant Taylor
IBM Cloud

Christian Templin
IT Specialist: Software Services for Business Process Manager, IBM Cloud

Ramiah Tin
BPM Architecture & Development, IBM Cloud

Thanks to the following people for their support of this project:

- Deana Coble, IBM Redbooks Technical Writer
- LindaMay Patterson, IBM Redbooks Technical Writer
- Ann Lund, IBM Redbooks Residency Administrator
- Thomas Edison, IBM Redbooks Graphics Editor
- Debbie Willmschen, IBM Redbooks Editor

Thanks to the authors of the previous edition of *Leveraging the IBM BPM Coach Framework in Your Organization*, published in April 2014:

John Reynolds, Mike Collins, Eric Ducos, David Frost, David Knapp, Ivan Kornienko, Bodo Naumann, Pat O'Connell, Paul Pacholski, Gerhard Pfau

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:
  http://www.redbooks.ibm.com/rss.html

1

Delivering modern UI for IBM BPM using the Coach Framework and other approaches

The IBM Coach Framework is a key element of the IBM Business Process Manager (BPM) platform. With the Coach Framework, process authors can create and maintain custom web-based user interfaces that are embedded in their business process solutions. This ability to create and maintain custom user interfaces is a key factor in the successful deployment of business process solutions. This book helps you fully benefit from the power of the Coach Framework. It also helps you to determine when other approaches and tools are applicable.

The term *coach* was introduced by Lombardi Software, which was acquired by IBM in 2010. The term was inspired by Vince Lombardi, considered by many one of the most successful football coaches of all time. Lombardi was known for his leadership skills, hence the screens generated by the product are called *coaches*. Coaches guide you through the steps required to perform your work.

History aside, coaches are a powerful element of IBM BPM solutions. With the release of IBM BPM version 8.0, coaches incorporate recent advances in browser-based user interfaces, and they are further improved in subsequent releases. This book focuses on the capabilities that the Coach Framework delivers with IBM BPM version 8.5.7. The content of this document is also pertinent to future versions of coaches.

This chapter includes the following sections:

- User interactions with business processes
- Coaches: Custom user interfaces for business processes
- Coach views: Custom user interface components
- Human services
- Using coaches outside IBM BPM
- How coaches can benefit your organization
- Conclusion

1.1 User interactions with business processes

BPM solutions can improve your business operations by automating processes and providing increased visibility into and analysis of process performance. With BPM solutions in place, process execution becomes more predictable and performance improves as you incorporate BPM functionality into your process logic. IBM BPM has driven improved process efficiency in many organizations over a wide variety of industries.

In addition to getting the right tasks to the right people at the right time, efficient business operations require tasks that provide a state-of-the-art user experience that enables workers to complete their work efficiently. As process authors you want to rapidly develop compelling user experiences for business operations. The Coach Framework can help you do that.

Business users interact with business processes in various ways, but those interactions can be broadly categorized as follows:

- Discovering what work must to be done
- Performing work
- Analyzing the work that has been done in the past
- Reviewing and managing the work performed by a team

For example, as a business user, you learn about the work that must be performed in various ways. You can log on to a specific website to view your task list; you can use a mobile application to access your next scheduled task; or you can receive notifications via email or SMS. In all cases, it is important for you to quickly understand the nature of the task and its relative importance to the other tasks that you are responsible for performing.

You perform the work that you are required to do in a number of ways. In some cases, you perform the work outside of the BPM solution using existing systems. In that case, you inform IBM BPM once the work has been completed. In other cases, the interfaces used for performing the work are implemented as part of the BPM solution. In many solutions, both approaches must be combined. Regardless of the approach, you must clearly understand the details of the task to be performed and the wider business context of the task. You must be provided with user interfaces for supplying and modifying the information that the task gathers or manipulates. Custom user interfaces are typically necessary to provide you with optimal tools for performing your work.

You analyze your work in a number of ways and for a number of reasons. During the execution of a process, you need real-time feedback to help you guide subsequent execution of the process. Similarly, managers require visibility into their team's work and tools to help them manage that work. Analysis also occurs after-the-fact to determine if the process logic is optimal for the business to run smoothly. In each of these cases, users need the system to present relevant information. In some cases, generic out-of-the-box user interfaces are sufficient. In other cases, custom user interfaces are necessary to provide users with the most relevant information for their specific needs.

IBM BPM includes IBM Process Portal, which can be used to perform many of these activities from your desktop computer or mobile device. It provides a standard set of capabilities that are applicable to all your processes. The Coach Framework is used to build the custom user interfaces for tasks, dashboards, and reports that are delivered by Process Portal and can be used to extend and customize Process Portal. The Coach Framework can also be used to create custom user interfaces that are delivered via alternative means such as email, corporate portals, mobile applications, or websites. It is also possible to develop custom user interfaces using alternative techniques and technologies.

1.2 Coaches: Custom user interfaces for business processes

Coaches are the screens (or forms) that process authors construct to provide the custom user interfaces required for their BPM solutions.

Custom user interfaces must present users with relevant information that helps them efficiently carry out tasks. Because every process is different and every activity is different, you have to optimize the user's experience. To do this you must build custom user interfaces that present and manipulate the most relevant information in the most effective manner.

This critical need for custom user interfaces for business processes is the driving factor in the development and evolution of the Coach Framework. The Coach Framework's primary focus is to enable process authors to create and maintain custom user interfaces that improve users' interactions with their organization's business processes. This subsequently improves the performance of business operations.

Maintenance of custom user interfaces in a BPM solution is crucial because of the frequency of changes to the logic of an organization's business processes. As market conditions change, the logic of the business process has to adapt.

As the logic of a business process changes, the users' interactions with the business process must change. Your ability to adapt the user interfaces of a business process has to keep pace with changes to the process logic; otherwise, your organization's ability to respond quickly to change is diminished. Using the Coach Framework, your organization is agile and adapts quickly and seamlessly to business change.

Coaches make it much easier to adapt your user interfaces when your business process changes. User interfaces that are created with the Coach Framework are packaged along with the other artifacts that make up a process application. When a new version, known as a snapshot, of the process application is deployed, all of the coach-based user interfaces are also deployed. This ensures that users are always presented with screens that are in sync with the correct process logic.

Without coaches, the deployment of the user interfaces for a process is a separate step that must be performed when a new snapshot of the process is deployed. With coaches, the process screens always match the process logic.

1.3 Coach views: Custom user interface components

Coach views are the reusable user interface building blocks that process authors use to create their coaches.

To fully appreciate the benefits of coach views, you must understand the problems that they help solve. Before the introduction of the Coach Framework in BPM 8.0, coach views did not exist. Coach views were introduced to solve a variety of problems that customers encountered when using heritage coaches, the coaches used before IBM BPM version 8.0. These problems are described in 1.3.1, "Controls: Atomic coach views" through 1.3.3, "Dynamic coach view interaction" on page 5.

1.3.1 Controls: Atomic coach views

Coaches have always provided a facility for nontechnical process authors to rapidly construct form-based user interfaces by selecting controls such as input text, buttons, drop-down lists,

check boxes, tables, and sections from a preconfigured palette. However, the functionality offered by heritage coaches was not sufficient. It restricted the author's ability to go beyond what was offered out-of-the-box; it did not deliver the sophisticated user experience that was expected by the business as new user interface requirements and technologies emerged.

The Coach Framework was designed to enable the Coach Editor within the IBM Process Designer to be extended beyond the use of out-of-the-box controls, referred to as *stock* controls. As part of the Coach Framework, coach views allow process authors to create new controls and add them to the palette. In fact, the default stock controls provided with IBM BPM are now implemented as coach views themselves. This allows customers and business partners to develop toolkits that include sophisticated custom controls and leverage the latest user interface technologies to deliver a state-of-the-art user experience. Some recent examples of custom controls include maps, mobile specific controls, bar code readers, electronic signatures, and charting controls. This ability to create and distribute additional coach views dramatically increases the power of the Coach Framework.

The Coach Framework has retained the benefits that heritage coaches offered to nontechnical process authors; they can simply select the controls that are available on the palette to construct their coaches as they did before. The development of new controls using coach views requires a more technical skillset, such as the skillset of a web developer with proficiency in HTML, JavaScript, Ajax, and CSS.

Before the use of coach views, extending heritage coaches was possible to some extent using custom HTML. This, however, introduced complexity into the coaches that made it difficult for nontechnical authors to maintain them. Coach views provide a place to encapsulate the complexities that are necessary to extend controls without exposing the complexities to the coach author. Unlike custom HTML, coach views are designed to explicitly enable developers to integrate third-party user interface technologies of choice, such as AngularJS, JQuery, DOJO, and HTML5. Coach views enable this integration by providing appropriate APIs and event handlers within the Coach Framework.

Coach views provide the building blocks for visual controls and can also be used to control the layout and behavior of a coach. This is a significant enabler for IBM BPM to support responsive user interfaces for both desktop and mobile devices.

1.3.2 Reusable coach elements: Composite coach views

Another problem that process authors encountered with heritage coaches was the ability to reuse aspects of one Coach with other Coaches. Coach views solve this problem.

Coach views can be defined by combining other coach views in the same way as a process author constructs a coach, that is, using a drag-and-drop mechanism. The resulting composite coach views appear on the palette along side the atomic controls, so that they can be used on multiple coaches or nested in other coach views.

Composite coach views are often created when a number of coaches within a process include a common section of controls, for example, a header that contains overview information about a process or a business object like an address.

Composite coach views can be associated with the business objects that they are used to present and manipulate. When dragging business objects onto a coach, the appropriate coach views are selected and added to the coach automatically.

1.3.3 Dynamic coach view interaction

When a coach view is added to a coach, the author configures the coach view. The author does so by binding it to the business data that the component displays and to any configuration data that is necessary to control the appearance and behavior of the component at run time.

Most coach views respond when any of the data that they are bound to changes. This allows authors to create dynamic user interfaces by binding multiple coach views to the same data. When a user manipulates one coach view, all of the other coach views that are bound to the same data instantly react.

Coach views can also be bound to Ajax services that are invoked to retrieve information and update systems that are related to the business processes. This ability to bind to Ajax services allows authors to create highly dynamic UI components, such as select controls with fields that are updated dynamically while users are typing. This feature can be used also to create coach views that directly interact with services outside of BPM.

1.4 Human services

Human services are components that manage individual business-user interactions that might involve one or more coaches. Human services can be thought of as mini-applications that define the navigation between coaches and provide linkage to business logic and integrations to back-end systems.

Human services are either used to implement the user activities of a business process, or to implement dashboards that are invoked independently.

1.4.1 Task completion coaches

Task completion coaches provide the user interfaces that allow a business user to work on a specific activity of a business process. The lifespan of task completion coaches is tied to the lifespan of an activity in the process. The coach can only be opened while the activity is active.

When implementing a business process activity, a human service is initialized with data from the process when the process flow reaches the activity. This allows the coach authors to easily build user interfaces within the context of a specific activity of a business process.

1.4.2 Dashboards

Dashboards are user interfaces that are independent from the activities of a process and can be opened at any time. Dashboards can be used to extend Process Portal, they can be used to extend the BPM Admin Console, or they can be used outside of IBM BPM.

When extending Process Portal, Dashboards are available to specific teams so that the teams can access additional information that is applicable to completing their work. Dashboards can be used for many purposes, such as creating reports to monitor performance, managing workloads, or displaying business data related to a specific case or customer.

Similarly, Dashboards can be available to specific teams to access system-related information and perform custom administration activities within the BPM Admin Console.

Much of Process Portal is implemented using Dashboards, and many clients also use Dashboards to build their own custom portals.

1.4.3 Client-side human services

Human services were historically server-side components, where control was only passed to the web browser on reaching a coach within the human service flow. Server-side human services continue to be supported, but are now referred to as heritage human services.

Client-side human services were introduced in BPM 8.5.5. The entire CSHS is executed within the web browser, apart from server-side calls that are explicitly modelled within the Client-side human services. This reduces passing control back and forth between the server and the browser, which substantially improves the performance and user experience especially when accessed from a mobile device.

1.5 Using coaches outside IBM BPM

When you are logged on to Process Portal, the coaches that you work with are opened in the Process Portal browser window. However, coaches can also be available when you do not use Process Portal.

1.5.1 Launching coaches using a URL

Both task completion coaches and dashboards can be launched using a URL. This enables tasks to be launched outside of IBM BPM, for example, from an email, a custom web application, a portal, or a mobile application.

Every coach is embedded in a human service, and all human services can be launched using a URL. This makes it very easy to incorporate coaches into a wide variety of web-based interfaces. Links to launch coaches can be embedded on existing corporate web sites, or, when appropriately launched within iframes, on corporate web pages. You can discover human services that are exposed for this purpose via a BPM Representational State Transfer (REST) application programming interface (API).

1.5.2 Coaches within JSR 286 portlets

For companies that use IBM WebSphere Portal, the coach dashboards can be wrapped in JSR 286 portlets. This capability allows dashboards to be wired into WebSphere Portal composite applications in the same way as other custom Java portlets.

1.6 How coaches can benefit your organization

Now that you have a better understanding of what coaches are, you can better understand how coaches can benefit your organization.

1.6.1 Seamless integration of UIs and process logic

Coaches benefit your organization by seamlessly integrating the development and deployment of custom user interfaces with the development and deployment of associated process logic and necessary integrations that make up a complete business solution.

IBM BPM supports the use of *external* user interfaces for interaction with processes via representational state transfer (REST) APIs. However, when external user interfaces (UIs) are used instead of coaches, the process for developing, testing, and deploying the solutions is disconnected. Without coaches, explicit coordination and governance is required between the UI and process developers when packaging and deploying these components separately.

In most cases, the seamless integration of UIs with the process solution makes it easier for your company to implement a business process change that also requires a change to the process UIs.

1.6.2 Tailored UI components for your business

The ability for each business to create their own coach views enables organizations to create their own toolkits of UI components that are tailored for their specific needs.

Experienced developers can create rich user interfaces for key business data and package those UIs as building block components that can be used by less technical business programmers.

1.7 Conclusion

This chapter outlines how the IBM Coach Framework can be an extremely valuable resource for helping you to optimize your business operations and achieve process agility. The remainder of this book focuses on specific details and insight for how to best use the Coach Framework and other approaches to deliver BPM UIs within your own organization.

Chapter 2, “Creating user interfaces with coaches” on page 9 describes how to assemble your custom UIs from existing coach views and leverage new tools introduced in BPM 8.5.7. It describes how to build responsive user interfaces, how to style coaches using the new graphical theme editor, and how to design coaches using the new grid layout.

Chapter 3, “Building controls using coach views” on page 73 describes how to develop custom coach views. This chapter provides all of the details that are necessary for constructing new UI components when an existing component does not meet your needs.

Chapter 4, “SPARK UI Toolkit” on page 153 focuses on sharing real world lessons that have been learned by Salient Process, an IBM Business Partner which has extensive experience with developing custom coach views for their clients.

Chapter 5, “IBM Process Portal” on page 261 describes the standard features provided by the Process Portal; how to extend, configure, customize, and style Process Portal; the latest features including new saved searches, responsive UI and process federation, and how to build a custom Portal using the Salient Process Portal Builder toolkit.

Chapter 6, “Combining the Coach Framework with other approaches” on page 313 defines design patterns to expose coach-based human services over the Internet. It also describes how to integrate coaches into other solutions, integrate external UIs into your BPM solutions,

and use Mobile First with IBM BPM to build hybrid mobile applications and provide an offline working capability.

2

Creating user interfaces with coaches

In the previous chapter, you learned about human services, coaches, and coach views. Now, it is time to take a closer look at using *coaches* to create BPM user interfaces (UI). This chapter concentrates on the tools and controls that are provided by the product. These options are sufficient for developing UIs for simple mock-ups, playback scenarios, and production BPM solutions. If you have user interface requirements that go beyond these core capabilities, you can build your own coach views, which is explained in Chapter 3, “Building controls using coach views” on page 73.

This chapter includes the following topics:

- Getting started with client-side human services
- Building a simple coach
- Configuring controls
- Control toolkits
- Working with documents
- Creating a reusable group of controls (using a composite coach view)
- Adding validation to a coach
- Deriving data from other fields
- Making a coach responsive
- Designing a coach using the grid layout
- Using nested client-side human services
- Styling coaches using themes
- Displaying coaches in different languages (using localization)
- Performance considerations
- Conclusion

2.1 Getting started with client-side human services

As introduced in the previous chapter, the primary use cases for human services, coaches, and coach views are when building task completion UIs or dashboards. The focus for this chapter is on task completion user interfaces. *Task completion user interfaces* are the user interfaces used by a person who participates in a business process by performing user tasks. User tasks allow people to enter data, verify data returned by backend systems, give their approval, and so on.

> **Tip:** All the techniques that are discussed in this chapter also apply when modeling dashboards, which is detailed further in Chapter 5, “IBM Process Portal” on page 261.

More sophisticated scenarios are covered later in this chapter. This section looks at how to get started and explains what you need to know to build coaches using an example process called *Publish Paper*.

2.1.1 Creating a simple process flow

Start with a simple process flow. Create a process called *Publish Paper* in IBM Process Designer and add a single activity labeled Submit Paper Draft, as illustrated in Figure 2-1.

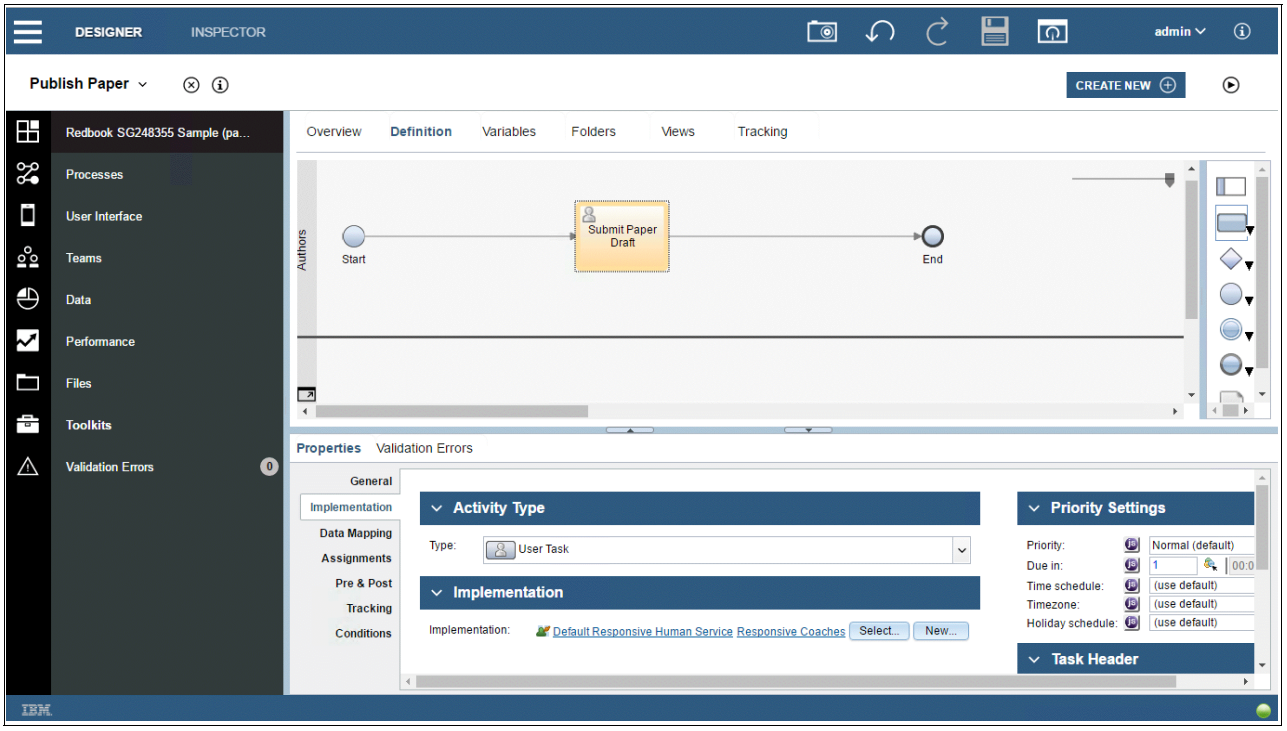


Figure 2-1 Simple process flow

By default new human activities are user tasks that use a standard Default Responsive Human Service implementation.

> **Development note:** You can run the process at this stage with the Default Responsive Human Service. During development it is helpful to focus on the process flow before developing the UI. For the purposes of this book, however, the focus is on creating the UI.

2.1.2 Creating a client-side human service

To replace the default user task with a custom implementation, go to the Implementation tab for the task, and click **New**, as shown in Figure 2-2.

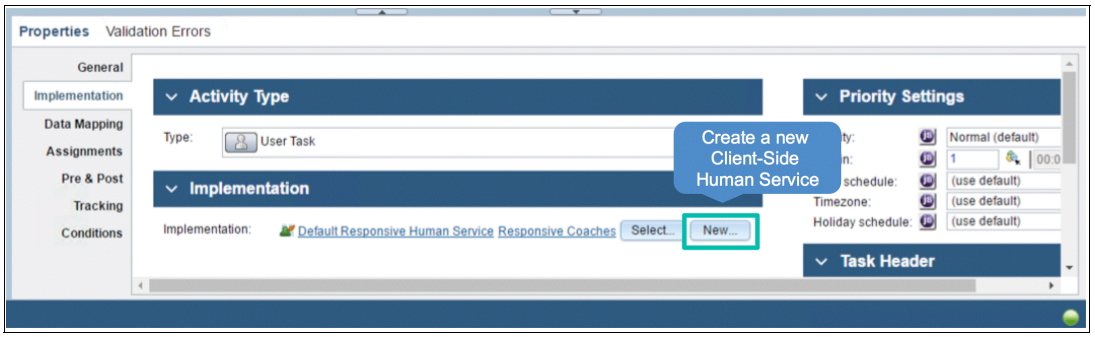


Figure 2-2 Creating a new client-side human service for a user task

Then, follow these instructions:

1. When prompted, enter the name for the new client-side human service. This scenario used `Submit Paper Draft`. Then, select **Intended for use on multiple devices**, and click **Finish**, as illustrated Figure 2-3.

> **Note:** The “Intended for use on multiple devices” option simply restricts the palette to hide controls that are not designed for responsive user interfaces.

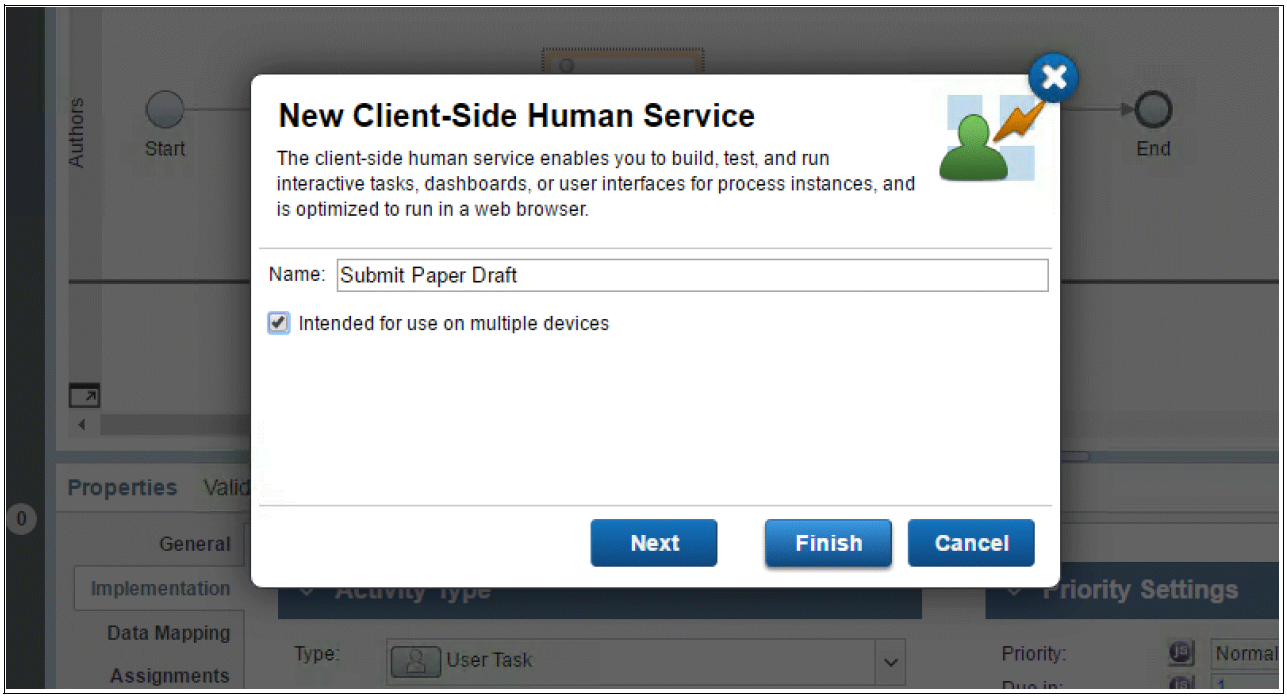


Figure 2-3 The client-side human service wizard

2. A client-side human service named *Submit Paper Draft* is created and is associated with the *Submit Paper Draft* activity within the process. To verify that everything worked as expected, select the **Submit Paper Draft** activity within the process. Check that the activity implementation is set correctly, as shown in Figure 2-4.

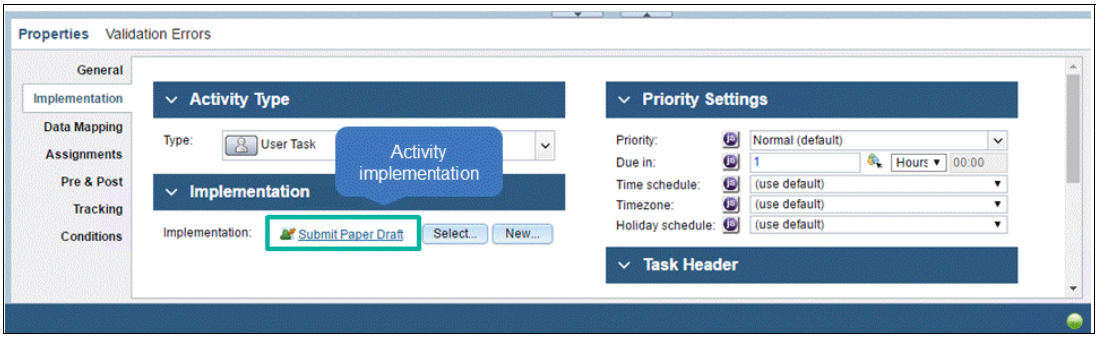


Figure 2-4 Implementation definition of the example "Submit Paper Draft" activity

3. Click the **Submit Paper Draft** link to open the Submit Paper Draft client-side human service in the service editor. This client-side human service defines the task completion UI for the activity. Initially it consists of a single coach with navigation for a single **OK** button, as shown in Figure 2-5.

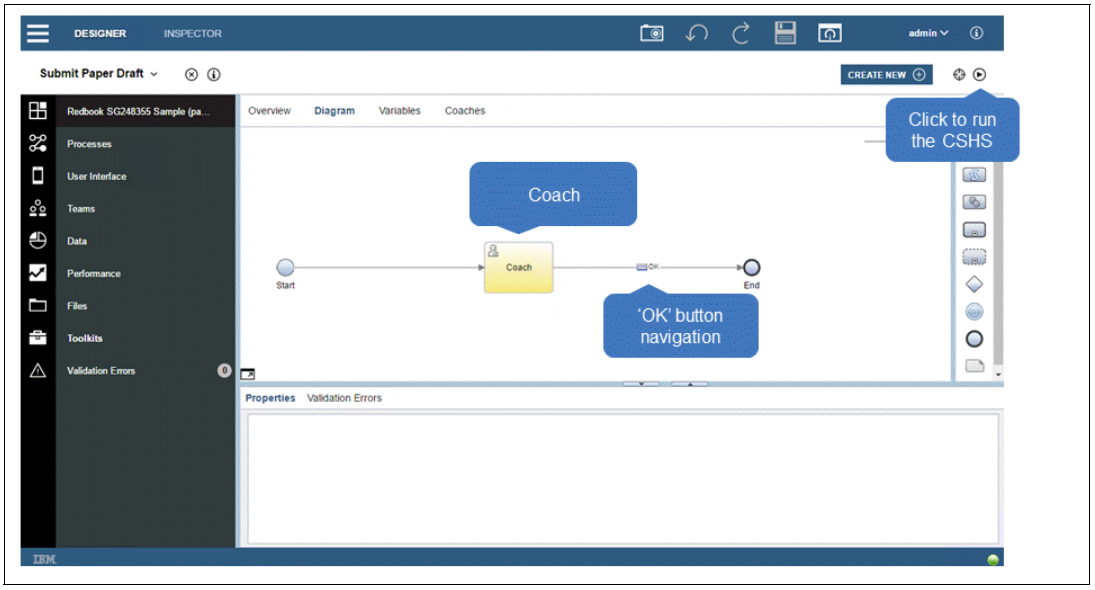


Figure 2-5 Submit Paper Draft client-side human service diagram

2.1.3 Running a client-side human service

You can now press the play button (Figure 2-6) to execute the Submit Paper Draft client-side human service.

Figure 2-6 Play button

A browser window opens and shows an empty form with a button (Figure 2-7).

Figure 2-7 Running the default coach implementation

Click **OK** to close the form and the text `Service completed` displays. You have just built and run your first task completion UI.

2.2 Building a simple coach

So now that you have everything in place, you can start working on designing your first coach. To do that, open the Submit Paper Draft client-side human service that you created in 2.1.2, “Creating a client-side human service” on page 11, and go to the coaches tab to open the coach editor, as shown in Figure 2-8.

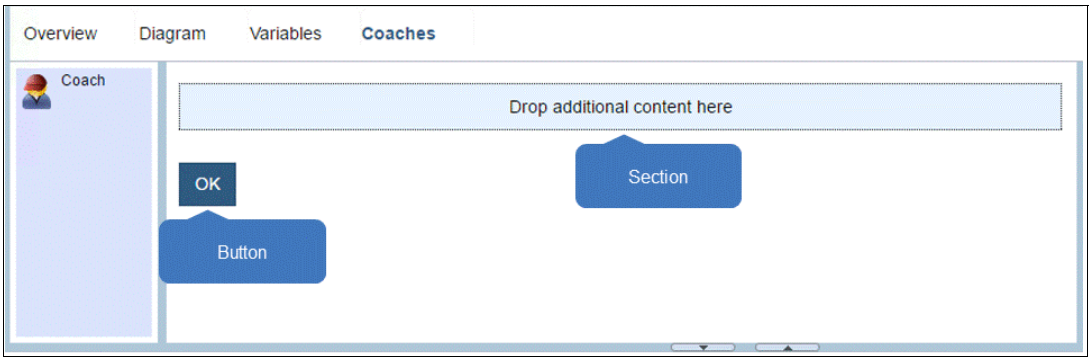


Figure 2-8 A new coach

By default a new coach includes a section with vertical alignment to add additional content and an OK button used to submit the coach.

2.2.1 The coach editor

Figure 2-9 shows the coach editor.

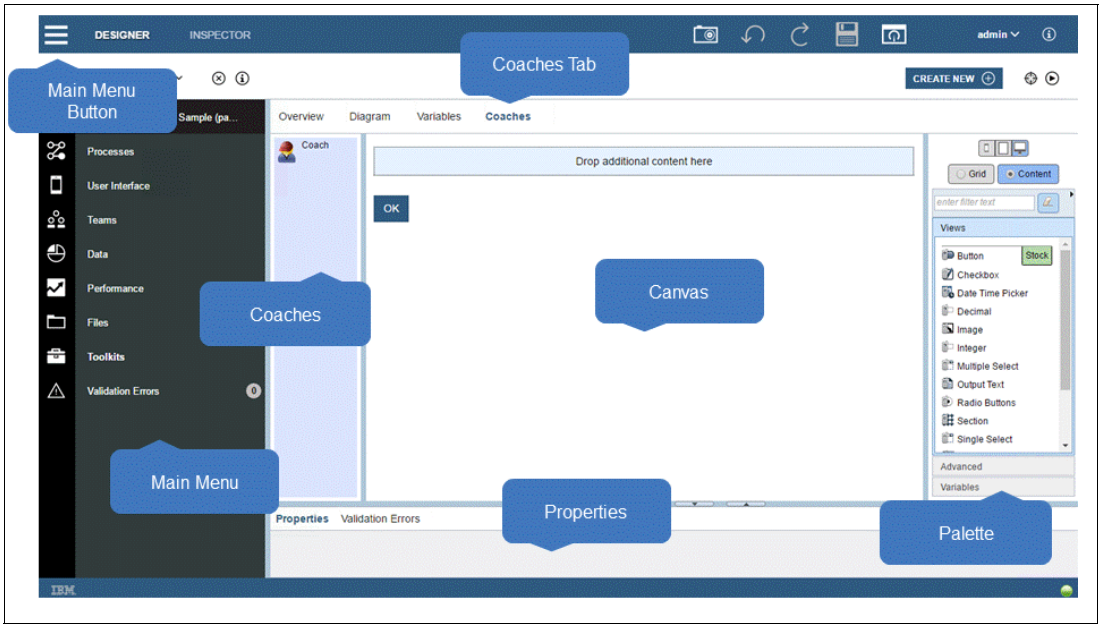


Figure 2-9 Coach editor

In the middle of the coach editor (Figure 2-9), you see the *canvas* of the coach UI. At the moment, there is a lot of white space because you have not started to construct your custom UI.

To the left of the canvas, you see the list of *coaches* in this client-side human service. Now, it includes only one named coach, which is the one that has been created for you by the activity wizard.

On the right side of the coach editor, you see the *palette* of UI controls that you can use to build the coach. Dragging any of the controls to the canvas adds it to the coach UI. By default the list shows the following stock controls that are provided as part of the product:

- Button
- Checkbox
- Date Time Picker
- Decimal
- Image
- Integer
- Multiple Select
- Output Text
- Radio Buttons
- Section
- Single Select
- Table
- Tabs
- Text
- Text Area

To learn more about the basic stock controls, refer to IBM Knowledge Center:

`http://ibm.biz/BPMResponsiveCoachesToolkit`

Sections and *tabs* provide a means to group and structure other controls on the form. Section content can either be aligned horizontally or vertically. Sections can also be nested to create a coach layout. Using the grid layout, detailed in 2.10, “Designing a coach using the grid layout” on page 50, is a better approach.

A note about controls: *Controls* are coach views that coach authors use to construct custom user interfaces, such as, buttons, text and sections. A *composite coach view* combines a group of controls (and other composite coach views) into a single coach element, so that they can be reused on multiple coaches without having to reconstruct them individually each time. This chapter predominantly focuses on controls, but all the concepts that are applicable to controls are equally applicable to composite coach views.

The palette also includes variables and advanced elements.

Advanced elements include *custom HTML*. Custom HTML can be added to a coach to enable HTML markup to be defined directly within the coach. Typically, a coach author would not use custom HTML, but 3.2.2, “Constructing the view of a coach view” discusses how useful this element can be for the coach view developer.

Variables list business data that is defined for the client-side human service. Dragging a variable to the canvas results in a matching control or composite coach view being selected and added to the coach, and a binding to the corresponding variable is created. You can find more information about composite coach views and their bindings in 2.6, “Creating a reusable group of controls (using a composite coach view)” on page 35.

The *Properties* pane provides the details relating to a control when it has been selected on the canvas. Properties are used to configure the control.

Clicking the Main Menu button either hides or shows the main menu. When hidden, the menu icons on the left remain, but the text is removed to provide more space for the coach canvas. Similarly, you can maximize the canvas area by reducing the coaches and the palette and properties panes by moving their borders within the coach editor.

2.2.2 What you see is what you get (WYSIWYG)

Next you use the coach editor to create a coach UI for the Submit Paper Draft activity by adding a label to the section, as shown in Figure 2-10. Notice that the label appears in the section on the canvas. The canvas has WYSIWYG behavior, so you do not have to run the coach to see how the coach is going to look at run time.

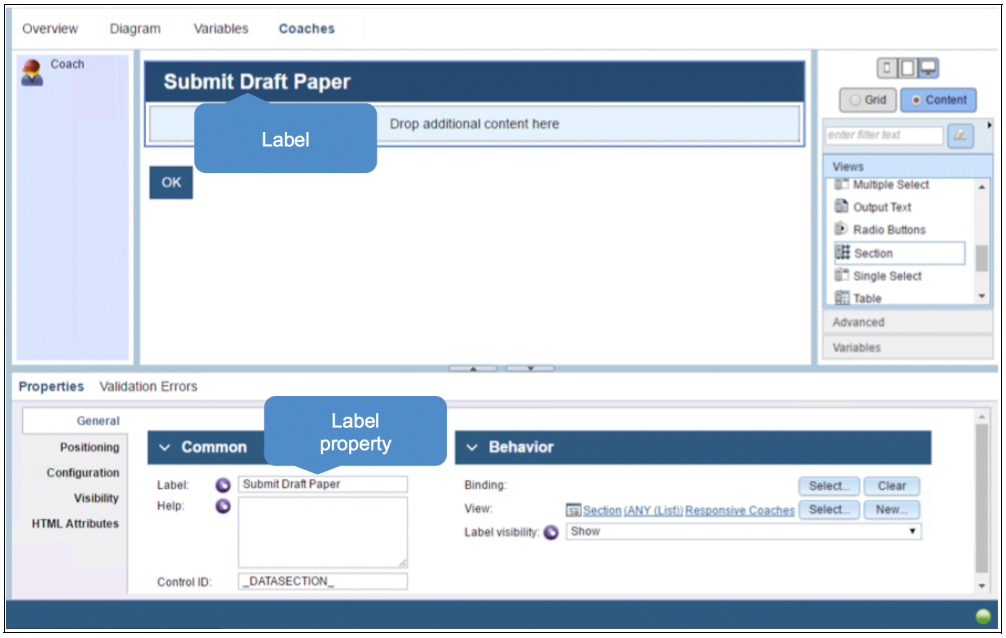


Figure 2-10 Defining the section label property

2.2.3 Using variables to add controls to a coach

To make the previously introduced basic scenario a bit more interesting, you can extend the *Publish Paper* process with a few more activities and lanes. This section describes a simple approval process where first authors and reviewers work on several drafts of a paper, before seeking approval, as shown in Figure 2-11.

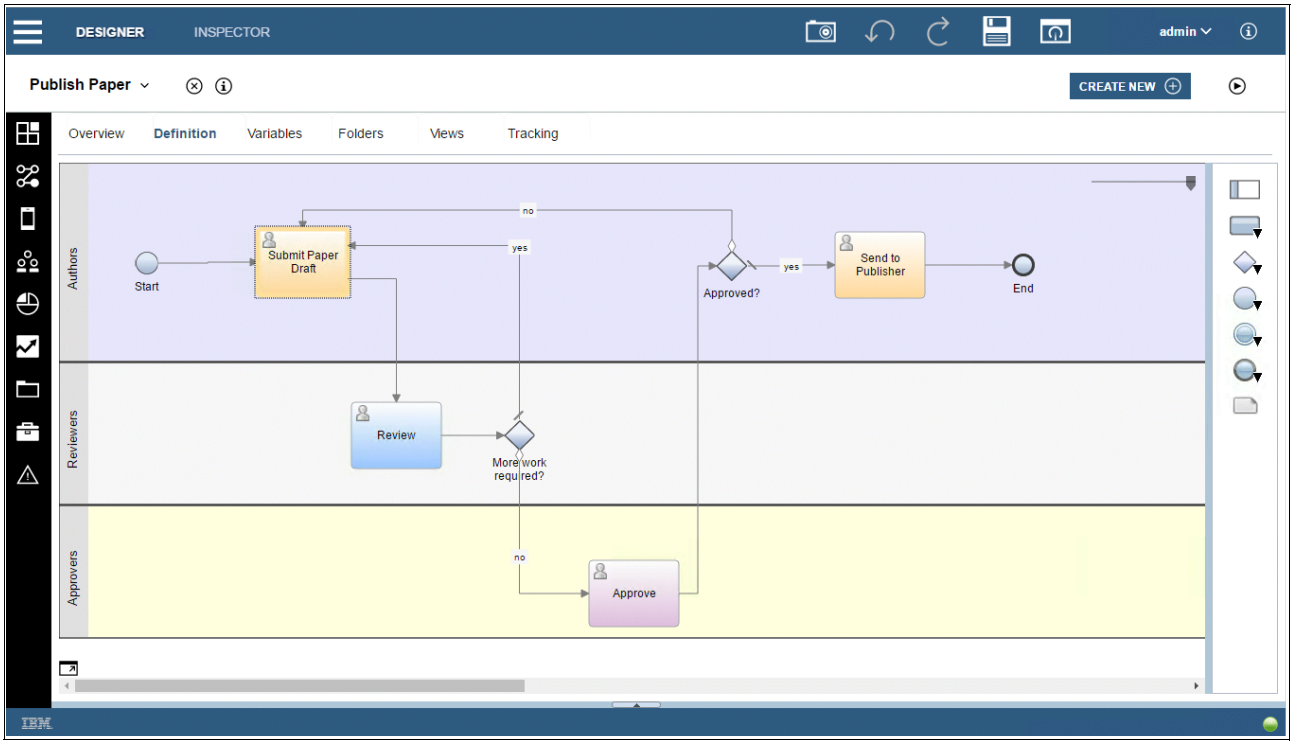


Figure 2-11 An evolved version of the Publish Paper process

The process also has added variables. Initially the Submit Paper Draft activity deals with the following data:

- `paperLocation` (String): This string describes the location where the paper can be found.
- `reviewComments` (String) (List): A list of review comments.
- `readyForApproval` (Boolean): A flag that indicates if the paper is ready for approval.

Now that variables are defined, you can add corresponding fields to the coach to capture the data. The variables now appear in the palette, as shown in Figure 2-12 on page 18. Drag the `paperLocation` variable from the palette onto the coach canvas and drop it onto the vertical section.

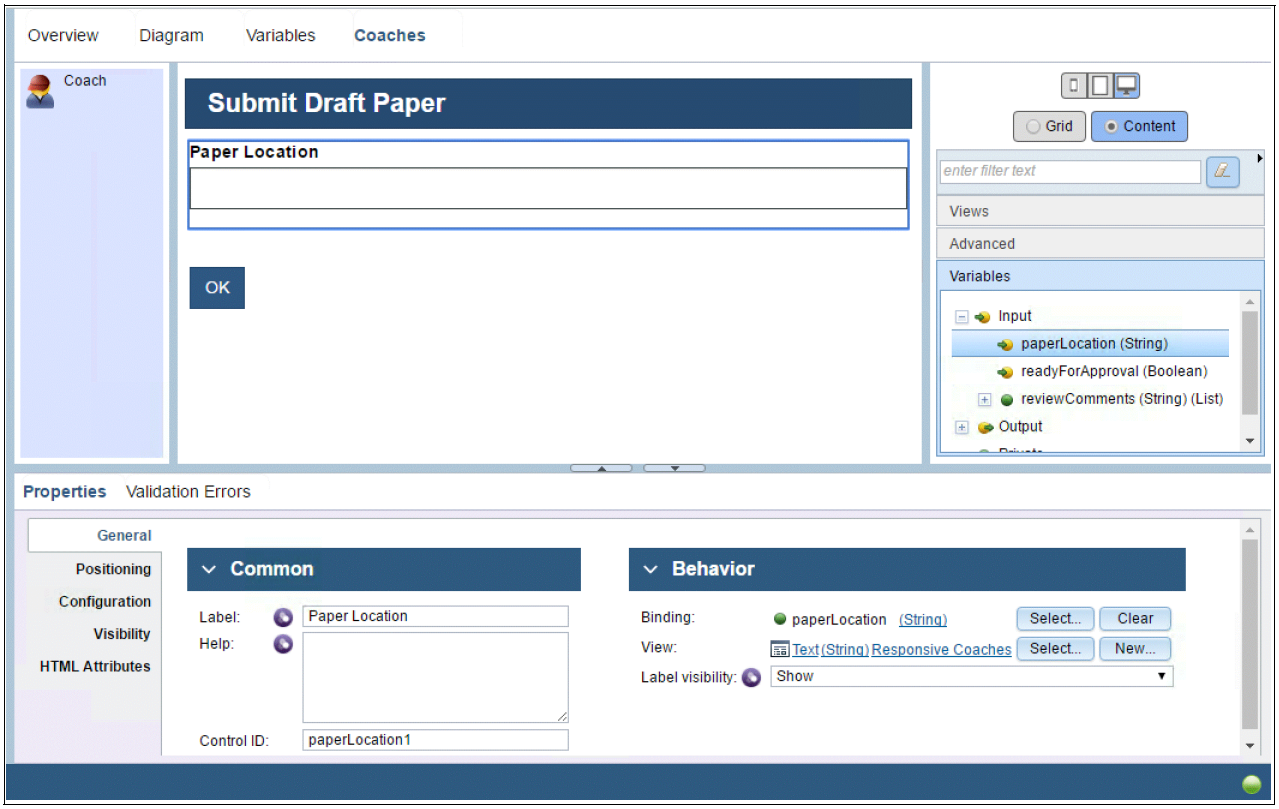


Figure 2-12 Adding a Text control using a String variable

A text entry field displays on the coach. In the Properties pane the Text control is automatically selected based on the String variable type, and the `paperLocation` input variable is set as the binding.

How binding works: *Binding* is an important concept of controls (and coach views in general). Coach views and variables are coupled via the binding. When the user changes the data of a control, for example by entering text, the variable bound is automatically updated. The change is also propagated to all other controls that are bound to the same variable.

Next, drag the `readyForApproval` variable from the Variables section of the coach editor menu to the canvas underneath the Paper Location field. Based on the type of the variable (Boolean), the Checkbox control is selected automatically, as shown in Figure 2-13 on page 19.

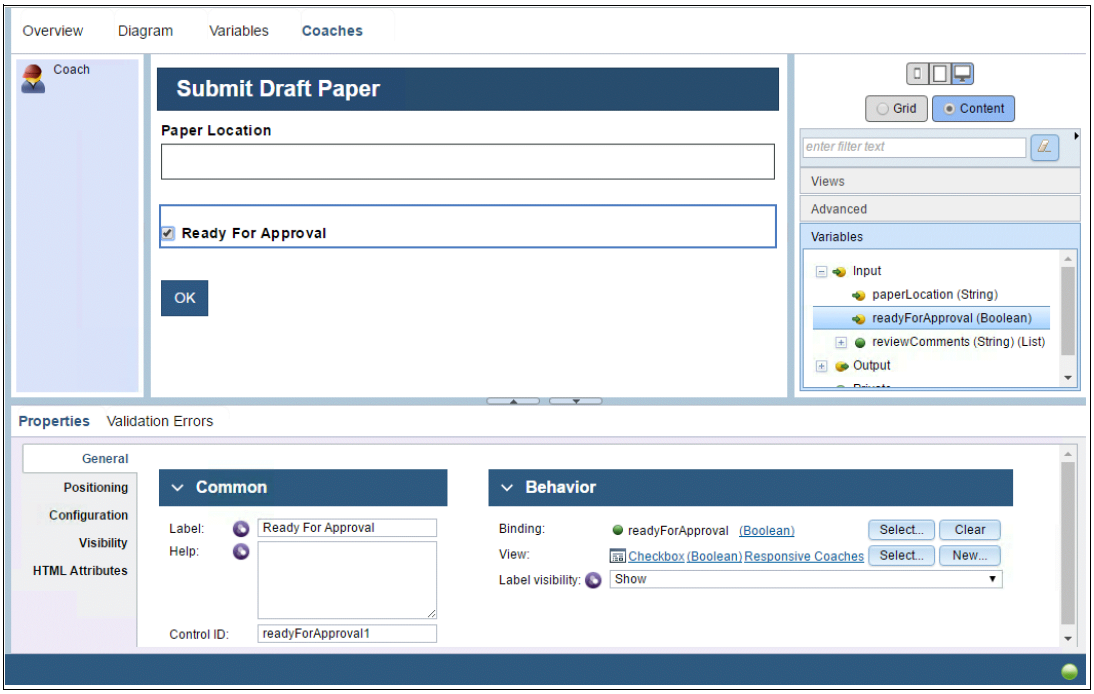


Figure 2-13 Adding a Checkbox control using a Boolean variable

Before adding support for review comments in the next section, run the coach in playback mode to see how things look in reality, as shown in Figure 2-14.

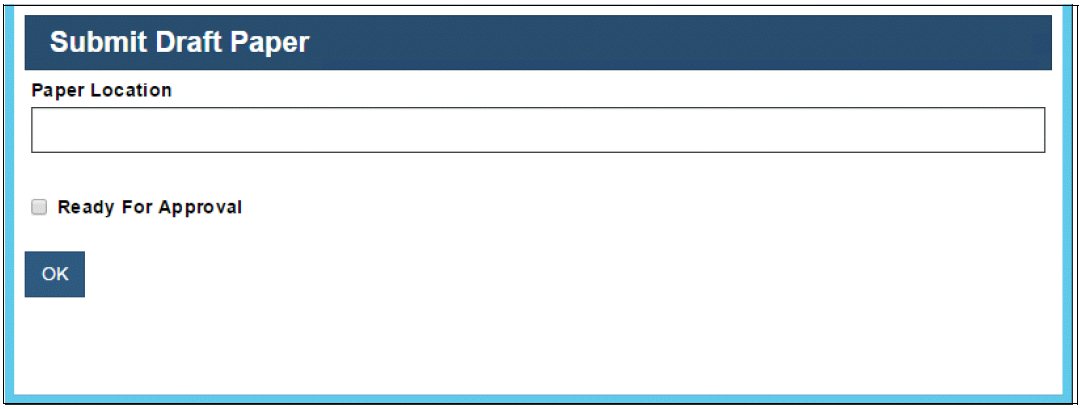


Figure 2-14 Running Submit Paper draft human service

2.2.4 Moving and deleting controls

Next, you can move and delete controls. You can move the OK button into the vertical section by simply dragging it on the canvas to the required position. The OK button was actually already in a section of its own. So after moving the button, simply delete the unneeded section. To delete the section, right-click it, and select **Delete**, as shown in Figure 2-15.

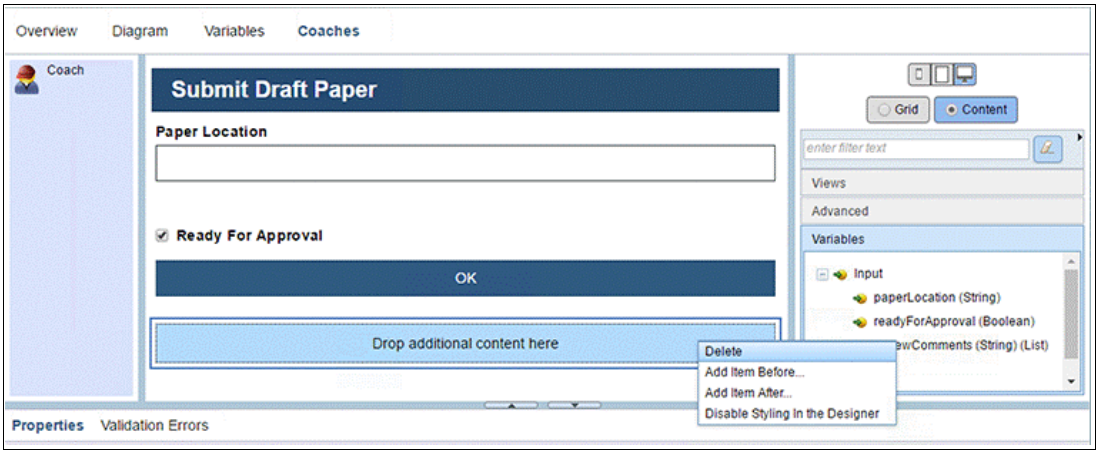


Figure 2-15 Deleting controls from the canvas

The next section explains how to implement the UI for the list of comments.

2.3 Configuring controls

Continuing with the same example, this section describes how to display, add, and remove review comments by using the Table control.

2.3.1 Setting the control binding variable

To add a Table control:

1. Drag the Table control from the palette onto the canvas.
2. Set the label to Review Comments.
3. Configure the Table to bind to the comments variable.

4. Click **Select** (next to Binding), and select the `reviewComments` variable, as shown in Figure 2-16.

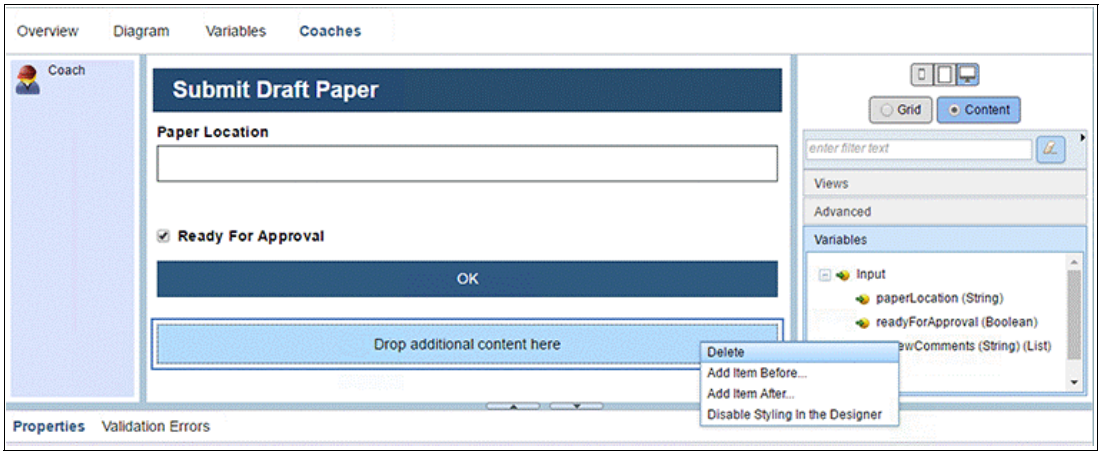


Figure 2-16 Setting the Table control's binding variable

2.3.2 Using pseudo variables

Now add controls to the display, and edit the columns of the table. In this case, the only column is the comment string. To accomplish this action, drop a Text control into the table and set the label for the Text control to define the column heading. For the custom UI in this example, the column heading is *Comment*, as shown in Figure 2-17 on page 22.

The Text control is bound to a pseudo variable called `currentItem` under `reviewComments`, where `currentItem` represents the list iterator.

Note: The second pseudo variable is `listSelected`. You use it if you want to refer to the currently selected item in the list.

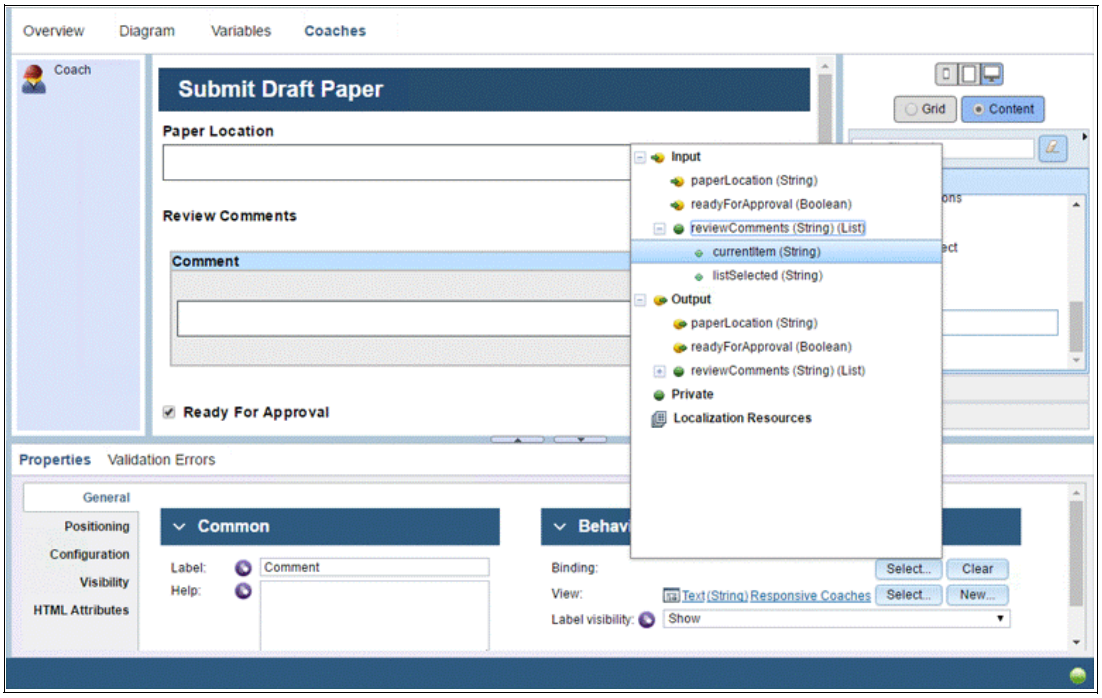


Figure 2-17 Pseudo variables

Next, to test-drive the coach, click the play button to run the service as before. Here is how the coach is rendered using sample data as shown in Figure 2-18.

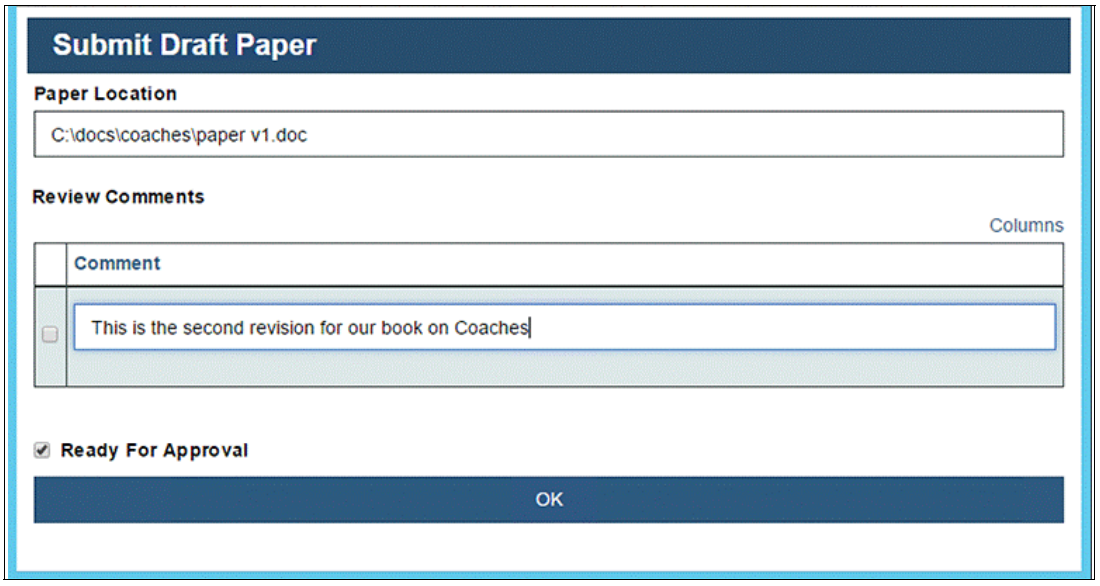


Figure 2-18 Rerunning Submit Paper Draft client-side human service

2.3.3 Setting control configuration options

Although this approach looks promising, the table within this custom UI has the following issues:

- You can select a line, which is not required in the use case.
- It does not allow you to add or remove comments.

These issues can be resolved using the Table control configuration options.

Taking advantage of configuration options: Configuration options are another important concept of controls (and coach views in general). The following configuration options exist:

- *Object typed* configuration options specify configuration data for the control, such as showing a particular button. Object typed configuration options have the same data sharing and notification behavior as binding variables.
- *Service typed* configuration options bind the control to services and can be used at run time to perform certain tasks, such as retrieving data or performing data validation. The provider of the control defines configuration options.

When using the control, a Configuration property sheet is automatically generated based on the configuration option definitions.

To configure a control, select the control on the canvas, and click the Configuration tab in the Properties section, as shown in Figure 2-19.

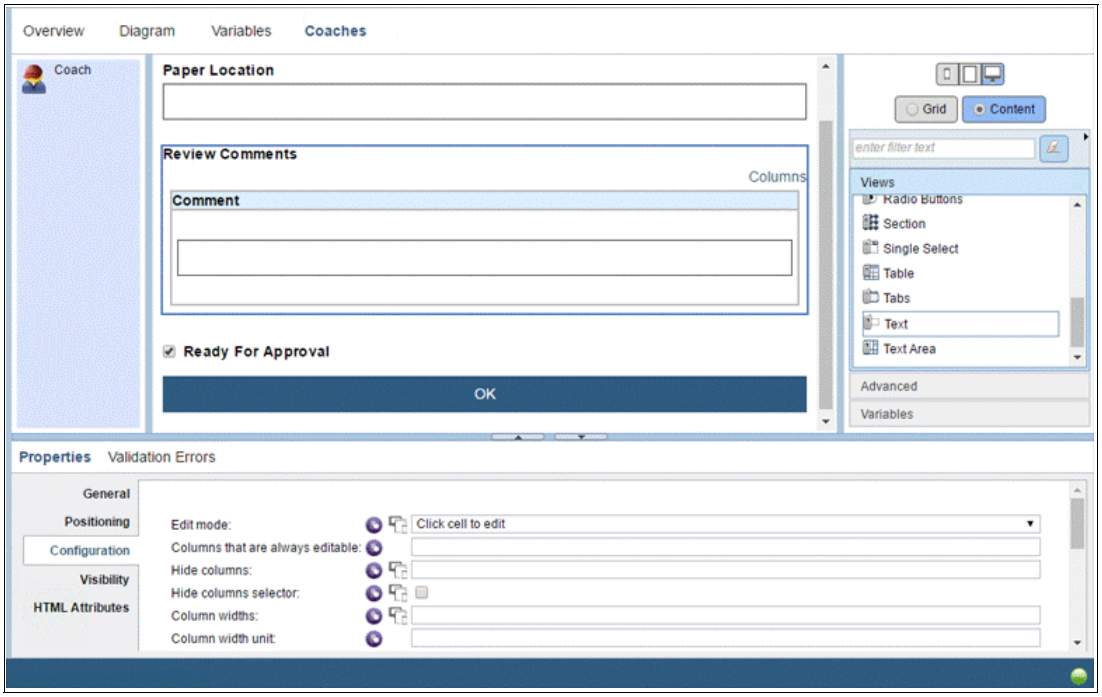


Figure 2-19 Control configuration options

By manipulating configuration properties, you can customize the behavior of a control.

To address the issues for this table control, you configure the table view, as shown in Figure 2-20. To make sure the selection option goes away, change the “Row selection type” field to **No Selection**. To allow comments to be added or removed, also select the “Show Add buttons” and “Show Delete buttons” options, respectively.

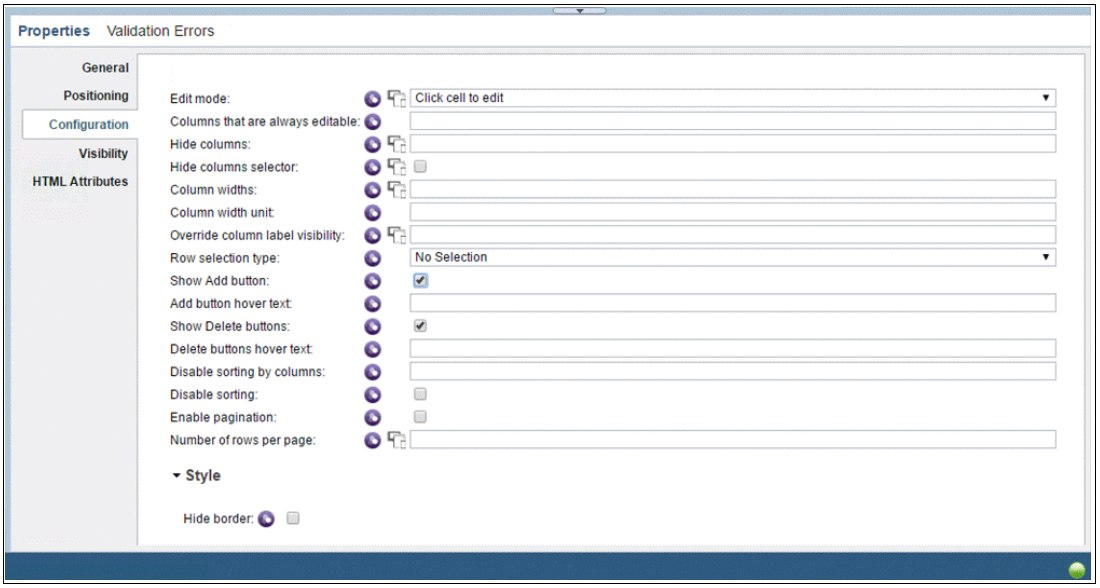


Figure 2-20 Configuration options for the Table control

Rerunning the coach now produces a result that is far closer to the expectations, as shown in Figure 2-21.

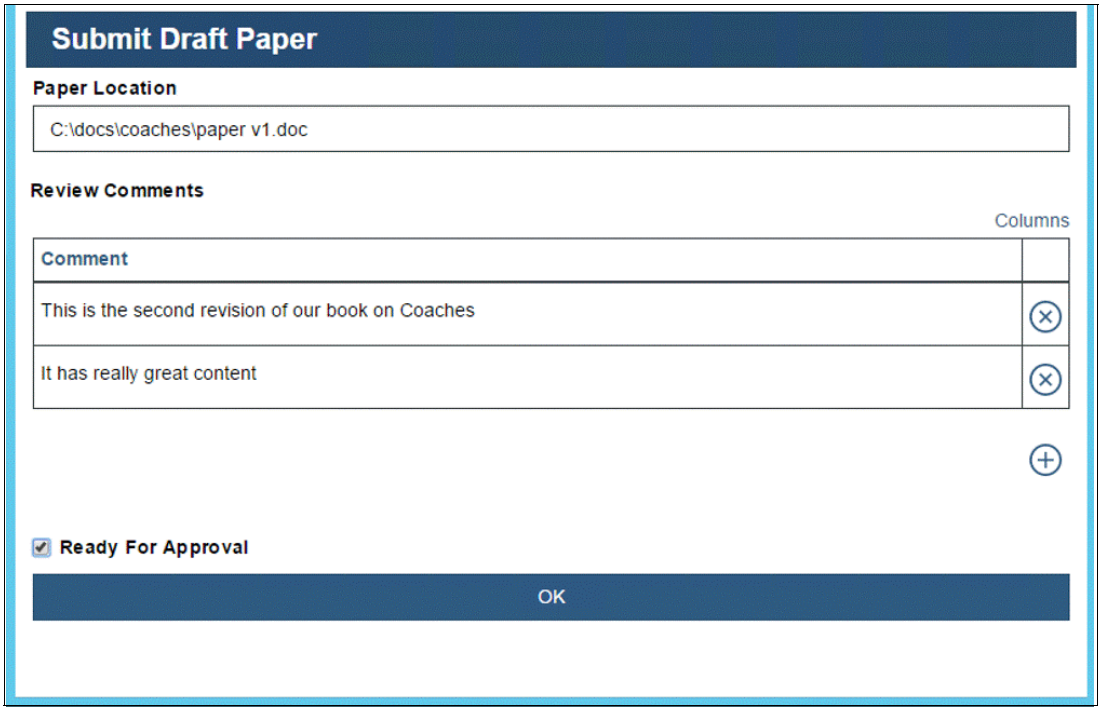


Figure 2-21 The Table control with add and remove buttons

2.3.4 Defining control visibility

Another powerful capability of coaches is the support for configuring the visibility of controls (and coach views in general). To define the visibility of a control, select the control on the canvas, and click the Visibility tab in the Properties pane (Figure 2-22).

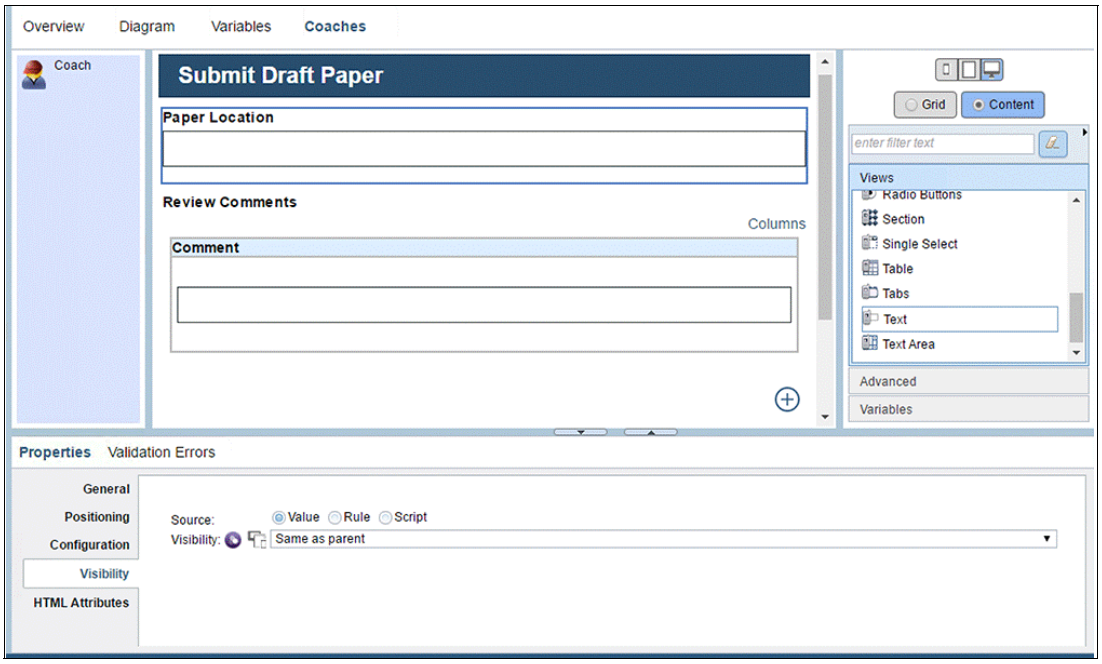


Figure 2-22 Defining the visibility of a control

By default the visibility of a control is the same as its parent. The parent of the Paper Location Text control is the Submit Draft Paper Section, and in turn the Section's parent is the coach, which means that everything is visible by default. That approach is in-line with the observation from the previous test run.

For this custom UI, you can modify the visibility of Paper Location based on the value of the Ready For Approval check box. When the check box is selected, the paper version is considered final, and thus no further changes of its location are allowed. You can define a rule for the visibility of the Paper location coach view by selecting the Rule radio button, as shown in Figure 2-23.

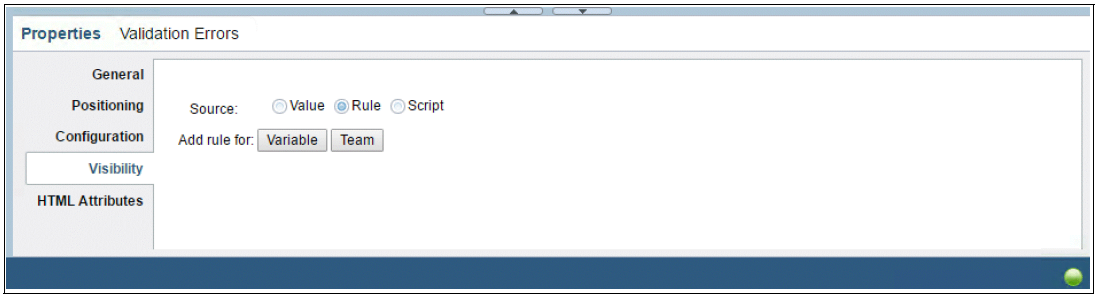


Figure 2-23 Selecting the visibility of a control to be based on a Rule

You can now add visibility rules that depend on variable values or team membership. In this example, you define a variable rule that makes the Paper Location control read only if the Ready For Approval check box is selected (in which case its value is true), as shown in Figure 2-24. To do that:

1. Click **Variable** to add a variable-based visibility rule.
2. Then, select the target visibility state “Read only” for the rule, and select the `readyForApproval` variable as the variable that this rule depends on.
3. Leave the condition as “Equal to” and set the value to `true`. Because this is a Boolean value, the coach Editor shows this as a check box in the rule definition, which corresponds to the Checkbox control that is used on the coach.

   If the rule does not apply, some default behavior occurs. The default behavior is defined by the Otherwise rule block, which by default assigns the control the same visibility of the control’s parent. Leave it that way.

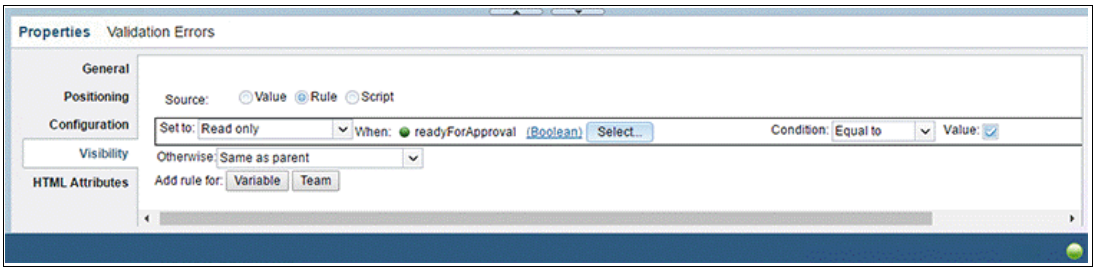


Figure 2-24 Defining a visibility rule for a control

At this point, you can add additional visibility rules for teams and variables to manage visibility, but leave things as they are and, instead, complete another test run.

As shown in Figure 2-25, what you were trying to accomplish works out nicely. Selecting **Ready For Approval** makes the Paper Location field read-only and clearing the option makes it editable.

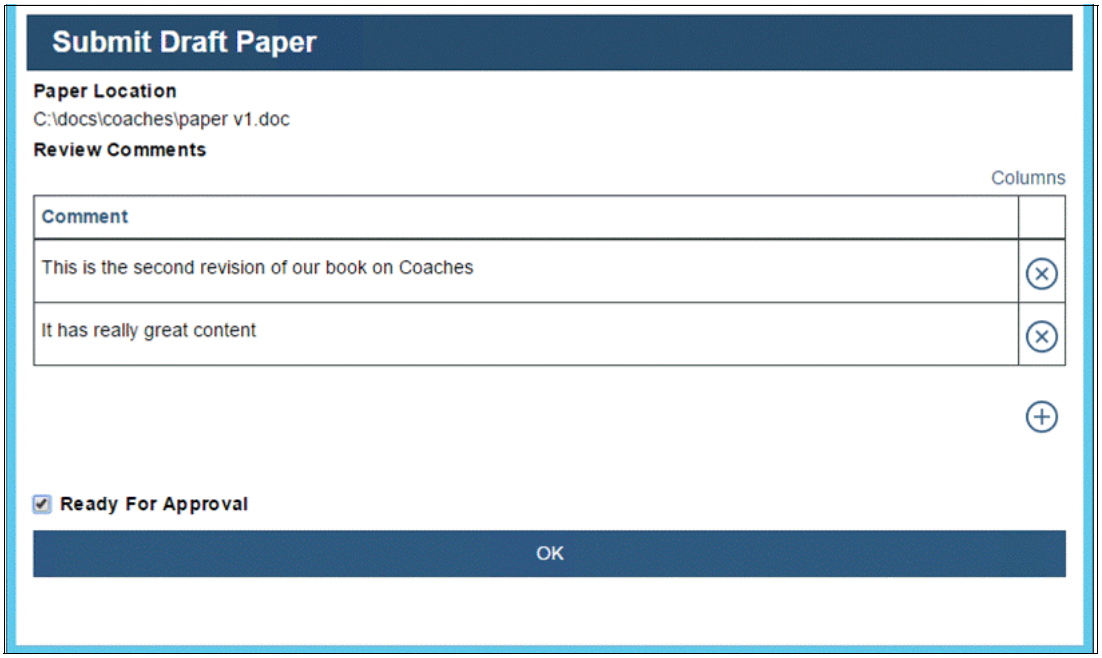


Figure 2-25 Testing the visibility rule by running the Submit Draft Paper client-side human service

For more information about setting control visibility, refer to IBM Knowledge Center:

http://ibm.biz/BPMCoachViewVisibilityProperties

2.3.5 Changing the positions of controls

Within this example, consider that there is too much white space between the table and the Ready for Approval check box. Conversely, the check box is too close to the OK button. So you can move the check box up by about 50 pixels without changing the relative position of the other controls on the coach.

The Position tab in the coach editor Properties pane provides a consistent way to position controls. You can use this to position the Ready for Approval check box, as shown in Figure 2-26.

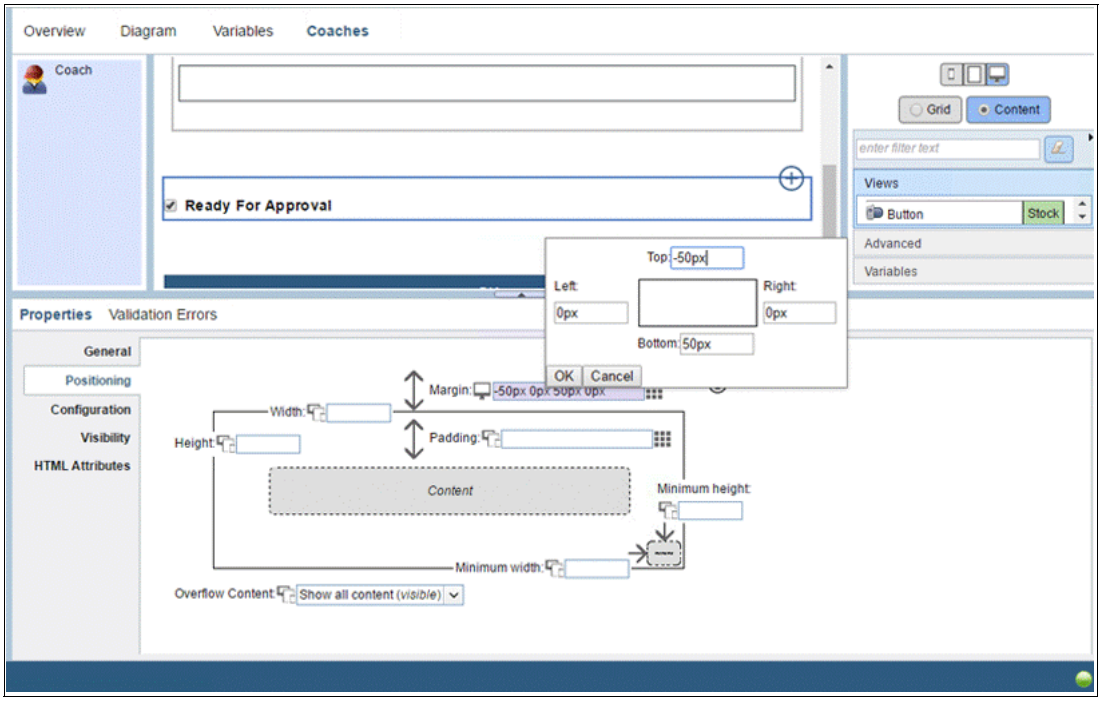


Figure 2-26 Defining the positioning of a control

In this example, you simply defined the top and bottom margins for the Ready for Approval check box, but there are a number of other attributes that can be used to govern the position of controls.

Note: The coach author gets immediate feedback when defining positioning attributes due to the WYSIWYG nature of the coach editor.

2.4 Control toolkits

The previous sections described the capabilities of coaches and the basic stock controls that are defined within the Responsive coaches toolkit. Because the Coach Framework is extensible, you can define new controls and add them to the palette.

Two more control libraries are provided as part of the product:

- Content Management
- Dashboards

To use these libraries, simply add appropriate toolkit dependencies using Process Designer, as illustrated in Figure 2-27. Using the Content Management toolkit as an example, click the plus icon next to Toolkits in the Process Designer main menu, and select the `8.5.7.0 snapshot` from the pop-up menu. The Content Management toolkit is then added as a dependency.

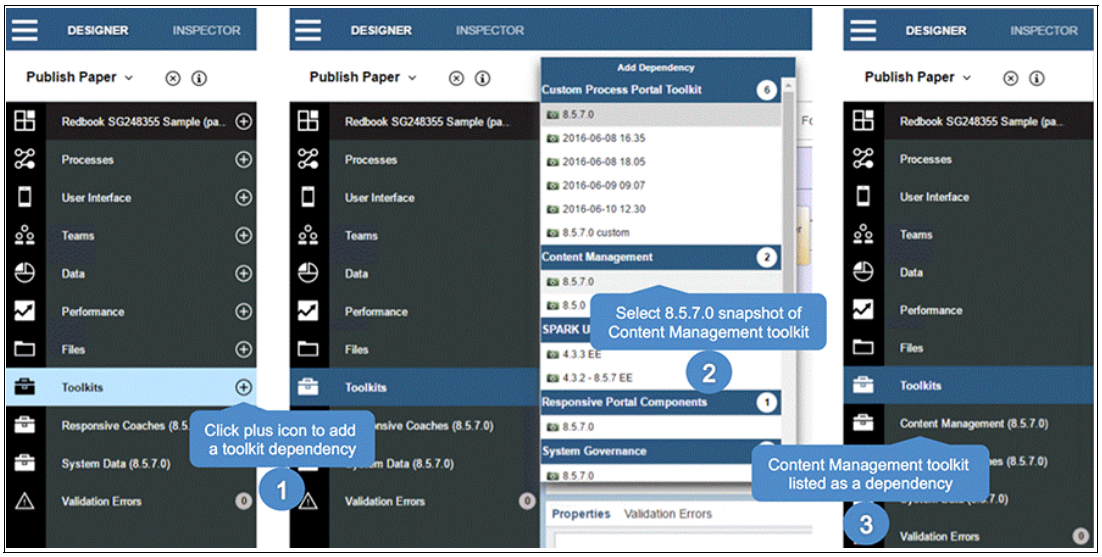


Figure 2-27 Add dependency for 8.5.7.0 snapshot of Content Management toolkit

The controls within the Content Management toolkit are detailed in the next section. The controls within the Dashboards toolkit are covered in Chapter 5, “IBM Process Portal” on page 261.

It is also important to point out that the mechanisms introduced here apply both to IBM provided controls and to control libraries that are provided by third parties, such as the Salient Process SPARK UI toolkit, which is introduced in Chapter 4, “SPARK UI Toolkit” on page 153.

2.5 Working with documents

Next, open the Submit Paper Draft coach again, as shown in Figure 2-28. After adding the Content Management toolkit as a dependency, notice that the document controls are now listed in the palette under the Content section.

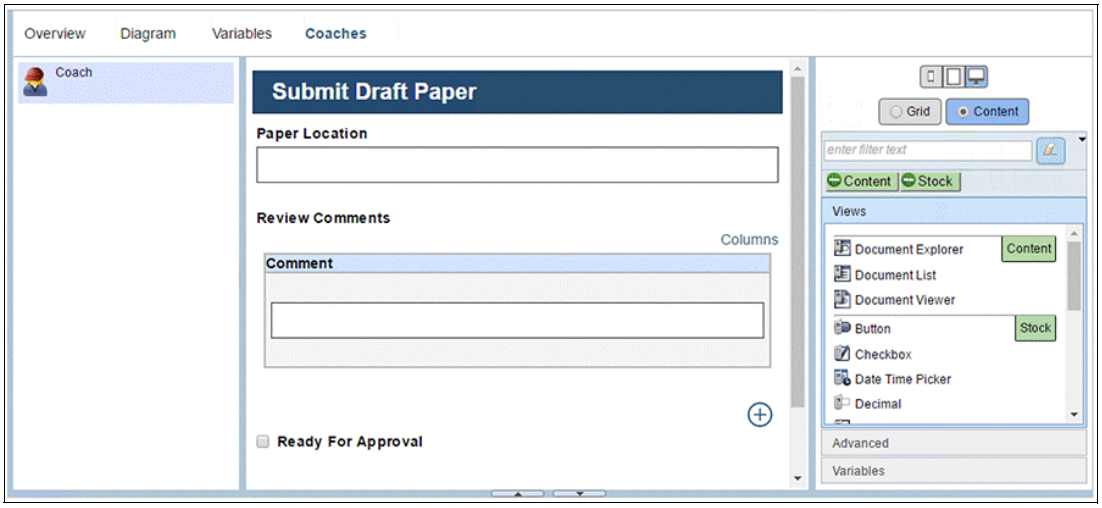


Figure 2-28 Content Management Controls now available in the palette

Using the palette: Controls in the palette are organized into groups according to their tags. The tag is used as the section heading and the controls within each section are listed alphabetically. Controls can be filtered either according to their tag or by name.

The content management controls include:

- Document Explorer
- Document List
- Document Viewer

To learn more about the content management controls refer to IBM Knowledge Center:

`http://ibm.biz/BPMContentManagementToolkit`

These document controls are supported by Asynchronous JavaScript and XML (Ajax) services with Content Management Interoperability Services (CMIS) operations. They can be used either with the internal BPM document store or an external Enterprise Content Management (ECM) system that supports CMIS. By default the BPM document store is an embedded IBM FileNet® ECM that is managed by IBM BPM. External ECM products can be used by simply defining them in the Process App Settings and configuring the document controls accordingly.

For this example, you can use the internal BPM document store to continue with the sample Publish Paper process app.

2.5.1 Document List control

Instead of simply referencing the document file name as input text within the sample, you can use the Document List control to attach documents and to store them as part of the process. The Paper Location control is no longer required and can be deleted. Add the Document List

control by dragging it from the palette and dropping it on the canvas, positioning it at the top of the section with the Submit Paper Draft coach, as shown in Figure 2-29.

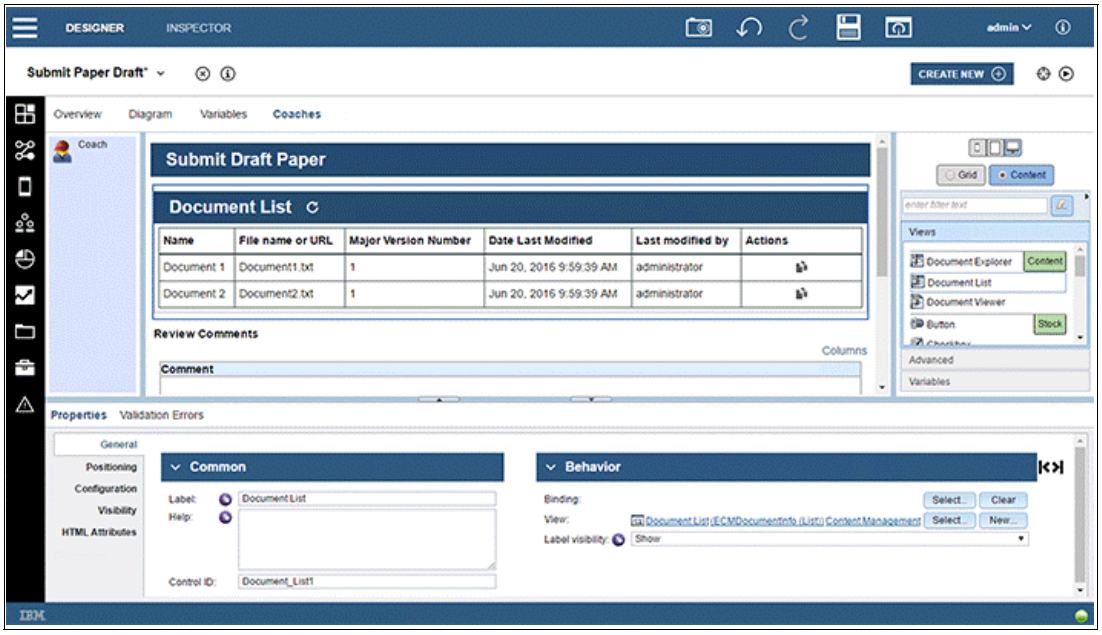


Figure 2-29 Document List control

You can now configure the Document List, as shown in Figure 2-30. Follow these steps:

1. Select the Allow Create, Allow Update, and Allow.
2. Delete options to add buttons to enable the user to create, update, and delete documents.
3. Check the “Open in new window” option to enable the user to view documents within an external viewer.
4. Set the “Search results per page” option to 5 to enable the user to see up to five documents at once and page through five documents at a time.

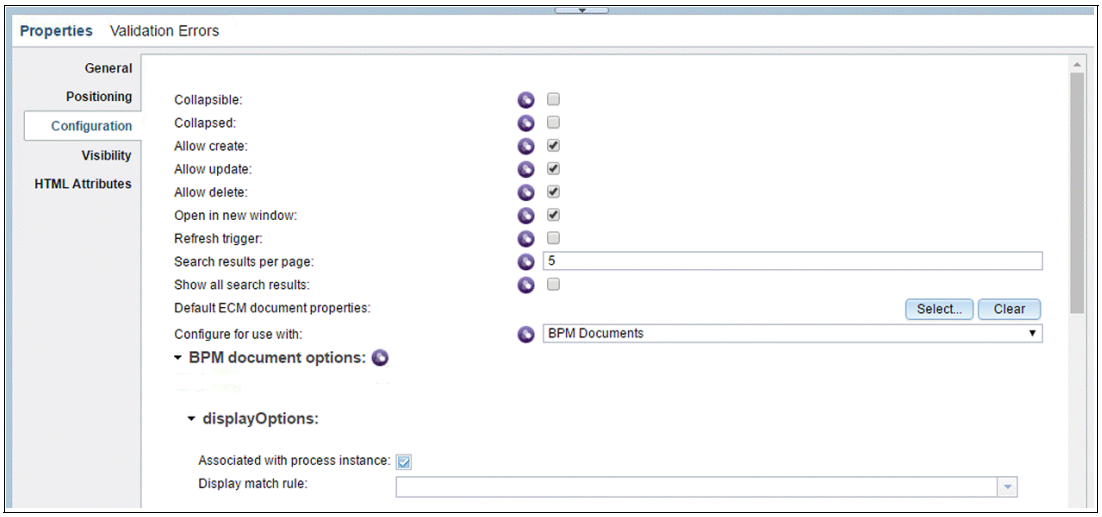


Figure 2-30 Document List configuration options

5. Another important configuration change is to ensure that you only display documents that are associated with the corresponding process instance. Doing so allows multiple

processes to run in parallel, each operating on a distinct set of documents. To make that change, leave BPM Documents as the chosen document store, drill into **BPM document options** and then **Display options**, and select **Associate with Process Instance**.

6. To verify the effectiveness of the configuration changes, start the process (not the client-side human service) in the playback environment, because you chose to display only documents that are associated with a running process instance. To launch a new process instance, open the Publish Paper process in Process Designer and press the play button, as shown in Figure 2-31. The Inspector view opens and shows the active tasks for this process instance. The Submit Paper Draft task should be active. Click the play button next to the task to launch it.

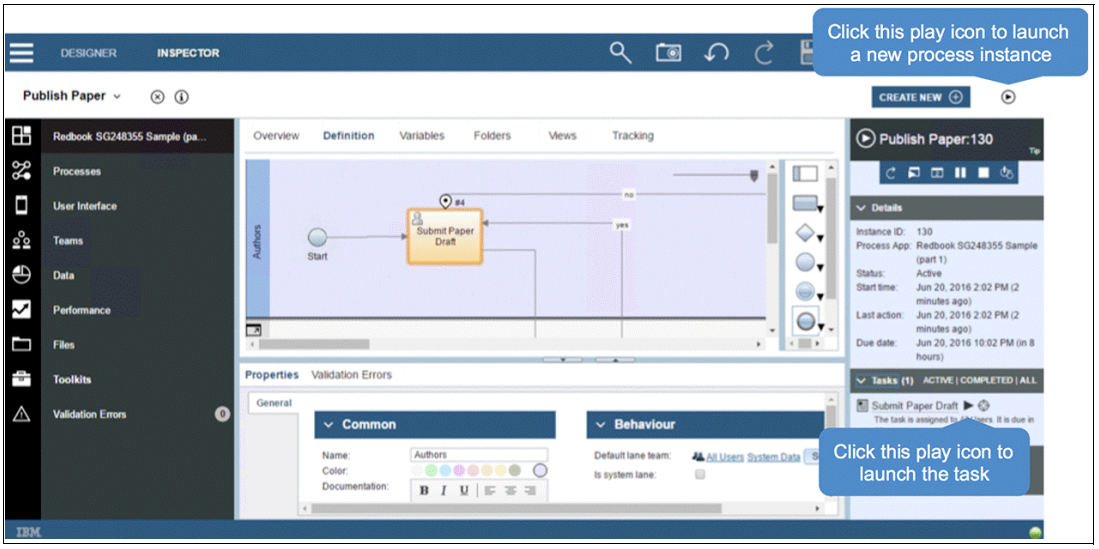


Figure 2-31 Launching a task from a new process instance

Figure 2-32 shows the Document List when the task is executed the browser window.

Figure 2-32 Testing the Document List control

The new Document List control displays, but no documents are added yet. Therefore, the list is empty. Continue the test by pressing the plus (+) icon on the Document List header to attach a document to the business process instance. In the dialog that opens, locate a document on your file system, and give it the name. Click **OK** to return to the coach with the new document listed in the Document List control, as illustrated in Figure 2-33.

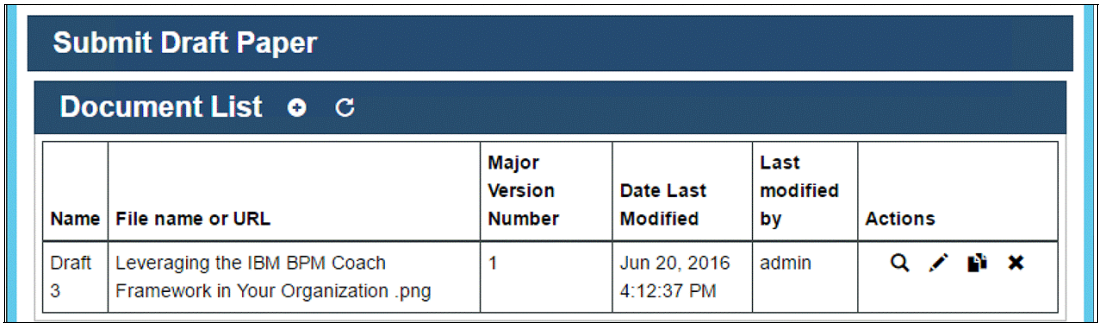


Figure 2-33 Documents listed in the Document List

After documents are added you see a number of action icons on the right side for each document listed, as follows:

- Selecting the magnifying glass (*view*) icon next to a document displays the document in an external viewer, as illustrated in Figure 2-34.
- Selecting the pencil (*update*) icon enables the user to upload a new version of the document.
- Selecting the pages (*revisions*) icon enables the user to view a specific version of the document.
- Select the cross (*delete*) icon removes the document attachment.

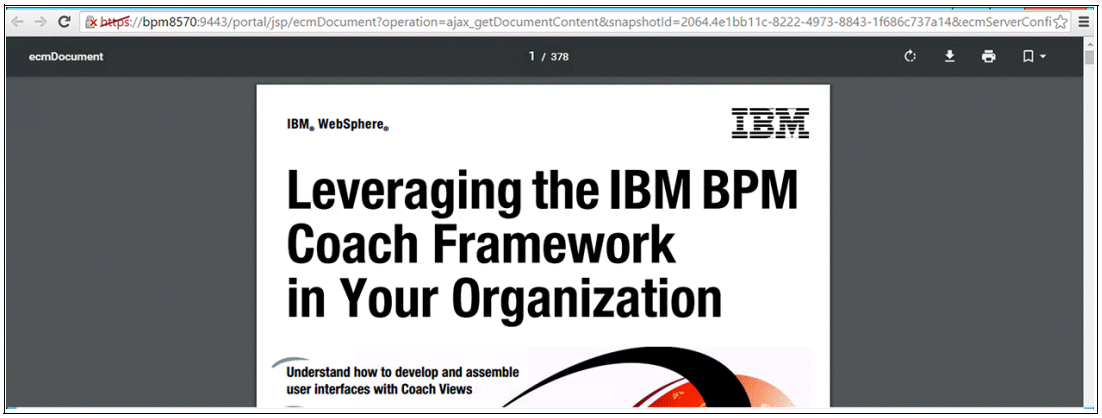


Figure 2-34 View the document in an external viewer

You can also configure the Document List in other ways. Display options provide the ability to configure the Document List so that it displays only documents that match specific property values. The “Associated with process instance” option is actually a predefined property that is used to find documents with a matching process instance identifier. When this option is not selected, the same document might be visible from multiple process instances.

Properties are automatically associated with documents when uploaded using the Document List control. The process instance identifier is associated with documents by default. However, you can also specify custom properties within the Document List upload options,

which are used to filter documents when displayed in the Document List, as described in the previous paragraph.

2.5.2 Document Viewer control

You use the Document Viewer to view the document content with the coach instead of launching it in a separate window. You can use the Document Viewer control either independently or in collaboration with the Document List control. Collaboration with the Document List control is achieved by using the binding variables to communicate between the two controls. Within the Submit Paper Draft client-side human service, set up a private variable, name `documents (ECMDocumentInfo) (List)`, as shown in Figure 2-35.

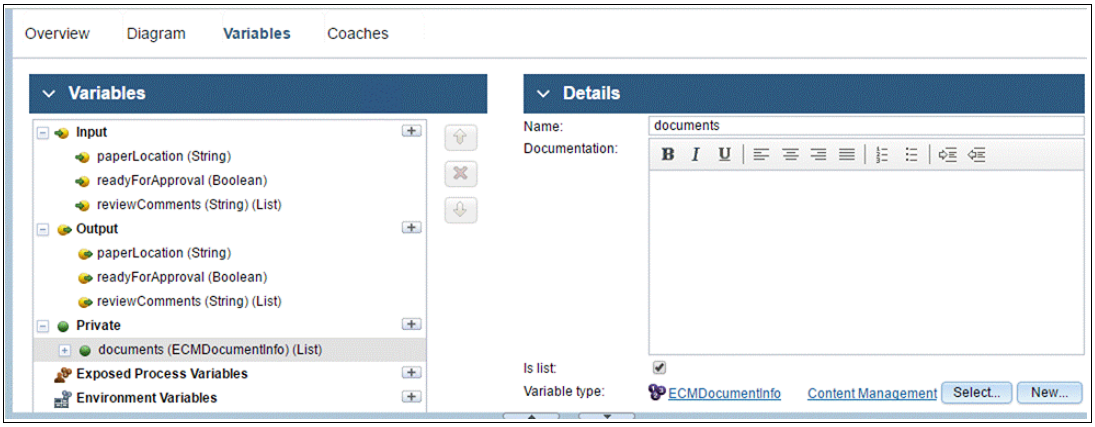


Figure 2-35 Use a private variable to communicate between the Document List and Document Viewer

Then, use the private variable to set the binding for the Document List control, `documents[] (ECMDocumentInfo)`, as shown in Figure 2-36.

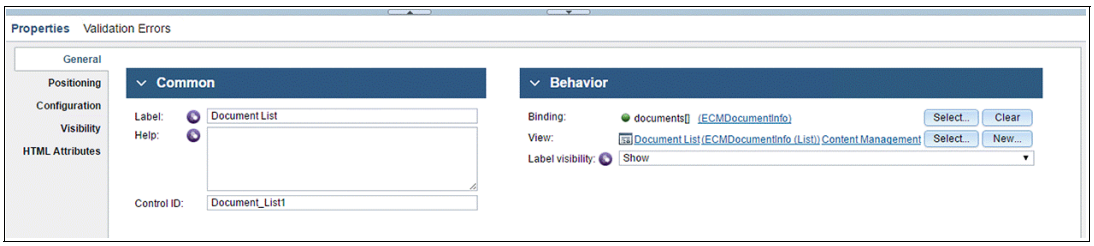


Figure 2-36 DocumentList control with Binding (ECMDocumentInfo)

You can add the Document Viewer control by dragging it from the palette and dropping it on the canvas so that it is positioned under the Document List within the Submit Paper Draft coach, as shown in Figure 2-37. Use `documents.listSelected (ECMDocumentInfo)` as the binding for the Document Viewer control.

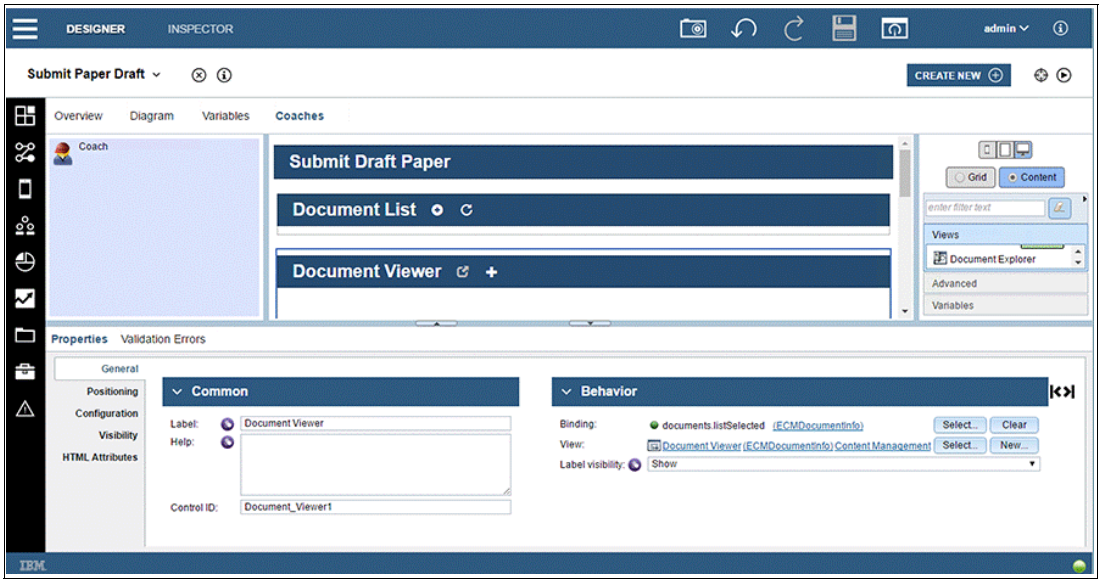


Figure 2-37 Document Viewer control with Binding: documents.listSelected (ECMDocumentInfo)

Finally, open the Document List configuration tab, and clear the “Open in new window” option, so that new documents open it the Document Viewer instead.

Now, to test these changes, rerun the process, and execute the task as before. The Document Viewer should now display the document in the coach, as shown in Figure 2-38.

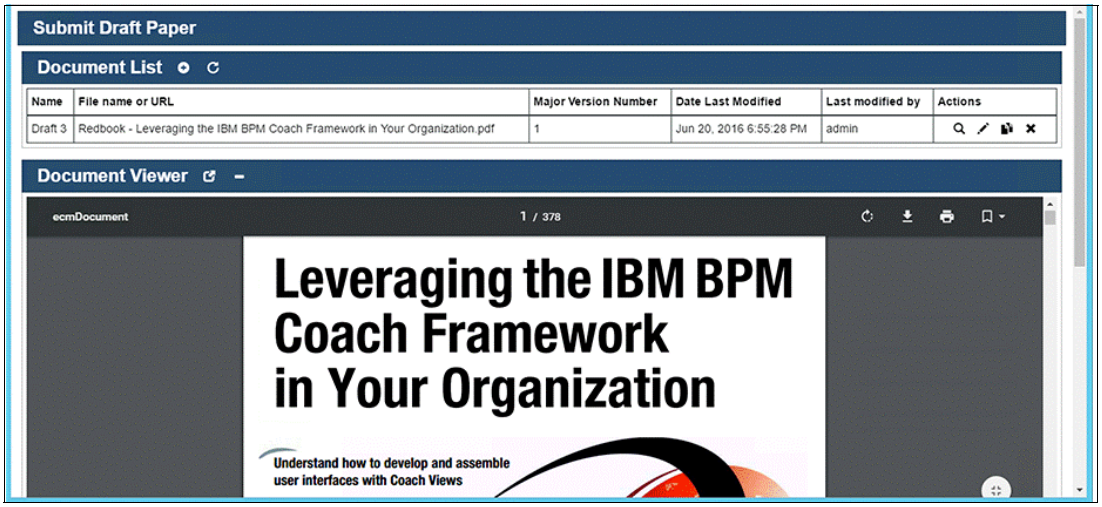


Figure 2-38 Testing the DocumentViewer control

2.5.3 Document Explorer control

The Document Explorer control has similarities to the Document List but is used to browse documents in a structured folder hierarchy. Similar to the Document List control, you can use the Document Explorer either in isolation or in collaboration with the Document Viewer to

display document content. Unlike the Document List, the Document Explorer provides a federated view of documents and folders that can be stored either locally within IBM BPM or referenced from one or more external ECM servers, as shown in Figure 2-39. Unlike the Document List, a search query cannot be provided. The Document Explorer shows all the contents of the folder currently in focus.

Note: Local BPM documents are stored in the embedded FileNet ECM and folders are stored in the BPM managed store (in the BPM database). External ECM document and folder references are also stored in the BPM managed store, but the referenced documents and folders are stored in the external ECM servers.

Figure 2-39 The DocumentExplorer control

You can also associate the Document Explorer with any process instance, so that it can easily be used within dashboards. In fact, you can use the Document Explorer to provide the case folder within the default Process Instance UI, as described in Chapter 5, “IBM Process Portal” on page 261.

Note: The Document Explorer can be configured with either a folder identifier or a process instance identifier. For the later, the corresponding process folder identifier is derived by the view. If neither is provided, the Document Explorer cant determine the Process folder identifier from the running process instance.

Local documents and folders are managed by IBM BPM itself and are removed when the process instance is deleted. Similarly, external ECM folder and document references are removed when the process instance is deleted, but the referenced folders and documents themselves that reside in the external ECM servers are not removed.

2.6 Creating a reusable group of controls (using a composite coach view)

When you have a group of coach controls that you want to reuse, either on the same coach or other coaches, you can create a *composite coach view*. This is the only time the coach author needs to build a coach view, but creating a composite coach view is relatively simple compared to using coach views to build custom controls, which is covered in Chapter 3, “Building controls using coach views” on page 73.

Continuing with the sample, you want to capture the details of the author on the coach. You first need to define a composite coach view here, because you want to reuse it on another coach later in the process.

A composite coach view is associated with a complex business object that represents the data that is associated with the individual controls on the coach view. Before creating the composite coach view, start with the business object.

Create a *Person* business object, as illustrated in Figure 2-40. This business object includes attributes to capture personal details, address, and costs, as follows:

- forename (String)
- surname (String)
- dateOfBirth (Date)
- address (Address)
- gender (Boolean)
- hourlyRate (Decimal)
- hours (Integer)

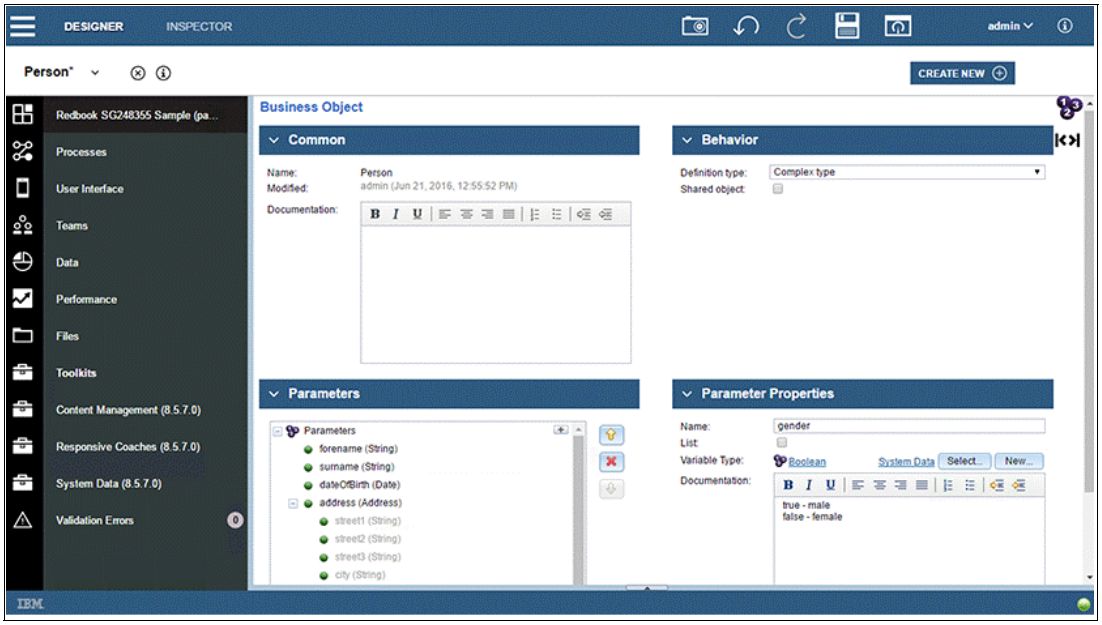


Figure 2-40 Example Person complex Business Object

Address is another complex business object that you also need to define:

- `street1` (String)
- `street2` (String)
- `street3` (String)
- `city` (String)
- `state` (String)
- `country` (String)
- `zipCode` (String)

Now that you have defined the Person business object, you can create the corresponding coach view. Follow these steps:

1. Create a new coach view, and give it a name (`Person`). Select the “Intended for use on multiple devices” option. The coach view editor opens automatically after the coach view has been created.
2. Within the Variables tab define the `person (Person)` business data variable, as shown in Figure 2-41 on page 37, which associates the Person coach view with the Person business object.

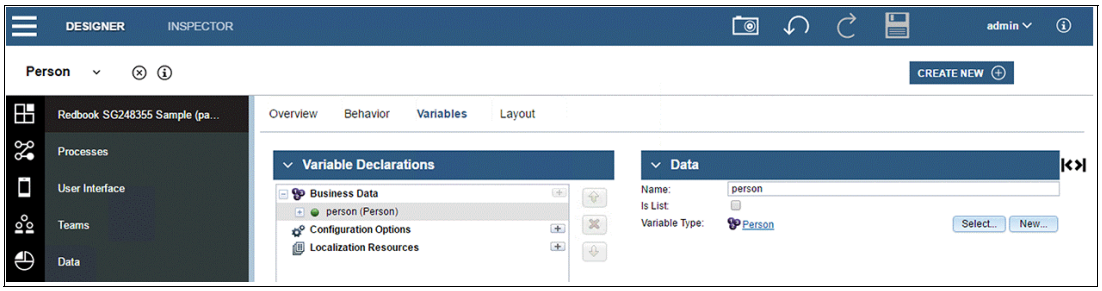


Figure 2-41 Example Person coach view, Variables tab

3. Select the Layout tab, and drag the person variable from the palette onto the coach view canvas. This action automatically assembles the controls that are associated with the data types that correspond to the business object attributes, as shown in Figure 2-42.

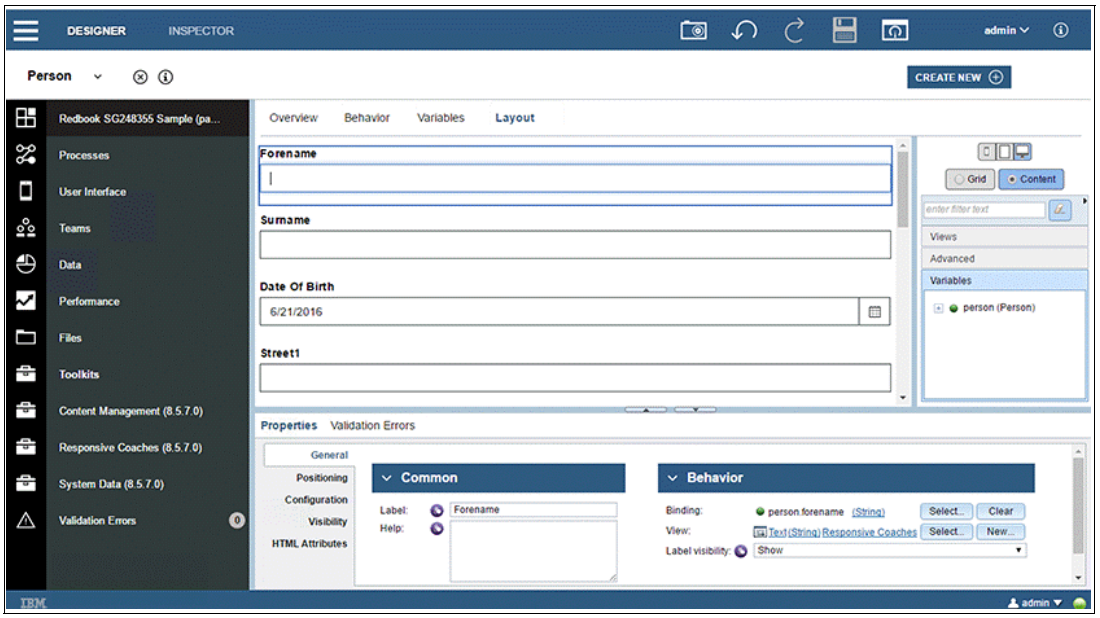


Figure 2-42 Example Person coach view, Layout tab

4. Now within the sample process, add three sections to wrap the personal details, the address fields, and the costs respectively. You can configure sections with either a vertical or horizontal layout to determine how elements contained within the section are arranged relative to each other. By default sections have a vertical layout. For this example, use vertical layout within the sample process.

   You can also change the configuration of the Gender check box to be rendered as a switch, as shown in Figure 2-43 on page 38, by setting the True label to **Male** and the False label to **Female**.

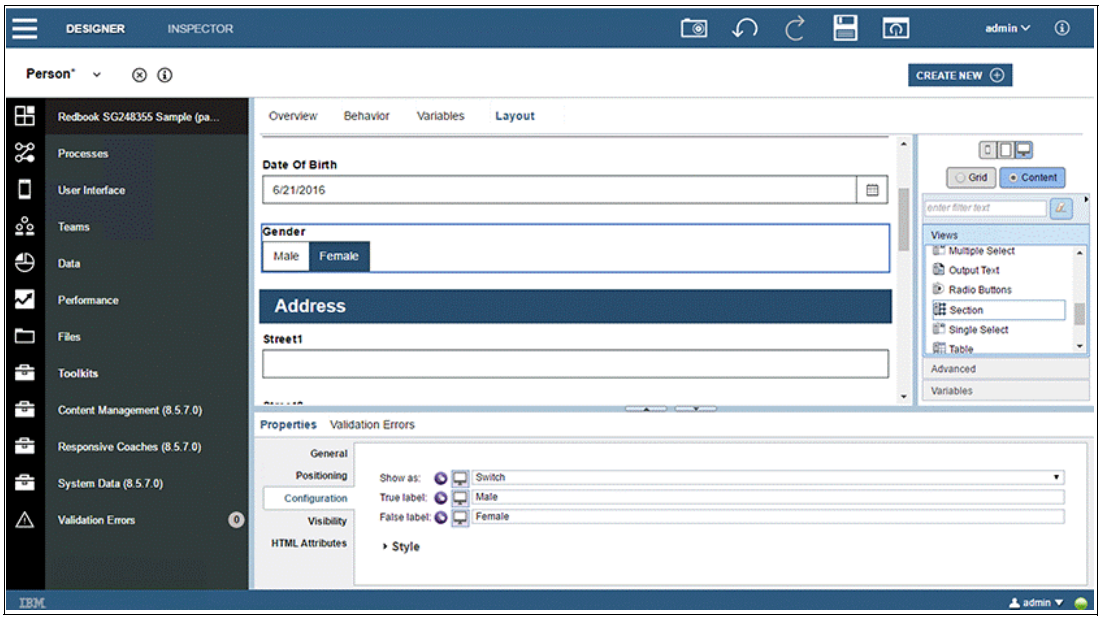


Figure 2-43 Configure Gender check box control

5. Next, go back to the coach. The Person coach view is now available on the palette. Drag the Person coach view onto the coach canvas, and position it at the top of the vertical section, as illustrated in Figure 2-44.

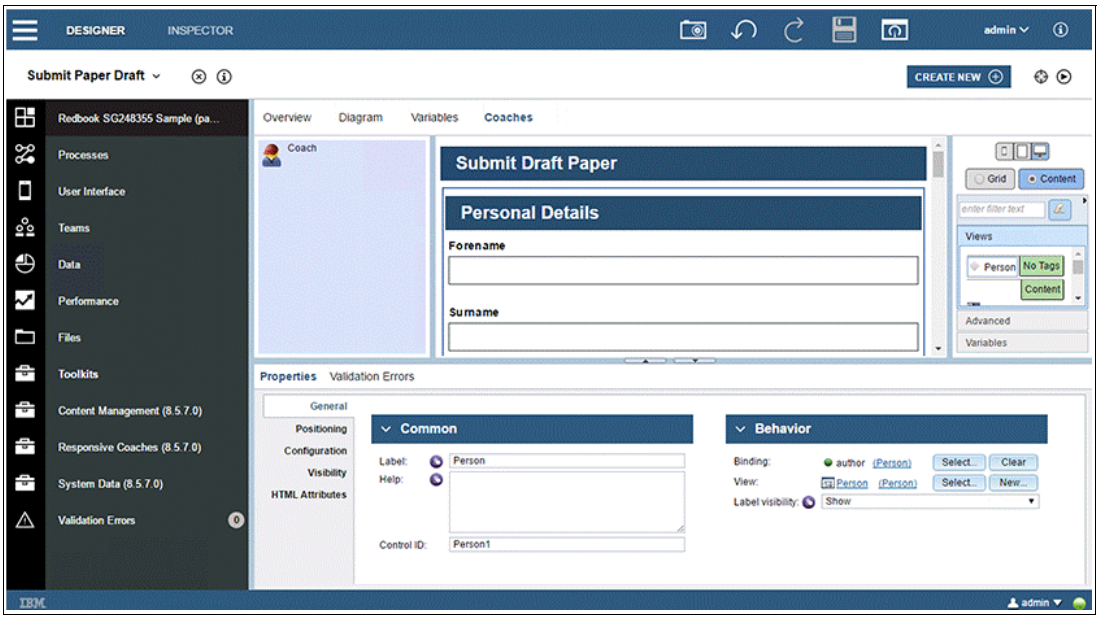


Figure 2-44 Example Person coach view, Layout tab

6. Now you need to define the coach view binding. Using the Variables tab, add input and output variables for `author (Person)`. Go back to the Coaches tab and use the `author` variable as the binding for the Person coach view, as illustrated in Figure 2-44. To enable the process to run add a private `author (Person)` variable to the Publish Paper process and add appropriate data bindings.
7. Finally, test the coach again. Notice that the Personal Details for the author display as required, as shown in Figure 2-45 on page 39.

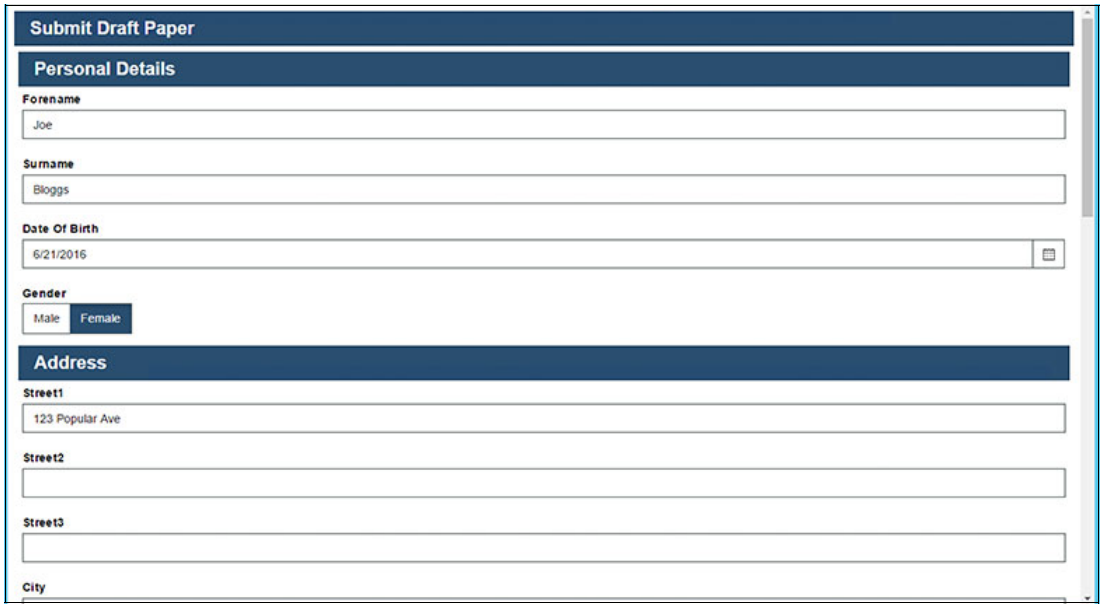


Figure 2-45 Testing the composite coach view

Improved reuse: For improved reuse, you can create another composite coach view for Address and use this composite coach view within the Person coach view instead of adding the individual address fields. This method allows the Address coach view to be used independently.

2.7 Adding validation to a coach

When building a custom UI for a BPM solution, it is important to validate the data entered by the user to:

- Improve the data quality by filtering out unexpected data.
- Enhance the process efficiency by providing user guidance.
- Harden the solution's security by detecting false information.

IBM BPM 8.5.7 includes the following validation techniques when using client-side human services:

- Use required visibility (visual indicator only)
- Control configuration options (client-side validation)
- Data Change event handler (client-side validation)
- Client-side human service validation script (client-side validation)
- Server-side validation service (server-side validation)

These techniques are detailed further in the following sections.

2.7.1 Required visibility

The visibility of a control can be set to required, as shown in Figure 2-46 on page 40. This setting does not actually perform any validation logic or warn users when required fields are left blank, but it does provide users with a visual indicator at all times. An asterisk (*) displays

beside each require field. Required visibility must be combined with another validation technique to enforce required data entry.

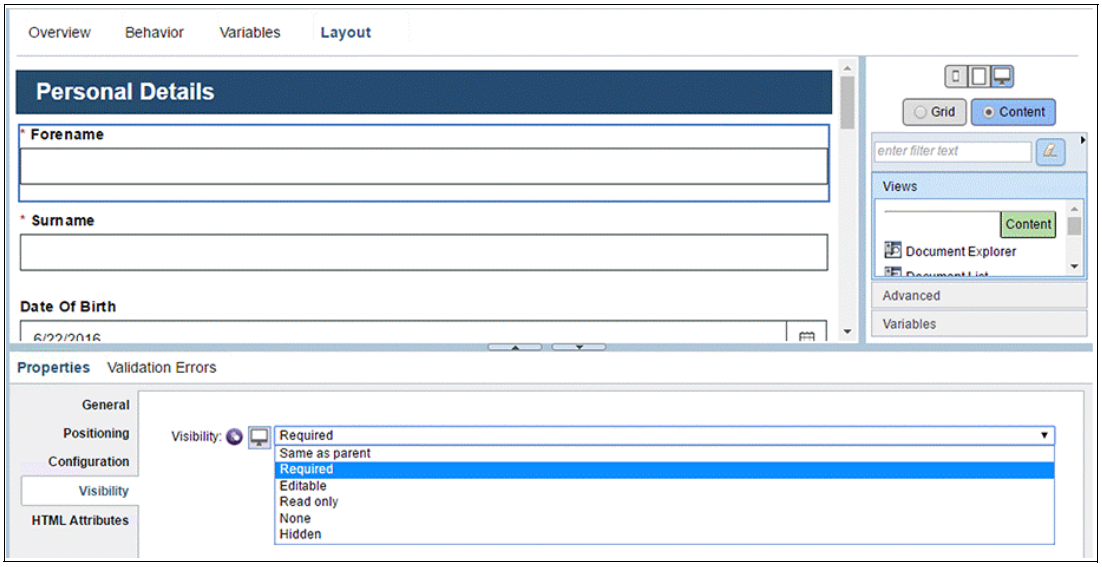


Figure 2-46 Using Required Visibility

2.7.2 Control configuration options

Within a coach you can use the configuration options of specific controls to prevent the user entering invalid data in real time. For example, the stock Decimal control provides validation options for maximum and minimum values, and the stock Text control provides validation options for pattern matching and restricting the length of the entered text.

Figure 2-47 illustrates the configuration of the Hourly Rate field within this sample.

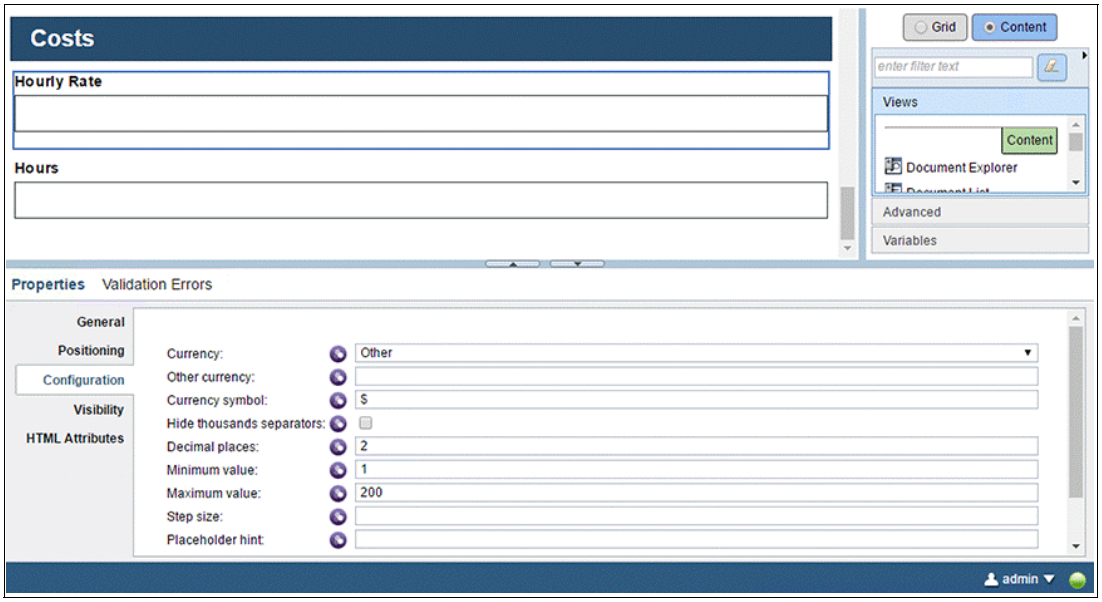


Figure 2-47 Validation options for stock Decimal control

In some cases the user is prevented from entering invalid data, such as with maximum length in the Text control. Otherwise, the user is warned when invalid data is entered, as shown in Figure 2-48. In this case, on closing the warning dialog box, the field data is returned to its previous value, which also prevents the user from entering invalid data.

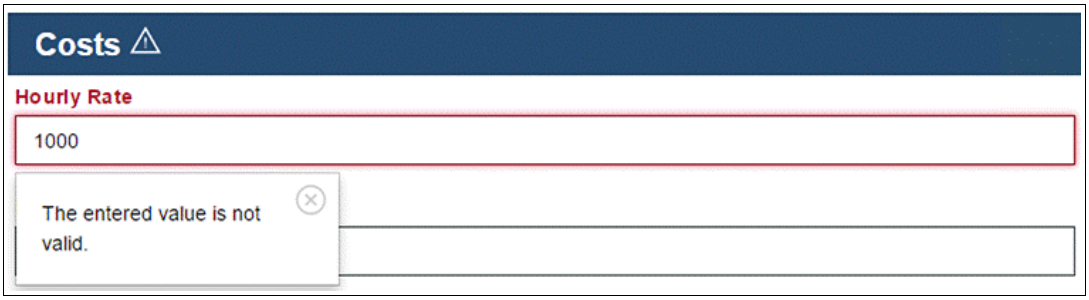


Figure 2-48 Validate field using configuration option

2.7.3 Data Change event handler

Each coach within a client-side human service has a Data Change event handler. You can add a client-side JavaScript that executes each time the user updates data on the coach. Within this sample, the JavaScript shown in Example 2-1 is added to the Data Change event handler for the coach in the Submit Paper Draft client-side human service, as illustrated in Figure 2-50.

Example 2-1 Validation logic for the sample Data Change event handler

```
var today = new Date();
if (tw.local.author.dateOfBirth.getTime() > today.getTime()){

   tw.system.coachValidation.addValidationError(
"tw.local.author.dateOfBirth",
"Date of birth must be a date in the past");
}
```

Validation warnings display as they did previously (as shown in Figure 2-49 on page 42).

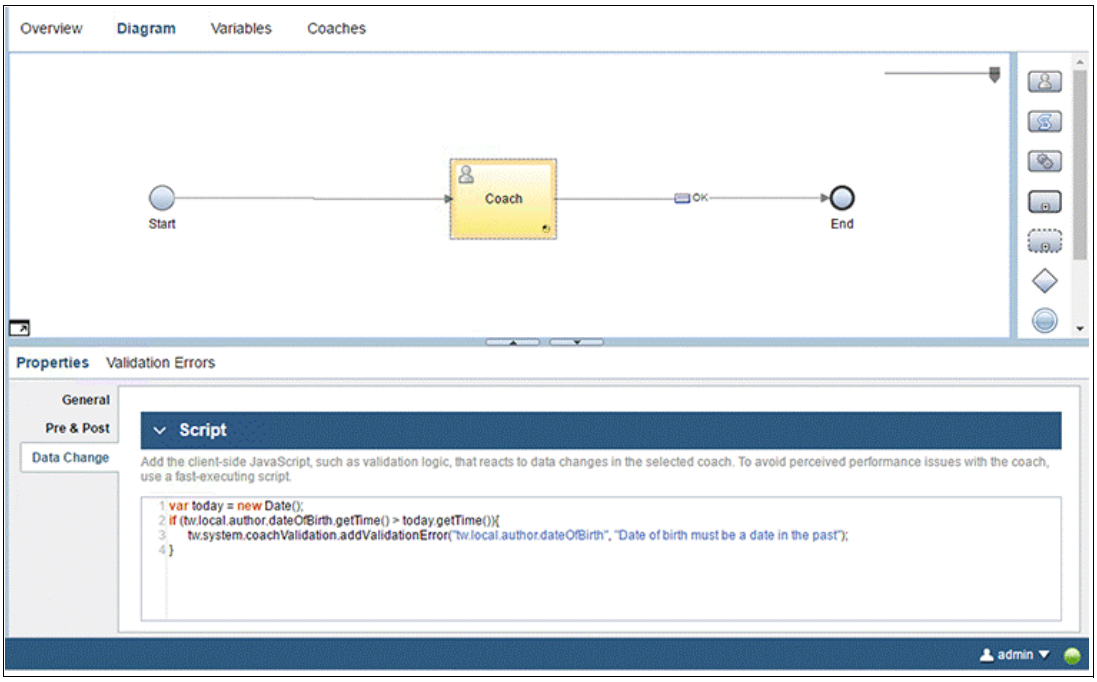


Figure 2-49 Validate coach using the Data Change event handler

By default, this event handler does not prevent the coach from being submitted where validation errors remain, but this issue can be remedied easily by disabling buttons when there are validation errors, as shown in Example 2-2.

Example 2-2 Script used to disable buttons within the Data Change event handler

```
if( tw.system.coachValidation.validationErrors.length == 0 ){

   tw.local.ready = "DEFAULT";
}
else{

   tw.local.ready = "READONLY";
}
```

The sample `tw.local.ready` includes a private variable added to the Submit Paper Draft client-side human service that is used to set the visibility of the OK button, as shown in Figure 2-50.

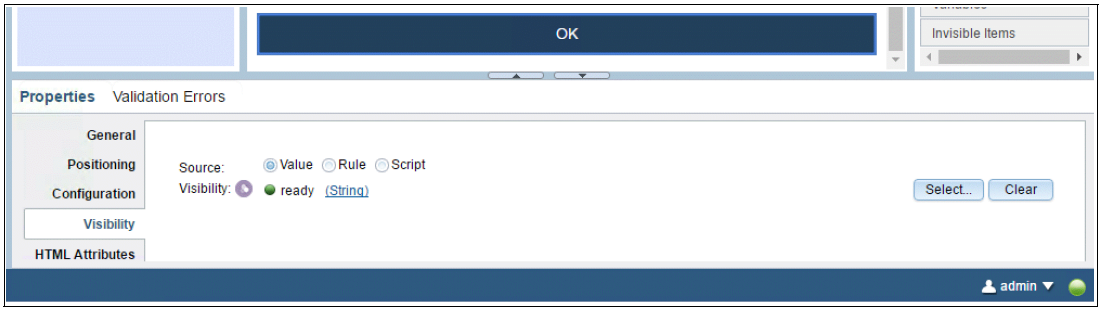


Figure 2-50 Setting the OK button visibility prevents submission while there are remaining errors

The Data Change event handler is called whenever the user updates form data and does not indicate what actually changed. So you must be careful not to define too much validation logic that is slow to execute, because doing so can impact the performance of the coach.

2.7.4 Client-side script

You can also add validation logic to a client-side human service in a client-side script. The script can be executed on firing a boundary event, such as pressing a button, as shown in Figure 2-51.

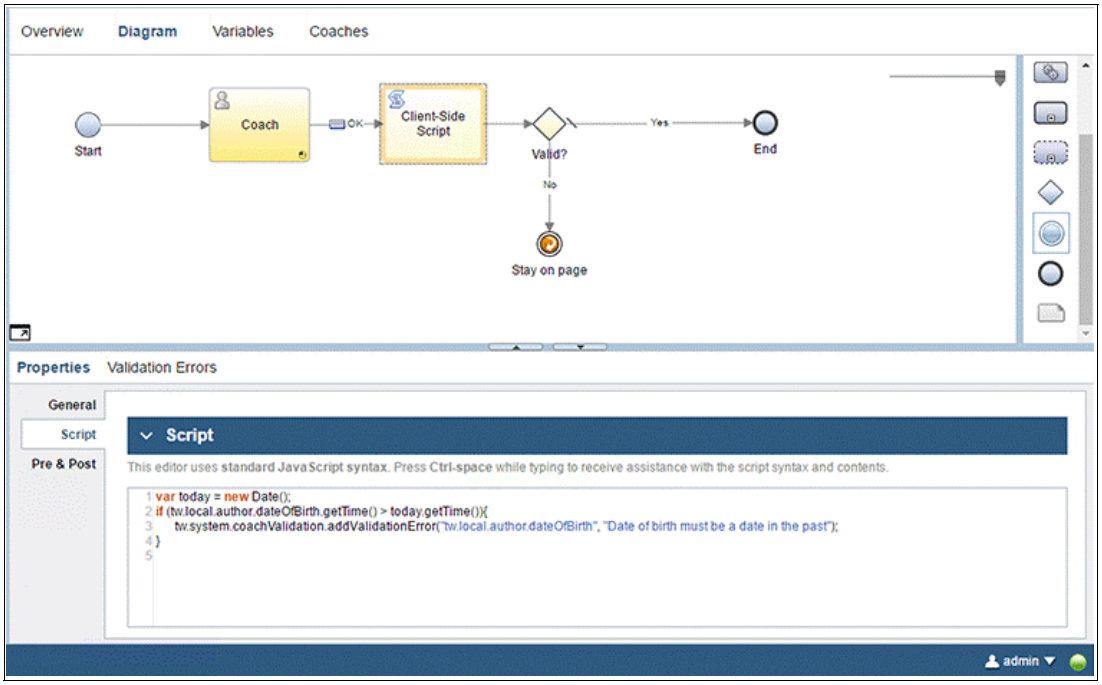


Figure 2-51 Validate coach using client-side script

The sample uses exactly the same JavaScript code previously defined for the Data Change event handler to define the validation logic for the client-side script in the Submit Paper Draft client-side human service (see Example 2-1 on page 41).

Immediately following the client-side script, use a decision gateway to test for any validation errors:

```
If tw.system.coachValidation.validationErrors.length == 0
```

If the coach is valid, and you can move on. Otherwise, if validation errors occur, you stay on the coach.

2.7.5 Server-side service

This service uses the same pattern as performing validation using a client-side script, except that a service is used to define the validation logic and is executed on the BPM process server.

2.7.6 When to use each validation technique

When adding validation to a coach, you need to determine which technique is the most appropriate. At a high level it comes down to the following basic considerations:

- Security
- Performance
- User experience

Client-side validation improves the user experience by providing immediate feedback to the a user and improves performance, because only valid requests are sent to the server.

Server-side validation is needed for security. Any client-side validation technique can be bypassed by tampering with the validation logic within the browser.

Server-side validation also provides the ability to integrate with back-end systems. External systems can either provide essential data that is required to perform the validation on the BPM server or can actually provide validation services that are executed outside of IBM BPM.

You can choose to always use *server-side services* for coach validation needs, but the advantage of using client-side logic is performance and improved user experience. Use client-side scripts as a convenient way to guide users but use server-side validation as the final security check, if needed.

Use the *required visibility*, *coach configuration options*, and *data change event handler* methods where you want to provide users with immediate feedback without having to submit the coach. Using *required visibility* does not prevent the coach from being submitted with outstanding validation errors. So, combine this method with other validation techniques to ensure that the coach can be submitted only when all validation errors are resolved.

Consider using coach configuration options first because this method both provides error warnings and prevents users from entering invalid data in the first pace. Only use data change event handler when coach configuration options are not available to maximize coach performance.

Refer to IBM Knowledge Center for further reading on coach validation:

`http://ibm.biz/BPMCoachValidation`

2.8 Deriving data from other fields

When building an interactive UI, coach authors often need to display information that is derived from other fields on the coach, so that the information is dynamically updated when the field values are modified.

For example, this might be a simple calculation, such as *costs = hourly rate * number of hours*, might require adding up all the values in a table column, such as *total hours = sum of all rows (hours)*, or some other logic, such as *if a < 10 then risk = “low”, if a < 20 then risk is “medium”, otherwise risk is “high”*.

The following illustration uses the first of these examples to update the sample process.

You can use the Data Change event handler introduced in 2.7.3, “Data Change event handler” on page 41 to define derivation logic for a coach. The JavaScript shown in Example 2-3 is added to the Data Change event handler for the coach in the Submit Paper Draft client-side human service.

Example 2-3 Derivation logic for the sample Data Change event handler

```
tw.local.cost = tw.local.author.hourlyRate *
                tw.local.author.hours;
```

To make this process work, you also add `cost (Decimal)` as a private variable to the Submit Paper Draft client-side human server and add a read-only `Decimal control` to the coach bound to `tw.local.cost`.

Figure 2-52 illustrates the result when rerunning the Submit Paper Draft client-side human service. The cost initially displays when the coach opens and dynamically updates when either hour rate or hours are modified.

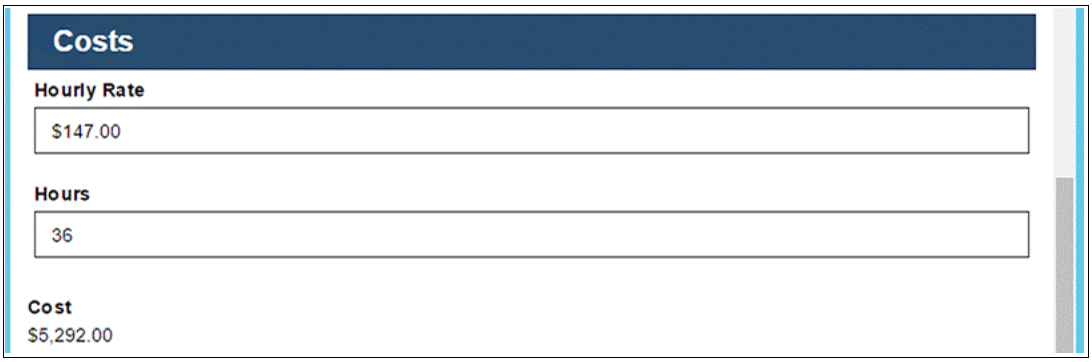


Figure 2-52 Derived information displayed on a coach

2.9 Making a coach responsive

You can use modern web-based UIs on a variety of devices, including desktop computers, tablets, and smartphones. To enable this without having to develop separate user interfaces for each device, the UI can be made to be *responsive*, so that the behavior and layout adapts depending on the available screen size (or form factor).

In IBM BPM 8.5.7, IBM Process Portal and stock controls are fully responsive. This capability enables knowledge workers and field staff to participate in business processes by allowing them to perform their work outside the office by using their mobile devices. IBM Process Portal is discussed in Chapter 5, “IBM Process Portal” on page 261.

The following sections describe how to make coaches responsive.

2.9.1 Multiple form factors

IBM BPM accommodates the responsive form factors listed in Table 2-1.

Table 2-1 Responsive form factors supported by IBM BPM

Form factor	Example device	Button Icon	Screen size (width)
Small	Phone (vertical orientation)		640 pixels or less
Medium	Tablet Phone (horizontal orientation)		641-1024 pixels
Large	Desktop computer		More than 1024 pixels

The button icons shown in Table 2-1 are visible in both the coach editor and coach view editor in Process Designer, as shown in Figure 2-53. You use these button icons to switch between the different form factors of a coach or a coach view.

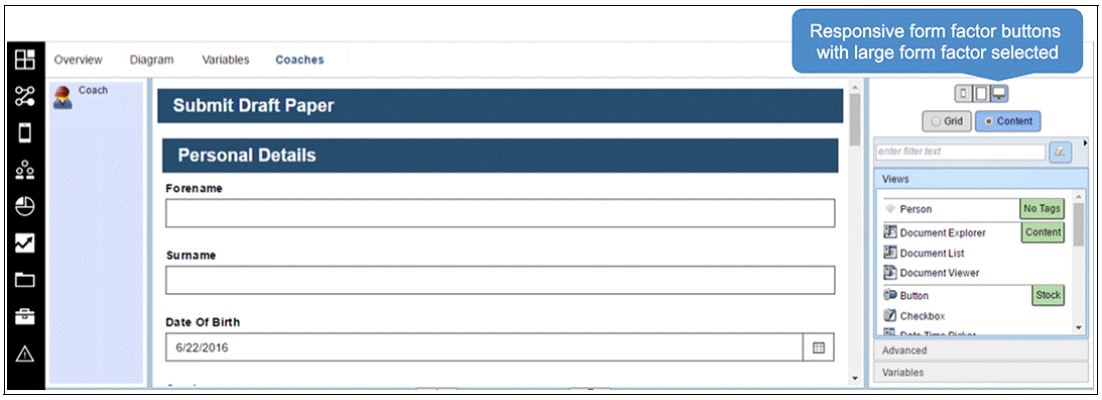


Figure 2-53 Responsive form factor buttons on the coach editor in IBM Process Designer

A coach can be designed for the desktop using the large form factor and then adjusted for medium and small form factors accordingly, but it's entirely up to the coach author to decide which form factor to start with. Thus far, the sample you've been working with uses the large form factor. Within this section, you continue with the sample to make it responsive for medium and small form factors.

Switching between the form factors automatically adjusts the available width for the canvas within the coach editor. Using the current coach design for the large form factor within the sample, you notice that the fields are adjusted automatically to fit the available screen size without needing to make any changes, as shown in Figure 2-54.

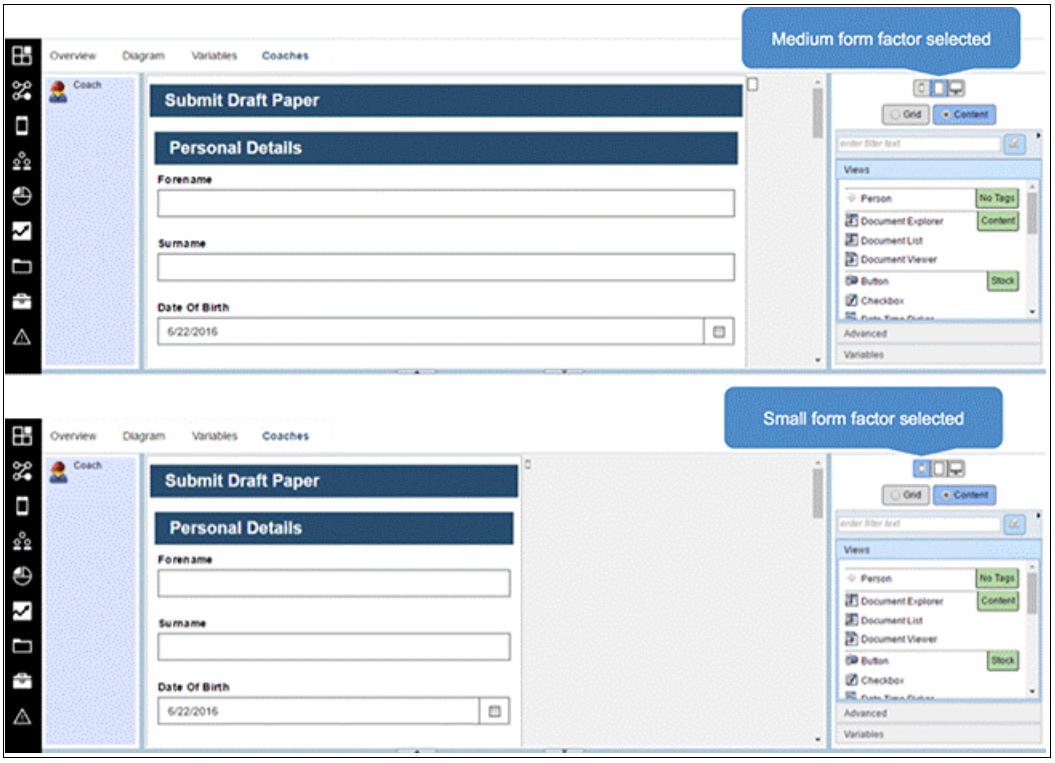


Figure 2-54 Medium and Small Form Factors selected in the coach editor

2.9.2 Controls' responsive properties

Within this sample, the design of the coach for small and medium form factors is adequate, but you can update the design for the desktop to take advantage of the available space.

Within the Person composite coach view, layout the fields with Personal Details and Address side-by-side, with Costs below the personal details. Use nested horizontal and vertical sections to achieve this layout, as illustrated in Figure 2-55. Also adjust both the Personal Details and Address sections to each occupy 50% of the available space using the control positioning properties.

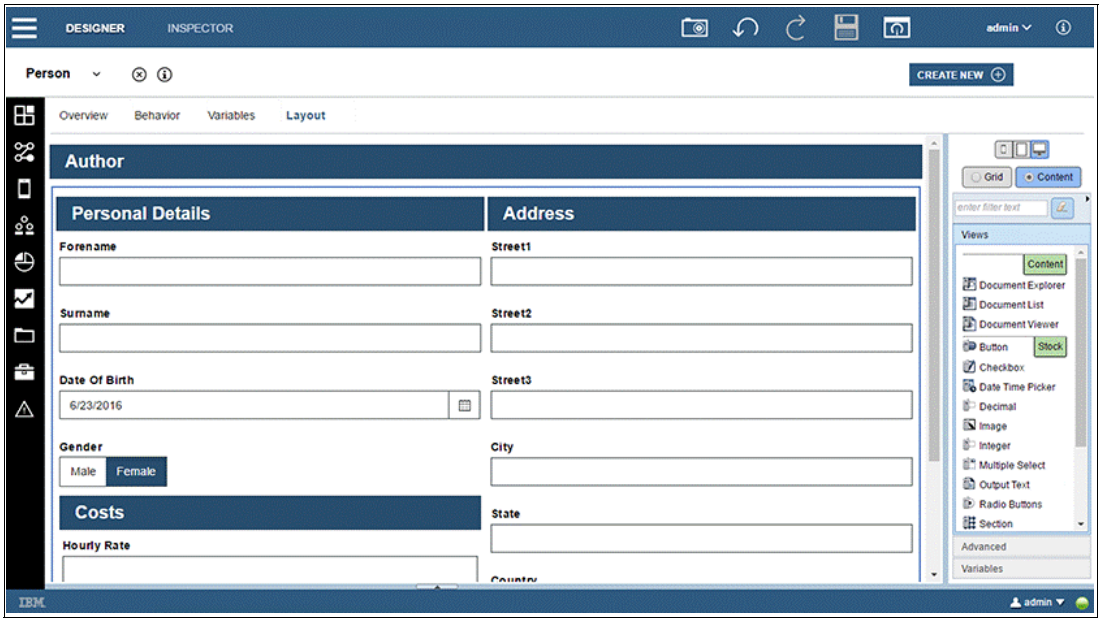


Figure 2-55 Responsive desktop design using sections to layout a coach (or coach view)

All the changes for the desktop are also applied to the small and medium form factors. The layout is the same for all form factors, but the position, configuration, and visibility properties can be configured independently. By default, settings that are defined for the large form factor are inherited automatically by the medium and small form factors; similarly, settings for the medium form factor are inherited by the small form factor.

The current design is now good for large and medium form factors, but it's a bit of a squeeze on smaller devices. Therefore, you need to configure the sections on a small form factor to display the Address vertically in between the Personal Details and Costs as you had it previously.

Switching to the small form factor and selecting the horizontal section enables you to change the orientation only for the small form factor, as illustrated in Figure 2-56 on page 49.

Notice the icon next to each configuration option. The icon shows where the value comes from. It indicates whether the setting applies to all form factors (only specified within the large form factor), inherits the setting from a larger form factor (specified in either the large or medium form factors), or applies to the current form factor (specified explicitly for the current form factor). In this case, the *small form factor* icon beside the Layout option indicates that this setting is applicable only to the small form factor, where the purple highlighting indicates it is set explicitly here.

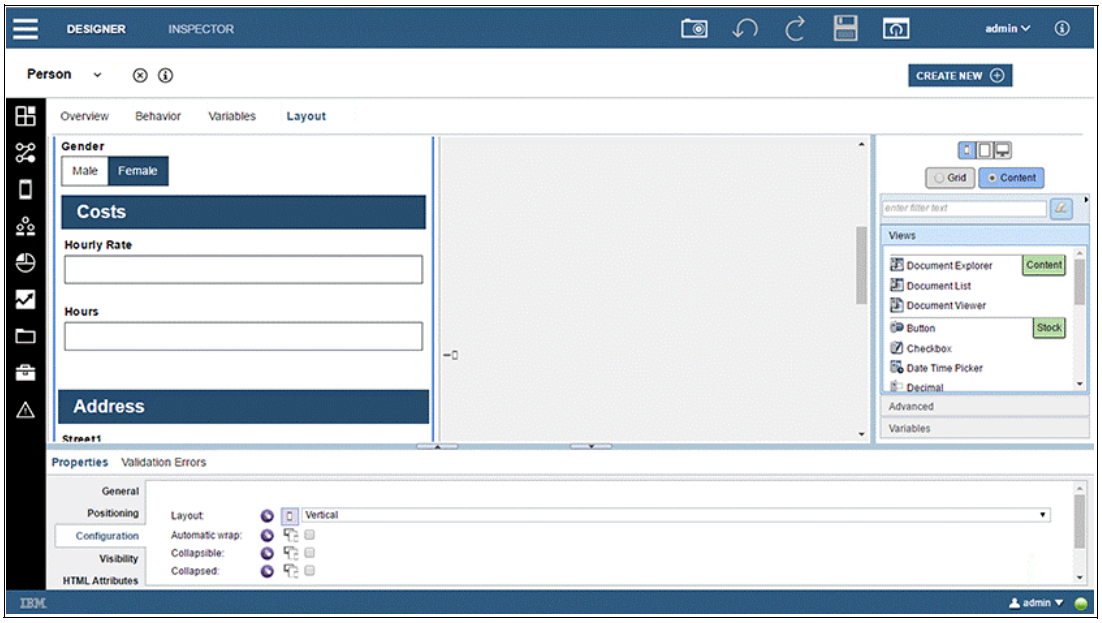


Figure 2-56 Responsive phone design using sections to layout the coach (or coach view)

You can also remove the positioning that you previously applied for the desktop for both the Personal Details and Address sections, so that they occupy 100% of the available space but only for the small form factor.

Because we applied these changes only for the small form factor, switching back to the large form factor still shows the Personal Details and Address sections positioned side-by-side, as illustrated in Figure 2-55 on page 48.

The layout now looks pretty good, but the changes to the small form factor did not quite give the desired look. Notice the Address details appear under the Cost details, and the alignment of the section headings and controls is a little untidy. To get these details to appear in the correct position for each form factor, using sections can be a little challenging. Instead, you can use the *grid layout* to make this easier, which is detailed in the next section.

2.9.3 Testing a responsive UI

You can test a responsive UI by rerunning the client-side human service on the desktop, which allows you to see how the UI adapts by changing the size of the browser window. Of course, this test does not show the native look and feel of a specific device, but you can use emulators to achieve that test without needing to test on a physical device. However, it is recommended to conduct tests for production solutions using the target devices with appropriate virtual private network (VPN) connectivity.

2.9.4 Responsive design considerations

A coach that is designed for the desktop might include too much information about a single coach for smaller form factors. When this situation occurs, you can modify the design in a few ways:

- Hide details about smaller form factors.
- Split large coaches into a number of smaller coaches.
- Present information differently for each form factor.

Details displayed on the desktop can be hidden on smaller devices to provide a better looking design, which is easier to use. However, this approach results in different information being available depending on the user's chosen device. Business users might perform tasks differently, depending on the available information. Thus, following a preferred practice, it should always be possible to access the same information irrespective of the chosen device.

Instead, you can split large coaches into a number of smaller coaches, providing navigation between them within a client-side human service. Although the user experience might be improved on smaller devices, this approach can actually make the user experience worse on the desktop.

Best results are often achieved by combining these two approaches. Where a single coach is the preferred option for the desktop, the same coach can provide the starting point for smaller devices. Specific information can be hidden on the initial screen when used on tablets and smartphones, but the information can still be accessible by navigating to other coaches. The navigation buttons on smaller devices is hidden on the desktop.

For example, within the sample process you've used thus far, you can display summary details for the author on the initial coach when accessed on a phone and then provide access to full details about a separate coach.

The IBM BPM 8.5.7 stock controls are designed to behave differently depending on the form factor and adapt to the device look and feel as much as possible to give a better user experience. For example, native date picker and list selection behavior is used for Apple iOS and Android devices, whereas these controls on the desktop use the additional available space to provide more convenient capabilities when using a mouse.

When designing coaches (or your own custom controls, as detailed in Chapter 3, "Building controls using coach views" on page 73), you can apply the same principle. Ideally, individual responsive controls will solve this problem without having to impact the coach design. However, you can choose to display different controls or composite coach views, depending on the form factor, by defining the appropriate visibility settings.

2.10 Designing a coach using the grid layout

In recent years the art of designing engaging and professional looking websites has given rise to a plethora of modern web technologies and approaches. A prominent technique used to layout modern looking web pages is to use a *grid system*. IBM BPM 8.5.7 adopted this approach to create compelling UIs for business processes.

2.10.1 The Grid and Content views

Grid systems are implemented using a number of different approaches. IBM BPM 8.5.7 uses a popular 12-column grid layout, as illustrated in Figure 2-57 on page 51. The coach editor includes two buttons, positioned just above the palette, that you can use to switch between the Grid and Content views. Thus far, for the examples in this book, you have used the Content view that exhibits WYSIWYG behavior. The examples in this section take a closer look at the *Grid* view and how the grid layout impacts the Content view.

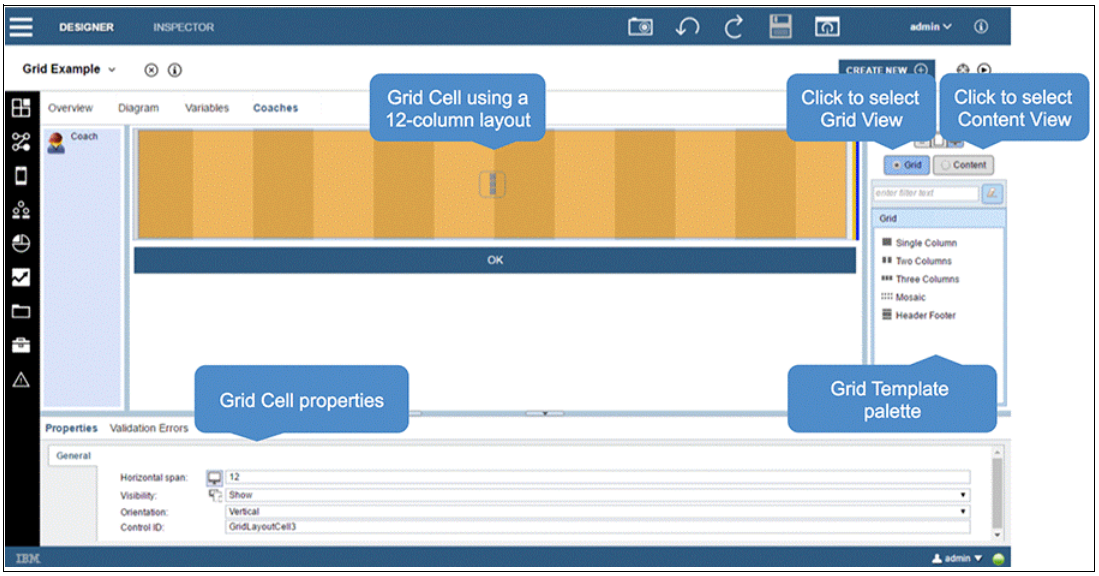


Figure 2-57 The coach editor showing the 12-column grid layout

On switching to the Grid view the coach content on the canvas is replaced with grid cells (if a grid exists). In addition, the palette displays a number of grid templates that you can use to drag onto the canvas, and the properties pane is used to display the properties of the currently selected cell. Figure 2-57 shows a *single grid cell* that illustrates the 12-column layout.

2.10.2 Grid templates

When creating a new coach, the coach author can choose to use a predefined grid template, as illustrated in Figure 2-58. These same grid templates are available on the palette. Even if the coach author starts without a grid, switching to the Grid view and adding grid cells from the palette as required is possible. The grid templates are simply a starting point. The coach author can modify the layout by adding, removing, resizing, and configuring cells as needed.

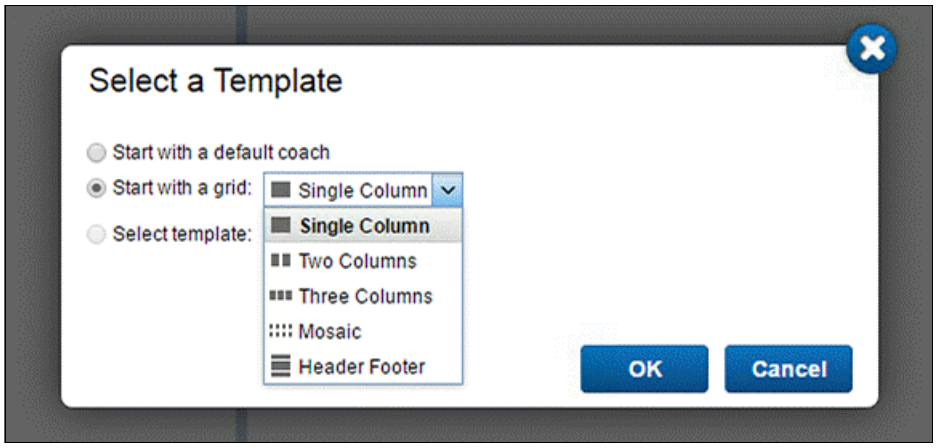


Figure 2-58 Create a new coach using a predefined grid template

Let's now turn to the sample process to demonstrate how to use the grid to define the layout of a coach. Opening the coach in the Submit Paper Draft client-side human service and switching to the Grid view, you notice that the original content is still visible.

Dragging the Header Footer grid template from the palette to the top of the coach moves the existing content down below the three new grid cells that were added, as illustrated in Figure 2-59. Within the sample process, you use the top cell to display the title of the coach, the middle cell to display the author details, and the bottom cell to display the documents and comments.

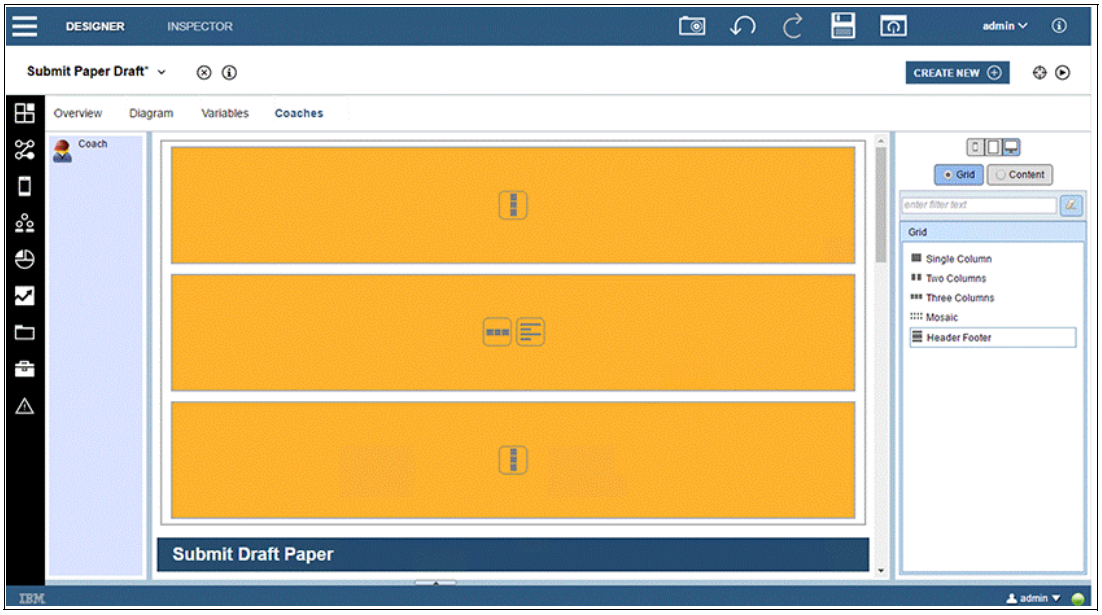


Figure 2-59 The Header Footer grid template shows in the Grid view

Terminology note: The grid root is actually a *container*. Containers can hold cells and other containers. *Cells* can only hold content. Cells are the orange boxes; containers are the other outlined boxes.

2.10.3 Grid cell properties

New grid cells are configured with vertical orientation by default, which is signified by the icon displayed in the center of the top cell in Figure 2-59. The icons in the middle cell indicate that it has horizontal orientation and left alignment. Cell orientation and alignment can be changed either by clicking the icons or setting the cell's properties, as shown in Figure 2-60 on page 53.

The width of a cell, defined by the Horizontal Span property, is specified in terms of the number of columns that it occupies. Each cell from the Header Footer grid template occupies the entire width of the coach. So the Horizontal Span is set to 12 columns, as shown in Figure 2-60 on page 53. Alternatively, a cell occupying half of the coach width has the Horizontal Span set to six columns, and a cell occupying a third of the coach width has a Horizontal Span set to four columns.

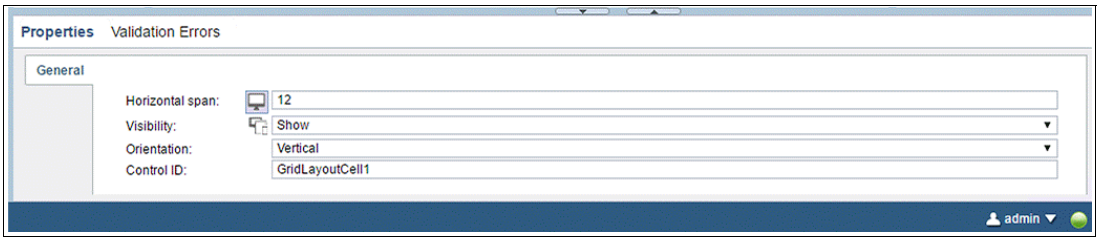


Figure 2-60 Grid cell properties

2.10.4 Adding content to the grid

Switching back to the Content view allows you to move the content into the grid cells. Cells without content are shown with a gray background, as illustrated in Figure 2-61. Click the plus (+) icon in the center of the cell to allow the user to select either a control or a composite coach view to add to the cell. Alternatively, you can add content by dragging elements from the palette or moving content from elsewhere on the canvas, as you did previously.

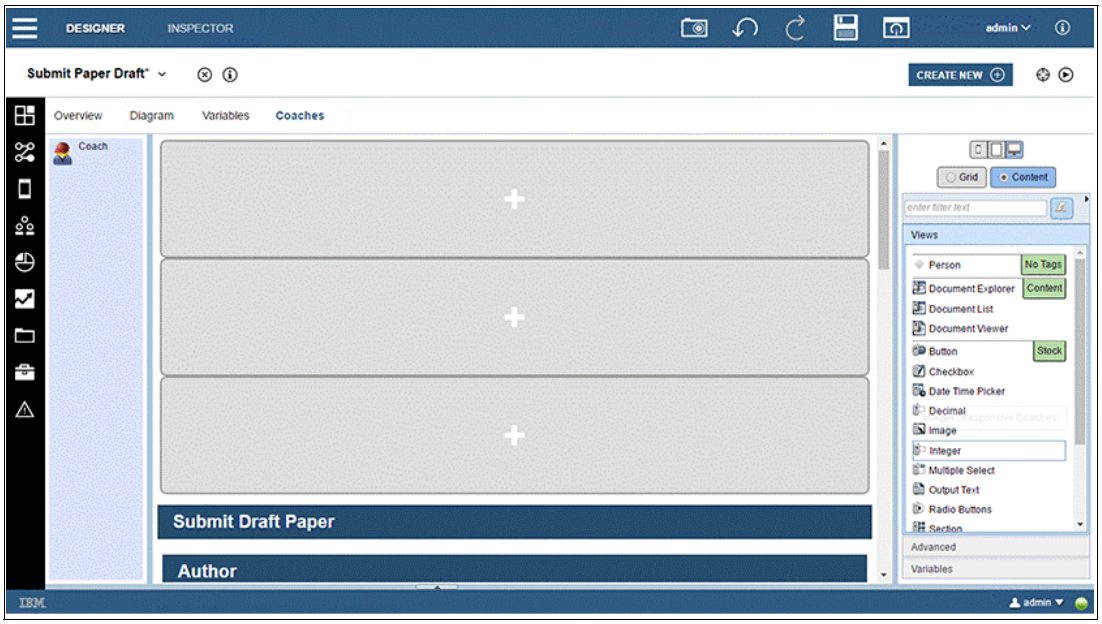


Figure 2-61 Header Footer grid template show in the Content view

In the sample process, you need to separate the author details from the surrounding section that defines the coach title. So move the Author composite coach view into the middle cell, as shown in Figure 2-62.

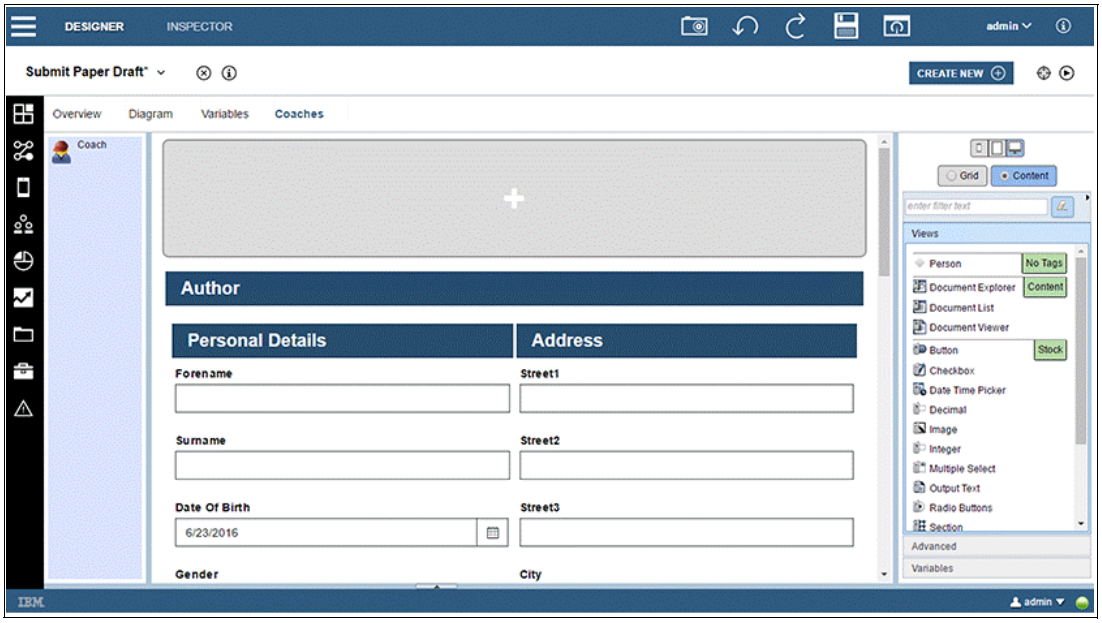


Figure 2-62 A cell with content show in the Content view

Switching back to the Grid view shows any populated cells with disabled content, as illustrated in Figure 2-63, so that the coach author can take the content into account when making changes to the grid layout.

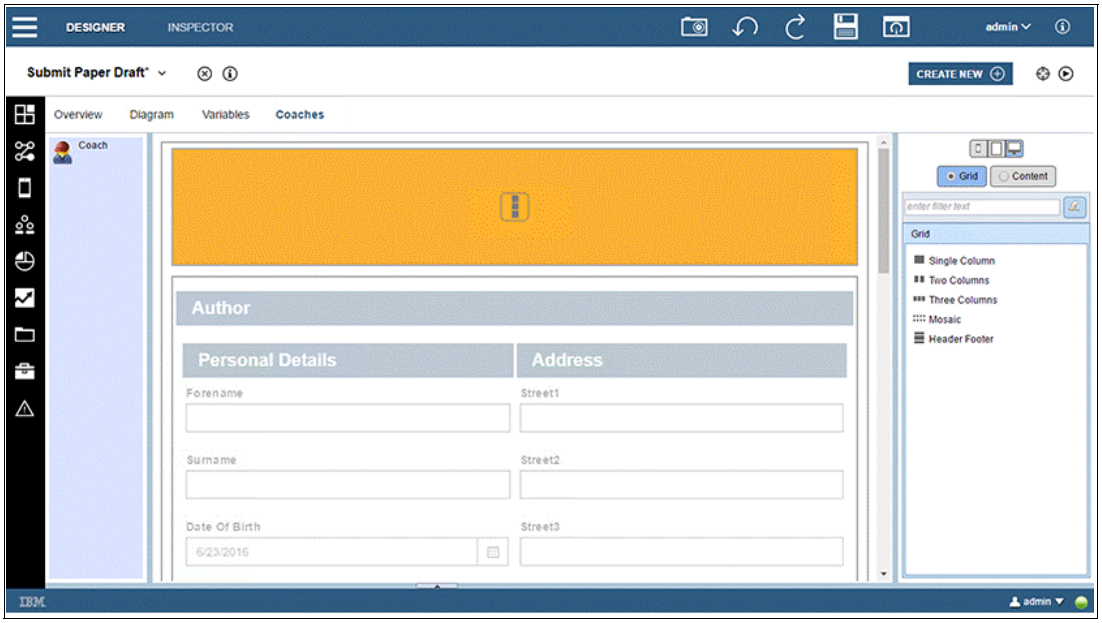


Figure 2-63 A cell with content show in the Grid view

Within the sample process, you now move the remaining content into the appropriate cells. As you drag content within the Content view, the grid cells appear so that you can easily drop the content into the required cells.

2.10.5 Using sections with a grid layout

One of major advantages of using the grid layout instead of using sections is improved alignment of content. Grid cells do not occupy any space, unlike sections. So cell content can use all horizontal space with the edges of controls that are aligned perfectly to the grid. Conversely, each time a section is added, the available horizontal space is reduced. Where sections are used to define the layout of a coach, deeply nested sections can result in poor alignment of controls and section headings.

Using the grid also results in a much leaner HTML payload that is delivered to the browser and has a much simpler implementation compared with using sections for layout purposes. This approach can improve the performance and memory footprint of your UI.

Sections can still be used within the grid to provide headings, but any content should be moved below the section if it needs to align with the grid.

2.10.6 Using a grid layout within a coach view

A grid layout can also be used for coach views, so that the coach view content can be aligned perfectly with the other content on the coach. Thus far, you have used sections to define the layout of the Person composite coach view within the sample process. Figure 2-64 shows a new design using a grid layout instead.

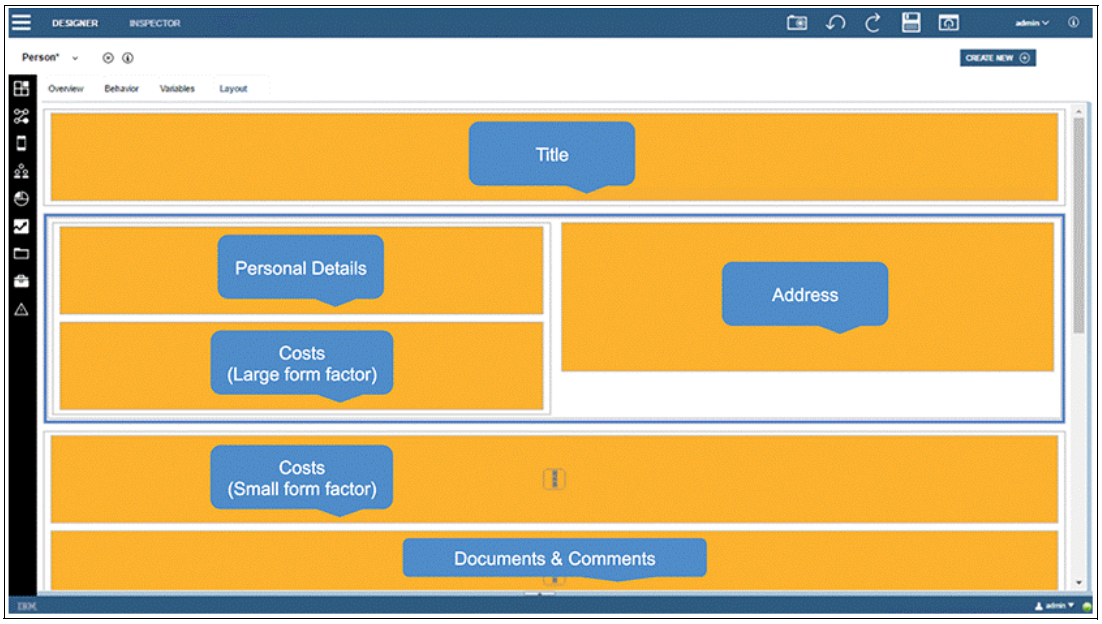


Figure 2-64 Grid Layout for Person composite coach view

2.10.7 Grid view visual tools

Now let's look at some of the visual tools provided by the Grid view that you can use to construct the layout.

Additional cells can be added quickly above and below and to the left and right of an existing cell (or container) by hovering over the it and clicking the green plus (+) icons that appear on the edges of the cell, as shown in Figure 2-65 on page 56. When adding cells to the left or right, the width of the existing cell is adjusted automatically to make space for the new cell, which effectively splits the original cell into two. If there is not enough room to fit the cell, the

cells flow to the next line. This method can be much faster than using the palette and resizing cells individually.

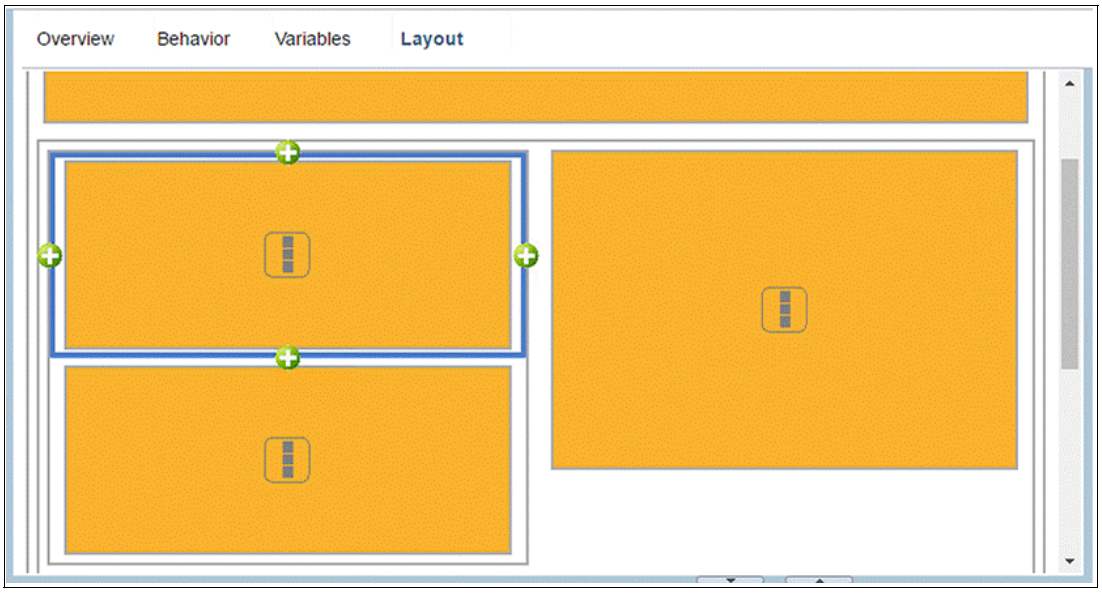


Figure 2-65 Adding grid cells using plus icons on the edge of an existing cell

You can drag and snap the gutter in-between horizontal adjacent cells to a different column boundary, as shown in Figure 2-66. This method adjusts the widths of the surrounding cells accordingly. This process can accelerate UI development, because it prevents having to update the widths of each cell individually in the Properties pane.

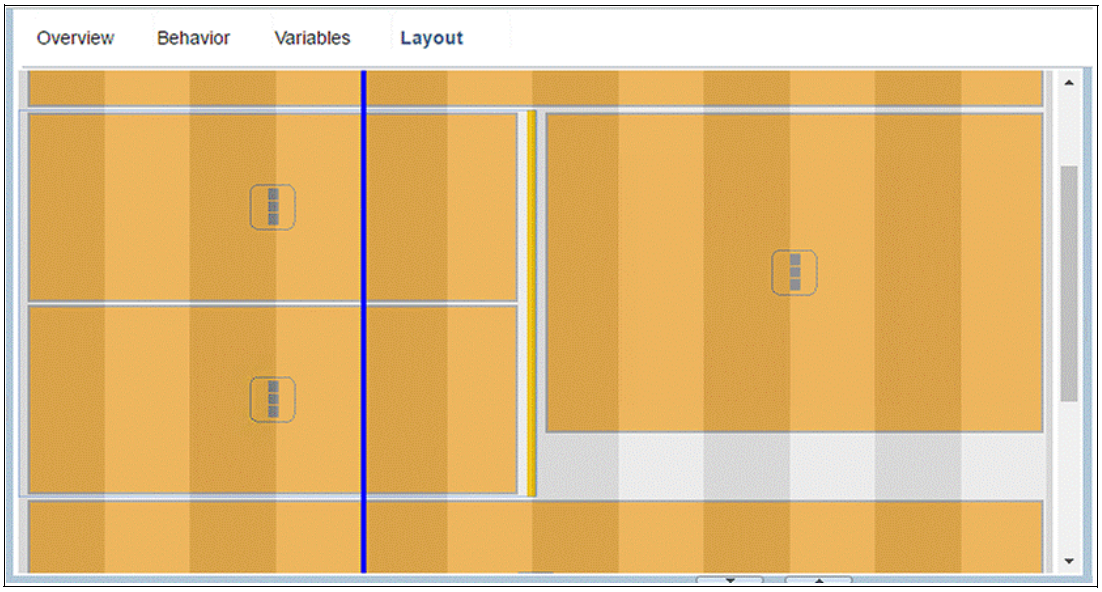


Figure 2-66 Resize grid cells by dragging the edge of a cell

2.10.8 Hiding grid cells

Now let's go back to the sample. When used on the desktop, the cell for Costs (Small form factor) is hidden, so that the Costs appear directly under the Personal Details. All cells on the

large form factor should occupy the entire width, except for Personal Details, Costs (Large form factor), and Address, which have Horizontal Span set to 6 columns.

When used on a smartphone, the cell for Costs (Large form factor) is hidden, so that the Address moves underneath the Personal Details and the Costs appear directly under the Address. All cells on the small form factor occupy the entire width. So Horizontal Span is set to 12 columns for both Personal Details and Address.

Figure 2-67 shows the result of the final grid layout design on a desktop and smartphone.

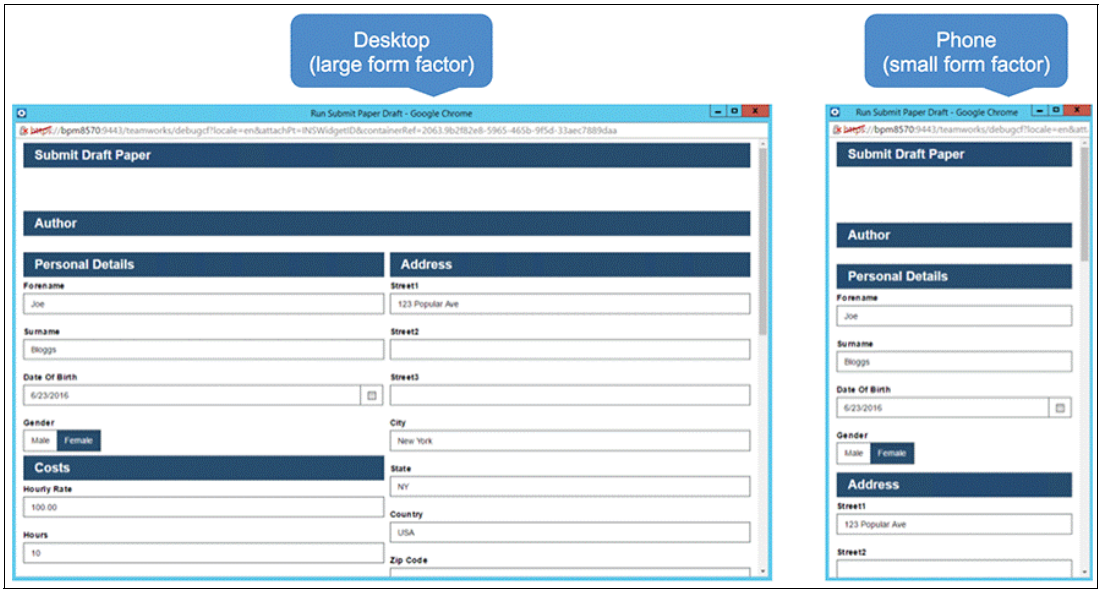


Figure 2-67 Testing the grid layout for desktop and smartphone

2.10.9 Reusing a grid layout

You can design a grid layout to reuse with one of the following methods:

- Copy a coach (with a grid layout)
- Create a coach view template (with a grid layout)

With the first option, you create a coach that defines the grid layout with no content. Keep this “empty” coach somewhere safe. Whenever you want to reuse the grid layout on another coach, you can create a copy of the entire coach within the client-side human service and add your new content, as required. This option allows you to modify the grid layout in each copy without impacting other coaches.

Alternatively, you can create a coach view that defines the grid layout, as shown in Figure 2-68 on page 58. This time each grid cell needs to include a Content Box control, so that additional content can be added when using the coach view on a coach. The Content Box control is selected from the Advanced tab in the palette. The advantage of this approach is that the grid layout is defined only in one place. So any changes to the grid layout is reflected on all coaches that use it.

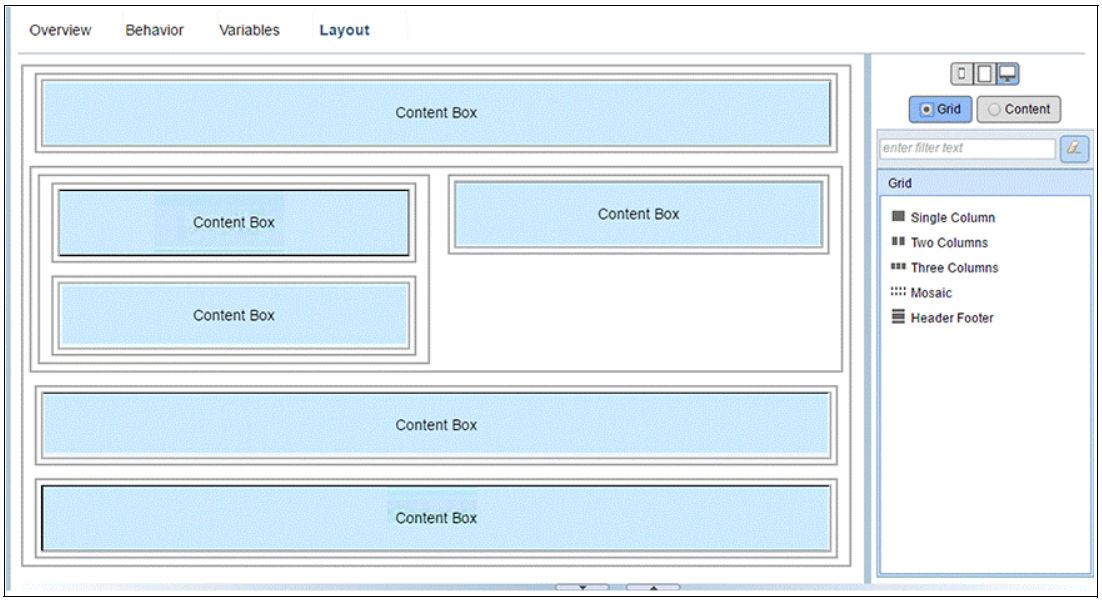


Figure 2-68 Reusable grid layout defined as a coach view (Grid view)

In the Overview section of the coach view, as shown in Figure 2-69, select the “Use as Template” option to make this layout available when creating a new coach, and select the “Intended for use on multiple devices” option, so that the coach view displays in the palette when creating responsive user interfaces.

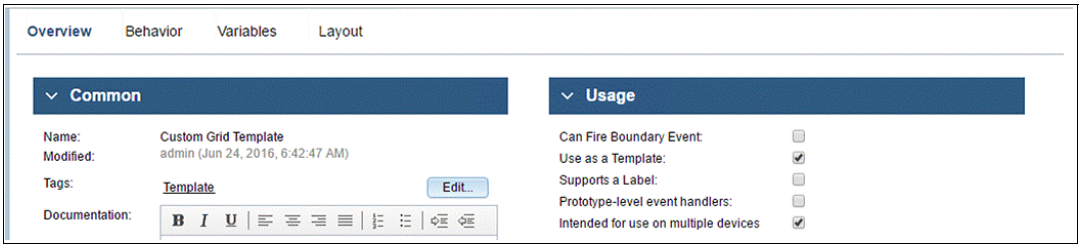


Figure 2-69 Reusable grid layout defined as a coach view (Grid view)

The next time you create a new coach, you can select the coach view template, as shown in Figure 2-70.

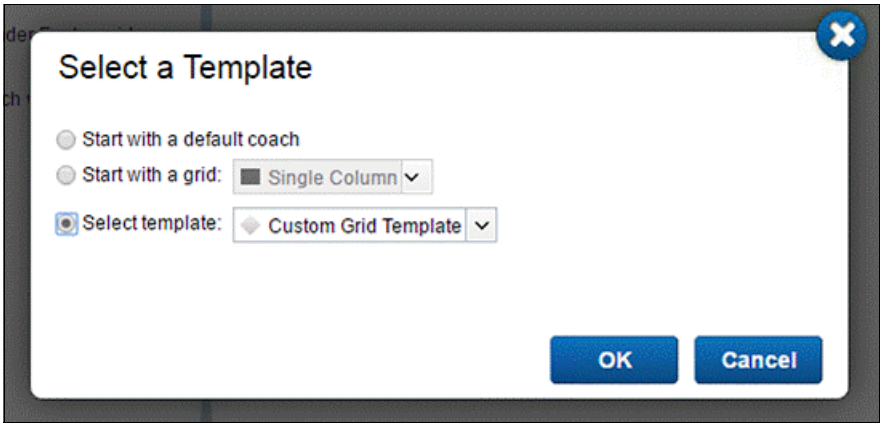


Figure 2-70 Select a coach view template (with a grid layout) for a new coach

2.11 Using nested client-side human services

To reuse an entire coach, you can clone a coach using copy and paste. Each copy is maintained separately, which allows you to modify the new coach without impacting the original coach definition. However, if you want to maintain only one definition of the coach that is used in multiple places throughout your process (or processes), you can use a *nested* client-side human service.

Nested client-side human services, introduced in IBM BPM 8.5.7, allow you to reuse a coach or sequence of coaches within one or more other client-side human services. When a top-level client-side human service is delivered to the browser, all nested client-side human services are included to avoid additional round trips back to the server. Apart from being client-based, nested client-side human services work in the same way as nested heritage human services, where data passed between parent and child client-side human service require a data mapping, as illustrated Figure 2-71.

Using business objects in nested client-side human services: Business objects are passed by reference. So there is no need to specify them as output parameters of the nested client-side human service. Therefore, when business objects are updated in the nested client-side human service, these changes are reflected in the parent client-side human service. Whereas basic types, such as String, Boolean, and so on, are passed by value, these need to be explicitly mapped as output variables if you want to update the equivalent variables in the parent client-side human service.

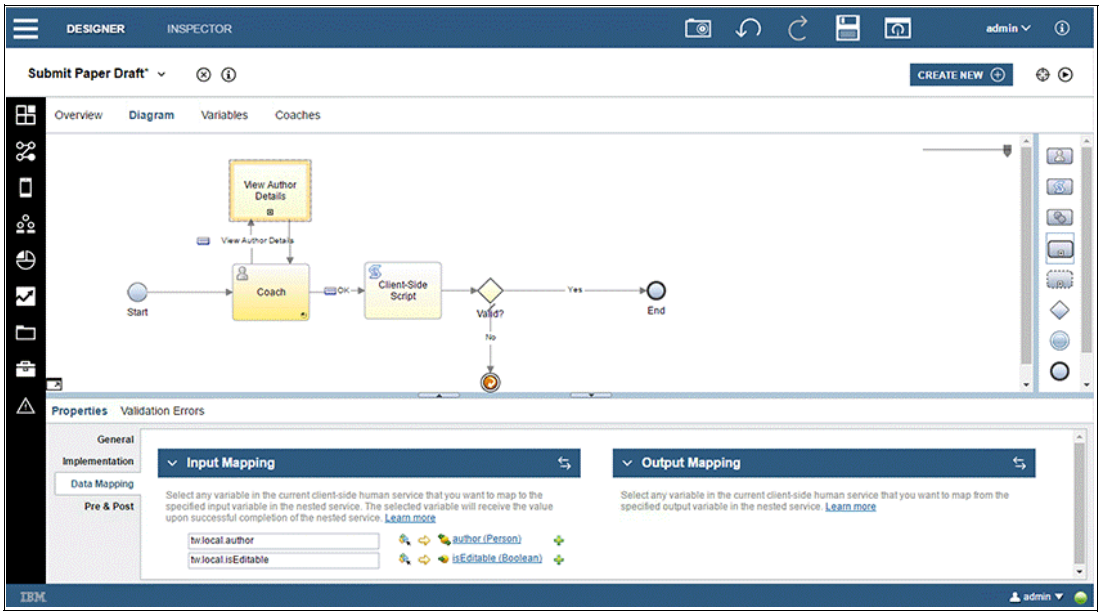


Figure 2-71 Nested client-side human service

Creating a new client-side human service: You need to select the “Use as a nested service” option, which is different from heritage human services. You cannot use a top-level client-side human service as a nested client-side human service. This limitation poses an interesting dilemma when thinking about creating unit test services.

Creating unit tests for a top level client-side human service requires the unit test to be created from a process flow, because it is not possible to nest a top-level client-side human service within a unit test service.

Alternatively, if you can ensure that the logic for a client-side human service is always contained in nested a client-side human service, the top-level client-side human service will be used only to delegate to a nested client-side human service. This way, you can use another top-level client-side human service to create a unit test service for a nested client-side human service. However, this process adds a level of complexity to your BPM solution.

The sample process includes a nested client-side human service, so that the author details can be displayed in a separate coach when using a small form factor. The author details are still included within the main coach for the Submit Paper Draft process when used with larger form factors. Visibility rules are defined accordingly to replace the author details with a button to View Author Details on the small form factor.

2.12 Styling coaches using themes

For many years Cascading Style Sheets (CSS) were the primary choice for styling web-based UIs. CSS requires a technical awareness of the underlying HTML that can make it challenging for non-technical process authors to style BPM UIs. In recent years new technologies and approaches have become available. Less[1] is an open source CSS pre-processor that provides a layer of abstraction with a rich set of features that is more flexible, easier to use, enables theming, and is extensible. IBM BPM 8.5.7 adopted Less to provide a new theming capability for both responsive coaches and the new responsive Process Portal.

2.12.1 Defining a custom theme

IBM BPM 8.5.7 introduced a model artifact called *Theme* that you can use to define styles, including colors, fonts, weights, and padding by using Less variables. A default theme, called *BPM Theme*, is provided in the System Data toolkit. The BPM Theme is included in process apps by default, but process authors can switch the BPM Theme with a custom theme within the Process App Settings, as illustrated in Figure 2-72.

When creating a new theme, you can either use an existing theme as a baseline or import a theme definition from your local file system. The sample process app used the BPM Theme as a baseline to create a custom theme called *Sample Custom Theme*.

[1] For more information about Less refer to: http://lesscss.org/

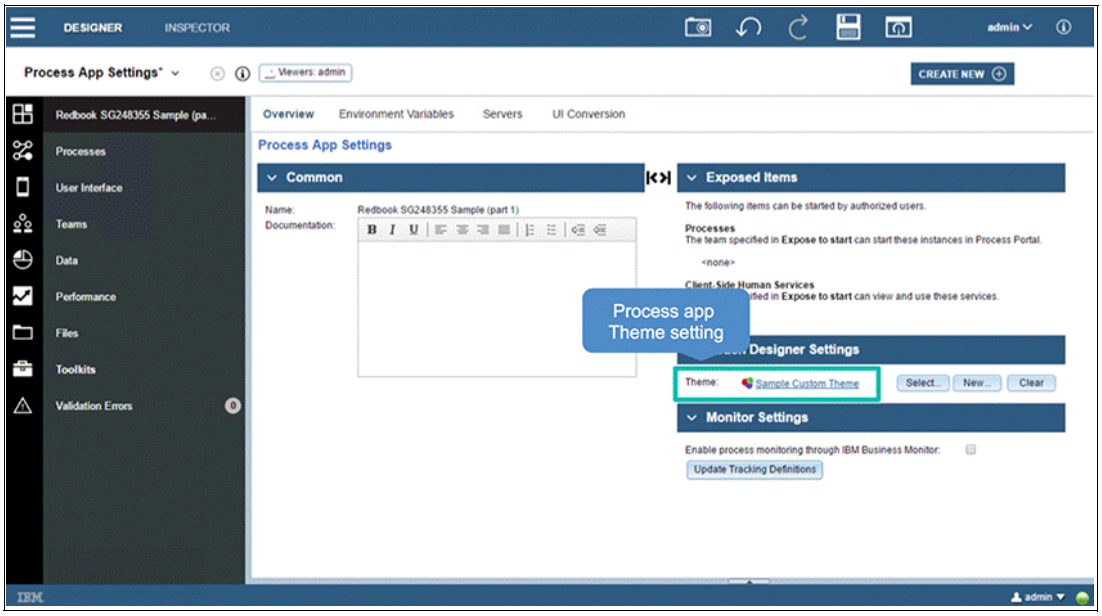


Figure 2-72 The Process App Theme setting

2.12.2 The Graphical Theme Editor

IBM BPM 8.5.7 simplifies styling further by providing a Graphical Theme Editor, as illustrated in Figure 2-73.

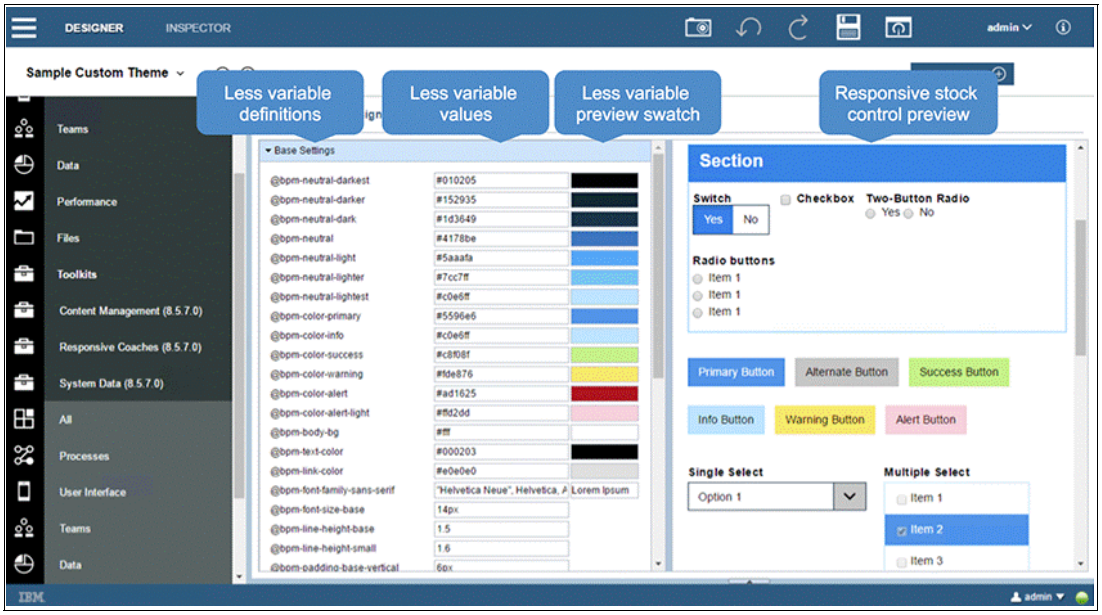


Figure 2-73 The Graphical Theme Editor design tab

The Graphical Theme Editor includes two sections. The left panel shows the Less variable definitions, their values, and a preview swatch. The Less variables are grouped into categories for convenience, as illustrated in Figure 2-74 on page 62. The right panel shows a preview of how the responsive stock controls appear when the theme is applied.

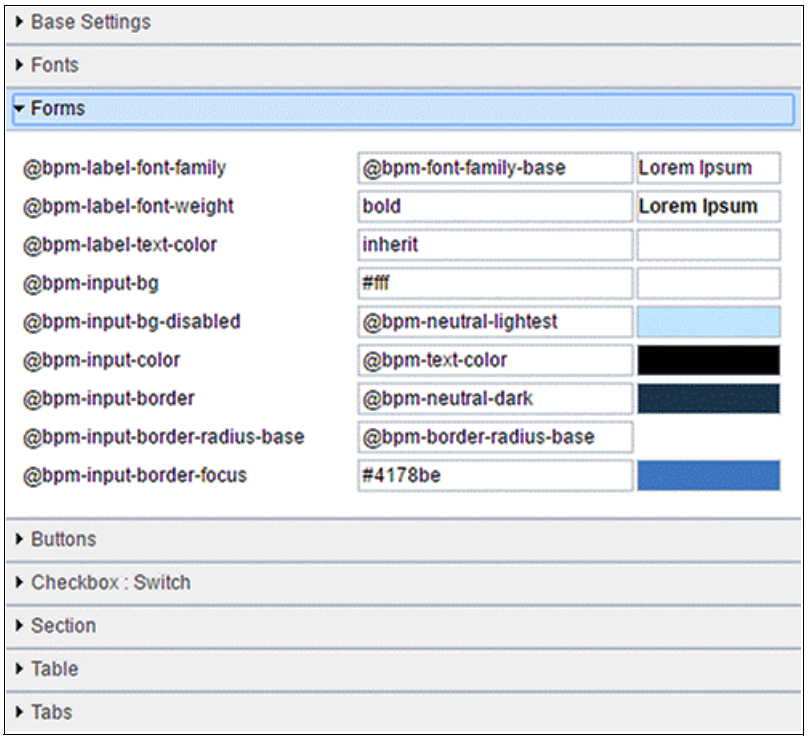


Figure 2-74 Less variable categories in the Graphical Theme Editor

You can either edit the Less variable values directly or click the corresponding preview swatch to open a context sensitive utility, such as a color or font picker, as shown in Figure 2-75 and Figure 2-76 on page 63, respectively.

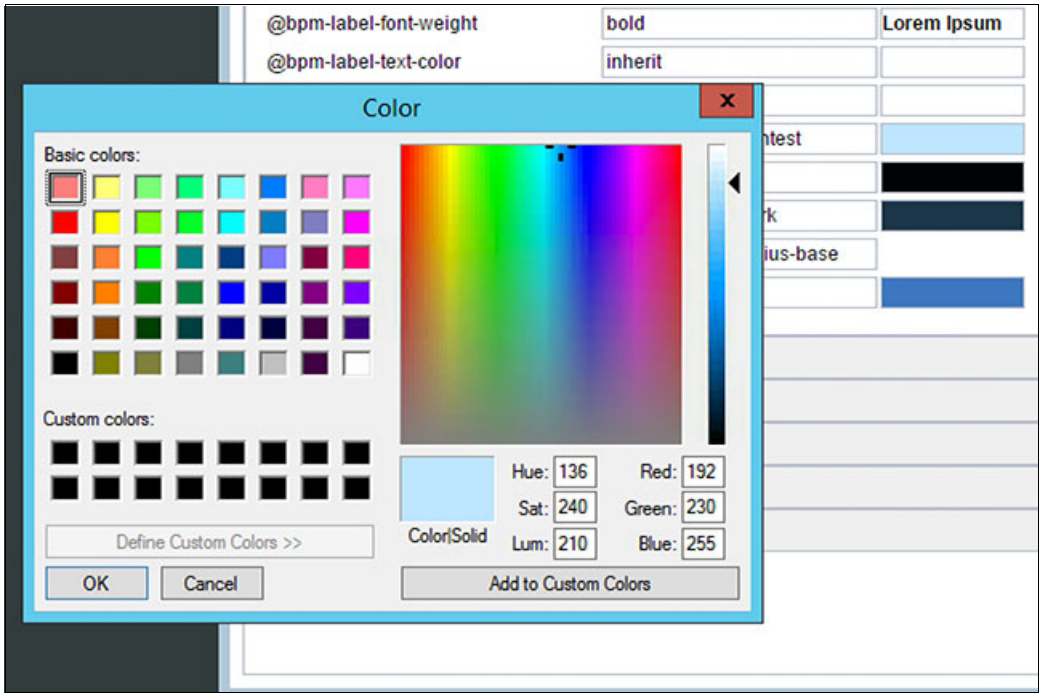


Figure 2-75 Color picker for Less variables

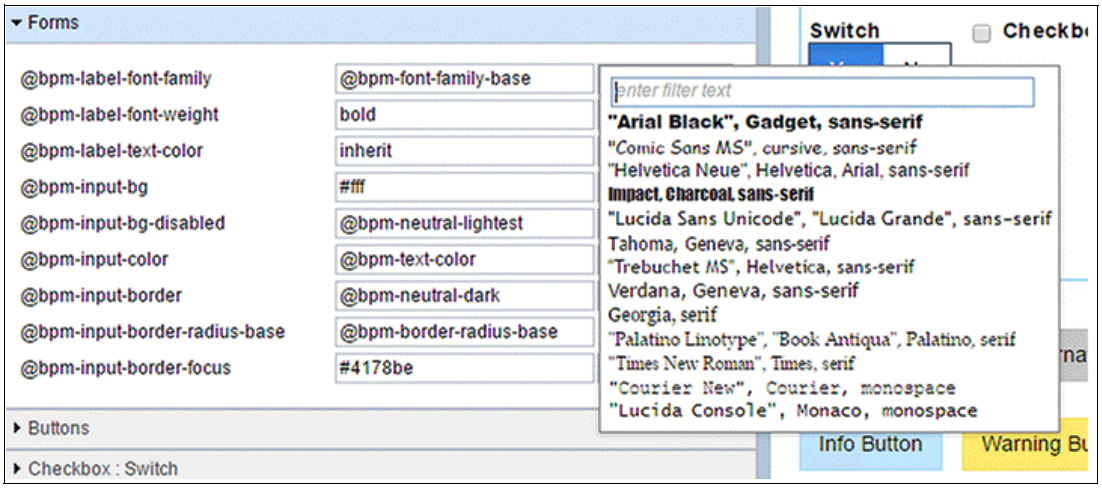


Figure 2-76 Font picker for Less variables

After you change the Less variable value, the theme is updated immediately, and the affected controls on the right side are updated to show you the result.

Moving the mouse pointer over a control within the preview pane, displays a tooltip that indicates which Less variables are used to style that control, as illustrated in Figure 2-77. Having this information available makes it easier than having to reverse engineer the coach HTML to determine which CSS attributes are used to style the control, as required in earlier versions of IBM BPM.

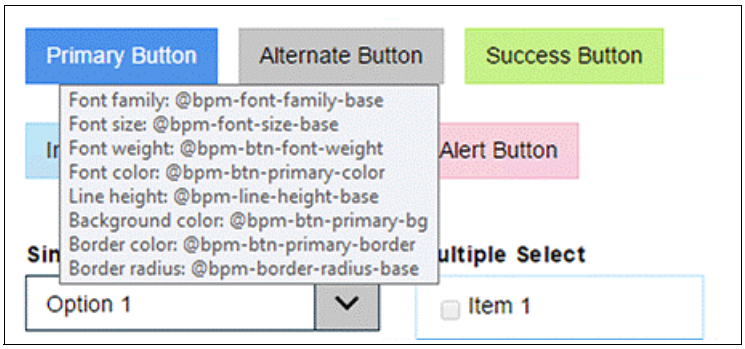


Figure 2-77 Less variables used to style a specific control (Primary Button)

Within the sample process, the Sample Custom Theme is updated with the following less variable values:

```
@bpm-color-primary = #ce61da
@bpm-body-bg = #f2d8f5
@bpm-table-bg = #f9f588
```

The theme is automatically applied. Although this might not be to everyone's taste, re-running the Submit Paper Draft client-side human service demonstrates the result, as illustrated in Figure 2-78.

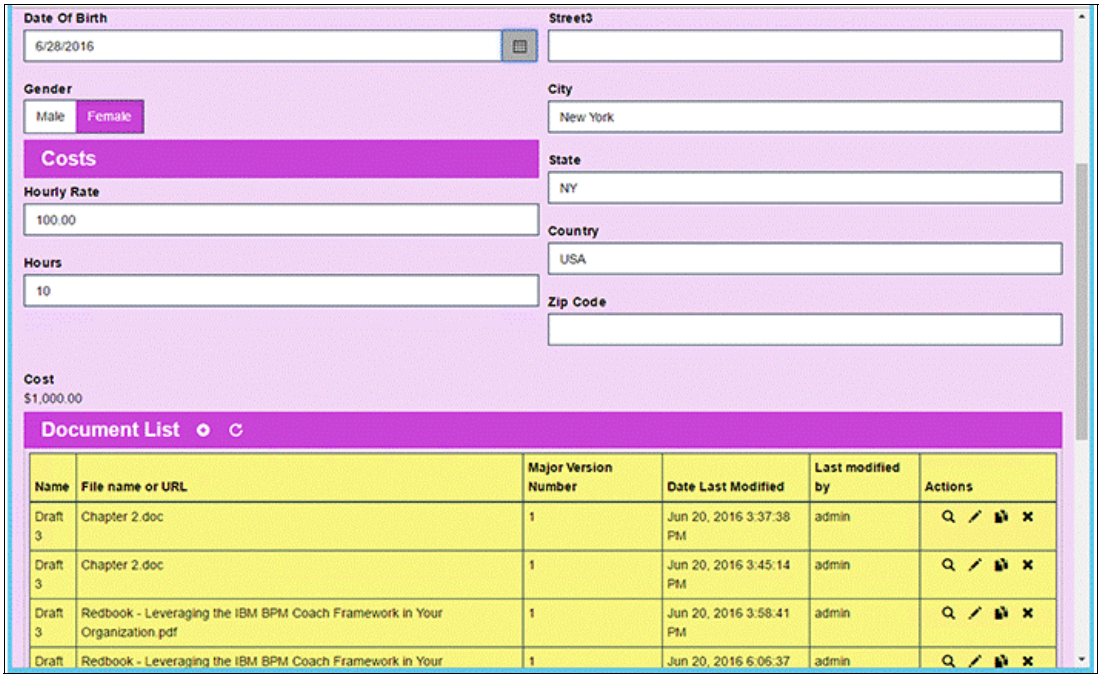


Figure 2-78 Submit Paper Draft coach using Sample Custom Theme

2.12.3 The Source Editor

Selecting the Source tab for a theme presents a text-based definition for the entire theme in the Source Editor, as shown in Figure 2-79.

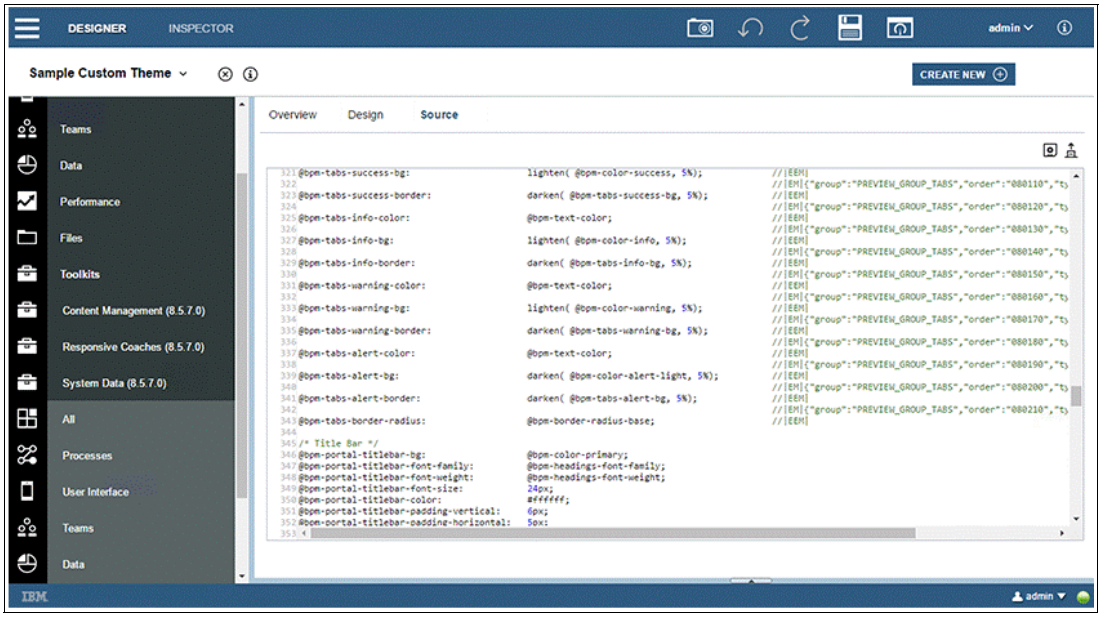


Figure 2-79 Text-based theme definition

You can update Less variables directly in the source editor. This process is useful for importing a theme definition that was created outside of IBM BPM by pasting it into the source editor. Similarly, you can export theme definitions by copying them from the source editor into an external tool. You can also extend the theme by adding your own Less variables, which can be used with your own custom controls (as detained in Chapter 3, “Building controls using coach views” on page 73).

Metadata in the comments is used to organize the variable definition within the graphical theme editor Design tab. You can even add new Less variable categories as shown in Example 2-4.

Example 2-4 Adding a Less variable category for Portal

```
//|EM|{"group":"Portal","order":"090010","type":"color"}|DE|
@bpm-portal-titlebar-bg: @bpm-color-primary; //|EEM|
//|EM|{"group":"Portal","order":"090020","type":"font-family"}|DE|
@bpm-portal-titlebar-font-family:          @bpm-headings-font-family; //|EEM|
//|EM|{"group":"Portal","order":"090030","type":"font-weight"}|DE|
@bpm-portal-titlebar-font-weight:          normal; //|EEM|
```

2.12.4 Fine-grained and broad-brush styling

A BPM theme includes about 120 Less variables, which enables precise styling of BPM user interfaces. However, many of these variables values reference other Less variables.

For example, in the default BPM Theme, both the `@bpm-btn-primary-bg` button background color variable and the `@bpm-section-header-primary-bg` section header variable reference the `@bpm-color-primary` primary color variable.

Therefore, when changing the primary color, both the background color for the button and section header will change accordingly.

The `@bpm-section-header-primary-bg` section header variable actually uses the following Less function:

```
darken( @bpm-color-primary, 5%);
```

This function results in a different color value, which is relative to the referenced color value.

Using references extensively throughout the theme, you can define a custom theme whereby you need to update only a few values to control the look and feel of all of your coach-based UIs. Many of these common variables are grouped into the Base Settings category in the Graphical Theme Editor tab for the default BPM theme.

2.12.5 Style configuration options

The responsive stock controls are implemented to respond to theme-level styling, which improves the consistency of the look and feel of BPM UIs and can significantly speed up development.

However, some controls provide configuration options to enable the coach author to select the style of specific instances of a control based on the context in which it is used. For example, the responsive Button control has the following styling options that determine the color of the button, as shown in Figure 2-80:

- Primary
- Alternate
- Success
- Information
- Warning
- Alert

These styling options map to Less variables within the theme definition. This function provides the coach author with the ability to style individual controls on a coach without loosing the benefits of theme-level styling.

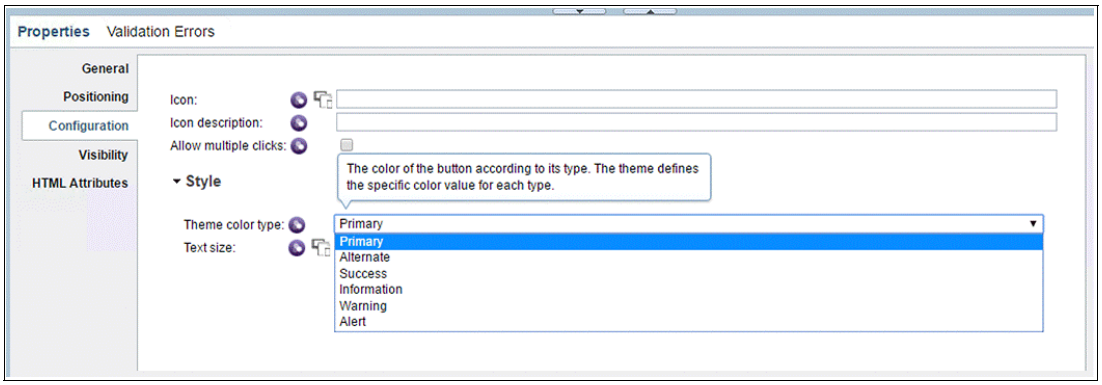


Figure 2-80 Button style configuration options

2.12.6 Applying a theme at run time

Themes can be applied either at design time or at run time.

We have already discussed how a theme is applied at design time, by updating the theme within the Process App Setting. In this case the theme definition is defined within either the process app or a toolkit dependency. However, you can also maintain a corporate theme that is applicable to all process apps. If you apply themes only at design time, the process author will need to update each process app, so that each app is using the correct version of the theme and then redeploy each app migrating all process instances. With IBM BPM 8.5.7, you can avoid this issue by applying the theme at run time instead.

As long as you deploy the correct theme definition to the target process server in one process app, you can run the **BPMUpdateTheme** command to apply the theme for another process app dynamically, as shown in Example 2-5.

Example 2-5 BPMUpdateTheme command

```
BPMUpdateTheme  -sourceContainerAcronym process_application_or_toolkit_acronym
-sourceContainerSnapshotAcronym snapshot_acronym [-themeName theme]
-targetContainerAcronym process_application_or_toolkit_acronym
-targetContainerSnapshotAcronym snapshot_acronym
```

Tip: Test a new theme in a test environment before applying it in a production environment, because it might produce undesirable results.

Refer to IBM Knowledge Center for further details:

http://www.ibm.com/support/knowledgecenter/SSFTDH_8.5.7/com.ibm.wbpm.ref.doc/topics/rref_bpmupdatetheme.html

2.12.7 Styling IBM Process Portal

The responsive Process Portal in IBM BPM 8.5.7 is completely coach-based (apart from the login screen). So you can apply a theme to the Process Portal process app in exactly the same way as with any other process app.

The theme definition includes specific Less variables (with the `@bpm-portal` prefix, as shown in Figure 2-79 on page 64) that correspond to various aspects of Process Portal.

See Chapter 5, “IBM Process Portal” on page 261 for more details about Process Portal.

2.13 Displaying coaches in different languages (using localization)

Localizing process applications enables business users in different geographic locales to interact with coach-based BPM UIs in their own languages. The business user selects language and locale settings within the profile using Process Portal. (See Chapter 5, “IBM Process Portal” on page 261 for more details.) After the business user specifies a personal language preference, the Process Portal UI displays automatically in the chosen language. Custom UIs, such as task completion and custom dashboards, also respond to these language preferences where localizations are defined within the coaches.

This section describes how to display coaches in different languages.

Language preferences: Selecting language preferences in Process Portal actually changes the language settings in the browser. Coaches that run using the Process Designer web editor are translated. So, it is not necessary to launch tasks in Process Portal to test UIs in different languages during development.

2.13.1 Defining a localization resource

Localization resources for a process application are contained in resource bundles, which are a collection of files that define key-value pairs for all the translated strings that are displayed in the application. Each supported language includes a corresponding file that contains translated values for all of the keys. You can define key-value pairs for specific works or entire phrases.

Localization resources are maintained within Process Designer, as illustrated in Figure 2-81, which shows the `LocalizationResource` localization resource that was created for the sample process application. This localization resource includes the following keys:

- `personalDetails`
- `forename`
- `surname`

The default value for `personalDetails` is Personal Details, which is used when there is no matching resource bundle for the user’s chosen language. In this case, the default language is English. Similarly, the default values for `forename` and `surname` are Forename and Surname,

respectively. For the purpose of the example, these are the defined values, but for a production solution, you will continue to define keys and default values for all aspects of your coach that you want to be translated.

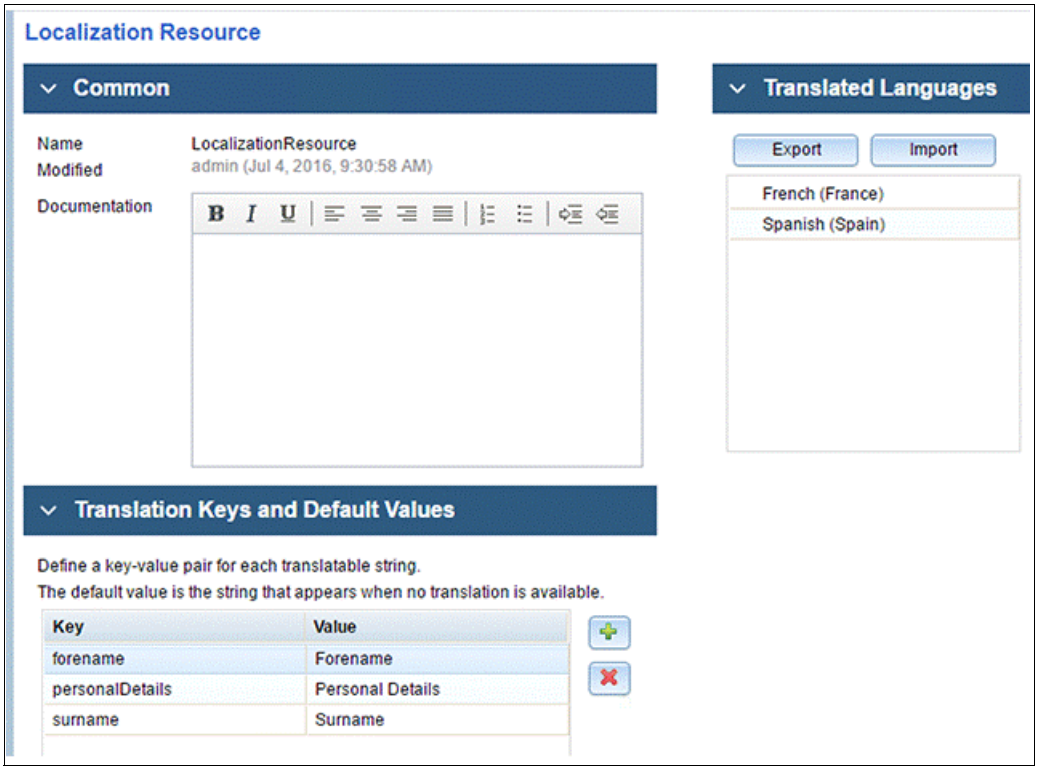


Figure 2-81 Defining a localization resource

The right side of the localization resource editor shows translations for both French and Spanish. To move resources:

- Click **Export** to download a compressed file that contains one resource bundle for the default values and one resource bundle for each of the translated languages.
- Click **Import** to upload a set of resource bundles contained within a single compressed file.

These resources enable you to maintain the translations outside of IBM BPM, which can be more efficient than using an editor directly, especially when translations can be auto-generated. Example 2-6 shows the French localization resource bundle for the sample process.

Example 2-6 French localization resource bundle

```
#
#Mon Jul 04 09:44:04 CDT 2016
forename=Pr\u00E9nom
personalDetails=D\u00E9tails Personnels
surname=Nom de famille
```

Tip: Define localization resources in toolkits so that they can be reused in multiple process applications.

2.13.2 Localizing user interfaces

The sample process app uses the example localization resource to demonstrate how localizations are applied to coach-based UIs. It uses the Person composite coach view, but the same principle applies to both coaches and coach views.

Opening the Person composite coach view and selecting the Variables tab shows the reference localization resources, as illustrated in Figure 2-82.

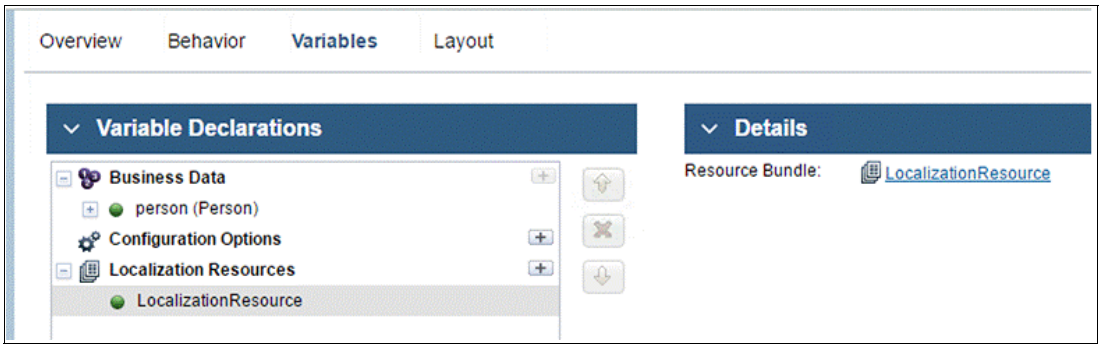


Figure 2-82 Declare localization resource references within a coach or coach view

Note: You must reference localization resources within the coach (or coach view) variable declarations before they can be used.

Switching to the coach (or view coach) editor, you can replace literal text with localization resource keys. Figure 2-83 illustrates how the localization resource has been used to display the Forename label.

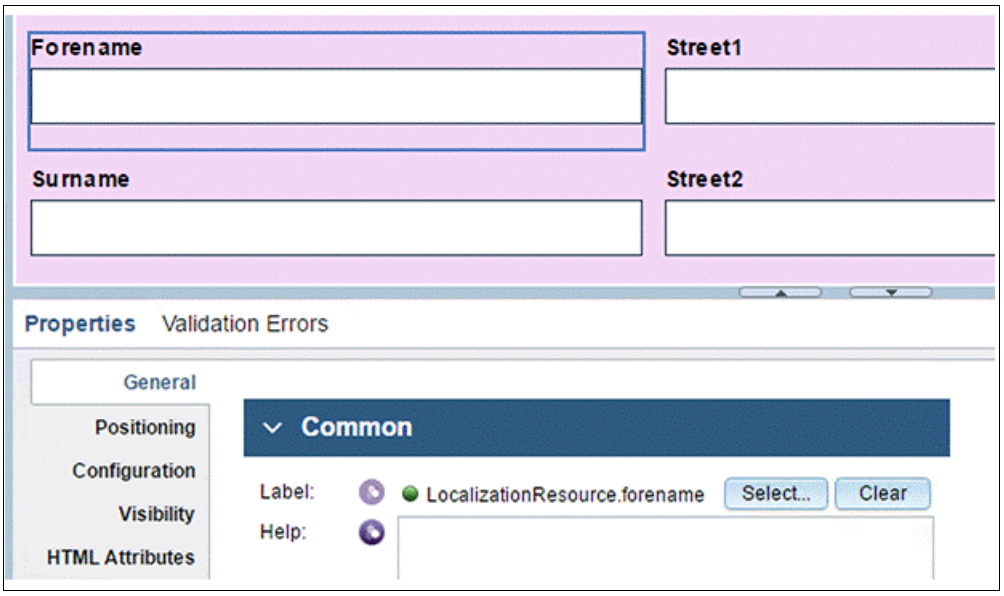


Figure 2-83 Use localization resource within a coach or coach view

For business users with a French language preference, this localization resource is translated as shown in Figure 2-84.

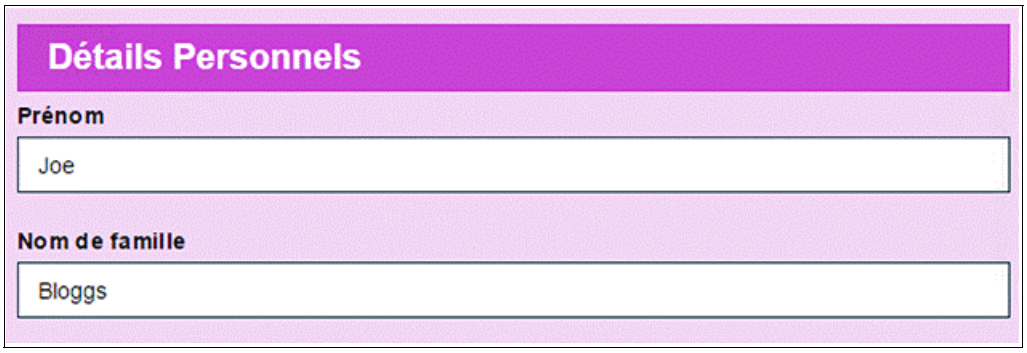


Figure 2-84 French translation

For more information about BPM localization, see IBM Knowledge Center:

http://ibm.biz/BPMLocalization

2.14 Performance considerations

Now that we have discussed the capabilities available from a modeling perspective, let's take a quick look at some performance considerations. How you model a coach can have an impact on the performance observed by the user. Therefore, you need to consider the performance recommendations discussed in this section. For further information about this topic, including runtime deployment performance considerations, see Chapter 4, "SPARK UI Toolkit" on page 153.

2.14.1 Consider the expected browser version when designing coaches

Rendering speed and efficiency vary significantly by browser vendor and release. When designing a coach, consider the browser that will be used in production. This consideration is particularly important for older browsers. Generally speaking, the more complicated the coach UI, the longer it takes to render on older browsers. Specific issues include:

- Large Document Object Model (DOM) trees
- Large JavaScript scripts
- Deeply nested coach views

2.14.2 Avoid building large coaches with too many controls

Coaches are intended to provide users with only the details that are necessary to enable them to perform a specific activity, so that they can focus on completing the activity without being distracted by irrelevant information. Following this principle generally leads to smaller coaches that perform well and provide a good user experience.

However, sometimes it is necessary to provide users with large datasets to enable them to complete their tasks. In these situations, it is tempting to build large coaches with all the data available on one coach. This type of coach can significantly impact the performance and user experience of the UI.

Instead consider one of the following approaching to avoid this situation:

- Use tabs to reduce the number of controls that are displayed when the coach first loads.
- Split large coaches into a number of smaller coaches with appropriate navigation.
- Use dashboards to provide general information that can be accessed at any time instead of loading the details directly within a task.
- Use controls with (*lazy loading*) to avoid retrieving unnecessary data when it is not currently visible (for example, by using the stock Tab control).

2.14.3 Minimize the number of server-side calls

Server-side calls add a significant performance overhead for a client-side human service. Server-side calls are made when:

- Using a nested server side service in a client-side human service
- Using Ajax services within your coach
- When boundary events are fired

Designing a client-side human service is different from designing heritage human services. Well designed client-side human services perform better because there are fewer round trips to the server. Adding too many nested server side services to a client-side human service significantly reduces the performance. In many cases, you can use nested client-side human services or scripts instead. Only use nested server side services when absolutely necessary, for example for security considerations.

Ajax has benefits and challenges. Minimal use of Ajax services can improve the user experience and performance of a coach by retrieving only data from the server as required. Alternatively, too much use of Ajax can significantly reduce the performance of a coach. Browsers can open a finite number of connections simultaneously. So too many Ajax calls might cause contention for resources and can degrade the performance of the coach.

2.14.4 Use client-side human services instead of heritage human services

Client-side human services were largely introduced to improve performance of BPM UIs and especially for mobile devices. Heritage human services inherently increase the number of server side calls compared to client-side human services. In addition to returning to the server after completing each coach, there are other situations where heritage human services make calls back to the server, as detailed in the following paragraphs.

For each boundary event that occurs within a heritage human service, a network request is made to the server and the current state of the coach (for example, all data values) is persisted. Both of these operations can be expensive. So use the fewest number of boundary events while still satisfying business requirements.

Business objects that are bound to coach views are persisted when boundary events occur within a heritage human service. This persistence can be a costly operation, especially for large business objects.

2.14.5 Minimize the size of business objects in the UI

If the business object used in the process flow is large and complex, create a separate business object that contains only fields that are relevant to the coach UI and that binds that business object to the coach view. After the coach actions complete, merge the contents of the coach-bound business object into the (larger) process flow business object. This method

is more efficient than passing unnecessarily large business objects to a client-side human service or persisting large business objects in a heritage human service when boundary events occur.

2.14.6 Judiciously use the Table control

Table controls are powerful and are the appropriate control for many business scenarios. However, tables can be expensive to render on some browsers, particularly older browsers. When using the stock Table control, ensure the total number of cells (number of rows x number of columns) is no more than are necessary. This method minimizes the browser rendering time.

Note: The Table control within the Spark UI toolkit uses server side pagination to improve performance. For more information, refer to Chapter 4, “SPARK UI Toolkit” on page 153.

2.14.7 Pick an appropriate delay time for auto-complete fields

For some coach controls, auto completion can be configured. Auto completion uses an auto completion service that is called after a certain delay to search for auto completion options. Use caution because these searches are performed against the server (for example, searching for a user name). The delay option specifies the time to wait after the user enters information in the field to when the request is sent to the server. If you choose a short delay, several requests might be sent before the user finishes typing, causing an unnecessary load against the server and also delaying user response time in the coach. Set a delay value that ensures that the user has finished typing. Several hundred milliseconds is a good compromise between responsiveness and reducing server load.

2.15 Conclusion

With what you learned in this chapter, you should be able to build modern UIs using IBM BPM stock controls. This chapter described how to assemble coaches based on the coach views provided as part of the product and how to configure the stock coach views to fit your needs. It included information about how coaches can be wired together, how validation can be added, and what to keep in mind to allow for good response times when users are interacting with the coaches that you built. It also discussed how to create composite coach views for business objects and how to use them.

The next chapter, Chapter 3, “Building controls using coach views” on page 73, describes how to go beyond that by building custom coach views from scratch to get full control over the components that constitute the UI.

3

Building controls using coach views

In Chapter 2, “Creating user interfaces with coaches” on page 9, you learned about how to build a client-side human service and coach as a coach author. The coach author implements user interface (UI) business logic by assembling controls. The coach view developer creates controls consumed by the coach author.

The coach author and coach view developer are two roles that are part of an IBM Business Process Manager (BPM) implementation team. The two roles can be assumed by one team member.

The Coach Framework is a powerful tool because it defines application programming interfaces (APIs) to allow a coach view developer to create a variety of building blocks that are consumed by the coach author. The coach views can use other JavaScript libraries or packages, such as the Dijit or Dojox packages from the Dojo Toolkit, AngularJS library, jQuery library, and other libraries and packages. This chapter describes the structure of a coach view and its APIs so that you can build coach views to be used by coach authors.

This chapter includes the following sections:

- Overview of a coach view structure
- Developing coach view basic functions
- Developing advanced functions of a coach view
- Patterns of coach view development
- Performance considerations
- Checklist for developing a custom coach view
- Conclusion

This document uses samples to describe the concepts of the coach view:

- The sample snapshot `SG24-8355-ch3-start.twx` allows you to build a coach view.
- The sample snapshot `SG24-8355-ch3-completed.twx` allows you to see the implemented coach views as a reference.

See Appendix A, “Additional material” on page 395 to download the samples.

Tip: Before you start developing custom coach views, enable JavaScript debugging in your IBM BPM environment. This capability allows the more readable Coach Framework JavaScript code and controls to be loaded. For more information, see:

`http://ibm.biz/BPMEnableJSDebugging`

3.1 Overview of a coach view structure

Coach views are the building blocks of a coach. Figure 3-1 shows each coach view, which consists of three parts in runtime. The three parts are as follows:

- *Data*. The bound data includes business data (that is, the binding object) and configuration options.
- *View controller*. The view controller has event handlers in different stages of the coach view lifecycle. The JavaScript code in it can process two kinds of inputs:
  - Coach view lifecycle event handlers invoked by the Coach Framework
  - Document object model (DOM) event handling invoked by end-user interaction with the DOM node of the coach view
- *View layout*. The view layout includes the HTML definition and styles of the control and JavaScript to manipulate the DOM node of the coach view.

Note: DOM event handlers can be placed in coach view lifecycle event handlers or inline JavaScript, but they are not part of a coach view lifecycle.

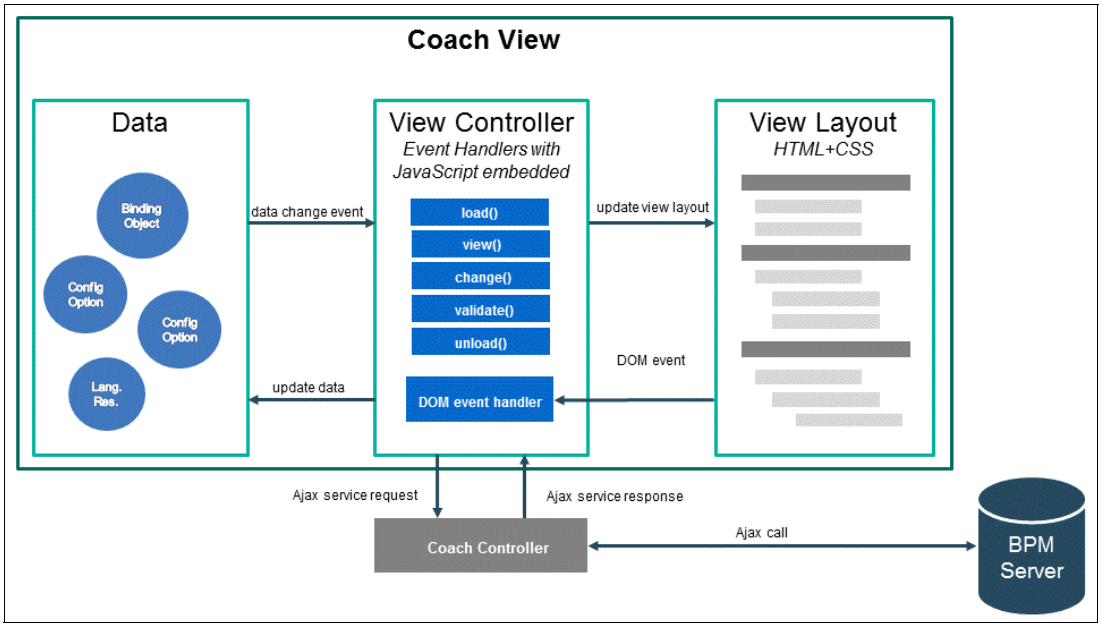


Figure 3-1 Overview of a coach view structure

From event handlers, the coach view can make Ajax service calls to the BPM server. After the response, the coach view event handler can update data and view presentation. Boundary events can be fired from event handlers, but this does not trigger Ajax service calls directly from the coach view in client-side human services.

In design time there are extra artifacts to better support coach authors who use coach views in IBM Process Designer. These artifacts include a palette icon, a layout image, URL binding for the layout image, and an HTML snippet file and JavaScript helper file for IBM Process Designer preview.

At run time the Coach Framework watches the data bound to Coach Views but is limited only to the top-level property of the bound variables by default. If the same human service variable is bound to multiple coach views, the `change()` event handler of each coach view is triggered every time the variable value is changed, either programmatically or by a user. This trigger provides a mechanism for coach views to communicate with each other.

Tip: In the coach view variable tab, you can also bind localization resources that can be used to translate the coach view configuration option label, the group name, and documentation text into local languages in design time. At runtime, this resource can be used in a composite coach view to translate nested coach views.

3.2 Developing coach view basic functions

This section describes how to build a runtime simple coach view, called Actionable Icon, which displays a specified icon with different colors and sizes. If enabled, clicking the coach view can trigger a boundary event to model the coach flow in a human service. For example, the coach can have a refresh icon to retrieve the newest data from the server. This coach view can also be reused to display some icons to indicate the different status of certain objects.

You can build a simple coach view by doing the following:

- Defining and accessing data in the coach view
- Constructing a view of the coach view
- Programming the coach view event handler to control the coach view's behavior
- Testing and debugging the coach view

3.2.1 Defining and accessing data in the coach view

A coach view can be associated with a data binding, which is called business data in the IBM Process Designer tool. The view context allows access to the data so that other parts of the coach view can use that data. For example, a table coach view, which is bound to a list of any type, can read the length of the bound list to know how many rows the table has.

This property is also the signature of the coach view. When dragging a service variable from the palette to the coach canvas, if the coach view business data matches the variable type, it can be used to represent the variable.

To create a simple coach view and set its binding object to the `iconClass, String` variable:

1. Click the plus (+) icon for User Interface, and select **Coach View**, as shown in Figure 3-2.

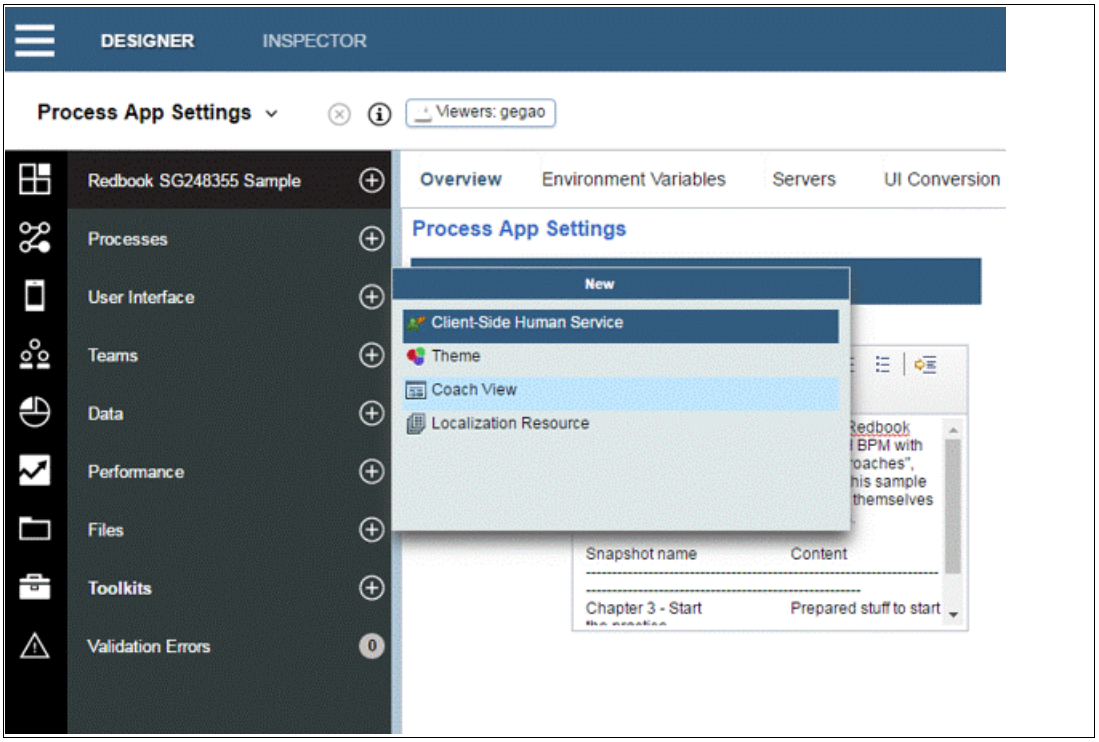


Figure 3-2 Create new coach view

2. You are prompted for the new coach view name. You are also prompted for how to initialize the view of the new coach view. You can start with a blank view or with a predefined grid layout, or you can select an existing template. Start with a blank view with the name *Actionable Icon* and click **Intended for use on multiple devices**. Figure 3-3 shows this view.

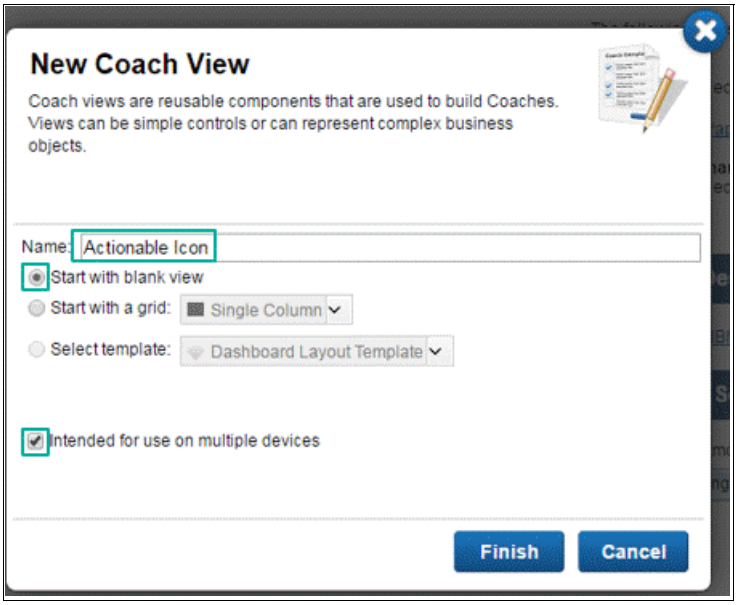


Figure 3-3 Set name and initial settings of the Actionable Icon coach view

3. When the new coach view is initialized in IBM Process Designer, you can define its business data. For this sample, enter the name `iconClass (String)`, where *String* is the variable type, as shown in Figure 3-4.

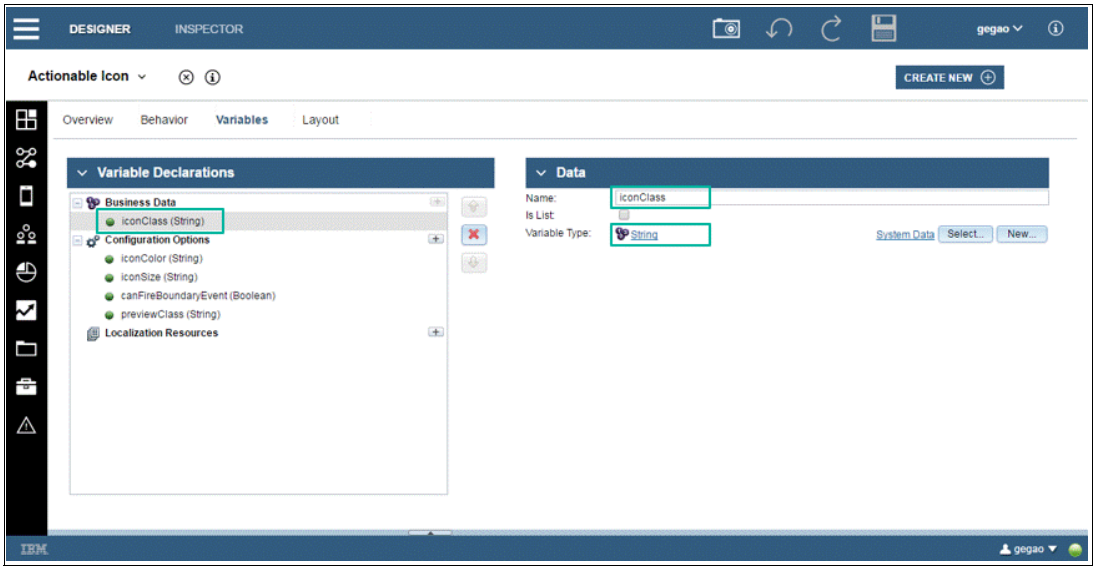


Figure 3-4 Set name and variable type of business data for the coach view

The binding object of each coach view is watched by Coach Framework in the web browser. If the binding object is changed, all coach views bound to it are notified. This mechanism binds different coach views together for interactions among each other. Therefore, business data can be used as data input for the coach view; it can also be used as data output to other coach views bound to the same variables.

Read and write configuration options

Configuration options can be used so that a coach view can interact with other coach views. Typical configuration options contain data only used within your process application, for example, to help you select the correct value from a list of options or to keep state information.

A coach view can have configuration options bound to it to configure its behavior in web browsers. As with business data, variables bound to configuration options are also watched by Coach Framework. Therefore, Coach Framework can also be used for data input and output of the coach view. Conceptually, the only difference between business data and configuration options is that business data is taken as a signature of the coach view and is regarded as the most important data.

Add the configuration options listed in Table 3-1 on page 74 to your sample coach view. The configuration options are not all of the variable type *List*. Because the icon is displayed in the same way on different screen sizes, leave the *Respond to Screen Size* setting cleared.

Table 3-1 Configuration options for the Actionable Icon coach view

Option name	Option type	Label	Description
iconColor	String	Icon color	Hexadecimal HTML color code of the font icon, for example, #F1F2F3. The default color is light blue #7CC7FF.
iconSize	String	Icon size	HTML size of the font icon in pixels, for example, 12 px. The default size is 32 px.

Option name	Option type	Label	Description
canFireBoundaryEvent	Boolean	Can fire boundary event	The icon only fires the boundary event on click if set to true. The default setting is false, and the icon does not fire a boundary event, even if it can be wired in the human service.
previewClass	String	Preview class	This option does not provide any runtime function. It is used for the coach author to better see the coach layout in design time.

The result is shown in Figure 3-5.

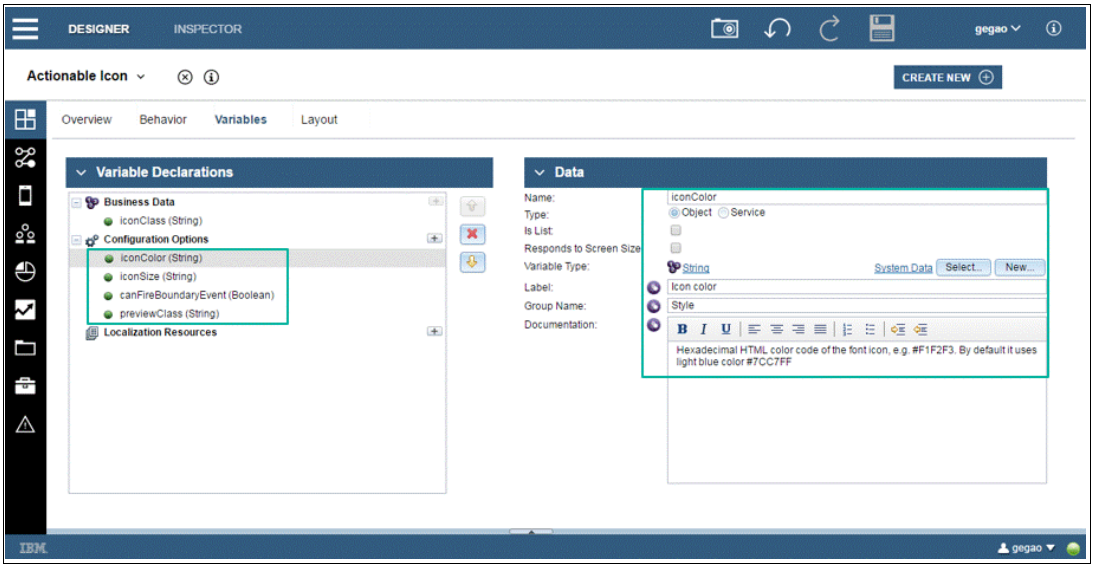


Figure 3-5 Set name, variable type, label, and documentation of configuration options

Tip: Use a label and documentation that are easy to understand for the variable for ease of use for the coach view builder.

The coach view label is shown in the Configuration tab of the coach view in the coach editor. Figure 3-6 shows how hovering the mouse over the label displays documentation as a tooltip.

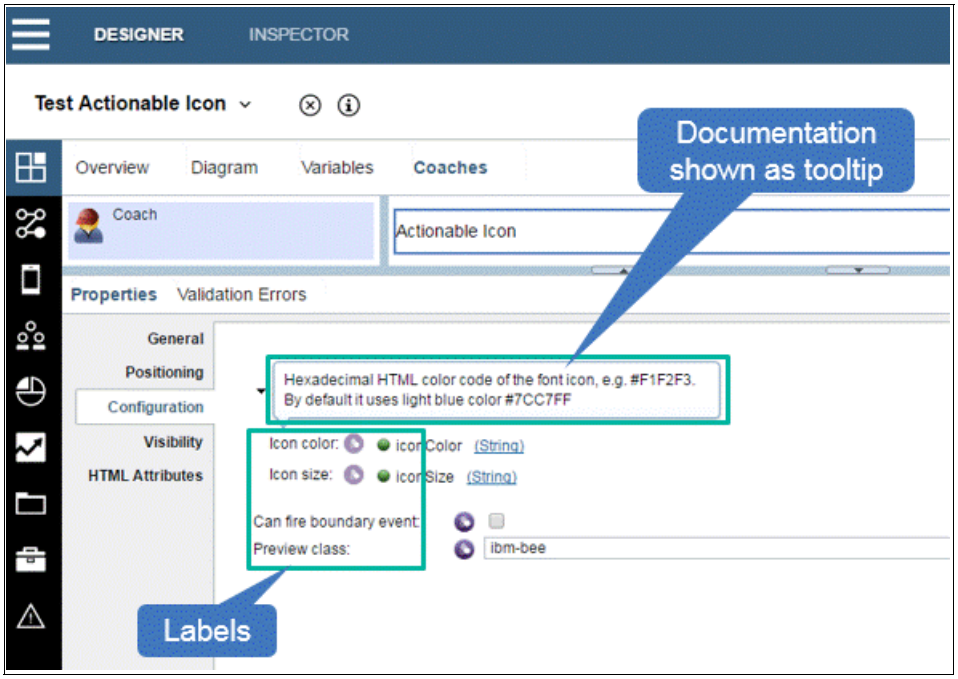


Figure 3-6 Label and documentation tooltip

You can also group related configuration options together with a display name set in the group. Configuration options with the same group name are organized into a collapsible category in the coach editor so that a coach author can have a better overview of the available options. Figure 3-7 shows an example of how you can select Style as the group name for both the `iconColor` and `iconSize`.

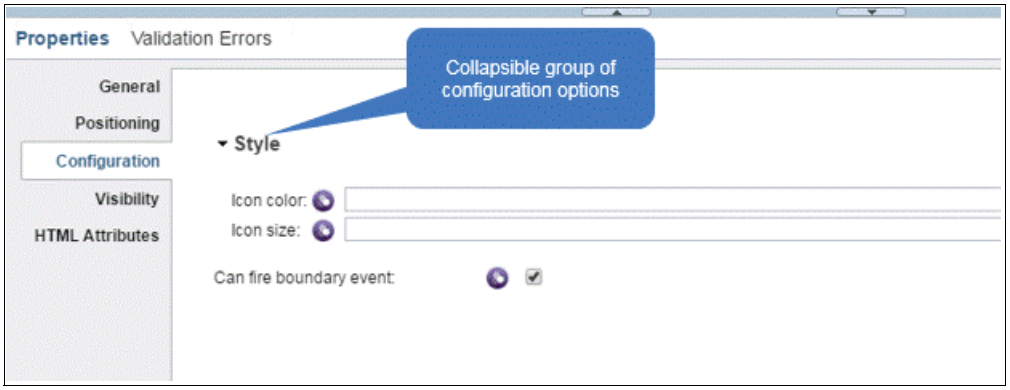


Figure 3-7 Collapsible group of configuration options

Manipulating coach view data using the coach API

When you have set the necessary data variables for the coach view, the data is accessed by the coach view by the API provided by Coach Framework. The view object represents a coach view and is a self-contained runtime JavaScript object that can render itself in a browser. When you are editing JavaScript in a coach view editor, the `this` keyword refers to the coach view. All coach view data is stored in the *context* object.

To get the value of the binding object variable, use the code shown in Example 3-1. This line of code checks first if the business data variable is bound to the coach view in the coach. The operator `!!` returns true only if the object is not null and not undefined. If the condition is satisfied, the code calls the `this.context.binding.get("value")` coach API to read the value. Otherwise, it sets the variable with a *false* default value.

Example 3-1 Code snippet to read the binding object value

```
var iconClass = (!! this.context.binding) ? this.context.binding.get("value") :
"ibm-bee";
```

You can use the `this.context.binding.set("value", newVal)` API to assign a new value to the binding object.

Example 3-2 shows an example of how, similar to business data, the value of variables bound to the coach view configuration options can be read or written.

Example 3-2 Code snippet to read the configuration options value

```
var iconColor = (!! this.context.options.iconColor) ?
this.context.options.iconColor.get("value") : "#7cc7ff";
var iconSize = (!! this.context.options.iconSize) ?
this.context.options.iconSize.get("value") : "32px";
```

To assign a new value to the variables bound to the configuration options, use the `this.context.options.<option_variable_name>.set("value", newVal)` API.

Copy the code snippets in Example 3-1 and Example 3-2 to the coach view `view()` event handler, as shown in Figure 3-8.

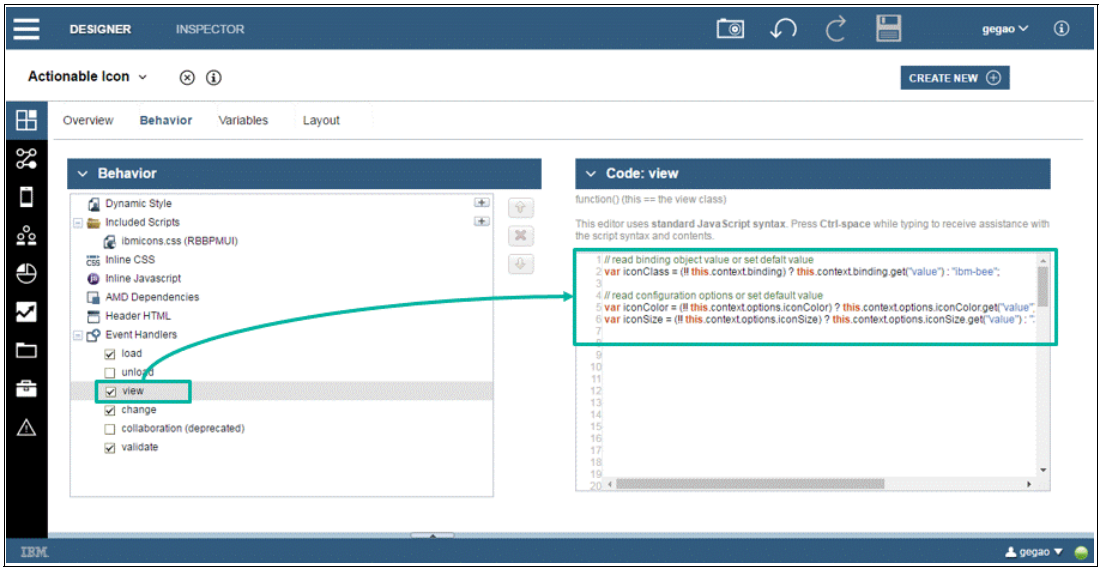


Figure 3-8 Code to read the binding object and configuration options in the `view()` event handler

When you bind to variables in the `load` event handler, use JavaScript conditional operators. These operators allow you to define defaults for variables that can make your coach view more robust. Defaults allow you to handle unbound configuration options or bindings in a more controlled way.

Tip: The following is a typical conditional operator:

```
yourVariable = (condition) ? value1 : value2;?
```

If the condition evaluates to true, `value1` is used. The default value is `value2`.

The Coach Framework by default watches on all bound variables in the coach in the web browser. (For more information about bound variables, see 3.2.1, “Defining and accessing data in the coach view” on page 75). This, however, does not apply to properties or subproperties of a complex type object. The `bindAll()` method can be used to watch on changes of first level properties of a complex type object. For deeper level properties, you can use the `bindAll()` method multiple times. If you do not want to watch on the change of some objects, you can use the `unbind()` method to remove watchers.

For more information about the coach API used to access coach view data, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewBindingAndConfigOptions`

The full coach API documentation can be found on the coach API website:

`http://ibm.biz/BPMCoachAPI`

When reading the binding objects and configuration options, do not store the objects in a local variable as cache because the value of these objects can be changed unpredictably at any time. For more information, see “Avoiding caching bound data” on page 99.

3.2.2 Constructing the view of a coach view

The view layout is one of the core components of the coach view. It represents data, which is displayed to process participants. The views rendered in the web browser are based on HTML and CSS code defined in the coach views. The structure represented in the web browser is based on a DOM tree, which is parsed from HTML code. You can construct the view for the coach view in the IBM Process Designer by using any of the methods:

- Assembling a composite coach view in the Layout tab, which is described in Chapter 2, “Creating user interfaces with coaches” on page 9
- Using custom HTML to define HTML in the Layout tab. The HTML code in custom HTML can be defined directly or loaded by a managed file or service variable
- Use the coach API and dynamic JavaScript to construct the DOM in the `view()` event handler

Custom HTML item

Figure 3-10 on page 85 shows how, using custom HTML, you can drag one or more custom HTML items from the palette to the coach canvas. After dragging the items, put the HTML definition in the HTML section of the coach view’s properties. You can put the code as text directly in the editor; or, you can put your code in an HTML file edited outside of IBM Process Designer, upload it, and link to it from custom HTML by selecting the Managed File option. Because using the Managed File option requires extra steps for each change to be tested in the HTML file, you can use direct editing as text, assuming the HTML editor suffices. You can also load HTML code snippets using a data variable of the coach view. This provides a flexible way to change the HTML definition during runtime.

Because coach views are reusable, it is common to have several coach views that are instantiated in a single coach. Sometimes a unique ID of a DOM node in a coach view’s

instance is required. If this is required, you can use the `$$viewDOMID$$` placeholder keyword. At runtime, this keyword is replaced by the coach view DOM ID. To generate a unique DOM ID for your custom coach view, in the Layout page, create a custom HTML control and add HTML code that is similar to Example 3-3.

Example 3-3 HTML sample code to generate a unique ID for a coach view

```
<div id="$$viewDOMID$$_myId1">?
<span id="$$viewDOMID$$_myId2"></span>?
<input id="$$viewDOMID$$_myId3" type="button" class="Mybutton" name=myjbtnName"
value="default">
</input>
</div>
```

Note: The `$$viewDOMID$$` placeholder keyword has limitations. For example, in a table control with embedded controls repeating for each row, the generated view ID can be the same in different rows. The generated view ID of a control in a coach might also have a conflict with the controls in a composite coach view in the coach. Use this placeholder only if it is *absolutely necessary*.

For more information about generating a unique ID for a coach view, see IBM Knowledge Center:

http://ibm.biz/BPMGenerateUniqueCoachViewID

When initially loading the coach, simple JavaScript variables with double curly braces can be used as replacement placeholders. These variables will be replaced with the value of those variables when loading the coach. Example 3-4 shows sample code with variable replacement in a custom HTML item.

Example 3-4 HTML sample code with variable replacement in a custom HTML item

```
<div>Hello, {{tw.local.user.name}}.</div>
<div>Cheers, {{tw.businessData.address.street}}.</div>
```

Table 3-2 list the three namespaces that can be used for variable substitution.

Table 3-2 Namespaces

Namespace	Context scope	Description
tw.local	Human service and coach	Reference to the local variable defined in the human service variable tab
tw.businessData	Coach view	Reference to the variable defined as coach view business data
tw.options	Coach view	Reference to the variable defined as coach view configuration options

Note: Use of double curly braces has limitations because this substitution occurs only one time during coach generation. If at runtime a variable changes, it is not updated. It is recommended to use JavaScript code in the `change()` event handler and to update the DOM as required.

For more information about variable substitution for a coach view, see IBM Knowledge Center:

http://ibm.biz/BPMCoachViewCustomHTML

Using coach APIs and dynamic JavaScript to construct views

Coach Framework provides useful APIs for you to manipulate a DOM tree of the view layout.

The context object of the coach view object is the core of all coach APIs. It provides two fundamental properties to locate the DOM element of the coach view:

- The `context.element` holds the root DOM element of the coach view.
- The `context.viewid` holds the unique identifier of the coach view.

When searching the DOM element of the coach view, the `context.element` can narrow the search scope to the coach view instance only. Within the scope you can further use HTML DOM search functions to locate the element you want to access. Search function examples follow:

- The `querySelector(/*String*/cssSelectors)` search function returns the first element, which matches the specified CSS selectors in the search scope.
- The `querySelectorAll(/*String*/cssSelectors)` search function returns the NodeList object containing all DOM elements, which matches the specified CSS selectors, in the search scope.
- The `getElementByTagName(/*String*/tagName)` search function returns a list of elements that match the specified tag name in the search scope.

Tip: The `querySelector()` and `querySelectorAll()` functions are the preferred functions because they have better performance than the `getElementByTagName()` function.

A coach view can be embedded into a coach view that has a Content Box item. For example, Text or Select controls can be embedded into Section, Tab, or Table controls. The template coach view is another example of embedding coach views. In this case, the enclosing coach view is the parent coach view of the embedded controls. Figure 3-9 illustrates the concept of the parent coach view.

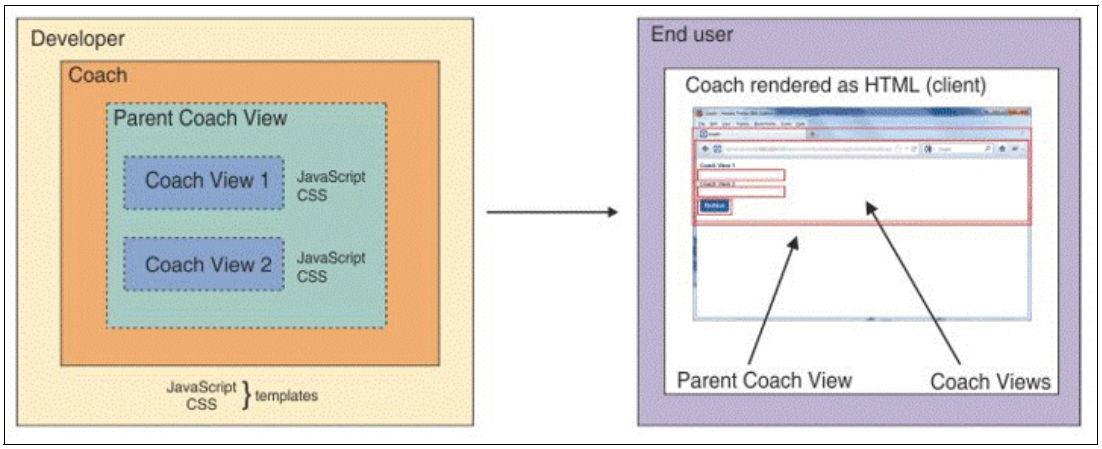


Figure 3-9 Enclosing the parent coach view

Coach Framework provides API functions to locate parent or child coach views, to create or delete child views, and to refresh the current coach view instance. The APIs are as follows:

- `context.subview[viewid]`
- `context.setUniqueViewId(uniqueViewId, targetNode)`
- `context.createView(domNode, index, parentView)`
- `context.deleteView(domNode)`
- `context.getDomNode(uniqueViewId, optParam)`
- `context.getSubview(viewId, requiredOrder)`
- context.parent`view()`
- context.refresh`view()`

For more information about the Coach Framework APIs, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewContextObject`

An atomic coach view is the smallest building block in a Coach Framework. Within an atomic coach view, at times the DOM node must be manipulated. For example, a drop-down selection box has DOM elements for text input and drop-down arrows. The behavior of these elements must be controlled separately during user interaction. For this type of requirement, you can use the Dojo base package API, which is available with Coach Framework or plain JavaScript. You can also use other third party library APIs, such as the following APIs:

- The `dojo/dom` module is the core module of the Dojo DOM API. It provides a method to locate the DOM node by ID.
- The `dojo/dom-construct` module is the module for DOM construction. It provides methods for placement, creation, and destruction of a DOM node.

For more information about `dojo/dom` and `dojo/dom-construct`, see the Dojo community website:

- `https://dojotoolkit.org/reference-guide/1.10/dojo/dom.html`
- `https://dojotoolkit.org/reference-guide/1.10/dojo/dom-construct.html`

The Dojo community provides the following tutorial for DOM functions:

`http://dojotoolkit.org/documentation/tutorials/1.10/dom_functions/index.html`

When the browser renders the coach, it must know the important attributes of the to-be-rendered DOM node. Examples of these attributes are the style and visibility of the node. Coach Framework provides APIs for the following attributes:

- `context.getInheritedVisibility()`
- `context.getStyle()`
- `context.setDisplay(isDisplay, domNode)`
- `context.setVisibility(isVisible, domNode)`

You can use Dojo base-package APIs to manipulate the style and visibility of a DOM element within the coach view in the same way that you do for construction of the view layout. You can also use other third-party library APIs as follows:

- The `dojo/dom-attr` module manipulates the DOM attributes.
- The `dojo/dom-class` module manipulates the DOM class.
- The `dojo/dom-prop` module manipulates the DOM properties.
- The `dojo/dom-style` module manipulates the DOM style.

For more information about the `dojo/dom-attr`, `dojo/dom-class`, `dojo/dom-prop`, and `dojo/dom-style` modules, see the following Dojo community websites:

- `https://dojotoolkit.org/reference-guide/1.10/dojo/dom-attr.html`
- `https://dojotoolkit.org/reference-guide/1.10/dojo/dom-class.html`

- https://dojotoolkit.org/reference-guide/1.10/dojo/dom-prop.html
- https://dojotoolkit.org/reference-guide/1.10/dojo/dom-style.html

Depending on your requirements, you can use the different approaches together. For this example, use the second and third approaches to construct the Actionable Icon coach view.

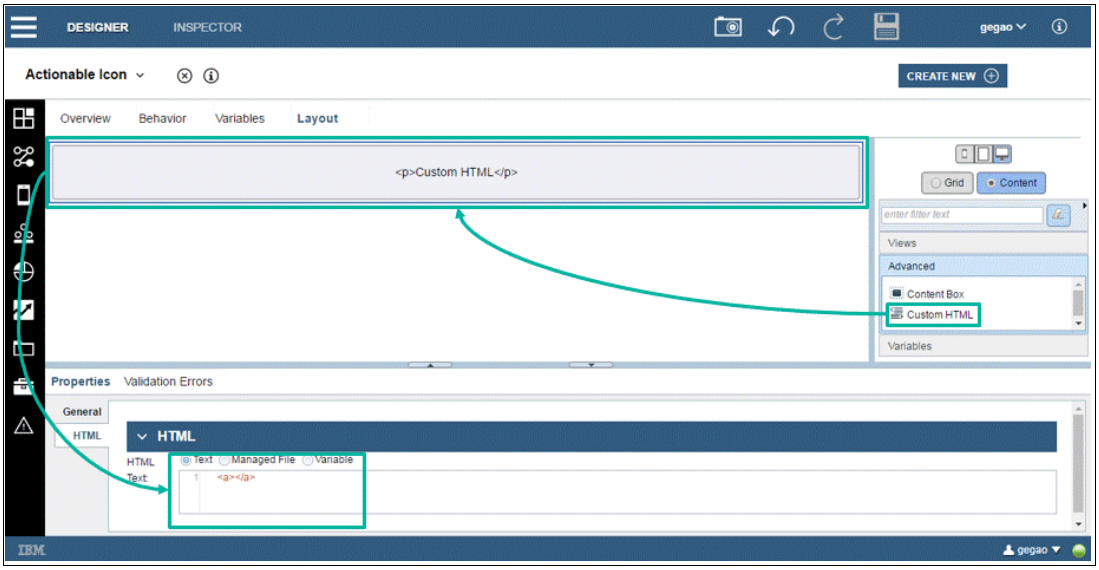


Figure 3-10 Drag the custom HTML item from the palette to the canvas and define the HTML code

Drag the custom HTML from the Advanced section on the right-side of the palette to the coach editor canvas. Paste the code snippet shown in Example 3-5 to the HTML definition shown in Figure 3-10.

Example 3-5 Code snippet of custom HTML in the layout

```
<a></a>
```

Now the Actionable Icon has a basic layout. In the runtime, the display of the icon is controlled by the CSS class, which is specified as the binding object of the coach view. The icon color and size are specified dynamically using configuration options, as described in “Defining and accessing data in the coach view” on page 75. To put these dynamic values into the view layout as DOM elements class and style, use the Dojo API programmatically. For more information about writing code in the coach view lifecycle event handlers to control the coach view behavior in runtime, see 3.2.3, “Programming the coach view event handler to control behavior” on page 88.

Use CSS and LESS for styling

When you create custom coach views, you can show the look of your coach views through CSS classes. The users of your coach view can easily change the look of your coach view without having to change the source code by copying and modifying it.

When you author coach views, use inline style definitions sparingly, as shown in Example 3-6, because inline styling is only available in the DOM node and cannot be reused. Also, inline styling does not support class mixing and inheritance in CSS.

Example 3-6 Sample code of inline styling

```
<div style="background:red;">
content
</div>
```

Instead, use explicit classes, as shown in Example 3-7.

Example 3-7 Sample code of an explicit CSS class definition

```
.myClass { background: red; }
```

As introduced in Chapter 2, “Creating user interfaces with coaches” on page 9, BPM 8.5.7 supports the CSS preprocessor LESS. If you want your coach view to change its style along with the change in the `theme` file, you can use the LESS variable in your style definition. For example, a theme-enabled class looks like Example 3-8.

Example 3-8 Sample code of style definition using the LESS variable

```
.myClass { background: @bpm-btn-primary-bg; }
```

Write the style definition in a `LESS` file and attach it to the coach view. Then use these classes when defining your coach view, as shown in Example 3-9.

Example 3-9 Sample code using the CSS class

```
<div class="myClass">
  content
</div>
```

Your coach view users can change the look of the coach by overriding the class, as shown in Example 3-10.

Example 3-10 Sample code using the overriding CSS style

```
myClassOverride .myClass
 { background: blue; }
```

As mentioned in 3.2.2, “Constructing the view of a coach view” on page 81, the web browser renders the page by interpreting the DOM tree and the attached styles, which are defined in CSS. Figure 3-11 on page 87 shows where to add the CSS in the coach editor. You can use one of the following methods to define the CSS styles of the coach view:

- Inline CSS
- Included CSS file
- Included LESS file

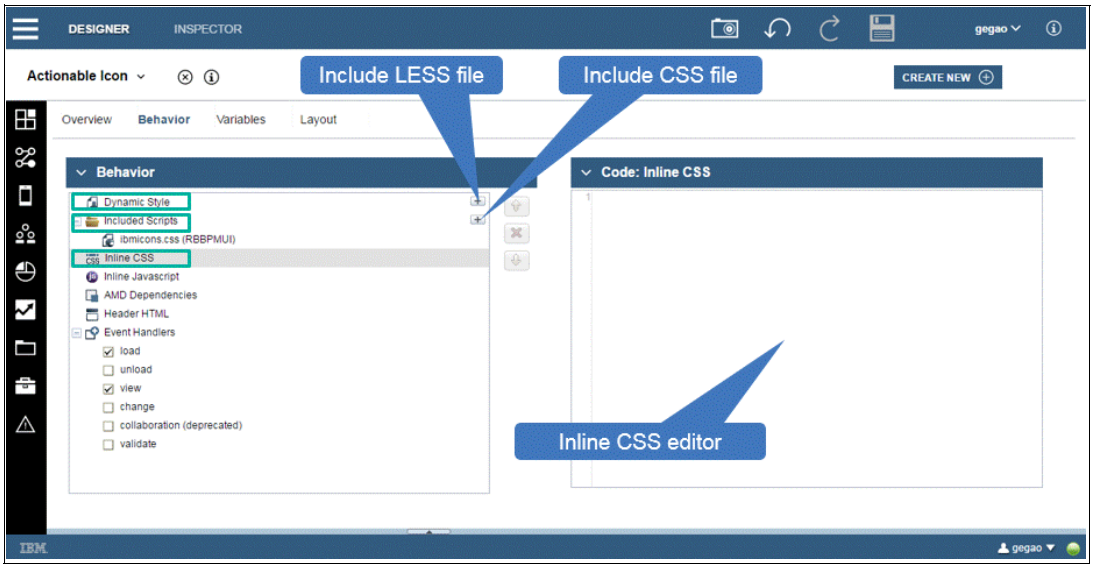


Figure 3-11 Coach editor supports adding CSS styles

The inline CSS is easy for development because the change applied in the inline CSS editor takes effect immediately when testing the coach view. However, the styles defined in inline CSS cannot be shared in other coach views. A CSS file can be included in different coach views so that the defined classes can provide a consistent style for all coach views.

Tip: A good practice is to define styles in the inline editor for development and unit test them using the coach view developer. When fixed, move the reusable CSS classes into one LESS or CSS file and attach that file to one atomic or template coach view for later reuse. The LESS file is preferred if you use a theme.

In your sample Actionable Icon coach view, use IBM Design icon fonts to display the icon. Icon fonts are fonts containing glyphs or symbols that can be used as regular fonts with CSS styling. In the sample application, the font files are already uploaded and referenced by the ibmicons.css and ibmicons.min.css style sheets with the CSS class name defined for each font.

Figure 3-12 on page 88 shows how to add the CSS file for the IBM Design icon fonts. Click the plus icon (+) under Included Scripts in the Behavior tab of the coach editor. Then enter ibmicons.min.css, which is a minified version of ibmicons.css for better performance, in the pop-up list search field to filter out the CSS file. Select the CSS file.

Tip: Although a CSS or JavaScript file can be included in different coach views and one coach view can have multiple instances in a coach, the same file is not loaded multiple times when the HTML document of a coach is loaded into a browser. The Coach Framework detects the duplication of the same included file in a coach and loads it in the HTML head section one time.

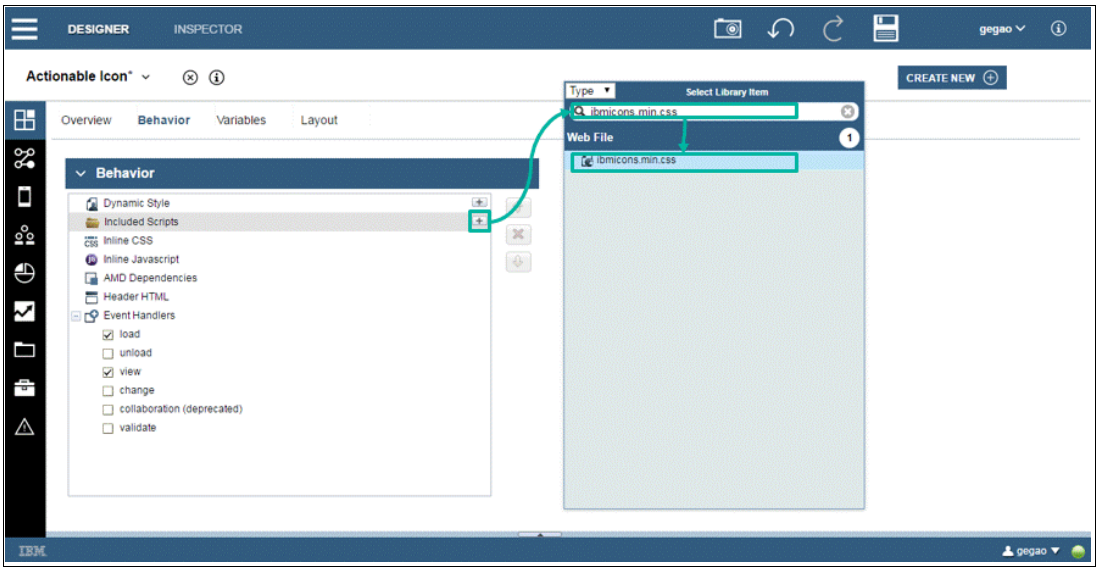


Figure 3-12 Add the CSS file as included script of a coach view

3.2.3 Programming the coach view event handler to control behavior

Section 3.1, “Overview of a coach view structure” on page 74 describes how the view controller plays a central role to control the data and view presentation of a coach view. The event handlers of the view controller have embedded JavaScript to achieve this task. Each event handler has a different function within the lifecycle of a coach view. It is important to understand the lifecycle model shown in Figure 3-13.

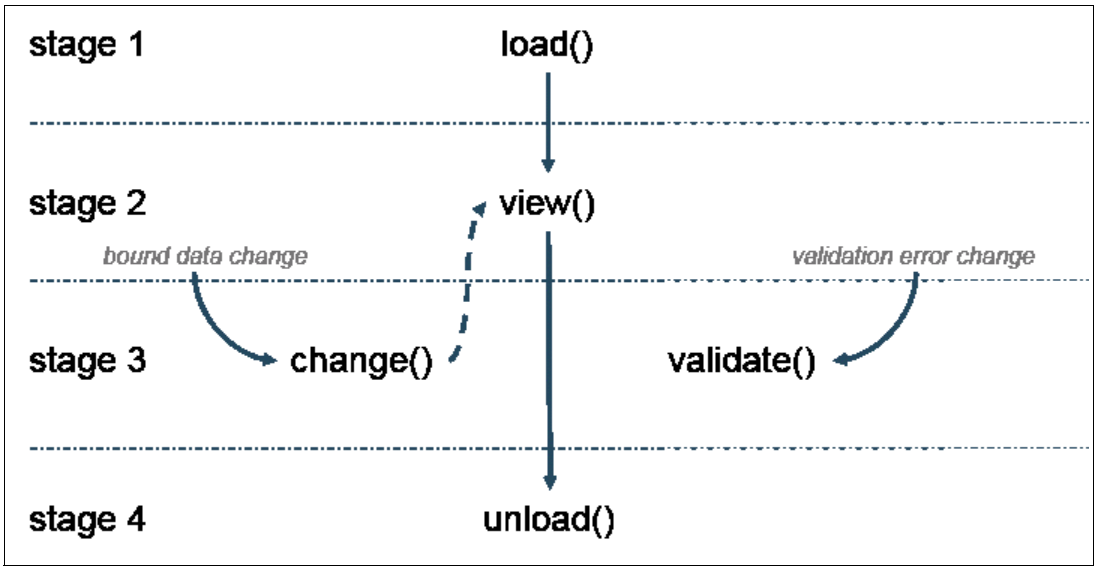


Figure 3-13 Coach view lifecycle event handlers

The coach view lifecycle has four stages. During browser runtime, the coach view always starts in stage one, when the `load()` event handler is called. The initialization logic can be put into stage one. After stage one, the coach view enters into stage two, when the `view()` event handler is executed to prepare for the code necessary for rendering. The rendered view then appears.

A change event of bound data can trigger the move from stage two into stage three; or a change event of a validation error observed by the Coach Framework can trigger the move from stage two into stage three. The `view()` event handler can be called from a `change()` event handler. The code in the `view()` event handler can be executed multiple times. If the `change()` event handler is not implemented, the `view()` event handler is called by default.

Because the change event of bound data and validation error can be triggered any time in any sequence, the event handlers in stage three can be started at any time and in any order. After the stage three event handlers are finished, the coach view falls back to stage two and waits for the next change event. If the browser page is closed, or the coach view is removed from the coach, the stage four event handler `unload()` is called for cleanup and only called for one time.

For more information about coach view event handlers, see IBM Knowledge Center:

http://ibm.biz/BPMCoachViewEventHandlers

The load() event handler

The `load()` event handler is used for initialization logic. For example, you can initialize local variables or locate or construct a DOM node for later use in other event handlers.

Figure 3-14 shows how you can copy the code snippet in Example 3-11 to the `load()` event handler.

Example 3-11 Code snippet of `load()` *event handler for the Actionable Icon sample*

```
// locate DOM element of to be displayed icon
this.iconNode = this.context.element.querySelector("a");
```

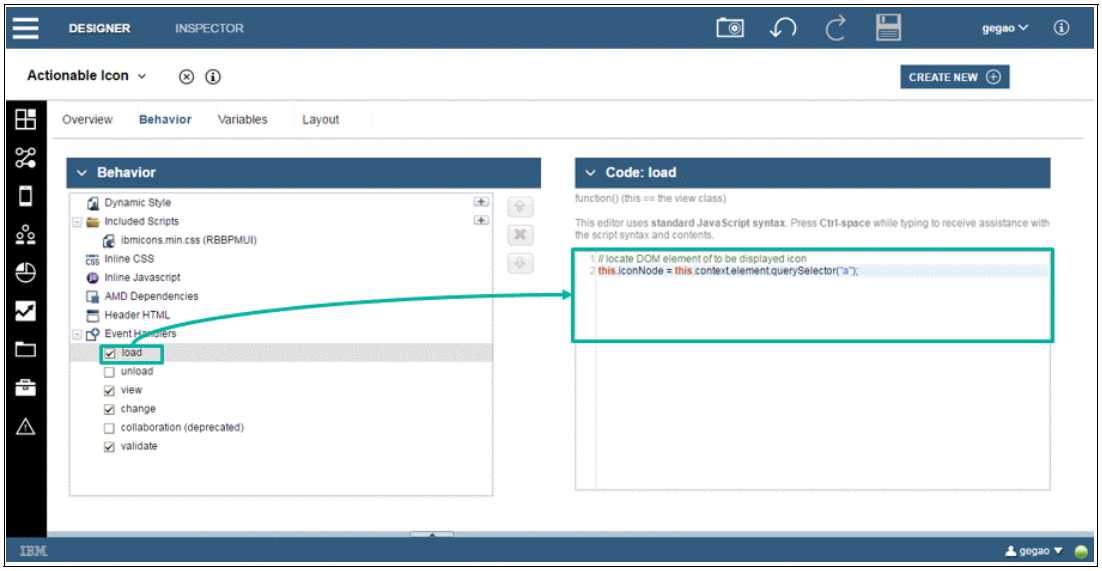


Figure 3-14 The load() event handler in the coach editor

The view() event handler

The `view()` event handler is used to prepare the view before it is displayed to the user. The preparation can include calculation of coach view visibility, construction of the view DOM nodes, setting of the initial DOM element attributes, and other preparation.

In your example, Actionable Icon, you first read the bound data (that is, the binding object and configuration options) and then use `querySelector()`to locate the DOM element for the icon display in the scope of the coach view instance (`this.context.element`).

You then use Dojo APIs to set the initial class and style of the icon to be displayed.

Tip: When editing code in coach view editor, half the width of the editor window is displayed. You can hover the mouse to the right side of the editor so that an icon for expanding the editor width appears. Click the icon to expand the editor to the full width of the editor area.

To finish the `view()` event handler code in your sample, copy the code in Example 3-12 into the Actionable Icon `view()` event handler. See Figure 3-15.

Example 3-12 Sample code of the locate and manipulate DOM element

```
// manipulate DOM element by Dojo API
domClass.add(this.iconNode, iconClass);
domStyle.set(this.iconNode, "color", iconColor);
domStyle.set(this.iconNode, "font-size", iconSize);
```

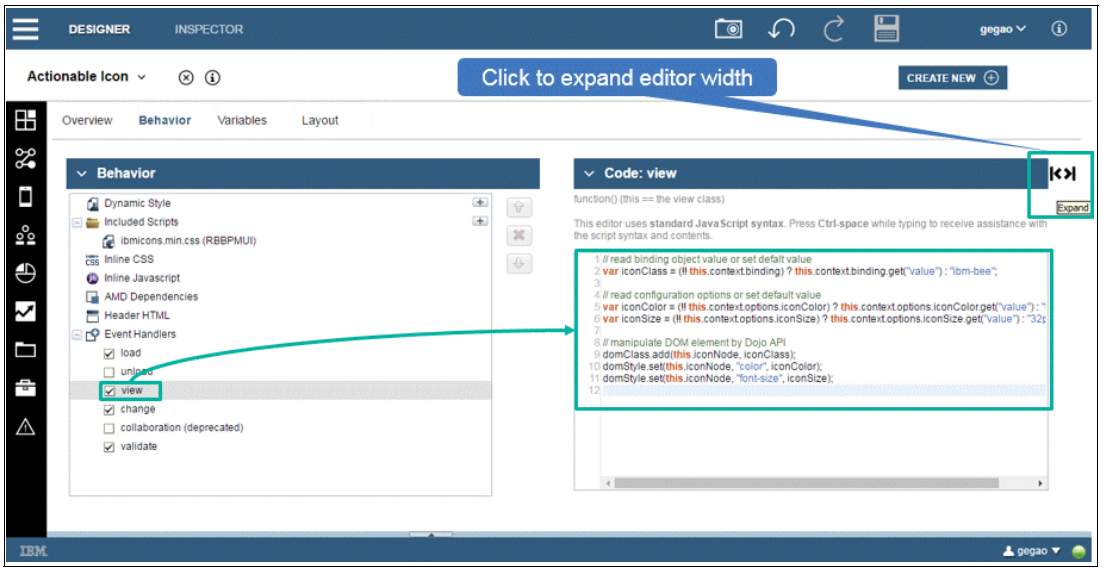


Figure 3-15 The view() event handler in the coach editor

Add AMD module dependencies

The sample Actionable Icon coach view now has constructed its view. However, it cannot yet be run. An error appears when you test the coach view in a human service because the Dojo base package API is modularized and all modules you use in your `view()` event handler must be added as Asynchronous Module Definition (AMD) dependencies to the coach view.

AMD is a concept in JavaScript. Implementation of AMD, for example, using the Dojo toolkit, must provide modularized functions and define its dependencies on other modules if required. By implementing in AMD style, the code is encapsulated in its own module scope and, therefore, is better organized in a logical way. It helps you avoid using code outside of the module, which might cause an error.

When developing a coach view, you must define the alias for each required AMD module. The alias holds the reference to all properties, methods, and events within the scope of the coach view instance. This prevents a naming clash between different coach views. Table 3-3 lists the AMD modules and their aliases that are required by the Actionable Icon coach view.

Table 3-3 AMD dependencies required by the Actionable Icon coach view

Module ID	Alias
dojo/dom-class	domClass
dojo/dom-style	domStyle
dojo/dom-attr	domAttr

Figure 3-16 shows how to add the listed AMD modules and alias to the sample coach view in IBM Process Designer.

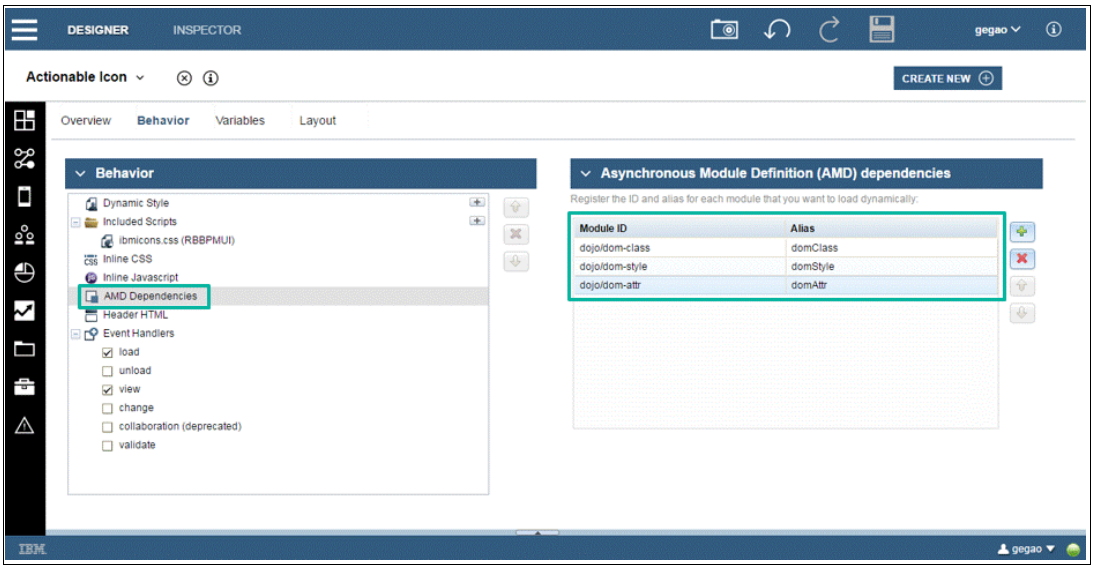


Figure 3-16 AMD dependencies in IBM Process Designer

The Coach Framework generates coach view JavaScript code based on the declared AMD dependencies. To better understand how the AMD works, review the generated AMD definition in your browser debugger. The code snippet in Example 3-13 shows that the aliases domClass, domStyle, and domAttr are defined as coach view global references and can be referenced anywhere in the scope of the coach view. The other module aliases *engine*, *CoachView*, and *require* are generated for each coach view for use by Coach Framework; therefore, do not use them as aliases in your coach view.

Example 3-13 Code snippet of generated an AMD definition of a custom coach view

```
define(
"com.ibm.bpm.coach.Snapshot_3b6e1eed_eac4_4c9d_b792_0db45645d9fe.Actionable_Icon",
["com.ibm.bpm.coach/engine",
"com.ibm.bpm.coach/CoachView", "require",
"dojo/dom-class", "dojo/dom-style", "dojo/dom-attr"],
function(
engine, CoachView, require,
domClass, domStyle, domAttr
) {
```

```
// code
}
)
```

You can now create and test a client-side human service with a coach containing the Actionable Icon coach view. Figure 3-17 shows an example of how to create a client-side human service with the name *Test Actionable Icon*.

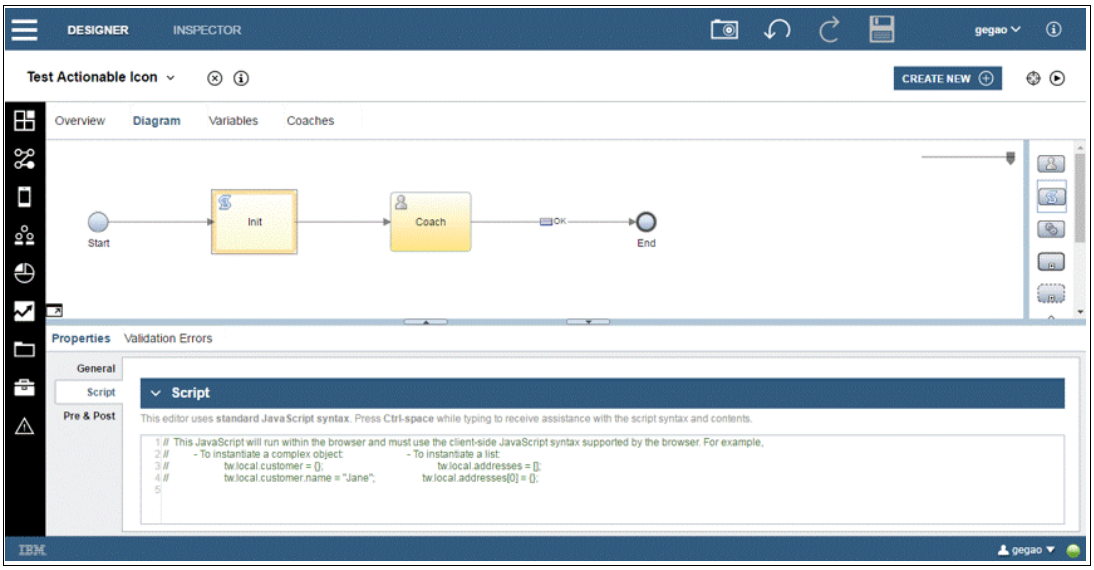


Figure 3-17 Client-side human services initial testing

After you have tested the client-side human service, drag one Actionable Icon to the coach canvas using the configuration options shown in Table 3-4.

Table 3-4 Configuration option for first test of Actionable Icon

Configuration option	Value
Icon color	#5596e6
Icon size	128 px

Figure 3-18 shows the configuration settings for the client-side human services testing for the Actionable Icon.

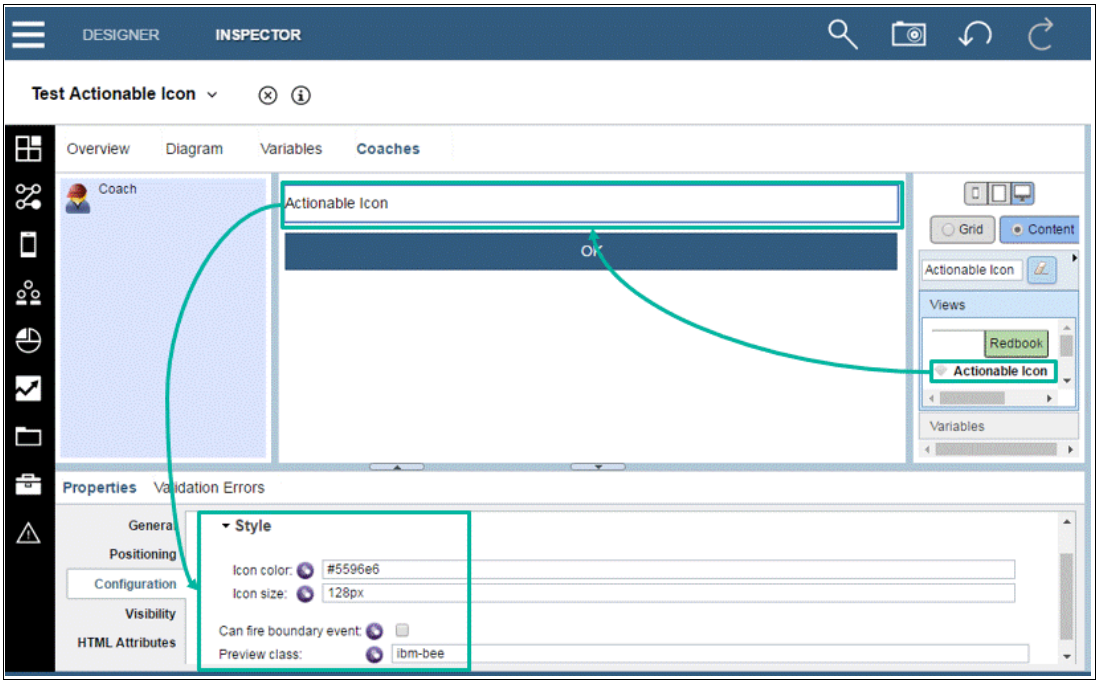


Figure 3-18 Configuration option settings for testing client-side human services for the Actionable Icon

Leave the binding object empty; the default icon class name *ibm-bee* is then displayed. Drag a client-side script to the service diagram before the coach for use in the `change()` event handler. (This use is explained in “Using the change() event handler” on page 94.)
Figure 3-19 shows how to see the testing results by clicking the run icon on the top-right corner of the window. The custom coach view has read the configuration options set in the coach and constructed the view with the default font icon class and styles applied. This is done by the `view()` event handler.

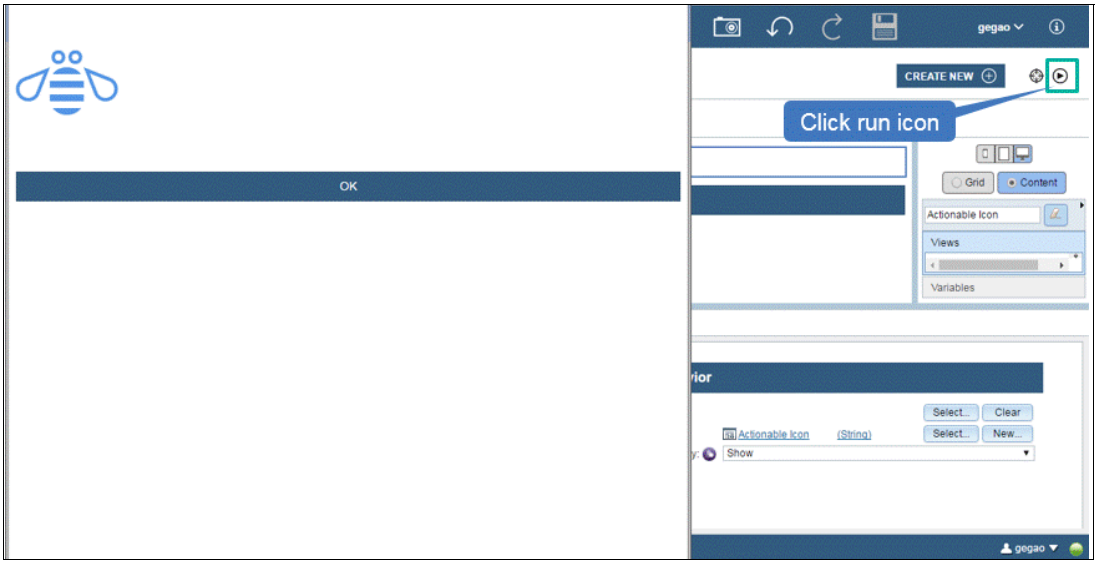


Figure 3-19 Running testing for client-side human services for the Actionable Icon with initial settings

Using the change() event handler

After the `view()` event handler has finished running, the coach view stays in stage two. Any change on the binding object or configuration options triggers the `change()` event handler. If the `change()` event handler is not implemented, the `view()` event handler is called by default.

After the `change()` function is called, it has one single input parameter, the event object. It contains information about the change. Table 3-5 lists the properties contained in the event object.

Table 3-5 Properties of the change() event handler input parameter of the event object

Property	Type	Description
type	String	Valid values are `binding`, `config`, `undefined`, or `null`. An `undefined` or `null` value means `binding`.
property	String	The property that was changed. If the changed type is configuration option data, this is the name of the variable defined in the coach view, not the service variable name bound to the coach view. If the changed type is the binding object, this is the name of the service variable bound to the coach view.
oldVal		For simple types, the previous value.
newVal		The new value.
insertedInto	Integer	For arrays, the index where the object was added to the list.
removedFrom	Integer	For arrays, the index where the object was removed from the list.

For more information about the `change()` event handler, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewChangeEventHandler`

To better understand how the `change()` event handler works:

1. Create several private service variables in the Test Actionable Icon testing service. Figure 3-20 shows how to create the service variables.

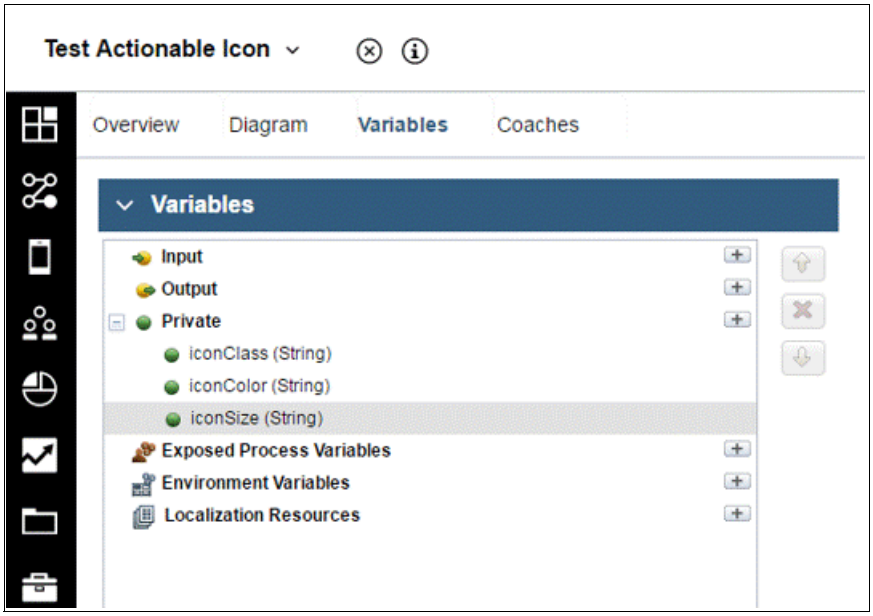


Figure 3-20 Defining variables in the Test Actionable Icon human service

2. Copy the code snippet shown in Example 3-14 to the human service step *Init* to initialize the value of these variables, which are bound to the Actionable Icon coach view.

Example 3-14 Code snippet for initializing coach view configuration options in testing human service

```
tw.local.iconClass = "ibm-bee";
tw.local.iconColor = "#5596e6";
tw.local.iconSize = "128px";
```

3. Open the coach and follow these steps:
   a. Drag three stock Text controls to the coach canvas. To make the controls more user friendly, change the label to *Icon class*.
   b. Bind `tw.local.iconClass` to the business data of the Actionable Icon coach view.
   c. Bind `tw.local.iconColor` and `tw.local.iconSize` to the Actionable Icon coach view configuration options.
   d. Bind variables to the binding object of the Text control, respectively; for example, bind the `tw.local.iconClass` variable to the output field for icon class.

By binding the same service variable to different coach view instances, a connection between the coach views is set up. For example, a change to the `iconClass` variable from the Text coach view triggers the Actionable Icon coach view `change()` event handler. This trigger happens because the Actionable Icon coach view configuration option is also bound to the same `iconClass` variable that is changed. Figure 3-21 shows this connection with a yellow, dashed line.

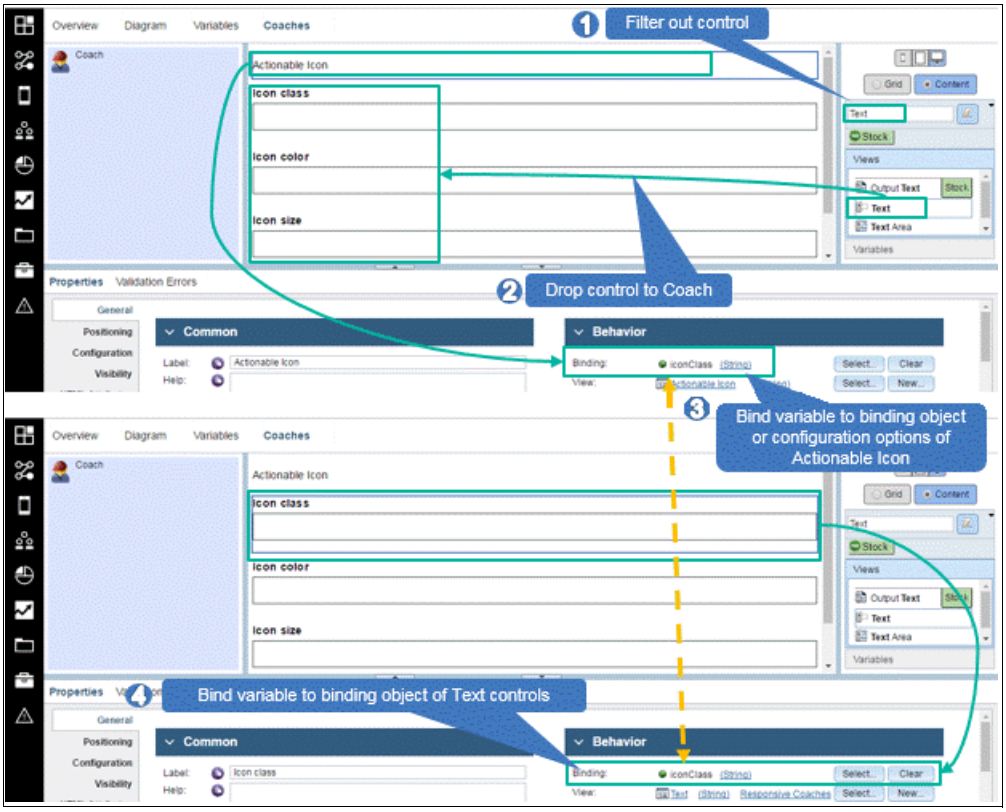


Figure 3-21 Binding variables to different controls in coach

Figure 3-22 shows how to put a console log into the change event to inspect the event object in the browser debugger.

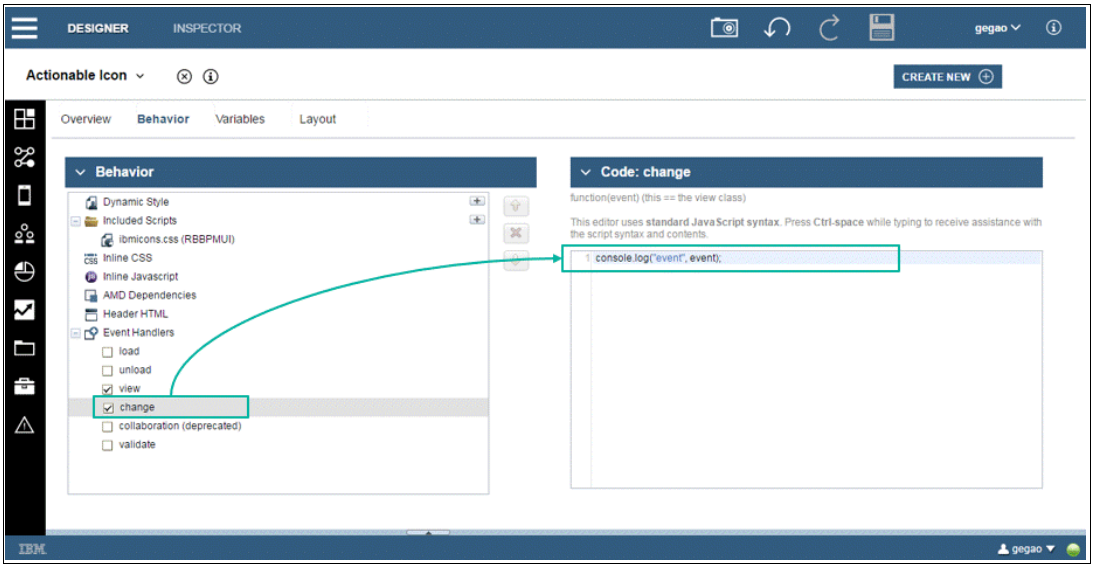


Figure 3-22 Adding a console log in the change() event handler to observe the event object

Figure 3-23 shows that with the browser debugger open, by changing the icon class name in the text field and moving the focus away from that field (by selecting another field for example), the event object is printed in the debugger console.

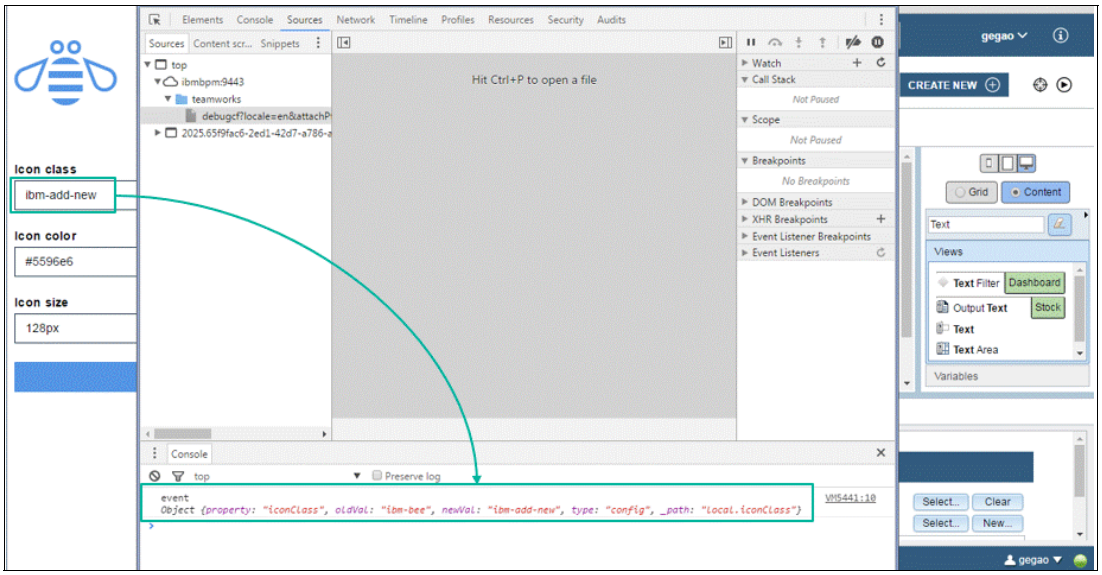


Figure 3-23 Event object displayed in the browser debugger

You can see that the old value and the new value are contained in the event. The type of the event is `config`, because the configuration option `iconClass` is changed.

Tip: Because the change event type of the configuration option is always explicitly specified as `config`, and all other possible values indicate the type of binding object, check the type against `config` by using an *if, else* clause in your code:

```
if (event.type == "config") {
// code to process configuration option change
} else {
// code to process binding object change
```

The *property* value is the configuration option variable name defined in the coach view.

Note: If the changed type is a binding object, the property value is *not* the binding object variable name defined in the coach view; it is the human service variable name, which is bound to the binding object in the coach. By using the *if, else* statement, you do not have to check the `property` value of the event object.

Filter change event

The changed value in the Text control is propagated to the `change()` event handler of the Actionable Icon control. Now you have to implement the event handler to update the view layout of the icon.

There are many configuration options and one binding object defined for the coach view. Every bound variable can trigger the `change()` function; coach view metadata such as `visibility` can trigger it as well. Because during the design process you do not know which source can trigger the change in runtime, you must explicitly process those changes affecting coach view behavior and ignore unexpected changes. This requires filtering on the event. For example, on the change of certain configuration options, a coach view makes an Ajax call to retrieve data from the backend; on the change of other data, this does not happen.

Note: Filtering events is important, especially when expensive operations are called in your change logic. For example, if a back-end integration that takes some seconds to run is called as a result of a change, ensure that it is not called unnecessarily.

In the Actionable Icon sample, update the DOM node of the icon if the bound configuration options `iconClass`, `iconColor` or `iconSize` change. Copy the code shown in Example 3-15 to the `change()` event handler, replacing existing code.

Example 3-15 Code snippet to handle changes on configuration options of Actionable Icon

```
if (event.type == "config") {
   if (event.property == "iconColor") {
      domStyle.set(this.iconNode, "color", event.newVal);
   } else if (event.property == "iconSize") {
      domStyle.set(this.iconNode, "font-size", event.newVal);
   }
} else {
   domClass.remove(this.iconNode, event.oldVal);
   domClass.add(this.iconNode, event.newVal);
}
```

Figure 3-24 shows how to update the DOM node in the Actionable Icon window.

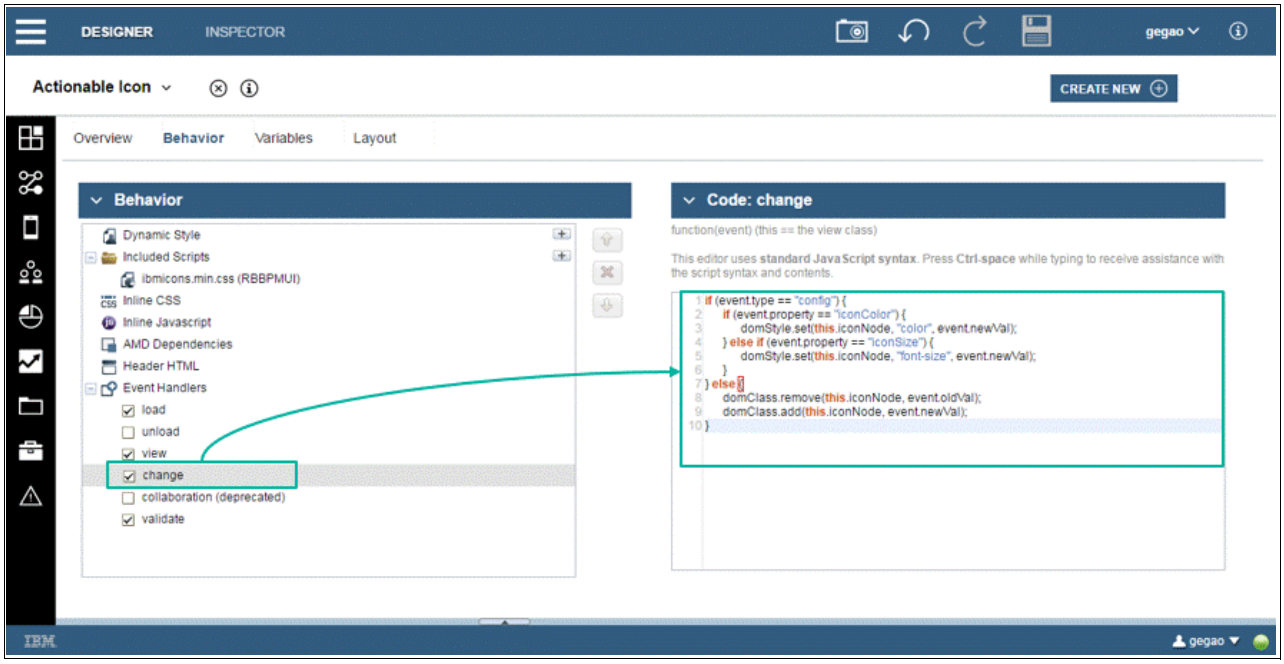


Figure 3-24 Coach view change() event handler

In the code, the expected changes are filtered out and handled respectively. The unexpected change is ignored.

When you run the testing human service now, you can change the icon class in the text control to another class name, for example `ibm-add-new` or `ibm-remove-delete`. You can also change its color code and size.

For a complete list of IBM Design icons and their class names, see the IBM Design website:

`http://ibm.biz/IBMIconFont`

Figure 3-25 shows the effect of updating the DOM node. It shows the change with two coaches before and after the update.

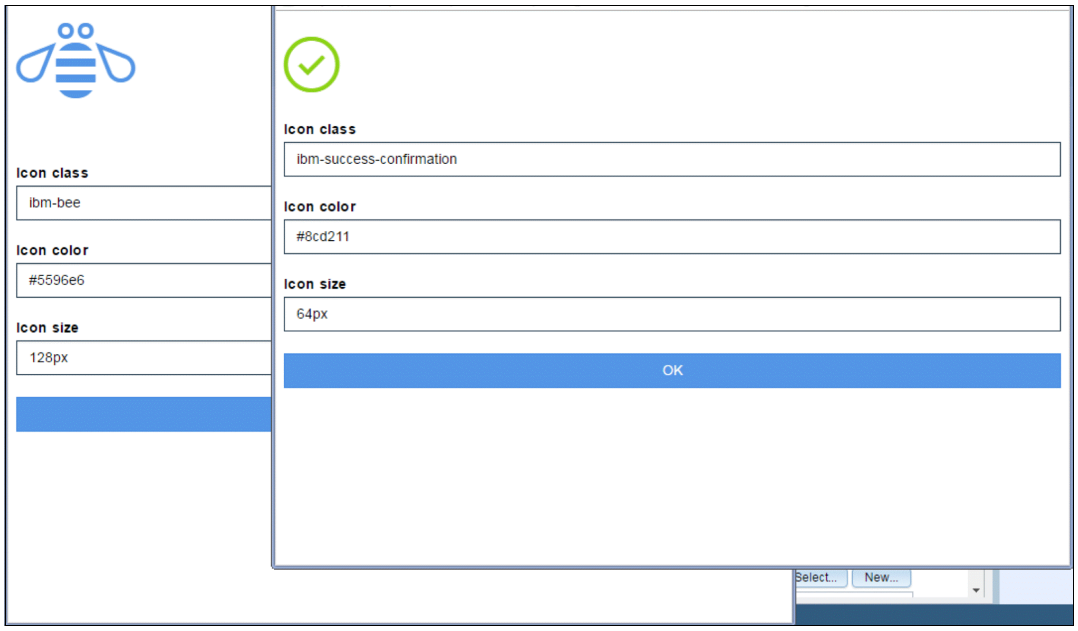


Figure 3-25 Comparison of the Actionable Icon change result

Avoiding caching bound data

As described in 3.2.3, “Programming the coach view event handler to control behavior” on page 88, the `change()` event handler can run at any time after the `view()` function is executed. Also, the `view()` function can be called later from the `change()` function; thus, any locally cached bound data might be stale.

Figure 3-26 gives an example of stale local-cached bound data.

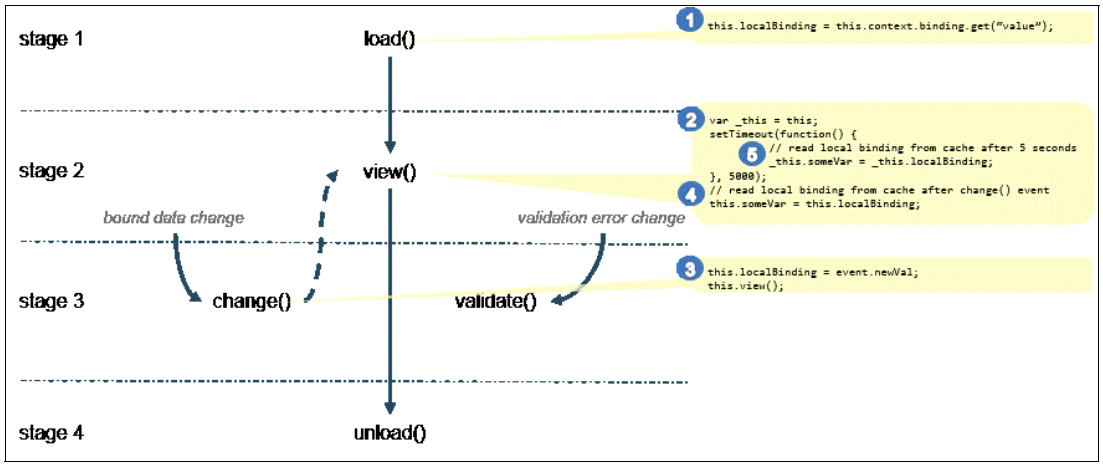


Figure 3-26 Stale locally cached bound data

The binding object is stored in a variable as cache in the global scope of the coach view in the `load()` event handler. In the `view()` event handler, the code within the `setTimeout()` function reads the value of the binding object from the local cache after five seconds. The code snippet in (3) can run before (4) and (5). In this case, the locally stored cache is already stale and the value to be read will be wrong. Locally cached bound data can only execute

sequentially within an event handler function where no asynchronous code execution, such as the `callback` function or `setTimeout()` function, is used.

The preferred practices to cache binding objects are as follows:

- Avoid caching binding objects or configuration-option objects and use them in other event handlers because they might change.
- Avoid caching binding objects or configuration-option objects within an event handler if there is asynchronous code execution in the event handler.
- When you are developing coach views, use the **`this.context.binding`** command to access the binding object of a coach view instance. Use the **`this.context.options.<option_name>`** command to access the options of a coach view instance.

The validate() event handler

The Coach Framework provides validation capability that you can use to validate the data that is in the coach before the flow proceeds to the next step in the service flow. Validation logic is described in Chapter 2, “Creating user interfaces with coaches” on page 9. A review of possible validation approaches follows:

- Client-side validation by the coach view itself, for example maximum or minimal value range of a Decimal control
- Client-side validation in the Data Change section within a coach
- Client-side validation in client-side script triggered by a boundary event in a client-side human services
- Server-side validation by calling service triggered by boundary event in client-side human services

Note: In heritage human services, validation in the server script or a nested service is always server-side because a boundary event in heritage human services always sends the event to the backend for flow navigation. Note that only client-side human services are covered in this chapter.

Except for the coach view, which provides its own validation logic, all other approaches can be programmatically controlled.

The `tw.system.coachValidation.addValidationError(variableName, errorMessage)` API is called to store the validation errors into the `tw.system.coachValidation` system variable. When the validation variable is returned, Coach Framework calls the `validate()` event handler of the coach views; the binding object of the `validate()` event handler contains the validation error. When this occurs, the coach view lifecycle turns from stage two to stage three. The handler contains the logic to display error indicators and the error messages that are defined in the validation service.

Figure 3-27 on page 101 shows how to copy the code snippet in Example 3-16 to the Data Change section of the coach in the test client-side human services.

Example 3-16 Code snippet for adding the validation error in Data Change of the coach

```
tw.system.coachValidation.removeValidationError( "tw.local.iconClass");
var listOfAllowedIconClasses = ["ibm-bee",
"ibm-add-new",
"ibm-remove-delete",
"ibm-cloud",
```

```
"ibm-reset-revert"];
if (listOfAllowedIconClasses.indexOf(tw.local.iconClass) == -1) {
tw.system.coachValidation.addValidationError( "tw.local.iconClass", "Entered icon
class is not valid. Allowed class names are: " + listOfAllowedIconClasses);
}
```

You can see in the code snippet that the validation error of the binding object of both the Actionable Icon and first Text controls, `tw.local.iconClass`, is first removed by using the `tw.system.coachValidation.removeValidationError(variableName)` API. Then the value of `iconClass` is checked. If it is not in the range of the allowed list, a validation error is added to the `tw.system.coachValidation` system variable.

For more information about the JavaScript API in client-side human services, see IBM Knowledge Center:

http://ibm.biz/BPMCSHSJSAPI

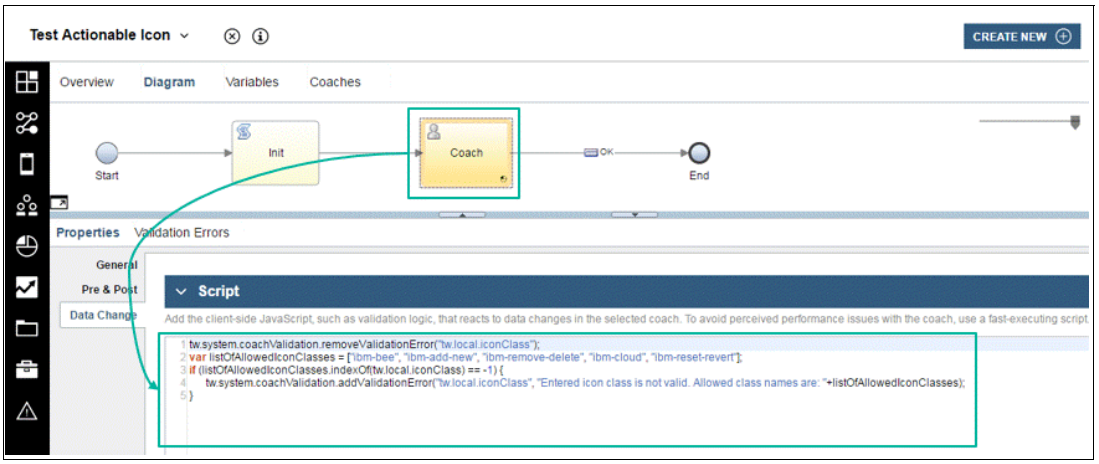


Figure 3-27 Adding the validation error to the system variable in the Data Change section of coach

If you run the testing human service and change the icon class name to an invalid name, such as ibm-bee-1, the `validate()` event handler is called. As you did for the `change()` event handler, add a console log to the `validate()` event handler (Figure 3-28 on page 102) to see what is passed in when Coach Framework calls the event handler.

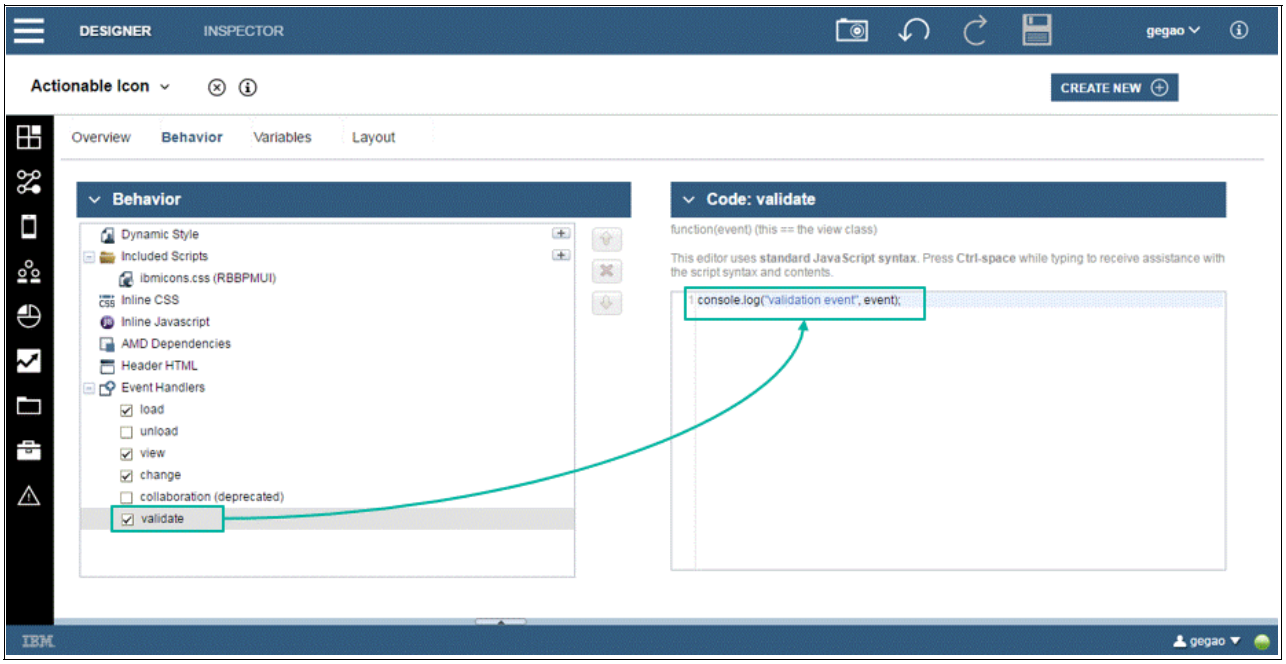


Figure 3-28 Adding a console log in the validate() event handler

If you run the testing human service and change the icon class name to invalid text, the Text control displays the validation error specified in the Data Change section and in the browser console. Figure 3-29 on page 103 shows the validation event. The event object has either an *error* or *clear* property type. The *error* property type indicates that there is a validation error to be displayed for the coach view; the *clear* property type indicates that the coach view must clear any existing validation error. If you have corrected the entered value, the *clear* event is triggered to notify the coach view to remove the displayed validation error. In our sample, the property type is *error* because a validation error on the icon class name has been detected. In the Data Change section code, one validation error message has been added to the system with the variable name `tw.local.iconClass`. This can also be seen in the event object. The `errors` list contains one element, which has the error message added before. The element has a property called *binding*, which indicates the relative path to the binding object of the coach view. Because the binding object is a simple type, the relative path to it is empty.

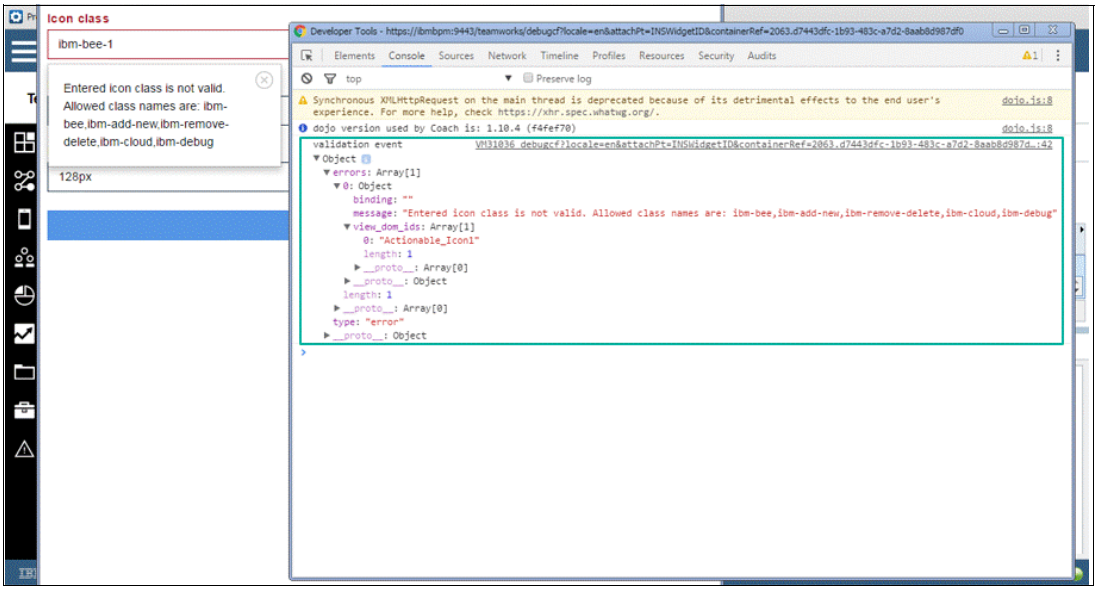


Figure 3-29 Event passed into the validate() event handler shown in the browser console

For more information about the event in the `validate()` event handler, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewValidateEventHandler`

Figure 3-29 shows that the error message of the first Text control is displayed in a pop-up dialog and the text field including the label is highlighted with red color. This is because both binding objects of the Actionable Icon coach view and the Text control are bound to the same `tw.local.iconClass` variable. The icon font has disappeared because no valid class name of the font exists. You must display visual information of the validation error on the Actionable Icon so that users know the variable bound to it is also wrong.

Tip: Although the implementation for displaying a validation error can be different, a basic rule is to present a consistent look and feel. Based on the main style of controls in your solution, you can style the validation error of your own control with shared common CSS.

In the sample, the other Text and Button controls are all from the Responsive Coaches toolkit, which uses the Angular JavaScript library for the underlying implementation. Because the sample does not use AngularJS, the HTML annotation used by AngularJS is not available. Use a pure HTML definition to simulate the HTML annotation by referencing the same styles. Then copy the code snippet in Example 3-17 on page 104 to the custom HTML item in the Actionable Icon coach view after the existing anchor node.

Example 3-17 HTML code snippet for displaying validation error with the responsive coaches style

```
   <div class="validation-error-container BPM_Resp_Icon" style="visibility:
hidden;">
      <div tabindex="0" role="alert" class="bpmValidationContainer popover bottom
in alert-container">
          <div class="arrow"></div>
          <div class="close-container">
          <button class="close" type="button" title="Close alert box">
            <svg class="bpm-svg bpm-svg-close-circle"><use
xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:href="#bpm-svg-close-circle"></use></svg>
         </button>
          </div>
          <div tabindex="0" class="popover-inner">
            <div class="alert alert-alert.type validation-error-message"
role="alert" type="alert.type"></div>
          </div>
      </div>
   </div>
```

The generated Text control HTML can be read in the browser debugger. When you compare the code snippet in Example 3-17 with the HTML definition of the Text control, you find the sample code does not contain any `ng-*` annotation, which is AngularJS-specific. Instead, you find CSS class names used in the sample code, for example, validation-error-container and validation-error-message. These are helper classes not defined in any style sheet; they help identify the DOM node by the CSS selector. Another class called `BPM_Resp_Icon` is used to match the predefined responsive coaches validation error styles. Combining with a class name of `BPM_Resp_*` at the highest-level DOM node, the predefined styles can be matched.

Tip: The helper class name can be used to identify certain DOM nodes. An example follows.

HTML definition:

```
<div class="validation-error-container BPM_Resp_Icon" style="visibility:
hidden;">
</div>
```

In runtime, use the CSS selector to obtain the DOM node:

```
this.context.element.querySelector("div.validationErrorMessage");
```

Copy the code snippet in Example 3-18 to the `load()` event handler to locate the validation error message DOM node. This DOM node is rendered later in the `validate()` event handler when a validation error event occurs.

Example 3-18 Code snippet to locate the validation error DOM node in the load() event handler

```
this.validationErrorContainerNode =
this.context.element.querySelector("div.validation-error-container");
```

Copy the code snippet in Example 3-19 to the `validate()` event handler to display the error message.

Example 3-19 Code snippet to render the validation error

```
if (event.type == "error") {
   domClass.remove(this.iconNode, this.context.binding.get("value"));
   domClass.add(this.iconNode, "ibm-close-cancel-error");
   domStyle.set(this.iconNode, "color","#ad1625");
   domStyle.set(this.validationErrorContainerNode, "visibility", "");
   var validationErrorNode =
this.validationErrorContainerNode.querySelector("div.validation-error-message");
   var closeButtonNode =
this.validationErrorContainerNode.querySelector("button.close");
   var _this = this;
   on(closeButtonNode, "click", function(evt) {
      domStyle.set(_this.validationErrorContainerNode, "visibility", "hidden");
   });
   for (var i=0; i<event.errors.length; i++) {
      var error = event.errors[i];
      if (error.view_dom_ids[0] == this.context.element.id) {
            validationErrorNode.innerHTML = validationErrorNode.innerHTML +
error.message;
      }
   }
}
```

The coach view must display an error icon with the same highlighting color of the responsive coaches if the class name of the binding object is invalid; responsive coaches use #ad1625 as coloring code. First, remove any existing class name from the icon DOM node. Then add the error icon font class name, `ibm-close-cancel-error`, and set the same highlighting color, `#ad1625`, to the icon. The DOM node for validation error display sets the visibility to normal so that it is shown. You use the helper class to obtain the error message DOM node. You also use it to set the `innerHTML` to the error message contained in the validation event object with the matching DOM identifier.

You can also see that the Close button is located by the CSS selector and connected to a click event. Use a new dojo API, `dojo/on`, to listen on the DOM event. Figure 3-30 on page 106 shows how to enable this function by adding an AMD dependency.

Before the declaration of the listener on the Close button, the coach view object *this* is assigned to a local variable `_this`. This occurs because within the scope of an anonymous function in the listener function, the coach view object `this` is not accessible. Instead it accesses the local variable defined in the enclosing scope. This is an important concept of the JavaScript called Closure. When the Close button is clicked, the DOM node of the validation error container is hidden.

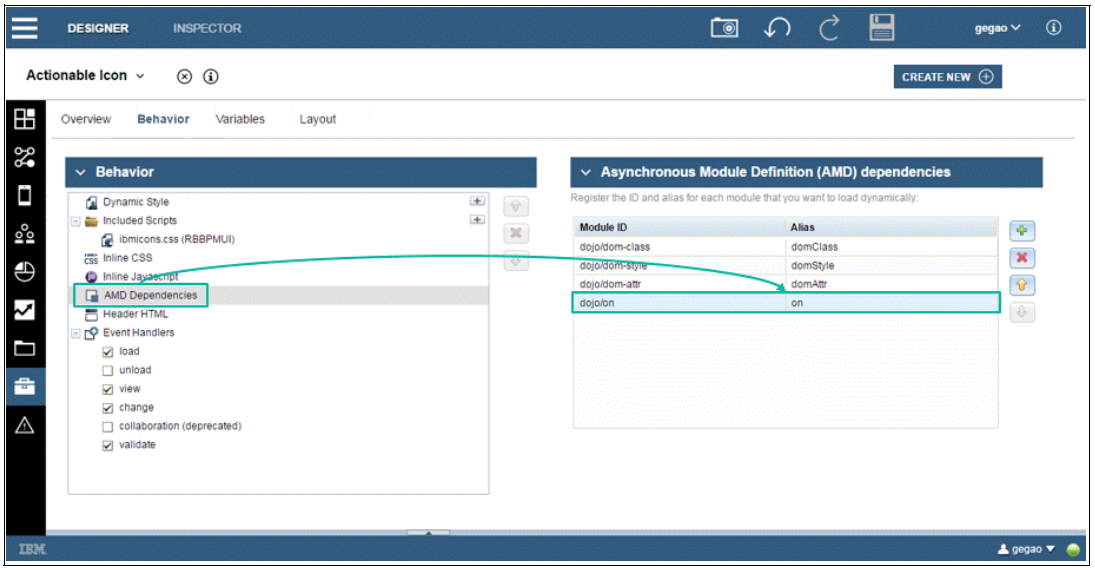


Figure 3-30 Adding an AMD dependency on dojo/on with the on alias

Copy the code snippet in Example 3-20 to the end of the code in the `validate()` event handler to handle the validation event type clear. This clears the displayed validation error message and restores the display with the corrected value.

Example 3-20 Code snippet to clear the displayed validation error

```
else if (event.type == "clear") {
   domClass.remove(this.iconNode, "ibm-close-cancel-error");
   domStyle.set(this.validationErrorContainerNode, "visibility", "hidden");
   var validationErrorNode =
this.validationErrorContainerNode.querySelector("div.validationErrorMessage");
   validationErrorNode.innerHTML = "";
   this.view();
}
```

In the code, the icon's DOM node first removes the validation error icon `ibm-close-cancel-error`. Then the validation error container node is hidden and the error message is cleared.

You can now run the testing client-side human services to see how the validation error message is displayed in the Actionable Icon coach view to indicate an error if the class name is invalid. Figure 3-31 on page 107 shows that when you change the class name to an allowed name, it is rendered as expected.

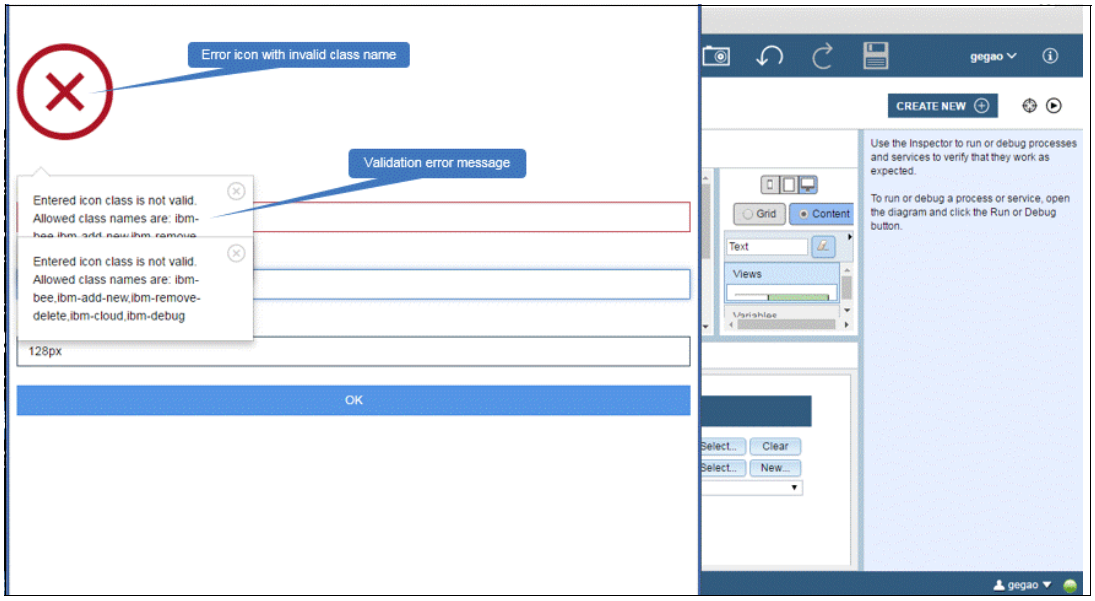


Figure 3-31 Rendering effect of a validation error in Actionable Icon

The unload() event handler

`The unload()` event handler is called in the last stage of a coach view, and is only called one time, when the browser page is closed or the coach view is deleted. When the coach view is deleted, Coach Framework recursively destroys the DOM nodes that contain widgets of that coach view. Typically, you do not have to programmatically control the destroy. If there are global listeners remaining, however, they are destroyed in the `unload()` event handler.

For more information about the `unload()` event handler, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewUnloadEventHandler`

Tip: When using a global `_this` or `self` variable as cache defined in Inline JavaScript to refer to the coach view object, a preferred practice is to null the variable in the `unload()` event handler to gain more resources. For example:

```
_this = null;
```

Similarly, handles returned by the `bind()` or `bindAll()` functions are also released. For example:

```
handle.unbind();
```

Boundary event

You have now built a coach view with bound data to communicate with other controls within a coach or to validate the data within the Data Change section of a coach view. In many cases, a coach must be wired to other steps in the human service, for example, a script or a service call. The wiring is triggered in runtime by an event to the next step. Because this event reaches outside of the coach boundary, it is called a *boundary event*.

Use boundary events for the following:

- Navigation to the next step in the human service flow
- Validation by a script or a service call
- Fetching more or new data from the server

Figure 3-32 shows how to declare boundary events using the following steps:

1. Select the Can Fire Boundary Event box in the Overview tab in the coach view editor. By declaring this event, a coach author can select the declared coach view as the source of a boundary event in the human service editor.
2. Drag another instance of the Actionable Icon to the coach canvas and name the label Reset Icon.

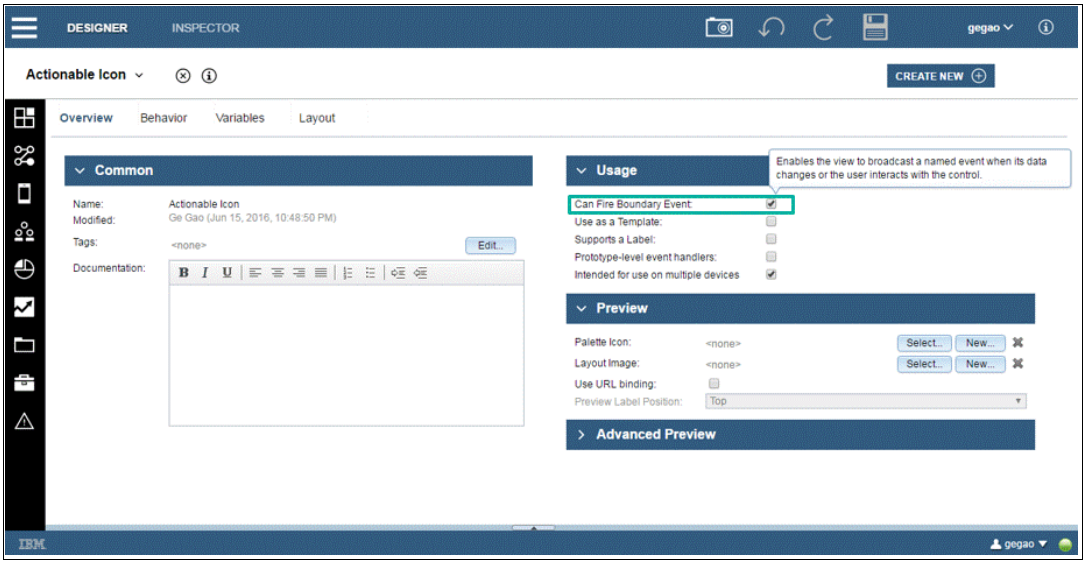


Figure 3-32 Enabling the firing boundary event of a coach view

3. Create a private human service variable called `resetIconClass` and set it to `ibm-reset-revert` in the Init script of the testing human service.

   Figure 3-33 shows how this variable is then bound as the binding object of the new Actionable Icon control.

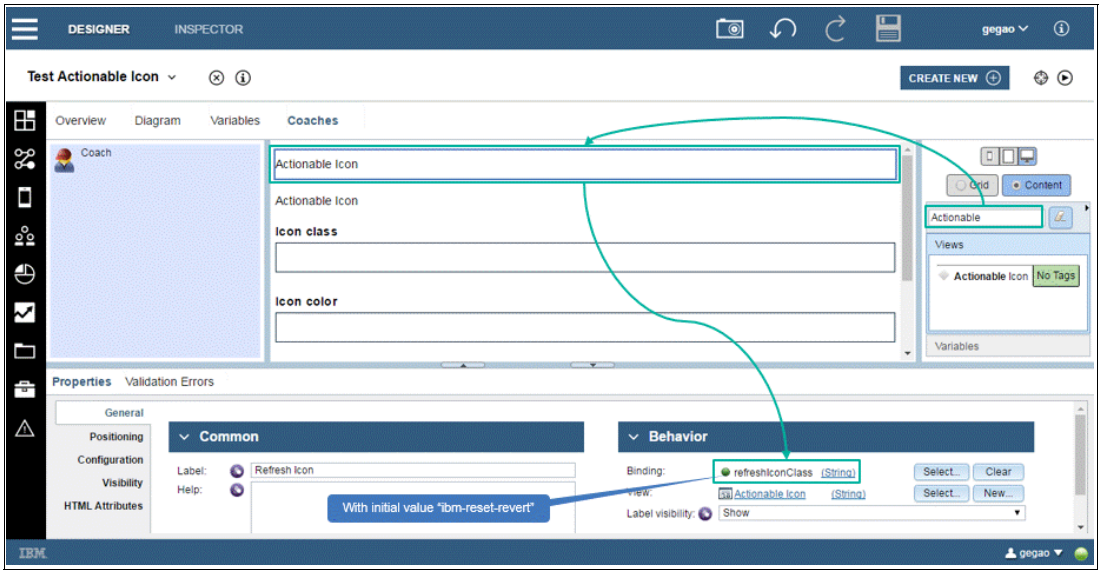


Figure 3-33 Adding the reset icon to the coach

Because some icons are used only for display (for example, displaying a status) and some icons are used to trigger actions (for example, the reset icon), you must set the

`canFireBoundaryEvent` configuration option of `Boolean` type in the Actionable Icon coach view. By default, this is set to false. For the newly created reset icon, set to true by selecting the check box.

4. Drag a client script to the coach canvas and name it *Reset data*.
5. Copy the code snippet in Example 3-21 into the script.

Example 3-21 Code snippet to reset the Actionable Icon default value

```
tw.local.iconClass = "ibm-bee";
tw.local.iconColor = "#5596e6";
tw.local.iconSize = "128px";
```

6. Drag a wire from the coach to the client script. The wire is named *Reset Icon*, which is derived from the label of the coach view instance, and can fire a boundary event.
7. If the automatically set boundary event is not the expected one, click **Select** next to the *End state binding* name to choose another one.

   After the script has finished resetting the service data, the server stays on the page so that coach views are updated with new data. All `change()` event handlers of the affected coach views are called. Figure 3-34 shows the finished human service diagram with the boundary event for refresh enabled.

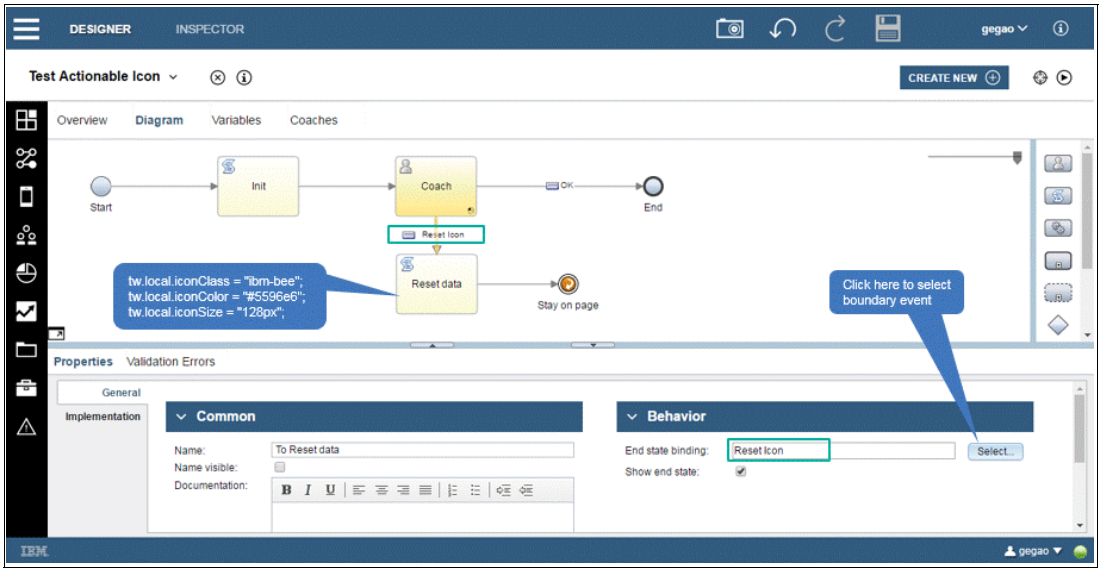


Figure 3-34 Wiring the coach to reset the script and stay on the page afterwards

If you run the coach, you can see that the reset icon is displayed on the top of the coach. Clicking the icon, however, has no effect on the coach. This is because you have not yet implemented the behavior on the DOM `click` event of Actionable Icon view. To fire a boundary event in coach view, use the `context.trigger(/*Function*/callback, /*Object*/options)` Coach Framework API, where both parameters `callback` and `options` are optional. A common use of the callback function is to restore the displayed style of the coach view when it is set to another style after the boundary event is fired. For example, the boundary event is greyed out after the boundary event is fired; you can set it back to the previous style in the callback function when the boundary event response is returned.

For more information about the `context.trigger()` API, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewContextObject`

You must copy the code in Example 3-22 to the end of the `view()` event handler. Check in the code snippet if the coach view instance is set to fire a boundary event. If it is not, the click action on the icon is ignored and no boundary event will be fired. If the coach view instance is set to fire a boundary event, add the HTML attribute `href` with a dummy URL to the anchor node. This allows the user to see a mouse pointer indicating this icon can be clicked when the mouse is hovered over it.

As noted in 3.2.3, “Programming the coach view event handler to control behavior” on page 88, you use Dojo `dojo/on` API to register a listener on the click event of the icon. This requires a local variable *_this* to be referenced in the closure. When the icon is clicked, the icon is disabled and its color is set to grey to avoid unexpected multiple clicks. Then the boundary event is triggered by calling the `context.trigger()` API. When the boundary event response of comes back, the icon is enabled again and the previous color is restored.

In the code, you read the current icon color configuration option through the API again instead of the previously stored value in local variable. This is because the callback function is not sequentially executed. As noted in 3.2.1, “Defining and accessing data in the coach view” on page 75, the coach view developer is not aware if there is change on the configuration option before the boundary event response comes back. It is recommended to avoid reading the local cache because the cached data might be stale. Copy the code in Example 3-22 to the bottom of the `view()` event handler.

Example 3-22 Code snippet to fire the boundary event if enabled

```
// enable click to fire boundary event
var canFireBoundaryEvent = (!! this.context.options.canFireBoundaryEvent) ?
this.context.options.canFireBoundaryEvent.get("value") : false;
if (canFireBoundaryEvent == true) {
   if (domAttr.has(this.iconNode, "href") == false) {
      domAttr.set(this.iconNode, "href", "#");
      var _this = this;
      on(this.iconNode, "click", function (evt) {
      debugger;
         _this.disabled = true;
         domStyle.set(_this.iconNode, "color", "#aeaeae");
         _this.context.trigger(function () {
            _this.disabled = false;
            var iconColor = (!! _this.context.options.iconColor) ?
_this.context.options.iconColor.get("value") : "#7cc7ff";
            domStyle.set(_this.iconNode, "color", iconColor);
         });
      });
   }
}
```

When you run the testing human service and change the icon class name to some other name, such as ibm-cloud, or change the color or size to some other value, you can click the reset icon on top to revert all your changes. Figure 3-35 on page 111 shows how to do this.

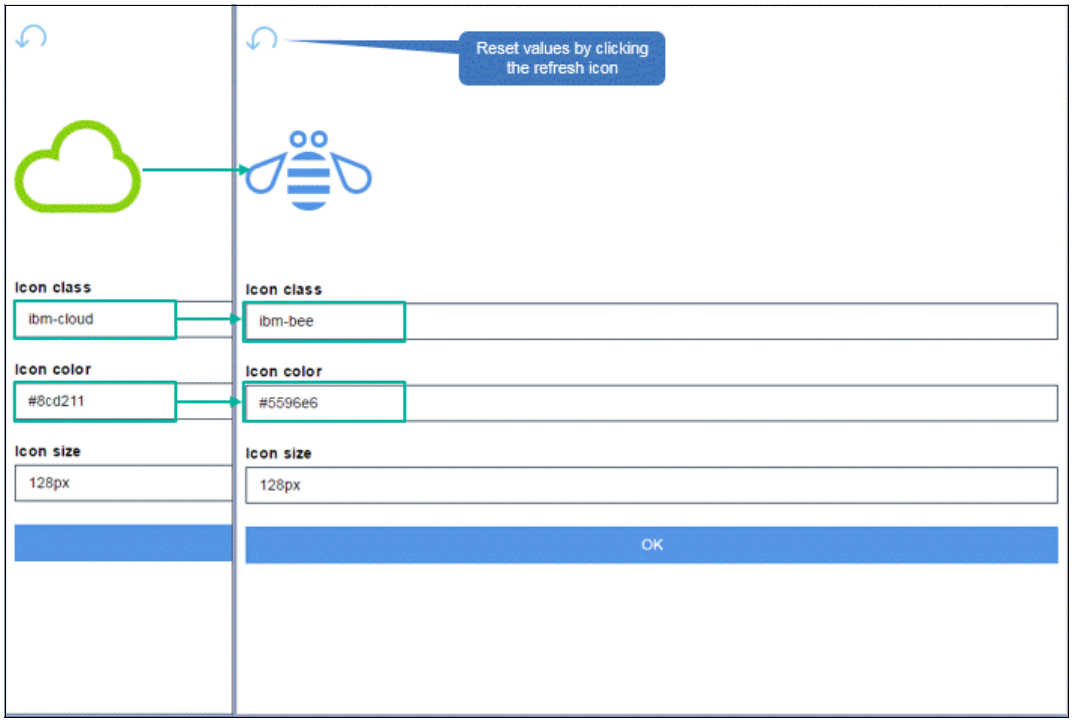


Figure 3-35 Effect of firing boundary event by clicking a refresh icon

3.2.4 Test coach view in a human service

The basic functions of the Actionable Icon coach view are implemented. You have created a simple client-side human service to test the coach view in a coach. It is a preferred practice to develop a coach view in a unit that tests a human service for the following reasons:

- This service can be used to develop a coach view in isolation so you are not working in the coach directly. This isolation allows you to reduce the side effects of other controls on a coach.
- When you complete development, you can use the human service as a kind of unit test, allowing you to debug and to reproduce errors. This can save time, especially for complex coach views with many parameters, because you do not have to create a unit test first before you start debugging.
- The existing sample human service helps other developers working on the coach view. In this case, the human service serves as documentation.

Your test human service must contain some of the following items:

- A service to perform basic initialization. If you use complex objects, you can create a service for each complex object. If you must initialize multiple complex objects, you can call the initialization services. Using services enables you to reuse initialization logic, which improves maintainability.
- A client script to set defaults.
- A first coach where you can manually adjust parameters of the coach view.
- The main coach.

- Optionally, a client script for final actions, such as nulling out complex object instances to reduce general memory usage. Figure 3-36 shows this client script.

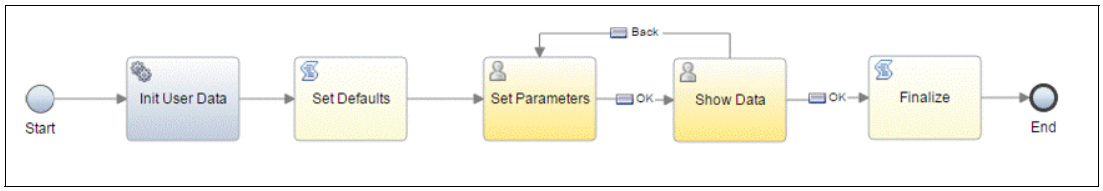


Figure 3-36 A typical testing human service for coach view unit test

Figure 3-37 shows that you can debug the code in the coach view event handlers by running the test service in debug mode. If you click the debug icon, a browser window opens to enable the developer to step through the service. The coach view developer can set breakpoints in the source code of event handlers. However, unlike coaches in heritage human services, opening coaches in client-side human services requires extra steps to make the source code of a coach view available in a browser debugger.

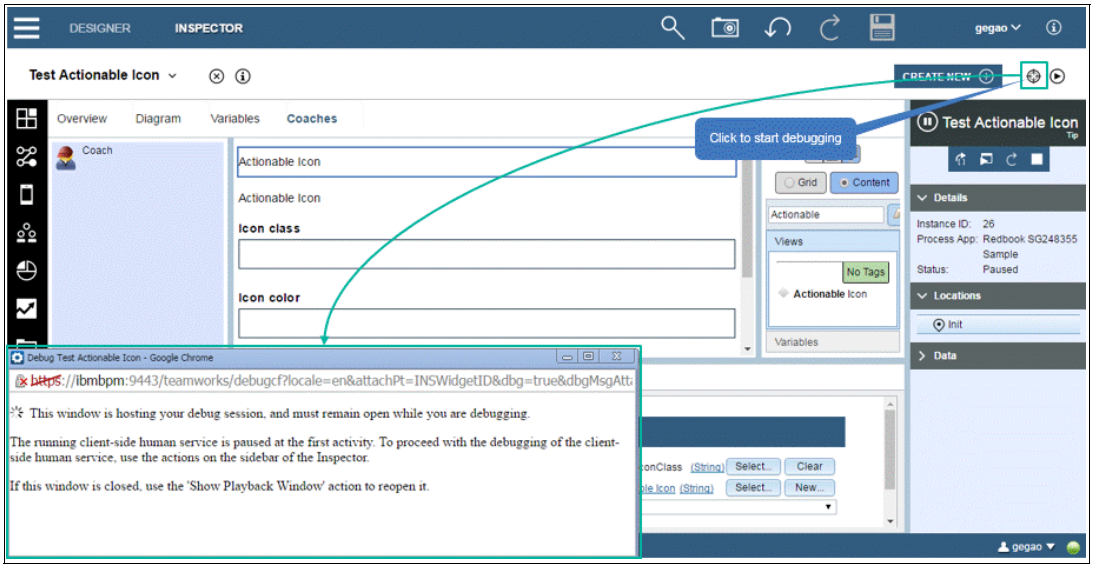


Figure 3-37 Start debugging coach view event handlers by clicking debug icon

Start the browser debugger by clicking the F12 key. After the browser debugger windows open, click the step over icon on the right side of the Process Designer. Figure 3-38 on page 113 shows these steps.

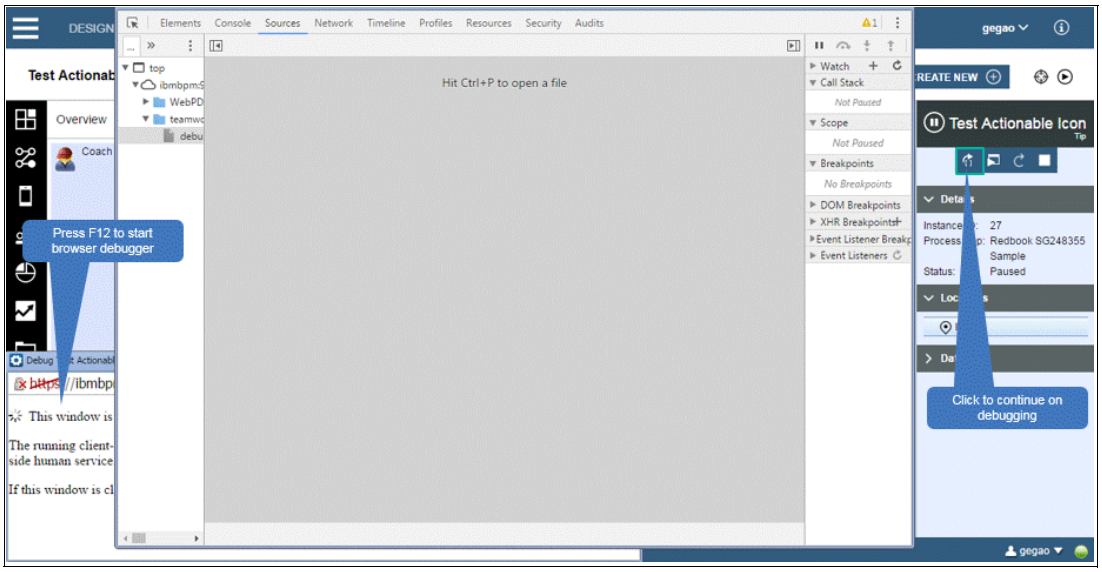


Figure 3-38 Continue client-side human services debugging by clicking step over icon

Click the step into icon shown in Figure 3-39.

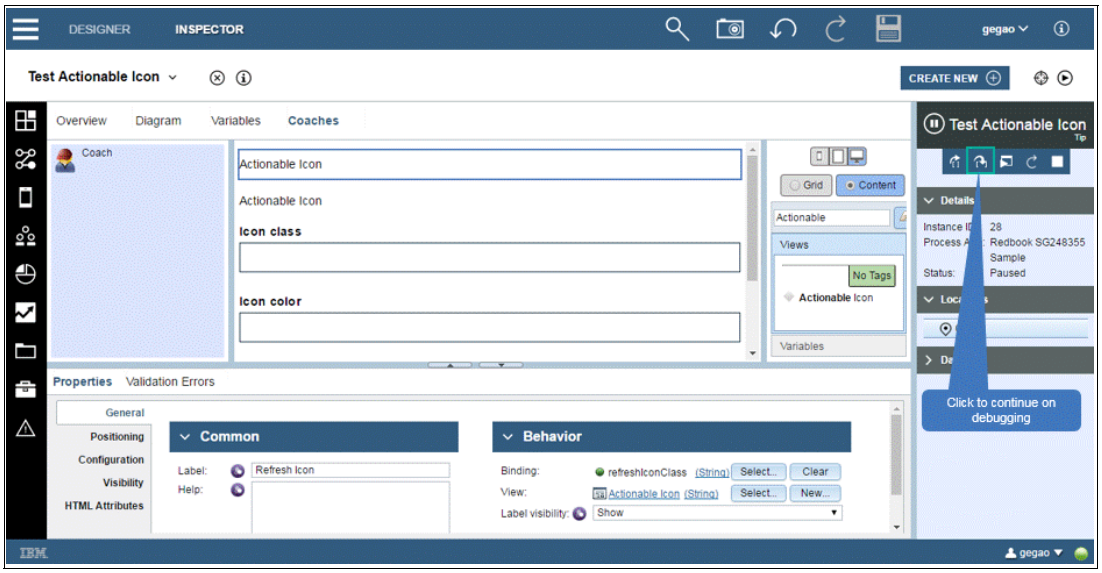


Figure 3-39 Continue client-side human services debugging by clicking step into icon

Figure 3-40 shows that the browser debugger window now opens to the front end and stops at a default breakpoint set by Coach Framework.

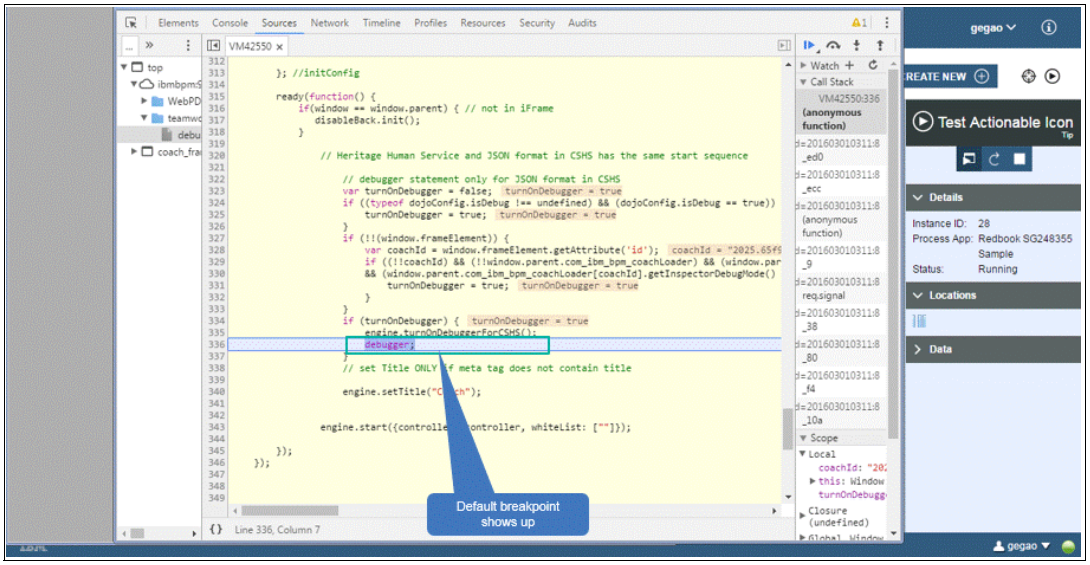


Figure 3-40 Generated code stops at system default breakpoint

Figure 3-41 shows how you can find your code by searching the coach view name, with whitespace replaced by an underscore. In the code, you can set a breakpoint in the event handlers code as required. These breakpoints are temporarily set and are removed in your next debugging task. To keep the breakpoint, you can set a manual breakpoint by including `debugger;` in the event handlers.

When the coach view is ready for use by the coach author, remove the manual breakpoints to avoid unexpected stops when other developers debug their coaches containing your coach view.

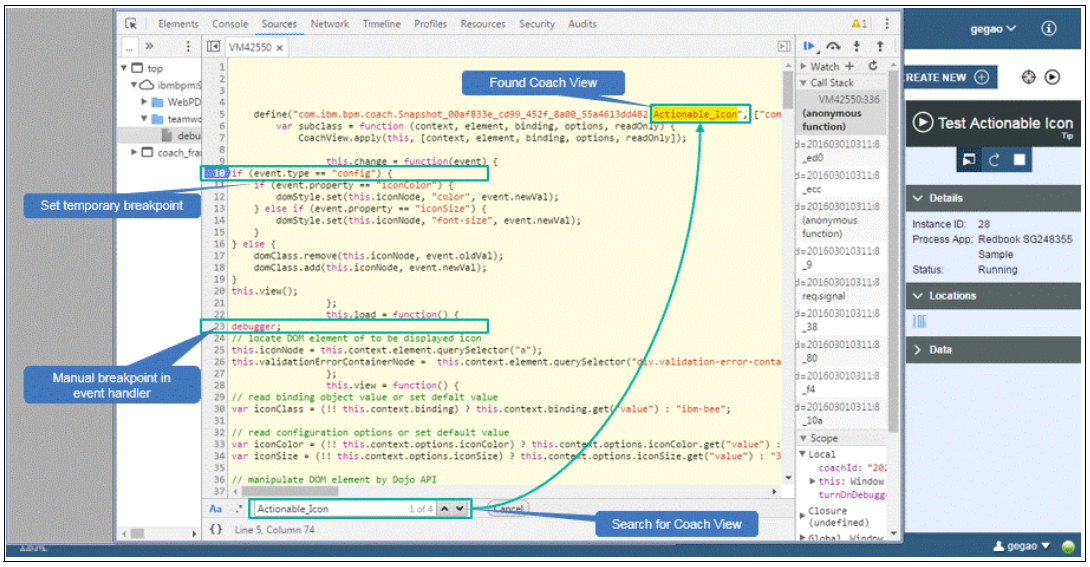


Figure 3-41 Find coach view and its event handlers by searching the coach view name

For more information about debugging event handlers in client-side human service, see IBM Knowledge Center:

http://ibm.biz/BPMDebugCoachViewLifecycleInCSHS

For more information about debugging event handlers in Process Portal, see IBM Knowledge Center:

http://ibm.biz/BPMDebugCoachViewsInProcessPortal

3.3 Developing advanced functions of a coach view

Section 3.2, “Developing coach view basic functions” on page 75introduces the fundamental concepts of the coach view, including its data, view layout, and view controller (event handlers) to program its behavior in runtime.

This section includes additional concepts of coach view development:

- How to manipulate the coach view data by communicating with the backend system
- How to control view layout attributes, such as label, visibility, bidirectional text, and accessibility
- How to embed other controls in a coach view
- How to better support the coach author by providing previews and tags

3.3.1 Communicating with backend data

The coach is rendered on the client side. Its initial data is read from a human service containing the coach. If the user needs to retrieve new data or send data back to the server within the coach, there are three options to use:

- Add an Ajax service as a configuration option of the coach view
- Call an Ajax service or call a `BPM REST` API programmatically by using a JavaScript API
- Call a service by firing a boundary event and return to the coach after the service call

IBM BPM provides Ajax service-type services. In coach views, you can define an Ajax service as a configuration option. A set of the stock controls, such as the `Single Select` control, supports the use of Ajax services to load content asynchronously.

Figure 3-42 on page 116 shows and example of how you can create a custom coach view that performs an Ajax call by using the Ajax service. You can use this method to gather business data asynchronously instead of loading it up-front on the server side; this can make coaches more responsive.

When data is loaded before the coach in a human service, this action delays the loading of the coach. When data is loaded asynchronously, you can load data as the page is being rendered on the client side. Because complex UIs might require processor-intensive work on the client side because of heavy JavaScript use, start the rendering as soon as possible and load data.

Note: Most browsers have a limit for concurrent Ajax connections. Be aware of these limits and avoid making excessive Ajax calls.

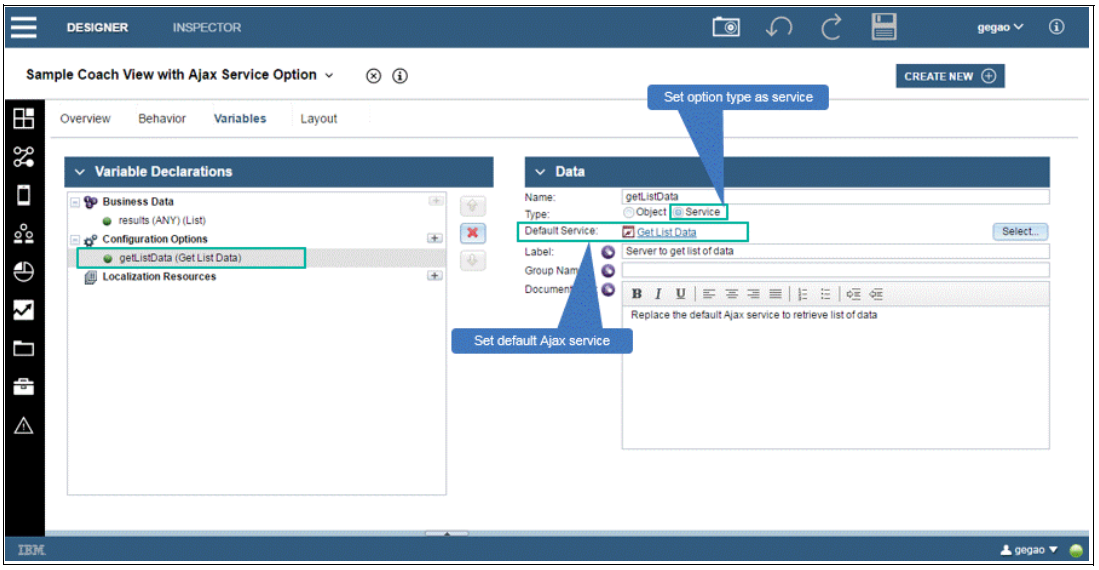


Figure 3-42 Sample coach view with Ajax service as a configuration option

Figure 3-43 shows the Ajax service used in this example, which is also built using Process Designer.

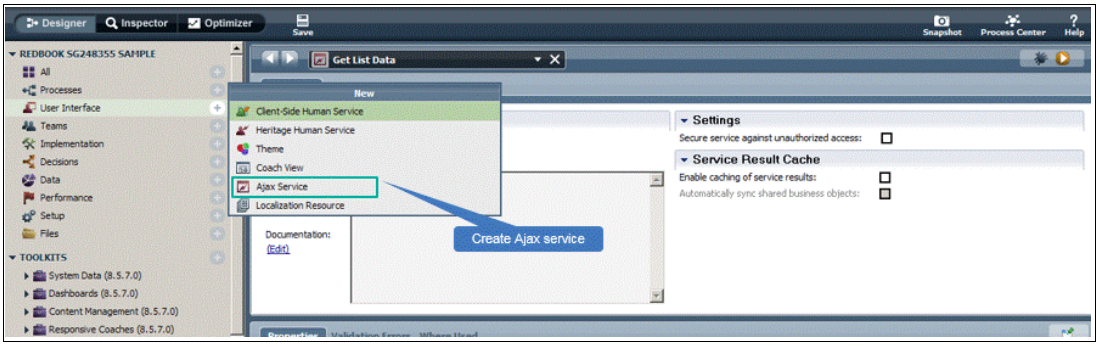


Figure 3-43 Create Ajax service in Process Designer

Instead of using the Ajax service construct directly, you can also call it through the IBM `BPM Representational State Transfer (REST)` API in taskless mode. The `context.options.<service_option_name>(args)` Coach Framework API is used to trigger the service. The Ajax service to be called defines one input parameter with type of `String` or a `JSON` serializable object. Before the API is called, you must serialize the input parameter by using `JSON.stringify()` if the input is a complex object.

For more information about this topic, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewsCallingAjaxServices`

Ajax service as a configuration option: To refer to an Ajax service that is defined as a configuration option, you can extract the URL for the service using the following API call:

```
this.context.options.myOptionService.url
```

This method is useful for more control over how the call is executed. This API wraps the underlying Dojo `dojo/request/xhr` XMLHttpRequest (XHR) API, which allows you, for example, to define a custom timeout for a call to a `REST` API.

More information about the dojo/request/xhr API, see the dojo/request/xhr website:

https://dojotoolkit.org/reference-guide/1.10/dojo/request/xhr.html

Example 3-23 shows a sample that uses the Dojo dojo/request/xhr API to call the REST service. In a coach view, start a human service using the code in Example 3-23.

Example 3-23 Code snippet to call the IBM BPM REST API by using dojo/request/xh

```
var xhrArgs = function(config) { return {
url: "/rest/bpm/wle/v1/service/RBO1@getAddressData", content: {
accept: "application/json",
params: JSON.stringify(config.inputs),
createTask: false,
parts: "all",
action: "start"
},
handleAs: "json"
} };
deferred = dojo.xhrPost(xhrArgs(
{ inputs: { myData: { name: myData.get("name") } } }
));
```

For more information about the IBM BPM REST API call to start a service, see IBM Knowledge Center:

http://ibm.biz/BPMRESTAPIStartService

Preferably, the URL is not hardcoded because, for example, the context root of the REST API /rest part of the IBM BPM REST URL might vary by environment. A preferred practice is to use the context.contextRootMap.rest coach API to get the context root of the BPM REST API of the runtime environment.

3.3.2 Additional view layout attributes of the coach view

There are a few general properties that must be considered by the coach view developer when providing a custom coach view. These properties are configured in design time in Process Designer and are generated as metadata by Coach Framework. The context.options._metadata API contains these metadata:

- Label of the control is stored in the metadata property label.
- Visibility of the control can be set as SHOW or HIDE and stored in the metadata property labelVisibility.
- Help text can be set by the coach author to show a tooltip to help the user finish the task completion UI. This information is stored in the metadata property helpText.
- Visibility of a control can be set by the coach author by Value, Rule, or Script. The calculated value in runtime is stored in the metadata property visibility.
- Base text direction setting is stored in the metadata property baseTextDirection.

There are two additional features to use when developing the coach view:

- Dynamic LESS style file
- Content box advanced item, which embeds other coach views

Label

The label of a UI control helps users identify the control in a web page and understand the meaning of the control. Coach Framework supports label with APIs. If your custom coach view has a label, you must add support for it to ensure that your coach view behaves as other stock coach views during runtime (for example, when the custom coach view is included inside the Table and Tab coach view), and at authoring time in Process Designer.

Same as binding object and configuration options, coach view support of a label also requires data input of the label text, the DOM node in view layout, and an event handler to control the display and visibility of label text.

As noted, the label text, visibility, and tooltip are stored in `context.options._metadata` as properties. They can be read in the similar form of a configuration option, as follows:

```
this.context.options._metadata.label.get("value")
```

To make the sample Actionable Icon support label, first define its view layout. Copy the code snippet in Example 3-24 to the custom HTML item directly before the anchor node.

Example 3-24 Code snippet to define DOM node for label in custom HTML item

```
<div class="BPM_Resp_Icon">
   <label class="control-label bpm-label-default"></label>
</div>
```

Tip: When you use the responsive coaches toolkit to build coaches, your custom coach view has a consistent style with the stock controls. To match stock control CSS styling, add a helper class with a prefix `BPM_Resp_` at the outer `<div>` tag. To make it fully aligned with stock control styles, observe the generated HTML code of the stock controls.

In the Actionable Icon coach view `load()` event handler, locate the DOM node of this label; it can then be manipulated in later event handlers. Copy the line in Example 3-25 at the end of current `load()` event handler.

Example 3-25 Code snippet to locate DOM node of label in `load()` *event handler*

```
this.labelNode = this.context.element.querySelector("label.control-label");
```

In the `view()` event handler, the label text is set to the DOM node. However, the coach view developer must also develop the label visibility correctly. When a custom coach view is included in the Table or Tab coach view, the label must appear only one time. In Figure 3-44 on page 119, the Actionable Icon coach view on the right shows the label only in the table header, and a poorly constructed custom coach view on the left shows the label inside the table cells repeatedly.

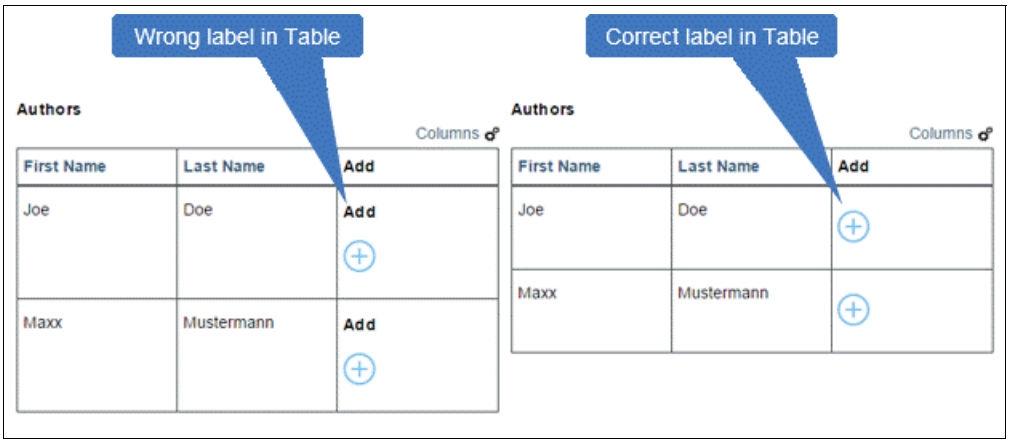


Figure 3-44 Comparison of label rendering in Table

To enable the label "hiding" in the container type of coach views (such as Table control) in the `view()` handler of a custom coach view, you must add code that is similar to the code shown in Example 3-26. For our sample Actionable Icon, you can copy the code to the `load()` event handler and place it before the firing boundary event code.

Example 3-26 Code snippet to render label with visibility control in the view() event handler

```
// control label
if (this.context.options._metadata.label === undefined
   || this.context.options._metadata.label.get("value") == ""
   || (this.context.options._metadata.labelVisibility !== undefined
      && this.context.options._metadata.labelVisibility.get("value") == "HIDE")) {
   // hide the label node
   this.context.setDisplay(false, this.labelNode);
} else {
   // display label text
   this.labelNode.innerHTML =
this.context.htmlEscape(this.context.options._metadata.label.get("value"));
   if (!!this.context.options._metadata.helpText) {
      // add tooltip
      this.labelNode.title = this.context.options._metadata.helpText.get("value");
   }
   this.context.setDisplay(true, this.labelNode);
}
```

Visibility

When you create a coach view, the Visibility property is automatically made available. In the Visibility page, the users of your coach view can set the runtime visibility properties in the coach by using the Value, Rule, or Script options shown in Figure 3-45 on page 120.

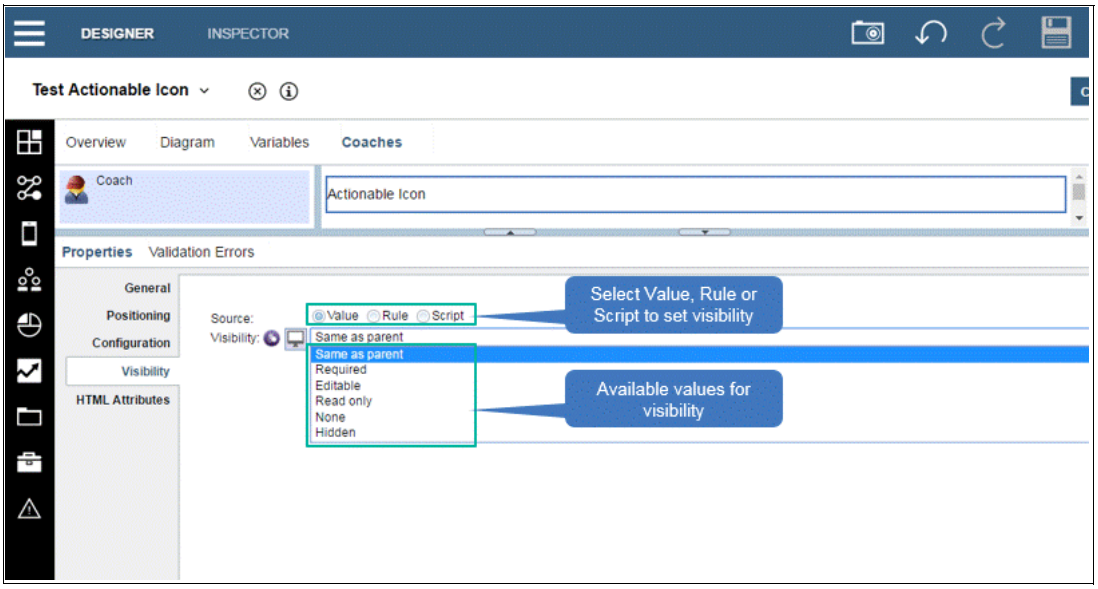


Figure 3-45 Select a value for visibility of a coach view in coach

These visibility options are available only for coach view instances in a coach and are not available in views. Figure 3-46 shows that coach views have only the “value” visibility style.

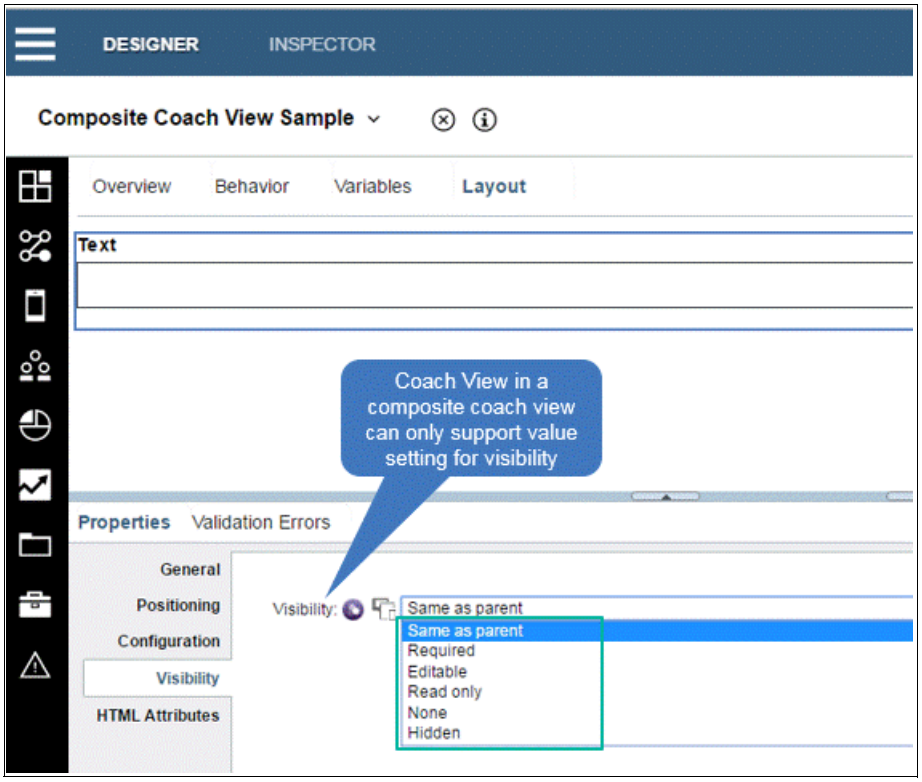


Figure 3-46 Visibility setting of coach view in composite coach view supports only value setting

A custom coach view must implement these visibility settings. If you change the Actionable Icon visibility value to *Hidden*, the control still appears in the testing human service at runtime because it is not yet implemented.

Internally, the visibility settings are DEFAULT (the code equivalent of “Same as parent”), EDITABLE, REQUIRED, READONLY, NONE, and HIDDEN. When you create a custom coach view, you must provide support for the visibility settings in the `view()` handler. The visibility logic must be implemented in the `view()` handler. It is not necessary to implement the `change()` handler. If the `change()` handler is not implemented, the `view()` handler is called if a change occurs in the bindings of the coach view.

Before you implement the sample Actionable Icon visibility, you must add one AMD module from the coaches toolkit, defined in the `utilities.js` in the `coach_ng_utilities.zip` file. This AMD module provides useful functions to handle visibility, bidirectional text, and localization.

Tip: Copy the `coach_ng_utilities.zip` file from the coaches toolkit to your own toolkit or process application for future maintenance.

To copy the `coach_ng_utilities.zip` file, select the Files category in the coaches toolkit and filter out the file. Then select **Copy item to** in order to copy the file to the target process application or toolkit. Figure 3-47 shows this process.

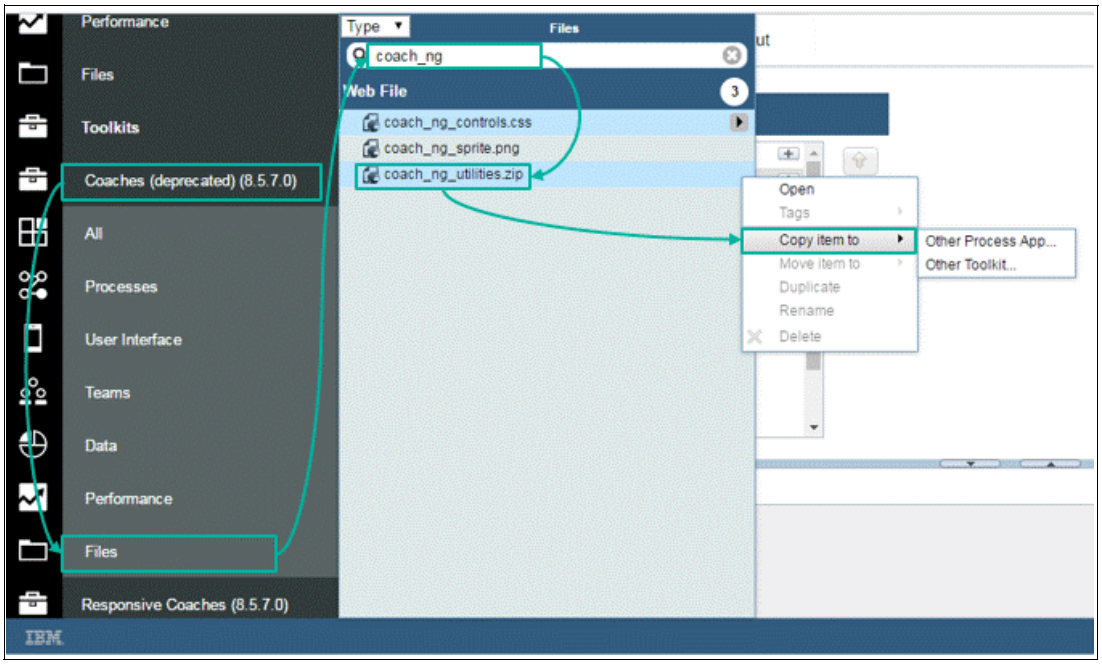


Figure 3-47 Copying the coach_ng_utilities.zip file from coaches toolkit to your toolkit or process app

The `coach_ng_utilities.zip` file is a custom Dojo build that contains AMD modules. Before adding the AMD module, you must add the code in Example 3-27 to the Inline JavaScript of the Actionable Icon sample.

Example 3-27 Code snippet to load AMD module from custom Dojo build

```
/*
@dojoConfigUpdateStart
   dojoConfig.packages.push({
      name: 'com_ibm_bpm_coach_controls',
      location: com_ibm_bpm_coach.getManagedAssetUrl('coach_ng_utilities.zip',
            com_ibm_bpm_coach.assetType_WEB, 'RBBPMUI') +
"/com_ibm_bpm_coach_controls"
   });
```

```
   dojoConfig.packages.push({
      name: 'com.ibm.bpm.coach.controls',
      location: com_ibm_bpm_coach.getManagedAssetUrl('coach_ng_utilities.zip',
           com_ibm_bpm_coach.assetType_WEB, 'RBBPMUI') +
"/com/ibm/bpm/coach/controls"
   });
@dojoConfigUpdateEnd
@layerRequiredStart
{
  "nonDebug": ["dijit/dijit_bpm",
               "com_ibm_bpm_coach_controls/controls"]
}
@layerRequiredEnd
*/
```

Note: You must update the acronym parameter of the `getManagedAssetUrl()` to the process application or toolkit, where the `coach_ng_utilities.zip` file is located. You must change this to `'RBBPMUI'`.

For documentation about loading AMD module files from a custom build, see IBM Knowledge Center:

http://ibm.biz/BPMCoachViewPerformance

Figure 3-48 shows how to add the `com.ibm.bpm.coach.controls/utilities` module to the Actionable Icon coach view as an AMD dependency.

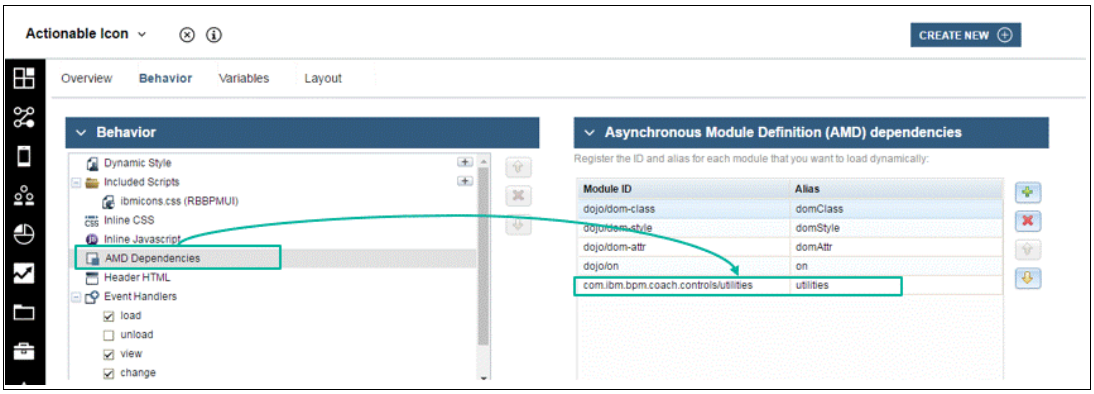


Figure 3-48 Add utilities as AMD module

The utilities module can be used in event-handler JavaScript code in the same way as other AMD modules are used. The `handleVisibility()` function can be used to return the calculated visibility value of the coach view and set specified DOM node or widgets with the expected visibility value. You can see in the code in Example 3-28 on page 123 that the object context is passed to the `handleVisibility()` function, with the second parameter as the icon DOM node in an array. This function also considers inheritance of visibility from the parent control. The visibility and disabled attributes are then set by the function according to the calculated result. Additional visibility processing can be then handled in your custom code in the `view()` event handler. For example, if the visibility is read-only, set the icon to be greyed out and show the mouse cursor as not-allowed. If the visibility is of the value EDITABLE or REQUIRED, the icon must be displayed with the specified color and a normal cursor. Copy the code snippet in Example 3-28 on page 123 directly before the firing boundary event code.

Example 3-28 Code snippet to render label with visibility control in `view()` *event handler*

```
// handle visibility
var visibility = utilities.handleVisibility(this.context, [this.iconNode]);
if (visibility == "READONLY" && this.iconNode) {
   // grey out icon node and set mouse pointer as not allowed
   domStyle.set(this.iconNode, "cursor", "not-allowed");
   domStyle.set(this.iconNode, "color", "#aeaeae");
} else if (visibility != "NONE" && visibility != "HIDDEN" && this.iconNode) {
   domStyle.set(this.iconNode, "cursor", "");
   domStyle.set(this.iconNode, "color", iconColor);
}
```

To limit an actionable icon to the fire boundary event only if its visibility is not set to be read-only, replace the code in Example 3-29 with code in Example 3-30.

Example 3-29 Code snippet of firing boundary event without dependency on visibility

```
if (canFireBoundaryEvent == true) {
```

Example 3-30 Code snippet of firing boundary event with dependency on visibility

```
if (canFireBoundaryEvent == true && visibility != "READONLY") {
```

Figure 3-49 shows the view when you set the visibility of the reset icon in the testing human service to read-only.

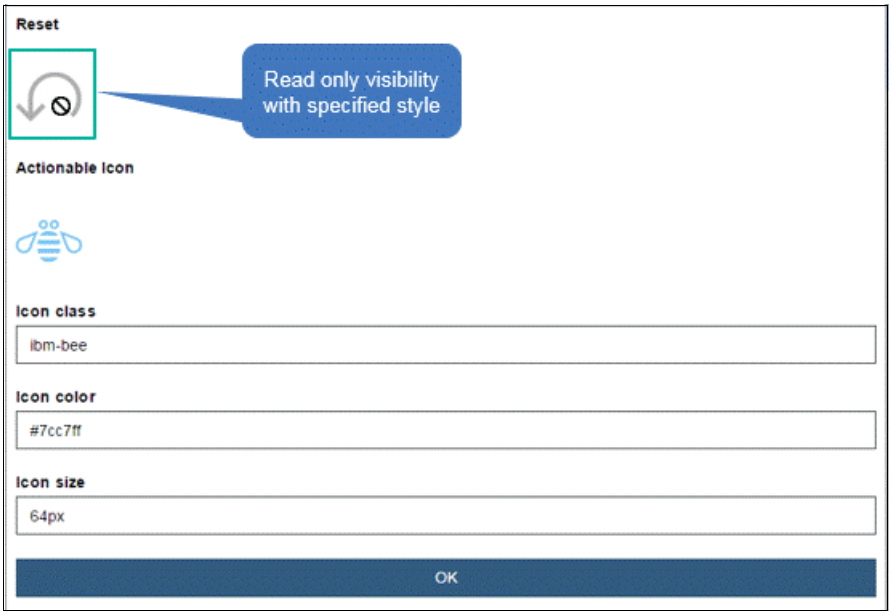


Figure 3-49 Effect of setting visibility to read-only

Base Text Direction setting

IBM BPM can support languages that are written from left to right and languages that are written from right to left. In Process Designer, base text direction support is turned off by default. Figure 3-50 on page 124 shows how it is enabled by selecting **File** → **Preferences** → **Capabilities** → **Base Text Direction**.

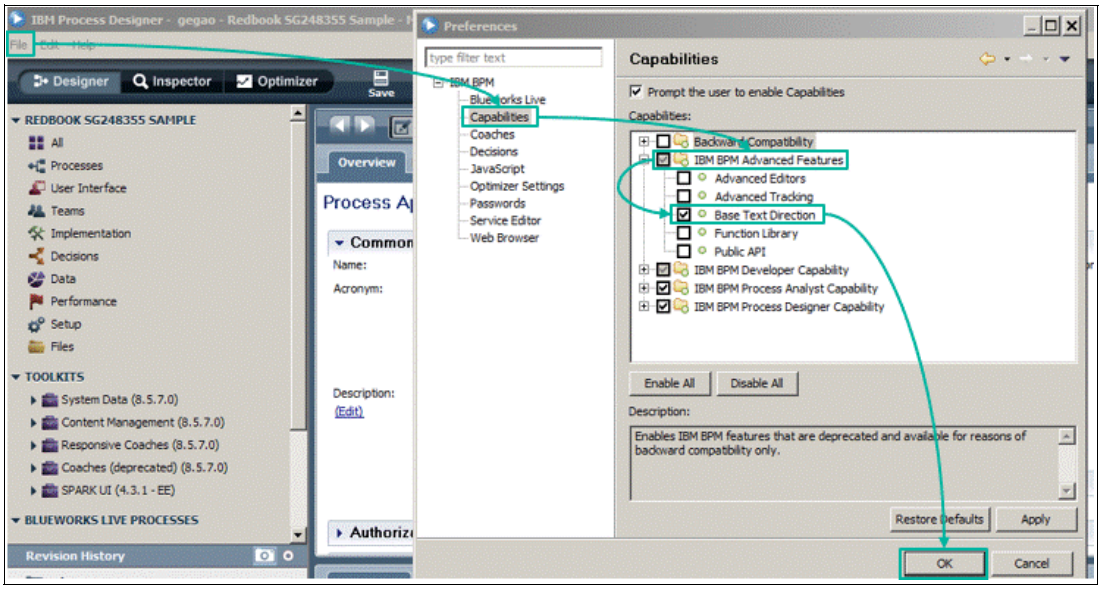


Figure 3-50 Enable Base Text Direction support in IBM Process Designer

Figure 3-51 shows that after base text direction is enabled the base text direction settings are displayed in the General tab in the Behavior window of a heritage human service.

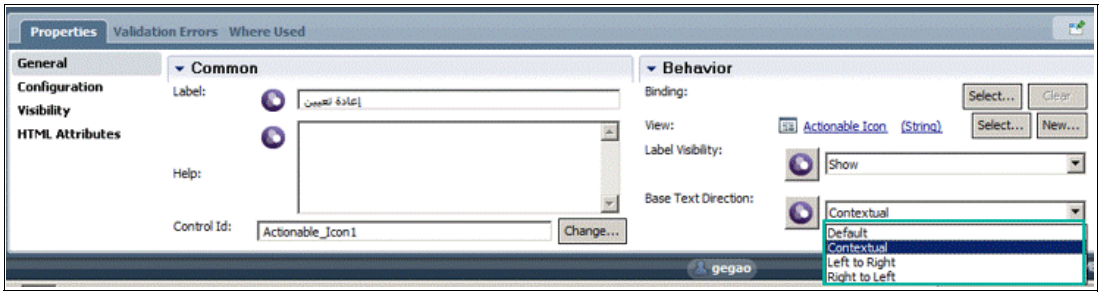


Figure 3-51 Set the base text direction for a heritage human service in IBM Process Designer

Table 3-6 lists available values of this setting and their descriptions.

Table 3-6 Available values of the base text direction setting and their descriptions

Value	Description
Default	Inherits the text direction that is set in the user's profile.
Contextual	Displays the text according to the first strong directional character in the string. For example, if the first strong directional character is from a right-to-left language, the text is displayed from right to left. This setting applies to all text elements that a coach view shows. For example, a Text stock control has an Arabic label, but its contents are English. In this case, the text in the label is right to left and the text in the field is left to right.
Left to Right	Displays the text from left to right no matter what characters are in the text.
Right to Left	Displays the text from right to left no matter what characters are in the text.

All Stock coach views provide base text direction support. To enable base text support in your custom coach view, you must retrieve the value of the base text direction attribute and write code to apply the direction in the view. As with visibility, the utilities module provides useful functions and constants to check this setting and apply the setting to the displayed text. You can append the code shown in Example 3-31 after the visibility handling code in the `view()` event handler of the Actionable Icon sample.

Example 3-31 Code snippet to render label with visibility control in view() event handler

```
// handle bidirectional label
var baseTextDirection = this.context.options._metadata.baseTextDirection &&
this.context.options._metadata.baseTextDirection.get("value") ?
this.context.options._metadata.baseTextDirection.get("value") :
utilities.BiDi.BASE_TEXT_DIRECTION._default;
var generalPrefTextDirection = this.context.bpm.system.baseTextDirection;
utilities.BiDi.applyTextDir(this.labelNode, this.labelNode.innerHTML,
baseTextDirection, generalPrefTextDirection);
```

You can test the base text direction by putting two Actionable Icon controls together with labels in English and Arabic, respectively. Figure 3-52 shows that, by setting the Arabic label with the Contextual option, the direction attribute of the DOM node label is automatically detected.

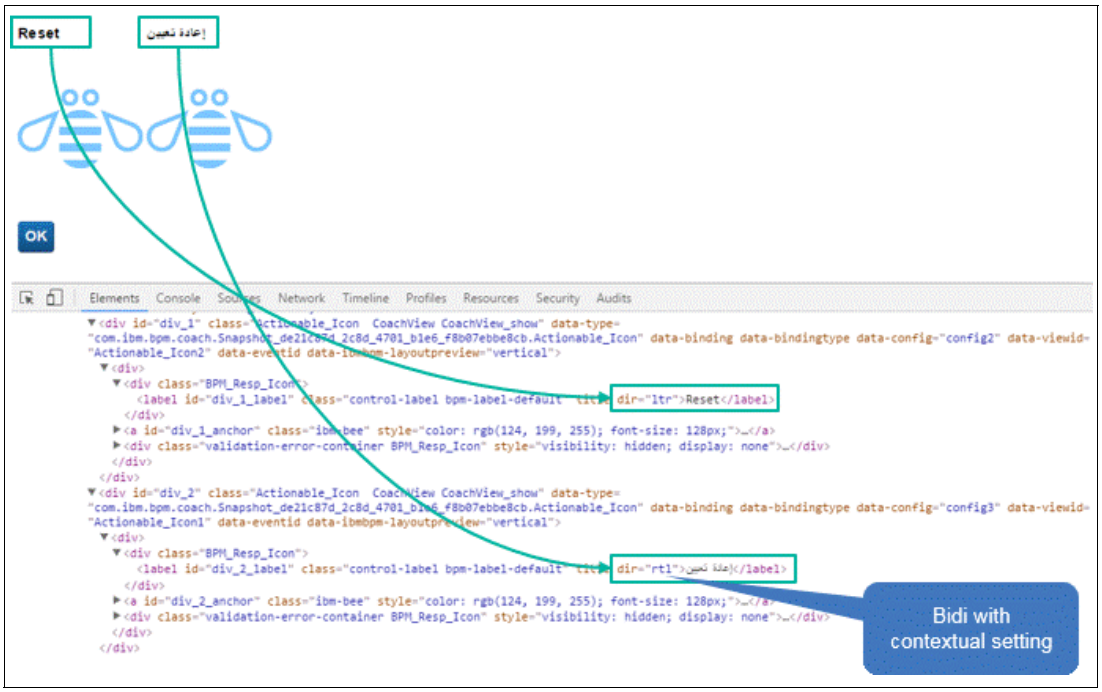


Figure 3-52 Bidirectional text setting with Contextual is detected according to displayed character

Localization of the coach view

If your custom coach view must support localization, the localization resources can be linked to the coach view in two ways. You can either link the localization resource in the Variable tab of the coach view, or you can provide a configuration option of type service and retrieve a list of NameValuePair from the service at runtime.

Example 3-32 shows that with the first approach you can again use the function provided by the utilities module.

Example 3-32 Code snippet to retrieve localization resources using the utilities module

```
var resources = utilities.initLocalizedMessages(this.context);
var errorText = resources["ErrorIcon_text"];
```

The `initLocalizedMessages(this.context)` function returns a list of NameValuePair. You can then reference the translated language by the key.

The same concept is used by the second approach by attaching an Ajax service as a configuration option. The Ajax service then returns a list of NameValuePair so that the event handler code can reference the language by the key as well.

Tip: For detailed coding of functions provided by the utilities module to handle visibility, base text direction, and localization, you can check the `utilities.js` in the `coach_ng_utilities.zip` file. A recommended practice is to maintain a copy of `utilities.js` and you can customize or add your own functions to better fit your application.

Content box

A content box allows you to add content to a custom coach view at design time. If you want to allow a coach author to specify the content of some parts of the coach view in IBM Process Designer, you can use the Content Box advanced item. When developing a coach view with Content Box, you might consider the lazy-loading pattern for better performance.

For more information about this topic, see IBM Knowledge Center:

- http://ibm.biz/BPMCoachViewsContentBox
- http://ibm.biz/BPMCoachViewsFrameworkVsViewManaged

3.3.3 Better support for the coach author in design time

Coach view developers do not only consider the user experience when the coach view is displaying content in browser runtime, they also must consider the user experience of a coach author.

Besides easily understandable labels, grouping, and documentation of configuration options mentioned in 3.2.1, “Defining and accessing data in the coach view” on page 75, there are additional artifacts available for a coach view in design time. These include the following artifacts:

- Palette icon
- Layout image
- URL binding
- Preview label support
- HTML snippet file for preview
- Helper JavaScript file for preview
- Tagging of coach view

Palette icon

On the right side of the coach editor, the coach author can select available coach views and drag them to the coach canvas. Figure 3-53 on page 127 shows how an easily identifiable icon can help a coach author locate the expected coach view.

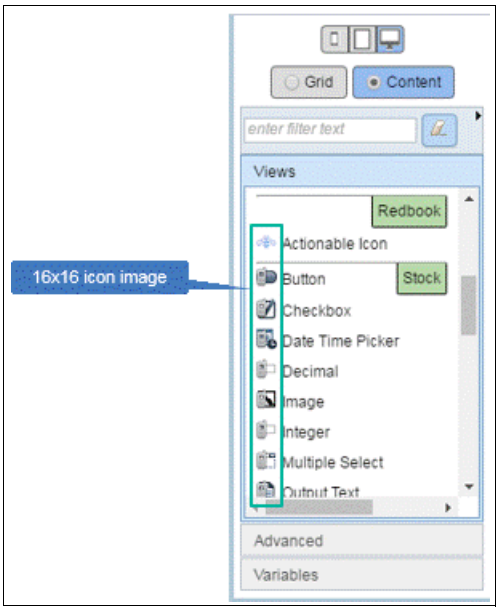


Figure 3-53 Icons for coach views in the coach editor palette

To support a palette icon, you must first create a 16 x 16 pixel size image file like the one in Figure 3-54.

Figure 3-54 Palette icon of coach view: 16 x 16 pixels

The icon image must be uploaded to the process application or toolkit as web files in your toolkit where the coach view is located. Figure 3-55 on page 128 shows how this is done.

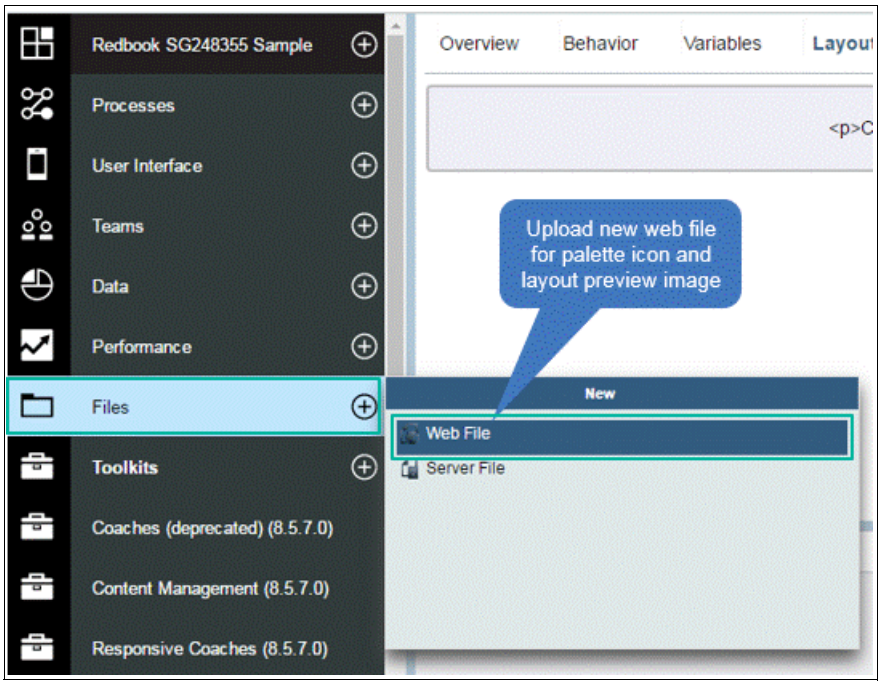


Figure 3-55 Upload a web file for the palette icon and layout preview image

Layout image

When a coach author assembles the coach in IBM Process Designer, it is useful to see how the coach might look like in runtime. The coach view developer can upload a static layout image as a preview in design time. Figure 3-56 shows this image.

Figure 3-56 Preview layout image helps understand layout in heritage human services

To set the palette and layout icons in your custom coach view, click the Overview tab. Then, select the palette and layout image files in the Preview section, as shown in Figure 3-57 on page 129. The preview palette and layout images for the Actionable Icon sample are already uploaded in the sample application. Select those images.

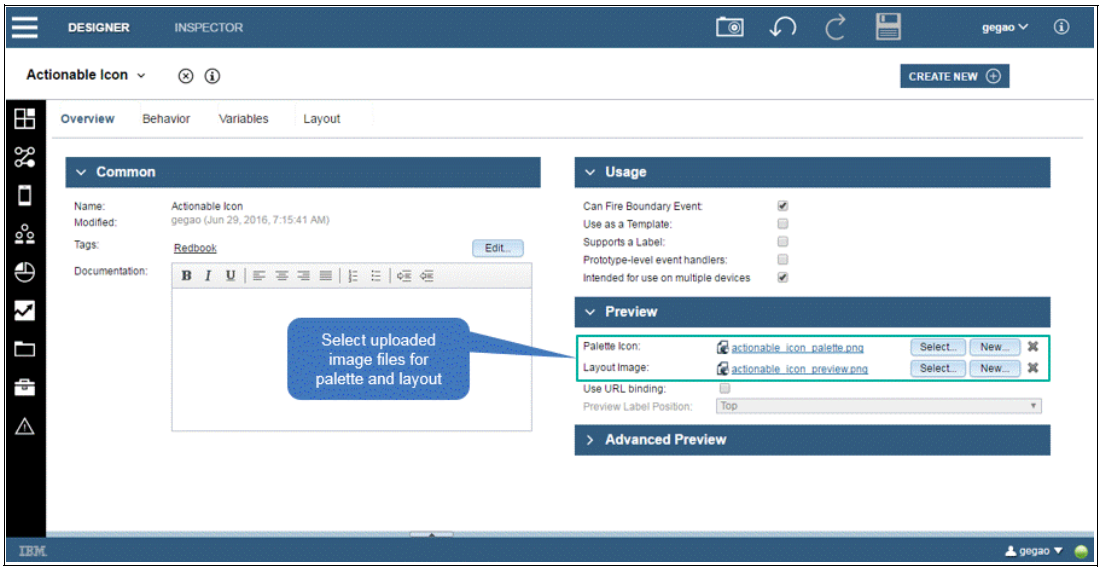


Figure 3-57 Set palette icon and preview layout image for coach view

Note: The preview layout image is static. It does not give the coach author a what you see is what you get (WYSIWYG) experience. The coach view developer uses preview JavaScript and HTML files to support WYSIWYG. A preview layout image is still required because the coach view can also be used in heritage human services, where preview files are not available.

URL binding

If a coach view has a binding object of type URL, which references to a managed asset image file, you can enable design-time preview with a dynamically loaded image using the bound URL.

Figure 3-58 shows how the stock control image uses this feature to better support the design time experience.

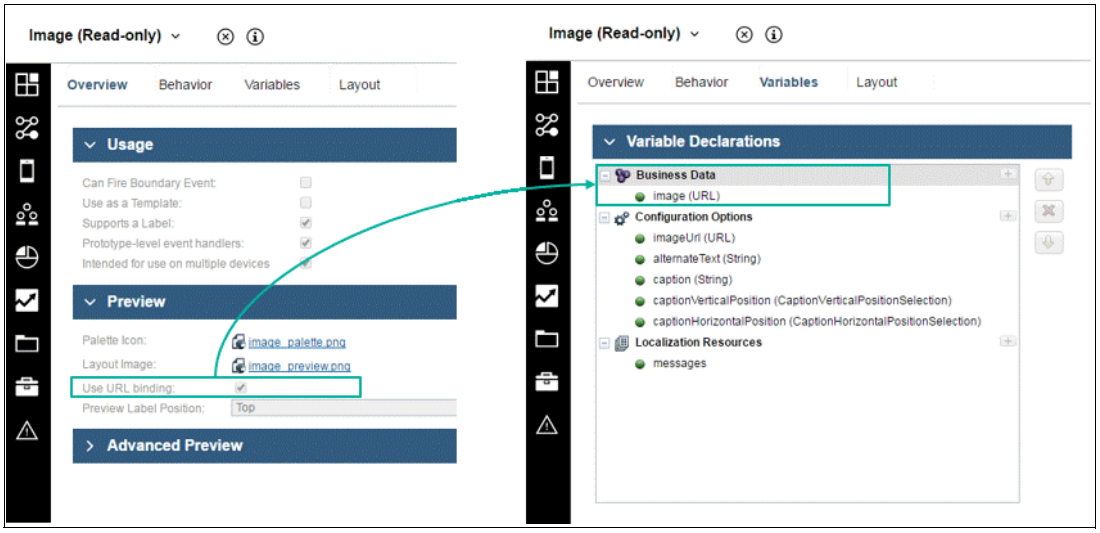


Figure 3-58 URL binding setting to load image by URL dynamically

Preview label support

If your custom coach view has a label, the label does not appear in the coach editor or coach view editor by default. Figure 3-59 shows that even though the label value is defined, it does not appear in the coach view editor.

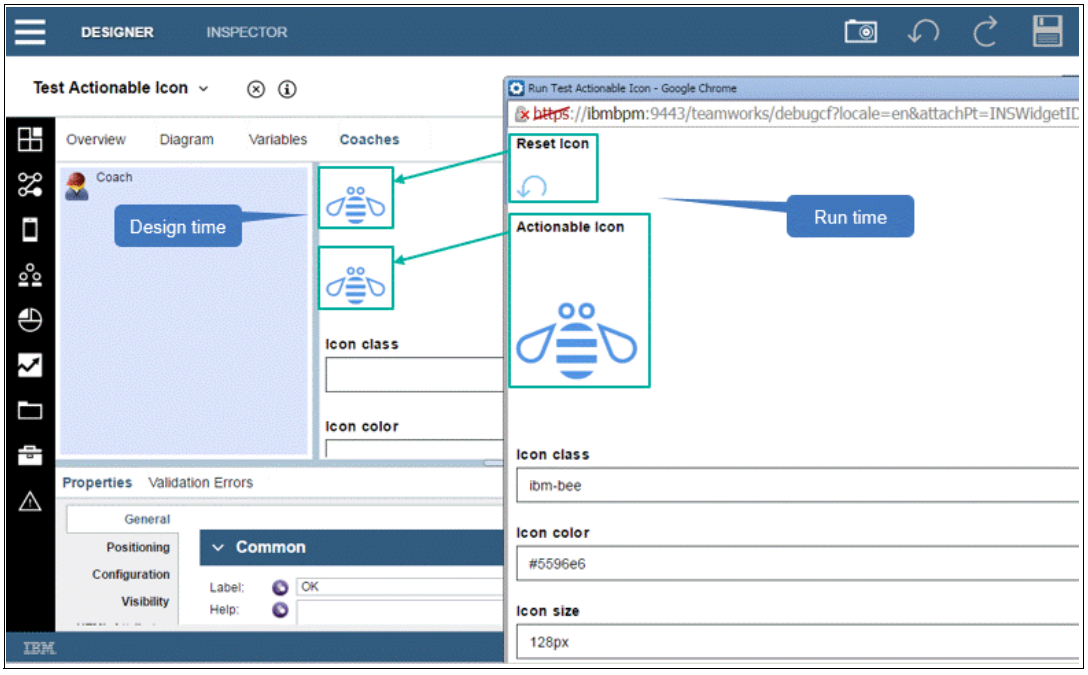


Figure 3-59 Missing label in design time

For the label to be visible at design time, you must select the “Support a Label” option in the Overview tab of the coach view Editor. Additionally, you must specify the default label position by using the Preview Label Position drop-down menu shown in Figure 3-60.

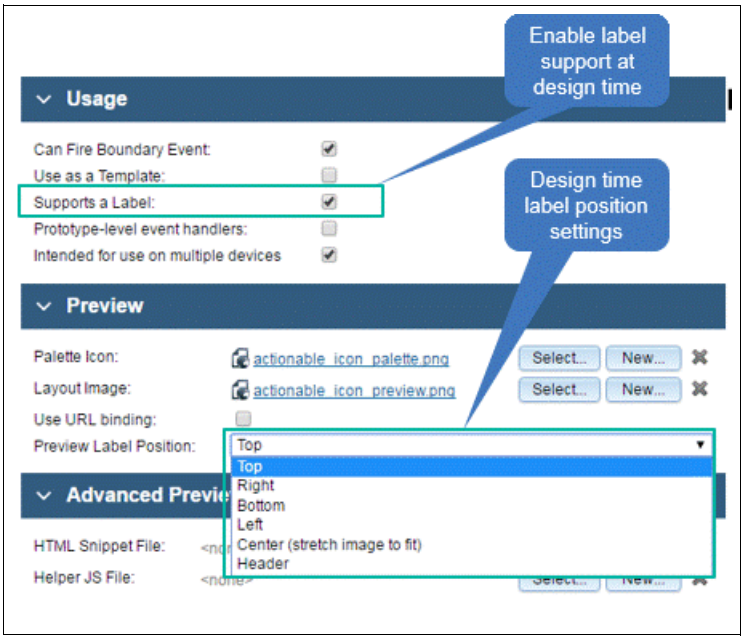


Figure 3-60 Setting label support and design-time position

This setting specifies only the design time preview support. If you have a configuration option to adjust the label position, the design time authoring does not reflect the setting; it shows the default. If you do not supply a configuration option for your label position and you have a hardcoded runtime label position, ensure that the preview label support matches the runtime support. For more information about design-time preview configuration, see IBM Knowledge Center:

http://ibm.biz/BPMCoachViewsDesignTimeAppearance

Preview HTML snippet file

More settings are required to provide a good user experience for the coach author. Because the preview layout image does not react to the change of configuration options of a coach view at design time, IBM Process Designer provides capabilities to enhance the WYSIWYG experience. The easiest way to enhance the WYSIWYG experience is to provide an HTML snippet file with a predefined API that can be rendered by Process Designer in design time. For example, if an HTML node is declared to have the class `DesignLabel`, Process Designer renders the label according to the current label settings in the coach editor.

You can extend your example to support design-time label rendering. An HTML file named `actionable_icon_preview.html` containing the code snippet in Example 3-33 is already prepared in the sample application.

Example 3-33 Code snippet for HTML preview snippet file

```
<div>
   <label class="DesignLabel control-label"></label>
</div>
```

Refer to Appendix A, “Additional material” on page 395 for information about downloading the file. This file must be added to the Overview tab of the coach view, as shown in Figure 3-61.

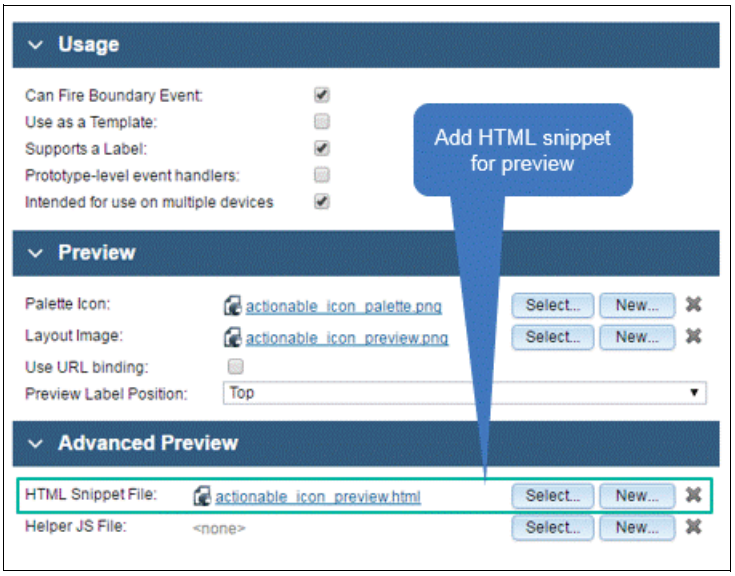


Figure 3-61 Add advanced preview HTML snippet file to coach view

Refresh the IBM Process Designer and open the coach in the testing human service. You can change the label in general properties section of the coach view. Figure 3-62 shows how the change is immediately shown in the coach canvas.

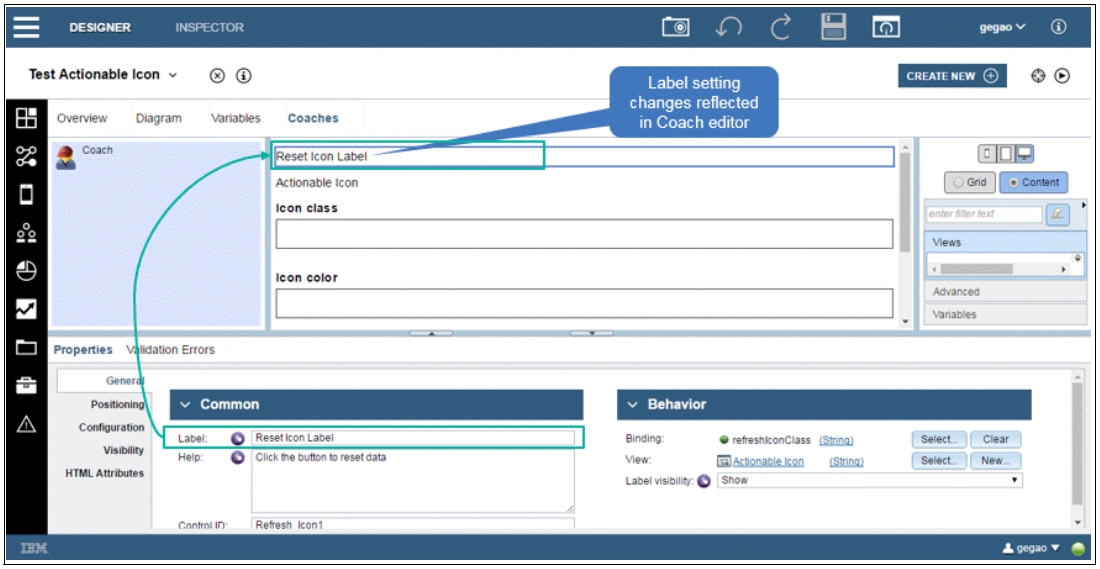


Figure 3-62 Label change is reflected in the coach editor at design time

Preview helper JavaScript file

In your sample, the Actionable Icon preview can be improved. The icon, color, and size are not shown at design time. To achieve this, a JavaScript helper file is required. Similar to the coach view, the preview also has its lifecycle in design time. Figure 3-63 on page 133 shows this lifecycle.

When a coach view is dragged to the coach canvas, the `createPreview()` event handler is called. This function is called only one time. Then the preview enters the second stage, where the `modelChanged()` and `propertyChanged()` event handlers are called because the initial binding object and configuration options are loaded. In stage two, any change on the binding object, configuration, or general properties of the coach view triggers both event handlers. If a binding object or configuration option is set with a simple value, the `modelChanged()` and `propertyChanged()` event handlers both contain the value as input parameters. If they are bound to a variable, the `modelChanged()` function returns the variable path and the `propertyChanged()` function returns null. Because the value of the human service variable is not available in the context of the coach, the preview cannot get their values if they are bound to variables. When a coach view is removed from the coach canvas or the human service is closed, the `destroyview()` function is called to clean up resources used for the preview.

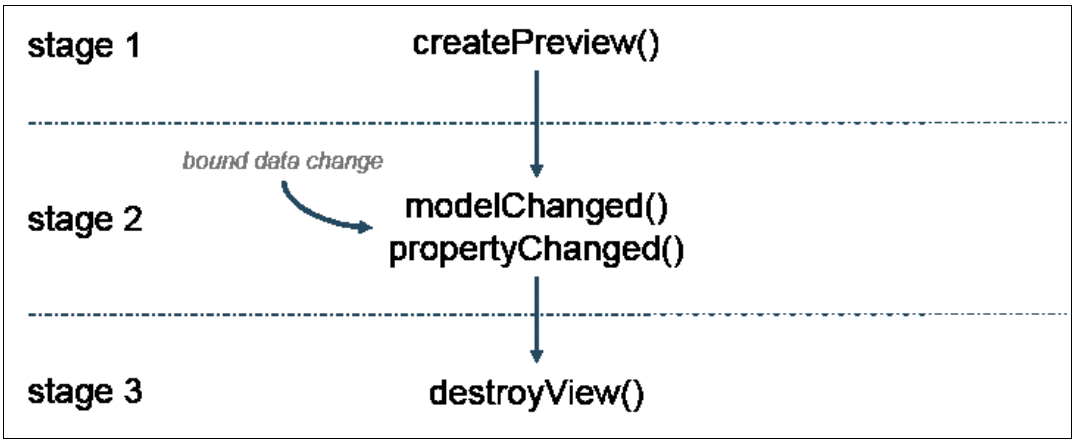


Figure 3-63 Lifecycle of event handlers for the coach preview

This section provides a detailed description of the Actionable Icon JavaScript. Because the icon class name is a binding variable, at design time its value cannot be detected. Because of this you use an extra configuration option, previewClass, to demonstrate the effect of the preview feature. In the sample application a JavaScript file named `actionable_icon_preview.js` is already prepared.

Figure 3-64 shows how you add this JavaScript file to the coach view.

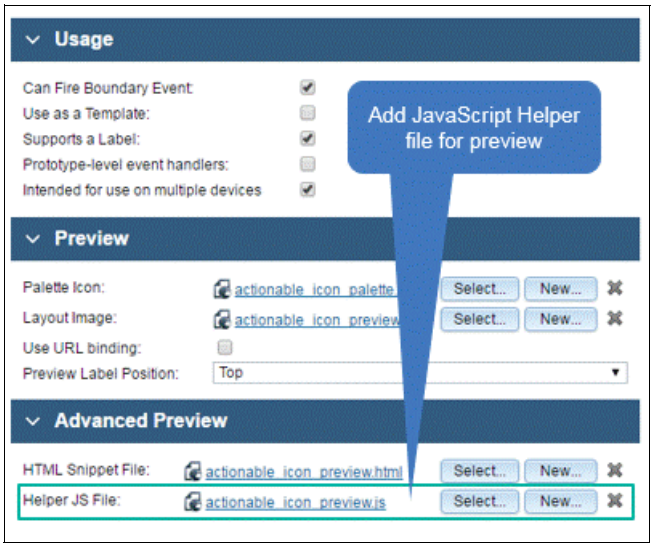


Figure 3-64 Adding an advanced preview JavaScript Helper file to the coach view

After you refresh IBM Process Designer and open the coach in the testing human service again, you can change the values of configuration options and see the immediate update in the coach canvas. Figure 3-65 shows this update. If not set, the configuration option takes the default value and renders the coach view, as shown on the left side of Figure 3-65.

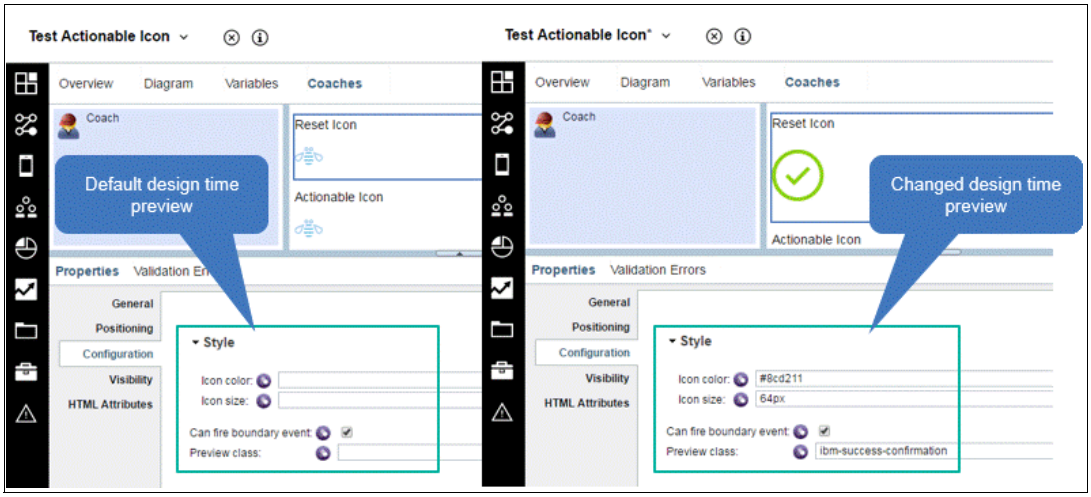


Figure 3-65 Comparison of design time default preview and changed preview

If you download the prepared `actionable_icon_preview.js` file, you can see it implements the two event handlers, `createPreview()` and `propertyChanged()`. The code snippet in Example 3-34 is used to initialize the preview. It requires the dojo/dom-construct module as an AMD dependency. It also requires the creation of an anchor node in the parent div node defined in HTML snippet file. This DOM node is referenced as the `containingDiv` input parameter. The `this.context.coachViewData` API stores any required data to be used in other event handlers. When a coach view is dropped to the canvas, there is no configuration for it. You must set the default styles in the `createPreview()` handler.

Example 3-34 Code snippet in createPreview() event handler to construct preview DOM node

```
require([ "dojo/dom-construct", "dojo/dom-class", "dojo/dom-style"],
this.lang.hitch(this, function(domConstruct, domClass, domStyle){
   var iconNode = domConstruct.create("a");
   domConstruct.place(iconNode, containingDiv);
   this.context.coachViewData.iconNode = iconNode;
   // set initial preview when first time dropped onto canvas
   domClass.add(iconNode, "ibm-bee");
   domStyle.set(iconNode, "color", "#7CC7FF");
   domStyle.set(iconNode, "font-size", "32px");
   callback();
}));
```

Before the DOM construction, initialization work must be done to load the required CSS styles. Example 3-35 shows this code snippet.

Example 3-35 Code snippet in createPreview() event handler to load required CSS styles

```
var head = document.getElementsByTagName("head")[0];
var link = document.createElement('link');
link.type = 'text/css';
link.rel = "stylesheet";
link.href = this.context.getManagedAssetUrl("ibmicons.min.css");
```

```
head.appendChild(link);
var cssUrl = this.context.getManagedAssetUrl("ibmicons.min.css");
var eotUrl = this.context.getManagedAssetUrl("ibmicons.eot");
var svgUrl = this.context.getManagedAssetUrl("ibmicons.svg");
var ttfUrl = this.context.getManagedAssetUrl("ibmicons.ttf");
var woffUrl = this.context.getManagedAssetUrl("ibmicons.woff");
var fontFace =' @font-face { font-family: "IBM Icon Font"; src: url("'
   + eotUrl + '"); src: url("'
   + eotUrl + '") format("embedded-opentype"), url("'
   + ttfUrl + '") format("truetype"), url("'
   + woffUrl + '") format("woff"), url("'
   + svgUrl + '") format("svg"); font-weight: normal; font-style: normal; }';
var fontStyle = document.createElement('style');
fontStyle.type = 'text/css';
fontStyle.appendChild(document.createTextNode(fontFace));
head.appendChild(fontStyle);
```

The HTML code to load the style sheet through the URL link is programmatically appended to the HTML head part; the font definition *@font-face* with the font files URL is similarly appended to the head part in HTML. The `this.context.getManagedAssetUrl()` API is for this task. You can use this code pattern to load required web files in the process application. A more practical way to do this is to organize the file loading code as a general function for better reusability.

In the `propertyChanged()` event handler, Dojo modules dojo/dom-class and dojo/dom-style are required. As with coach view event handlers, they are used to update the DOM node in preview. Example 3-36 shows the `propertyChanged()` event handler. If the changed new value is null or empty, the default values are used for rendering the preview.

Example 3-36 Code snippet in propertyChanged() event handler to update preview DOM node

```
propertyChanged:function(propertyName, propertyValue){
  require([ "dojo/dom-class", "dojo/dom-style"], this.lang.hitch(this,
function(domClass, domStyle){
    if (propertyName=="previewClass" && this.context.coachViewData.iconNode){
      domClass.remove(this.context.coachViewData.iconNode);
domClass.add(this.context.coachViewData.iconNode,
  (!!propertyValue ? propertyValue : "ibm-bee"));
    } else if (propertyName=="iconColor" && this.context.coachViewData.iconNode){
domStyle.set(this.context.coachViewData.iconNode, "color",
  (!!propertyValue ? propertyValue : "#7CC7FF"));
    } else if (propertyName=="iconSize" && this.context.coachViewData.iconNode){
domStyle.set(this.context.coachViewData.iconNode, "font-size",
  (!!propertyValue ? propertyValue : "32px"));
    }
  }));
},
```

For more information about the coach `preview` event handler, see IBM Knowledge Center:

http://ibm.biz/BPMCoachViewsDesignTimePreviewEventHandlers

Tagging the coach view

Tags can help organize artifacts used in your solution. They can also help locate the required coach view in the editor. Add at least one tag to your custom coach view. The coach editor in

Process Designer uses these tags to categorize the coach view on the palette. Figure 3-66 shows how you add tags to a coach view in Process Designer.

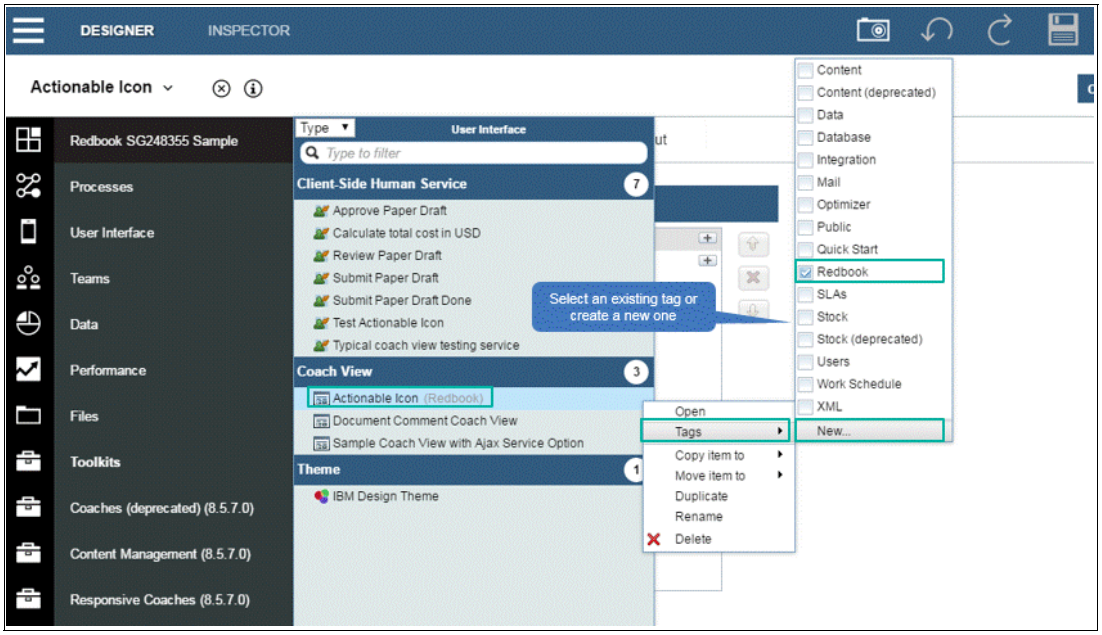


Figure 3-66 Selecting existing or creating new tags for a coach view

3.3.4 Developing coach views with existing UI libraries

Coach Framework is written with the Dojo base package, which provides convenient utility APIs to support DOM manipulation, events, Ajax calls, data stores, and other functions. The package can be used when developing a coach view. Coach Framework also provides APIs to support coach view development.

In addition you can use existing widgets in a JavaScript library or toolkit to avoid building complex controls. For example, you can develop a coach view with Dojo Dijit or DojoX modules and jQuery or AngularJS controls. This section explains how to write an AngularJS-based coach view.

For jQuery-based coach view development, see IBM Knowledge Center:

http://ibm.biz/BPMCoachViewExampleJQueryButton

Note: Sock controls in the deprecated coaches toolkit can be referenced as a sample of developing a Dojo Dijit-based coach view.

The Responsive Coaches toolkit in BPM version 8.5.7 is a set of controls written with AngularJS. The toolkit itself provides a good reference on how to write a custom coach view with AngularJS. As with other controls, an AngularJS-based control must also define its HTML view layout. Instead of constructing a DOM node through an API, AngularJS supports declarative markup. In this way, you can move much of your coding effort to the HTML definition. Example 3-37 shows a sample HTML definition.

Example 3-37 Sample code snippet of using AngularJS declarative markups in view layout

```
   <meter id="progressBar-progress-{{viewid}}" ng-disabled="disabled"
      aria-labelledby="progress-label-{{viewid}}" value="{{bindingValue}}"
min="{{minValue}}" max="{{maxValue}}" style="width:100%" >
   </meter>
</div>
```

The HTML definition is defined with a few variables, which can be watched by AngularJS in runtime. These listeners have to be registered so that any changes on the bound data can be passed in and out to AngularJS for update in the DOM nodes. To register the listeners, use the code shown in Example 3-38.

Example 3-38 Adding watchers to bound data for update using AngularJS

```
$scope.watchOption("maxValue", 100);
$scope.watchOption("minValue", 0);
```

Because AngularJS is not implementing an AMD specification, the toolkit provides a mechanism to load dependencies and bootstrap the control in the AMD style.

To use AMD style loading and bootstrapping, add the code in Example 3-39 to the Inline JavaScript section of your coach view.

Example 3-39 Inline JavaScript to add coachViewHelper in AMD style

```
/*
@dojoConfigUpdateStart
dojoConfig.packages.push({
name: 'com.ibm.bpm.coach.angular',
location: com_ibm_bpm_coach.getManagedAssetUrl('coachview.helper.min.zip',
```

```
com_ibm_bpm_coach.assetType_WEB, 'SYSRC')
});
@dojoConfigUpdateEnd
*/
```

Then you can add the `com.ibm.bpm.coach.angular/scripts/coachViewHelper` module to your AngularJS based coach view. Figure 3-67 shows how to add this module.

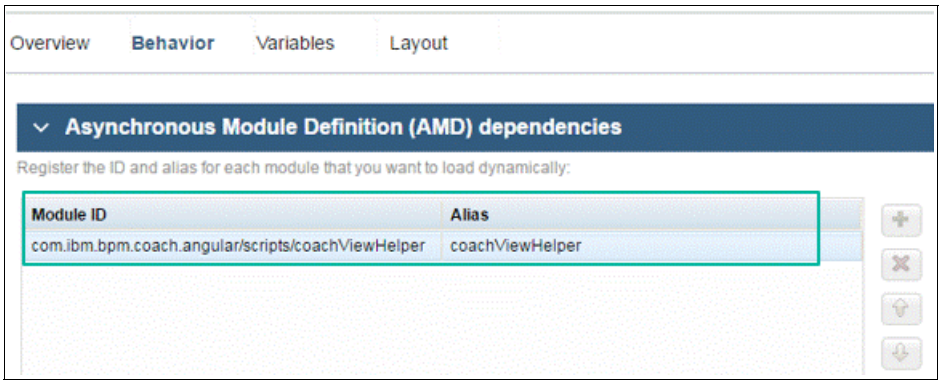


Figure 3-67 Adding coachViewHelper as an AMD module for the AngularJS-based coach view

This module provides APIs to load resources, bootstrap control, and call controller functions to the coach view.

For more information about `coachViewHelper`, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewsAngularJSHelper`

For an example of an AngularJS-based coach view, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewExampleAngularJSProgressBar`

3.4 Patterns of coach view development

When developing a coach view, there are patterns that can be reused. These patterns are necessary for coach view quality improvement.

3.4.1 Configuration option with drop-down selection

Limiting the available values of a configuration option is a common pattern to avoid unexpected configuration errors. If the configuration option has several selections, you can define a business object of simple selection type with a meaningful display name and predefined value. When you set the configuration option variable type to this object, it is shown in the coach editor as a drop-down selection control. The coach author can select only from one of the predefined options according to the display name. The benefit of this pattern is not only to avoid unexpected configuration errors, but also to provide a builder-friendly coach configuration option. For example, the ThemeColors business object is a predefined business object of type simple selection in the Responsive Coaches toolkit.

Figure 3-68 shows this business object.

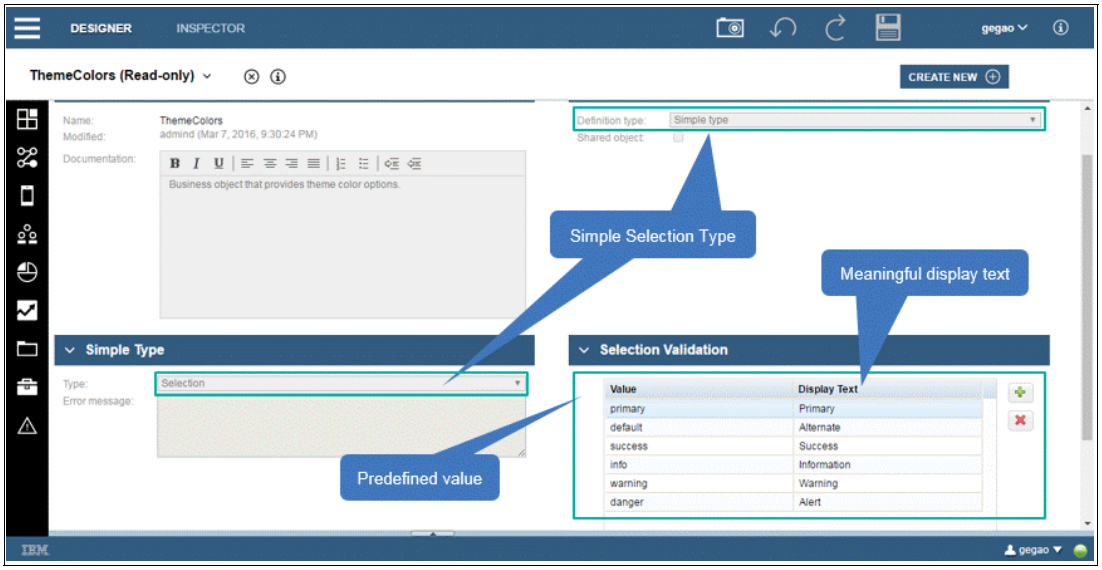


Figure 3-68 Defining a simple selection type business object for a configuration option

You can change the Actionable Icon `iconColor` configuration option to this object type. In design time, the `iconColor` configuration option is shown as a drop-down list instead of text. Figure 3-69 shows this drop-down list.

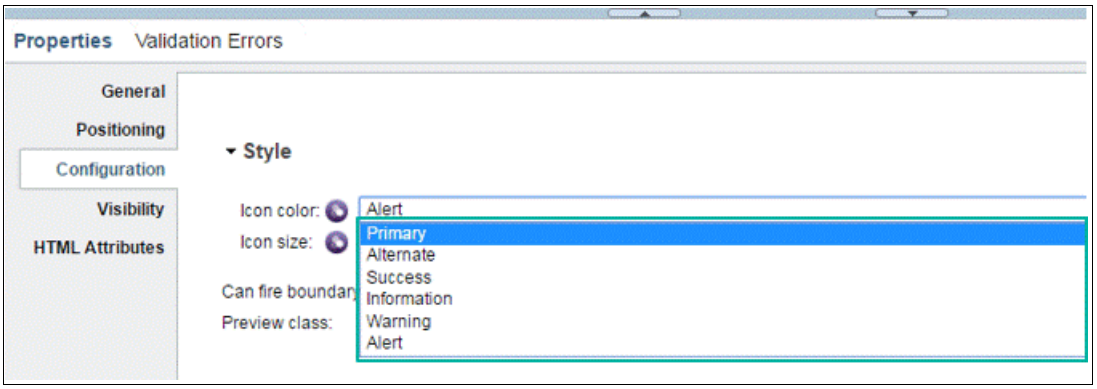


Figure 3-69 Configuration option drop-down list

3.4.2 Use LESS for theme participation

As mentioned in “Use CSS and LESS for styling” on page 85, you can define the dynamic style with LESS syntax to reference the variables defined in a theme. By using this pattern, the changes on the theme are automatically applied to your control’s style. This allows theme customization that keeps the UI controls with a consistent style and without individual change.

This section describes how to connect the Actionable Icon sample to the global theme with predefined colors. You create a LESS file that contains the dynamic styles code shown in Example 3-40 and then add the file to the Actionable Icon coach view. Figure 3-70 shows this connection.

Example 3-40 Sample LESS dynamic styles

```
.primary-color{
   color: @bpm-color-primary;
}
.alternate-color{
   color: @bpm-text-color;
}
.info-color{
   color: @bpm-color-info;
}
.success-color{
   color: @bpm-color-success;
}
.warning-color{
   color: @bpm-color-warning;
}
.danger-color{
   color: @bpm-color-alert;
}
.readonly-color{
   color: @bpm-neutral-light;
}
```

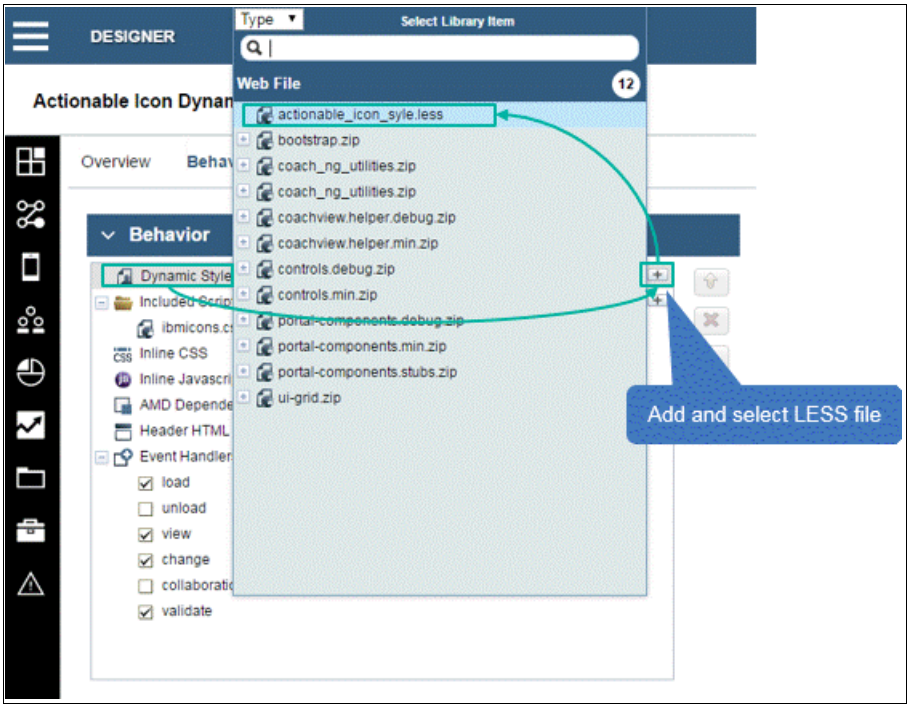


Figure 3-70 Adding dynamic style to the coach view

You must change the Actionable Icon event handler JavaScript of to use `domClass` to manipulate the class name defined in the dynamic style. Do this instead of directly

manipulating the DOM node styles. For example, you can replace the code in Example 3-41 with the code in Example 3-42.

Example 3-41 Old code snippet manipulating DOM node style

```
domStyle.set(this.iconNode, "color", iconColor);
```

Example 3-42 New code snippet manipulating dynamic style

```
domClass.add(this.iconNode, iconColor+"-color");
```

For a full implementation of `view()` and `change()` event handlers with dynamic style, you can read the sample code of the Actionable Icon Dynamic Style coach view in the completed sample in Appendix A, “Additional material” on page 395.

Tip: Choose a specific prefix, for example, company name, to avoid any class name collision when different coach view toolkits are used in a single coach.

3.4.3 Loading curtain

Depending on the size and the complexity of a web page, JavaScript and CSS might take a long time to complete initialization. The speed of initialization depends on the browser’s rendering engine. During this initialization, a web page is not ready for user interactions and might flicker as the page elements are loaded and initialized.

To avoid flickering, you can implement a Loading Curtain coach view that includes a progress indicator and a CSS style with an opaque background color high z-index. Figure 3-71 on page 142 shows the runtime effect of the loading curtain. This also provides a better user experience by indicating to the user that the page is being loaded.

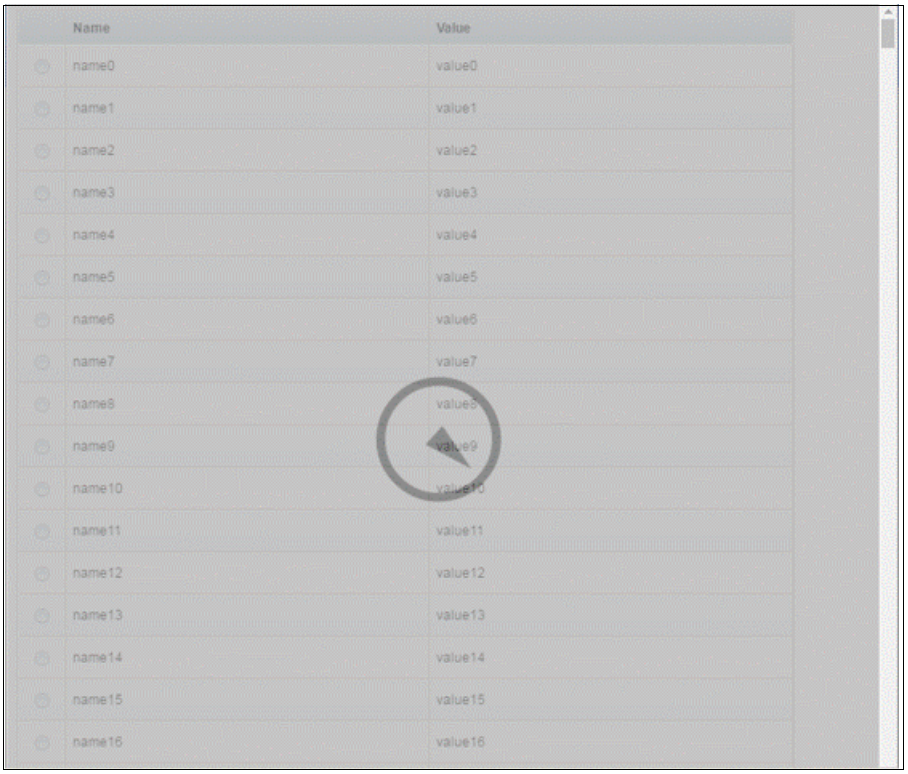


Figure 3-71 Loading curtain runtime effect

To implement a Loading Curtain coach view:

1. Add the Content Box in the Layout tab. Figure 3-72 on page 143 shows that the Content Box manages its own view.

> **Note:** Do not put controls requiring bootstrapping, such as AngularJS-based responsive controls, into the Content Box; bootstrapping is not supported by this sample. Instead, put controls requiring bootstrapping below the Loading Curtain control.

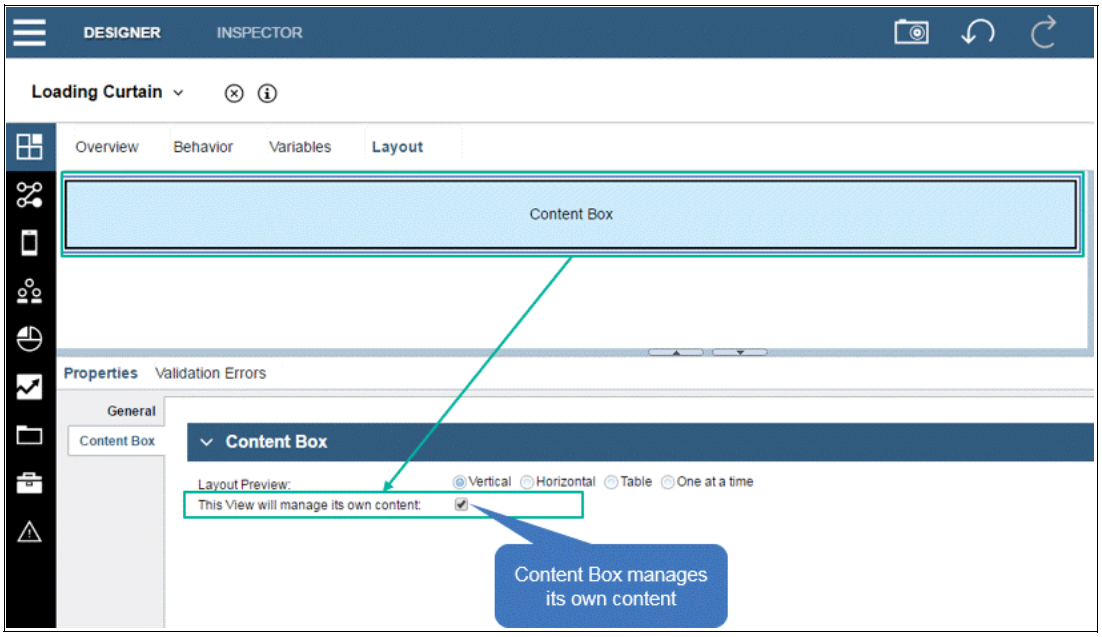


Figure 3-72 Adding Content Box and setting it to manage its own view content

2. Add the IBM Design font icon style sheet, `ibmicons.min.css`, and an AMD dependency, dojo/dom-construct. Figure 3-73 shows how to do this in the Loading Curtain window.

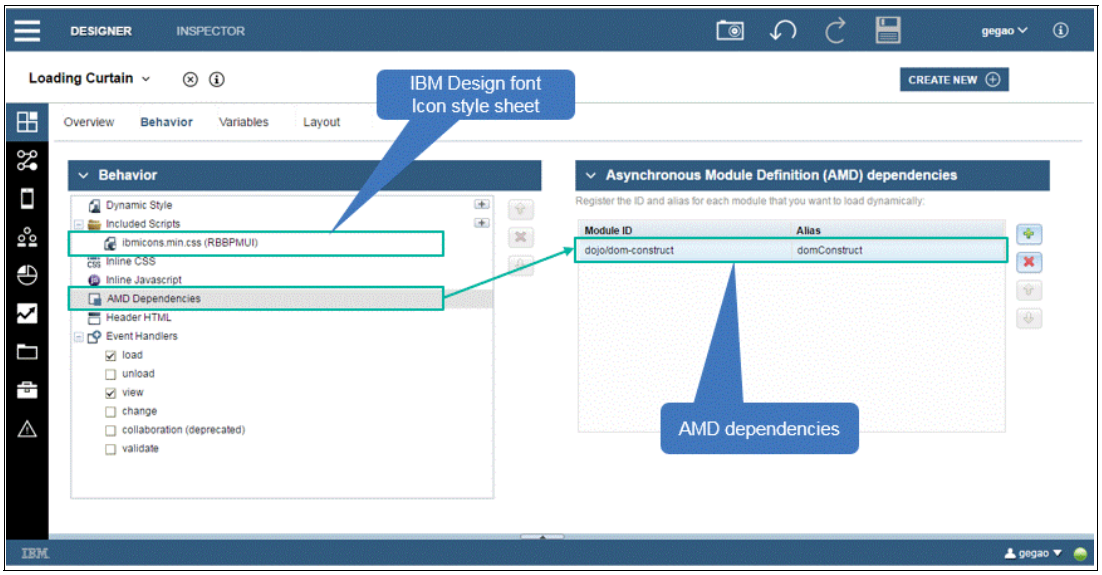


Figure 3-73 Adding a style sheet for font icons and AMD dependencies

3. Add styles for the loading curtain DOM node and loader indicator DOM node. Use inline CSS for this sample (see Example 3-43). Use LESS or CSS files in your delivery.

Example 3-43 CSS styles for curtain and loading indicator DOM nodes

```
.curtain {
    width: 100%;
    height: 100%;
    top: 0px;
    left: 0px;
```

```
    position: fixed;
    display: block;
    opacity: 0.7;
    background-color: #aeaeae;
    z-index: 99;
    text-align: center;
}
.spinner {
    animation: spin 2s linear infinite;
    font-size: 128px !important;
    color: #5a5a5a!important;
    display: inline-block;
    position: absolute;
    top: 50%;
    left: 50%;
    margin: -64px 0 0 -64px;
}
@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

4. Construct a DOM node for curtain and loading indicators. Example 3-44 shows the code snippet for initializing DOM nodes for curtain and loading indicators. The Content Box can be located by the predefined class name ".ContentBox".

Example 3-44 Code snippet to initialize DOM nodes for curtain and loading indicator

```
this.contentBox = this.context.element.querySelector(".ContentBox");
this.curtainDOM = domConstruct.create("div", {className: "curtain"},
this.contentBox, "first");
this.spinnerDOM = domConstruct.create("a", {className: "spinner ibm-gauge"},
this.curtainDOM, "first");
```

5. Render views in the Content Box and then destroy the curtain DOM node and its children elements. Example 3-45 shows the code required to do this. A `setTimeout()` function is used because the Content Box as the outer-most control is rendered after all its contained controls are rendered. Because you want to show the indicator before the rendering of contained controls starts, delay rendering them for 0.5 seconds. The loading indicator is then displayed first. After the rendering of Content Box-contained controls has finished, the curtain and its children elements are destroyed.

Example 3-45 Code snippet to render Content Box views and destroy loading curtain afterwards

```
var _this = this;
setTimeout(function() {
    _this.context.createView(_this.contentBox);
    domConstruct.destroy(_this.curtainDOM);
}, 500);
```

You can then create a CSHS to test it. This service requires a private service variable called `listOfNVP`, which is a list of the NameValuePair type. In an initialization script before the coach, you can add the code in Example 3-46 on page 145 to initialize a list of 500 elements. These elements are bound to a stock `Table` control below the Loading Curtain coach view in the coach.

Example 3-46 Code snippet to render Content Box views and destroy the loading curtain afterwards

```
tw.local.listOfNVP = [];
for (var i=0; i<500; i++) {
   tw.local.listOfNVP[i] = {};
   tw.local.listOfNVP[i].name = "name"+i;
   tw.local.listOfNVP[i].value = "value"+i;
}
```

In Appendix A, “Additional material” on page 395 in `SG24-8355-ch3-completed.twx`, you can find this sample coach view and its testing service named *Test Loading Curtain Completed*.

3.4.4 Publishing and subscribing an event through a controller coach view

When you have the coach view triggering a DOM event that is distributed to some other controls according to the event payload, you might consider using a utility coach view. A utility coach view is not visible but plays a role as a controller to forward or translate the event payload. This allows other coach views, which subscribe to some topic event of the controller, to receive events accordingly.

For example, as shown in Figure 3-74, the controller coach view subscribes to `topic1`, which is published by coach view 1 or 2; their events, however, indicate a different type of message and must be handled differently. The controller forwards or translate the message to coach view 3 or 4, depending on the received type in the event. If the type is “1”, the controller translates the message and coach view 3 receives it through topic2. For type “2”, the coach view 4 receives it through topic3.

This pattern has a benefit of loose coupling and can be scaled well. For example, you have a coach with a tree control on the left side that displays content of the selected tree node on right side. The content to be displayed depends on what kind of tree node is selected by the user. The controller can then retrieve content data by calling different kinds of Ajax services.

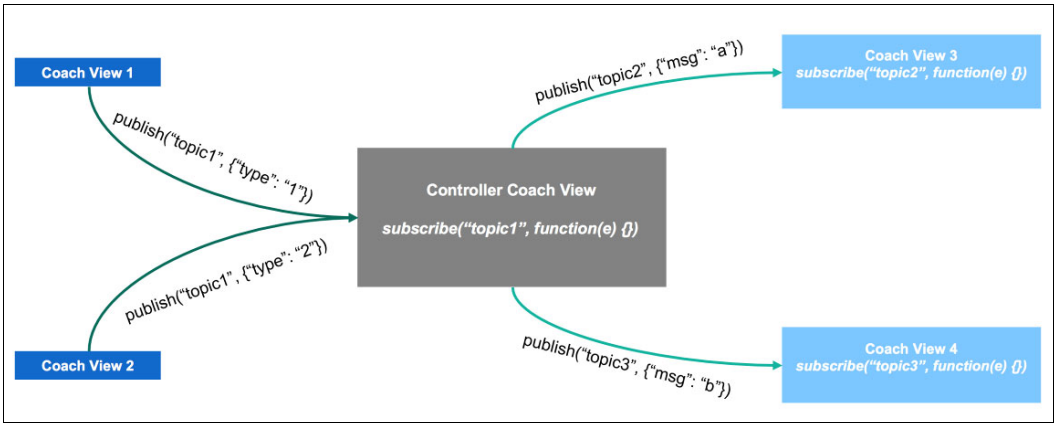


Figure 3-74 Sample pattern to forward or translate an event through the controller coach view

3.5 Performance considerations

This section describes performance considerations.

3.5.1 Ensuring the browser cache works

BPM solution is a distributed application with a client-server model. Unnecessary network round-trips impact performance on the client side. Caching static resources, such as Coach Framework-required JavaScript files, managed files, style sheets, pictures, and other static resources, can help reduce the network round-trips. Often users perceive bad performance because the cache in the browser does not work correctly.

Figure 3-75 shows how to check if the browser cache works correctly in a browser's developer tool. If you notice static resources are always retrieved from the backend server, you must check if your browser setting has enabled cache and if the HTTP header for cache control allows those resources to be cached at the client side. Although the BPM server enables caching of static resources, there is often an HTTP server standing before the BPM server in the backend infrastructure, where the HTTP response header might be changed.

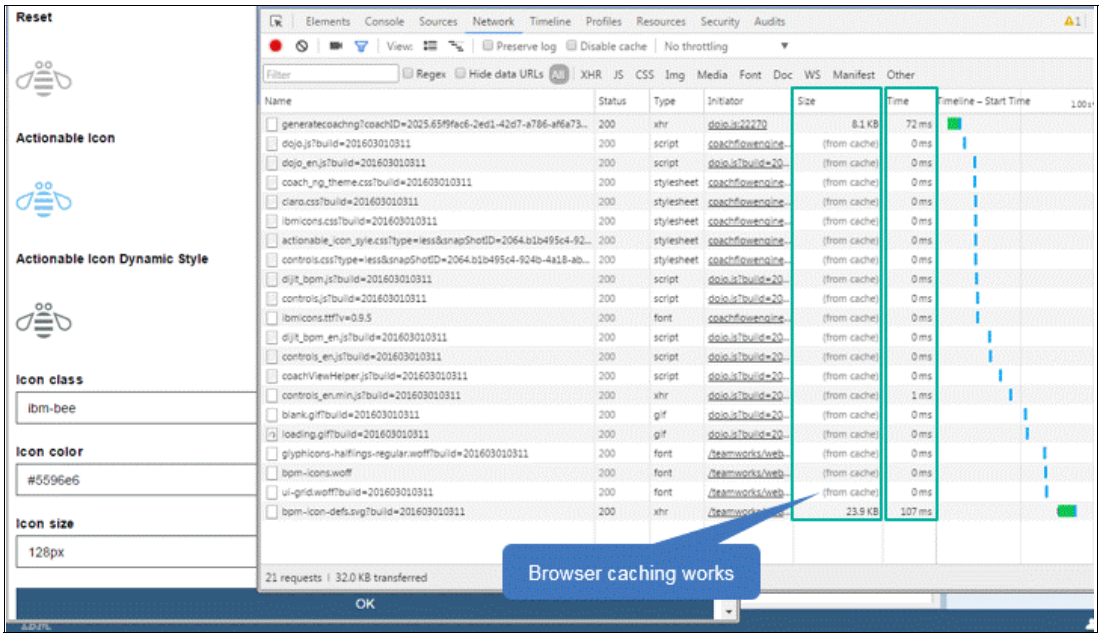


Figure 3-75 Checking if browser caching works correctly

3.5.2 Using Prototype to reduce the memory footprint

Prototype is a JavaScript concept for object inheritance. Each JavaScript object has a prototype and inherits properties and methods from the parent prototype. You can define coach view event handlers to be defined in the prototype level so that memory can be conserved. Depending on the browser type, prototype-level defined methods can be executed faster. To make the event handlers be defined in the event handler level, you set the flag in Overview of the coach view. Figure 3-76 on page 147 shows how to set the flag.

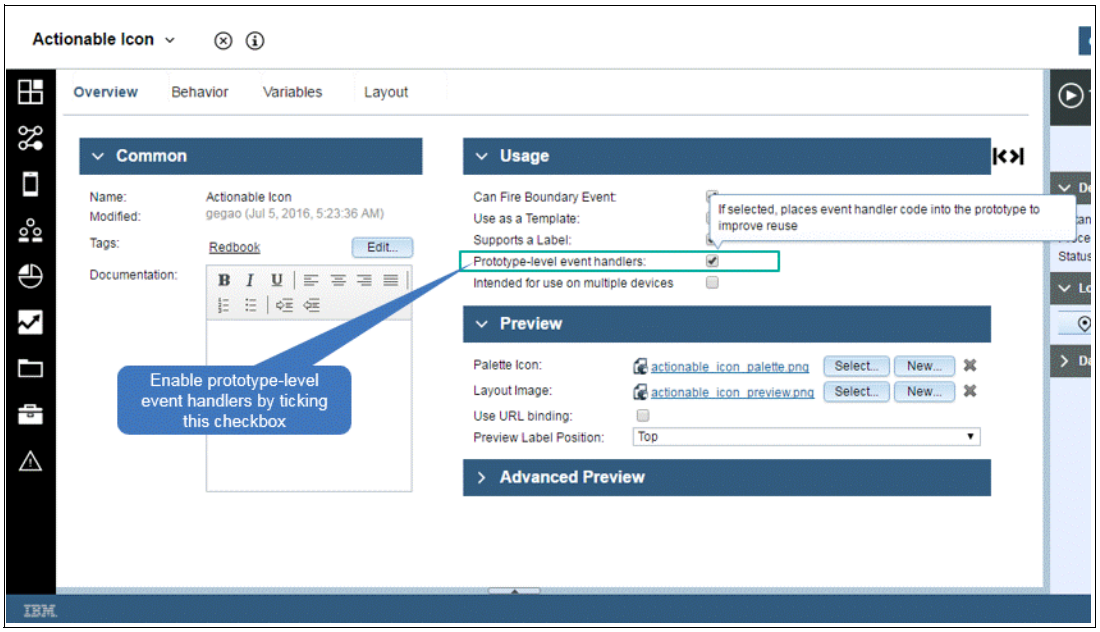


Figure 3-76 Enabling prototype-level event handlers in the Overview tab of the coach view

It is a preferred practice to enable prototype-level event handlers. All stock controls support this feature. There are differences between coding inline JavaScript with and without prototype-level event handlers. Table 3-7 shows the differences.

Table 3-7 Differences between inline JavaScript with and without prototype-level event handlers

Instance-level event handlers	Prototype-level event handlers

1. Define the variable in the inline JavaScript of the coach view:
`var myVariable = "123";`
2. Access the variable in the load event handler:
`if(myvariable == "123") {`
`...`
	1. Define the variable in the inline JavaScript of the coach view:
	`this.myVariable = "123";`
	2. Access the variable in the load event handler:
	`if(this.myvariable == "123") {`
	`...`

3.5.3 Using lazy loading

A lazy-loading pattern is commonly used to enhance the user experience by not creating controls that are not yet required to be displayed. There are typically two kinds of lazy loading:

- Lazy load the view of controls

  Lazy loading of controls can be implemented by using the Content Box, which manages its own content. The coach view calls the Coach Framework `this.context.createview()` API to start creating the views within the Content Box when they are to be displayed. This occurs, for example, when a user expands a collapsed section or switches to a hidden tab.

- Lazy load the data retrieved through an Ajax service

  Lazy loading of data through an Ajax service can be implemented by putting service calls into a function that is only executed on certain conditions. For example, in a drop-down list control, the list of available values is retrieved by the Ajax service from the backend server. This service is called in a function, where the service is not called until the entered input is more than three characters. Another benefit of this condition check is it reduces the number of to-be-returned available values by narrowing down the range. For example, by

searching an employee name in a large company, there are more results returned by a search for names containing three characters than by a search of names containing only one character. This reduces the backend server search load and also the data size returned.

3.5.4 Reducing service calls

Network round trips between the client and service are expensive for distributed applications, especially if the client-side UI rendering has to wait for a service call response. For example, a table has a select control in each row that contains a list of schools to be selected. Each control retrieves the same list for selection. But the list of schools depends on user interaction, for example, by the city name given by the user. In this case, you can program in the Select control to check if the required list is available in local cache. For example, check in the `com_ibm_bpm_global` global object, which is accessible by all coach views in the coach. If the cache is missed, the service call can be started to return a list of schools in the user-selected city. The response is then stored in the local cache for later use. If the cache is hit, the list of available schools can be directly read from the cache instead of calling the backend service. Figure 3-77 illustrates this concept.

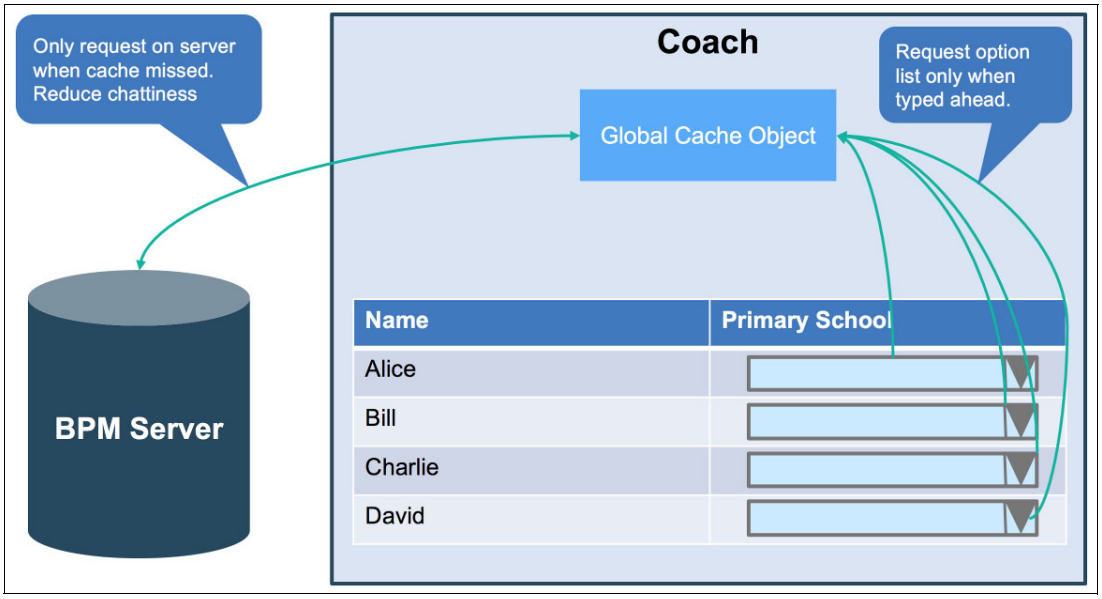


Figure 3-77 Reducing Ajax calls by the local cache

Tip: If you design a single-site BPM solution that can be accessed by users across different countries, be careful because the network latency can be different among the countries. A local test does not reflect all user experiences. Plan the test early in the design process to detect potential issues because of network latency. Also, check if unnecessary backend service calls can be saved or merged to reduce the number of network round trips.

3.5.5 Balancing UI design simplicity and usability

It is important to balance simplicity and usability in good UI design. In general, reducing the number of controls within a coach improves performance. Because each coach view represents both the DOM and JavaScript object, initialization time of the JavaScript and memory consumption increases as the number of coach views increases. In addition,

responsive coach views, based on AngularJS technology, have listeners to monitor the DOM value change. This also affects performance if too many coach views are used in a single coach. Responsive coaches display BPM UI on different devices well. However, when your application requires complex UI design, which does not work well for mobile devices, consider using other UI toolkits such as the Spark UI toolkit introduced in Chapter 5, “IBM Process Portal” on page 261 or write your own controls.

3.5.6 Do not mix controls written with different JavaScript libraries

Coach Framework supports coach views implemented with different third party JavaScript libraries. As noted in 3.3.4, “Developing coach views with existing UI libraries” on page 137, a coach view developer can use libraries such as AngularJS, jQuery, Dojo Dijit/Dojox, and other libraries. However, mixing the coach views of different JavaScript libraries might cause performance issue because of the requirement of a bigger size of multiple-loaded JavaScript libraries at runtime.

Also, mixing the use of coach views written with different JavaScript libraries is not verified by the Coach Framework or the third-party toolkit provider. This can also lead to unexpected runtime behavior.

3.5.7 Avoiding unnecessary change-events processing

Change events are triggered when a bound object has changed. However, not all of these change events have to be handled in the `change()` event handler. The code in the `change()` event handler checks the event type and changes properties explicitly before processing the change event. This allows change events that are not required to bypass the processing.

Also, avoid unnecessary change-event firing. For a complex object, use the `context.bind(path, callback, [scope])` API to explicitly bind to a property that must be observed by the framework. Always check the value of the configuration and binding before setting a new value. If a new value is the same as the old value, do not set the value again.

3.5.8 Using custom Dojo build layer

With the Dojo build system, your coach view can include the modules that it depends on in one file or in a small set of files. As noted in section 3.5.4, “Reducing service calls” on page 148, network round-trips in an infrastructure with high network latency reduces client-side performance. By creating your own Dojo build layer, the custom JavaScript, CSS, and HTML files can be minified and compressed into one file so that network round-trips are reduced to one. These files are then cached and loaded at browser runtime in an AMD style.

Because Coach Framework bundles a version of Dojo, when you make a custom Dojo layer ensure that you use the latest version of Dojo.

For more information about the custom Dojo build layer, see IBM Knowledge Center:

`http://ibm.biz/BPMCoachViewPerformance`

3.5.9 Handling editable and read-only modes differently

The behavior of a control is different in editable and read-only modes. Control in read-only mode does not change because of user interaction. In a complex UI, for example a coach with a big read-only table or with many read-only text fields, you might handle the editable and read-only modes differently.

The table in read-only mode does not require a control in each table cell because no user interaction is required. When the displayed data can be updated, it fulfills the requirements. Therefore, you might consider rendering the table control without rendering the embedded controls to achieve a better performance.

If many read-only text fields can be grouped together in one DOM node, you can also create an atomic control with a complex object containing all the required bound objects to the fields as sub-objects. The update to the sub-object can be then individually processed in the `change()` event handler. In this way, the number of coach view instances and bound objects is reduced and a better performance is achieved.

3.6 Checklist for developing a custom coach view

This section provides a basic checklist for developing a custom coach view. You must adjust this list according to concrete project guidelines and requirements.

- Data
  - Define the variable names of binding objects and configuration options that apply to naming conventions with your team.
  - Set an easy, understandable label for the configuration option.
  - Set easy, understandable documentation for the configuration option. For clarity, provide a sample in the documentation.
  - If design-time localization is required, that the label, group, and documentation are bound to the localization resources of the coach view.
  - If runtime localization is required, ensure that either localization resources are linked to the coach view or an Ajax service providing localization resources is set as a configuration option and resources are correctly processed.
  - Use `correct bind()` and `bindAll()` APIs, if necessary.
  - Avoid caching bound objects in asynchronous executed code. For example, store the cache in variables across event handlers or `setTimeout()` within a event handler.
  - Ensure that each custom coach view has a unit test human service with initial test data and the ability to change test data.
- View
  - Use the LESS file to define the dynamic style so that the coach view can participate in the solution theme.
  - Use an appropriate query to locate the DOM node.
  - Use $$viewDOMID$$ and double curly braces only if absolutely necessary.
  - Avoid using inline CSS styles in the completed coach view.
  - If required, ensure that the validation error is correctly rendered and consistent with global solution styles.
  - Ensure that the label and its visibility are processed correctly.
  - Ensure that the coach view visibility is processed correctly.
  - Ensure that the base text direction is processed correctly, if required (only available for heritage human services).
  - If possible, the enable lazy-loading option for delayed rendering or delayed data retrieval.

  - If possible, reduce the number of Ajax calls.
  - If necessary, handle editable and read-only modes of complex UI differently.
- Controller
  - If possible, enable prototype-level event handlers.
  - Explicitly filter out change events for processing.
  - Ensure that the callback function of the Coach Framework API is provided, if necessary.
  - Ensure that the `load()` and `error()` callback functions are provided when making XMLHTTPRequest calls.
  - Ensure that the boundary event is correctly fired and callback is processed, if necessary.
  - Ensure that no debugger breakpoint remains in the event handlers when delivering the coach view.
  - Ensure that a custom Dojo build is provided and correctly loaded if many custom modules exist.
- Design-time support
  - Provide the palette icon.
  - Provide the layout image.
  - Provide a URL binding if a binding object is of type URL and points to a displayable resource.
  - Ensure that a preview label is supported, if necessary.
  - For preview, ensure that HTML snippet files, helper JavaScript files, or both types of files are provided and work as expected at design time.
  - Tag the coach view according to the naming conventions defined in the project.

3.7 Conclusion

This chapter provides the information you require to build custom coach views that fit the purpose of your BPM solution UI requirements. You have learned the concept of coach view structure and how to implement its data, view layout, and view controller. Using one simple example, you have gained an understanding of how to use Coach Framework APIs to access business data and configuration options. You have also learned how to control coach view runtime behavior in event handlers and how to manipulate the DOM node of the coach view with Dojo core API. In addition, you have learned how to fire boundary events in a human service.

You have learned how to implement the advanced features of a custom coach view to integrate with the backend service using XHR calls. You have also learned how to support label, visibility, base text direction, localization, and to use the Content Box to embed other controls. The design-time support features make your custom coach view more consumable. Coach Framework is so scalable that you can use different JavaScript libraries to implement your custom coach views.

Remember the tips, recommended practices, patterns, performance considerations, and the development checklist in this chapter. They can help improve the quality of your implementation when delivering your custom coach view to be consumed by coach authors.

4

SPARK UI Toolkit

The SPARK User Interface (UI) Toolkit is an IBM BPM UI toolkit created by Salient Process, a Premier IBM Business Partner specializing in IBM Smarter Process consulting services and innovation.

IBM and Salient Process have partnered to make the SPARK UI Toolkit the UI toolkit of choice for IBM BPM customers. There are already efforts underway to incorporate the SPARK UI Toolkit into the IBM BPM product. For more information about this partnership and future plans for the SPARK UI Toolkit and IBM BPM UI, see 4.2, “Understanding the IBM and Salient Process partnership” on page 156.

This chapter includes the following topics:

- Introduction
- Understanding the IBM and Salient Process partnership
- Basic BPM UI concepts with SPARK
- UI layout
- Calling AJAX services
- Responsiveness
- Working tabular and repeating data
- Formulas
- Reporting and analytics
- Solutions for common patterns and requirements
- More on the SPARK UI Toolkit
- Conclusion

4.1 Introduction

The SPARK UI Toolkit enhances and streamlines the BPM UI creation process. It does so with controls and familiar UI development patterns that let UI developers focus more directly and efficiently on business problems.

This chapter provides a detailed introduction to key concepts, capabilities, and controls in the SPARK UI Toolkit.

Core concepts are described in 4.3, “Basic BPM UI concepts with SPARK” on page 157. The section builds upon that foundation to a full explanation of developing UI with the SPARK UI toolkit.

Note: As a UI developer, you are encouraged to read this chapter in its entirety to become expert in the use of the SPARK UI Toolkit. It is recommended that all developers using SPARK read, at a minimum, 4.3, “Basic BPM UI concepts with SPARK” on page 157 to take advantage of key benefits of the SPARK UI Toolkit.

4.1.1 Understanding the value of the SPARK UI Toolkits

The SPARK UI Toolkit offers the following benefits:

- Increases UI developer productivity up to three to four times faster than using traditional methods and decreases maintenance costs by avoiding UI complexity.
- Achieves the productivity increase through an efficient and intuitive development experience in combination with reduced skills expectations (primarily JavaScript, limited HTML or CSS, and no Dojo, AJAX, RWD, jQuery, or AngularJS required).
- Provides 90+ responsive and configurable controls, which can adapt to the form factor of the device running the coach and are suitable for both production and fast-build *proof-of-concept* scenarios.
- Includes, with every control, a simple and powerful event-based framework that creates a consistent approach for validation, formula-based computations, and cross-control interaction.
- Optimizes UI performance by using controls that support lazy loading and server-side pagination that can support complex UIs and large tabular data sets.

The SPARK External Participant Toolkit can extend the reach of IBM BPM to include external participants. For more information, see 6.3.2, “Expose human services indirectly through the EPS toolkit” on page 347.

The SPARK Portal Builder Toolkit provides a set of simple portal controls that build dashboards and custom portals including a Get Next Task capability. For more information, see 5.2.9, “SPARK Portal Builder toolkit” on page 305.

4.1.2 Developer experience

You can use the 90+ controls in the SPARK UI Toolkit to address a broad range of UI requirements. Though the core set of controls is relatively small (around 25), other more specialized controls (for example, Map, Slider, Signature, Video, Pop-up Menu, and Tooltip) help efficiently address more sophisticated UI requirements.

Detailed documentation and how-to articles are available at the following website for all SPARK UI Toolkit controls:

http://ibm.biz/BPMSparkUIToolkitControls

You can take advantage of patterns and capabilities that currently exist as preferred practices for IBM BPM UI development, as described in Chapter 2, “Creating user interfaces with coaches” on page 9. However, the SPARK UI Toolkit extends and streamlines the Coach Framework programing model in the following significant ways:

- Controls on a page or a view can easily refer to and talk to each other.
- Each control can react to all kinds of events, such as on click, on key press, on tab change, on timeout, on row deleted, and so on. Business logic can intuitively be expressed and attached to those events to create an interactive user experience.
- Reliance on data binding is optional. This eliminates complexity and *glue* artifacts commonly associated with BPM UIs. A data binding is only needed when a control must synchronize with meaningful business data.
- The enhanced SPARK UI development approach is designed after a well known effective and efficient development model of controls composed on a canvas with properties, methods, and events. The complexity stops there and all properties, methods, and events are well documented per control.
- When the values of controls and fields on a form are computationally related (whether numerical or not), spreadsheet formulas can be used to save time. These formulas automatically calculate and update values (in single fields or tables) on a page or view without you having to write code.

Note: BPM UI development can become more effective and efficient by exploiting the Coach Framework programming model enhancements provided by the SPARK UI Toolkit. The resulting solutions are consistently more lightweight, more maintainable, and more reusable.

4.1.3 Underlying patterns and principles

A foundational principle underlying the SPARK UI Toolkit is that it allows the developer to focus on solving business problems efficiently: you do not need to learn another complex JavaScript library or framework. With the SPARK UI Toolkit, you do not have to delve into the internal aspects of controls, duplicate assets, or create large amounts of glue constructs. There is no need to part with well-understood and intuitive UI development approaches.

Note: To use the SPARK UI Toolkit effectively, a you becomes familiar with the following four core patterns:

- Configuring controls
- Referencing and calling other controls
- Attaching logic to events
- Using formulas

These patterns are described in this chapter.

With an understanding of the four core SPARK UI patterns, you only have to look up the capabilities of controls and exploit those capabilities to implement business requirements. These controls include configuration options, events, and methods that you might not have

used before. Because all controls follow a strictly consistent approach across the four core patterns, gaining familiarity with a new control is a simple and predictable experience.

4.1.4 Modern, lightweight, consistent across BPM versions

With few exceptions (for example charting or maps), the SPARK UI Toolkit has no dependencies on extensive libraries and frameworks beyond the Coach Framework, HTML5, and CSS3.

This makes the toolkit extremely lightweight with controls that are inherently optimized to work with IBM BPM and the underlying Coach Framework. It also virtually eliminates reliance on thick layers of additional library or framework-specific processing and previous generation cross-compatibility behaviors.

The explicit HTML5 and CSS3 dependency, however, requires the use of a modern browser with comprehensive support for HTML5 and CSS3 such as Chrome, Firefox, Internet Explorer 11+, or Safari.

HTML5 and CSS3 support requirements for SPARK are more stringent than IBM BPM requirements. For example, whereas Internet Explorer 9 support is only deprecated for IBM BPM 8.5.7, it is not supported for the SPARK UI toolkit from IBM BPM 8.5.0.1 and up.

Note: The SPARK UI Toolkit requires a web browser with comprehensive support for HTML5 and CSS3.

IBM BPM is a constantly evolving platform. The introduction of new features can be valuable to some and disruptive to others, especially when a new feature entails the deprecation of an older feature.

SPARK takes full advantage of the latest IBM BPM product capabilities. For example, SPARK-based UIs in client-side human services (CSHS) can use the full SPARK programming model directly from client-side scripts to access page controls and manipulate them. The toolkit also attempts to normalize the experience across BPM product versions.

The cross-version normalization is done by relying on constructs that remain consistent from one product version to the next (for example SPARK-based UIs can be virtually Heritage- or CSHS-agnostic). By design, the toolkit works consistently across BPM 8.5.0.1 through BPM 8.5.7 and is jointly planned by IBM and Salient Process to be fully incorporated into the IBM BPM product.

4.2 Understanding the IBM and Salient Process partnership

In June of 2016, IBM announced a partnership with Salient Process to license the SPARK toolkits with the objective of incorporating the SPARK UI toolkit features into the IBM BPM Platform. More information about that partnership, including how you can order SPARK through IBM, can be found in the following IBM announcement:

http://ibm.biz/BPMSparkToolkitsAnnouncement

4.3 Basic BPM UI concepts with SPARK

This section presents key concepts that are specific to the SPARK UI Toolkit and can help you use the toolkit effectively and efficiently.

4.3.1 Controls and configuration properties

Control configuration, the most basic aspect of SPARK controls, is common with most other Coach Framework-based coach views. Figure 4-1 shows that it is also accessed in IBM Process Designer.

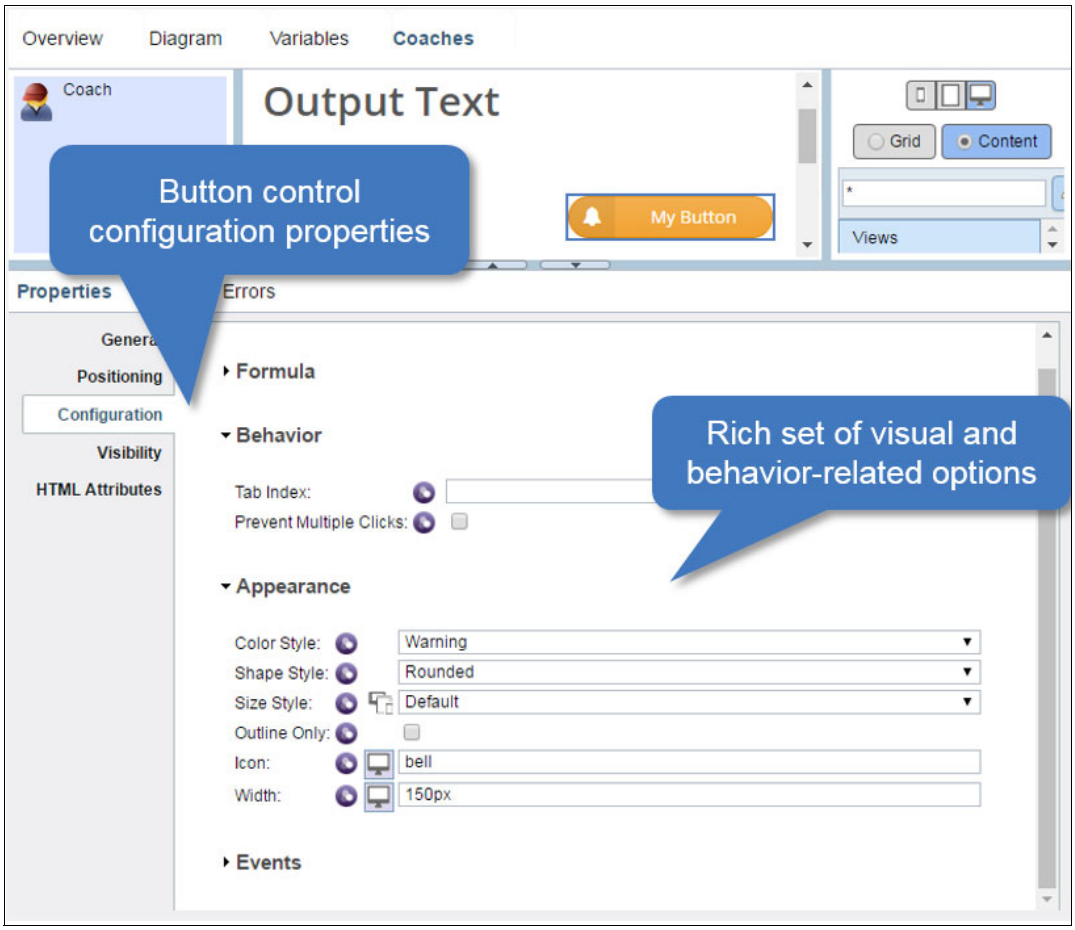


Figure 4-1 General SPARK control configuration

Common configuration categories

Many aspects of a control's appearance, behavior, and other runtime characteristics can be specified through configuration (and can be changed later at run time). For consistency in the UI development experience, most configuration options are grouped under consistent categories, such as:

- *Formula*: Used for the value computation of the control. For more information about formulas, see 4.9, "Formulas" on page 225.
- *Behavior*: Includes general behavior-related control-specific options.
- *Appearance*: Includes options such as styling, coloring, layout, typography, and labeling.

- *Performance*: Includes options to manage processing-intensive behaviors in repeating controls, such as tables and layouts for large data sets.
- *Responsive*: Includes view width-sensitive settings that automatically adjust layout and appearance of a control based on the view that contains it. Note that conventional page width-based adaptive configuration options are also available through the IBM Process Designer web editor.
- *Events*: Used to attached logic to various events specific to a control. For more information about formulas, see “Events” on page 161).

Figure 4-2 is an example of common configuration categories for a Button control.

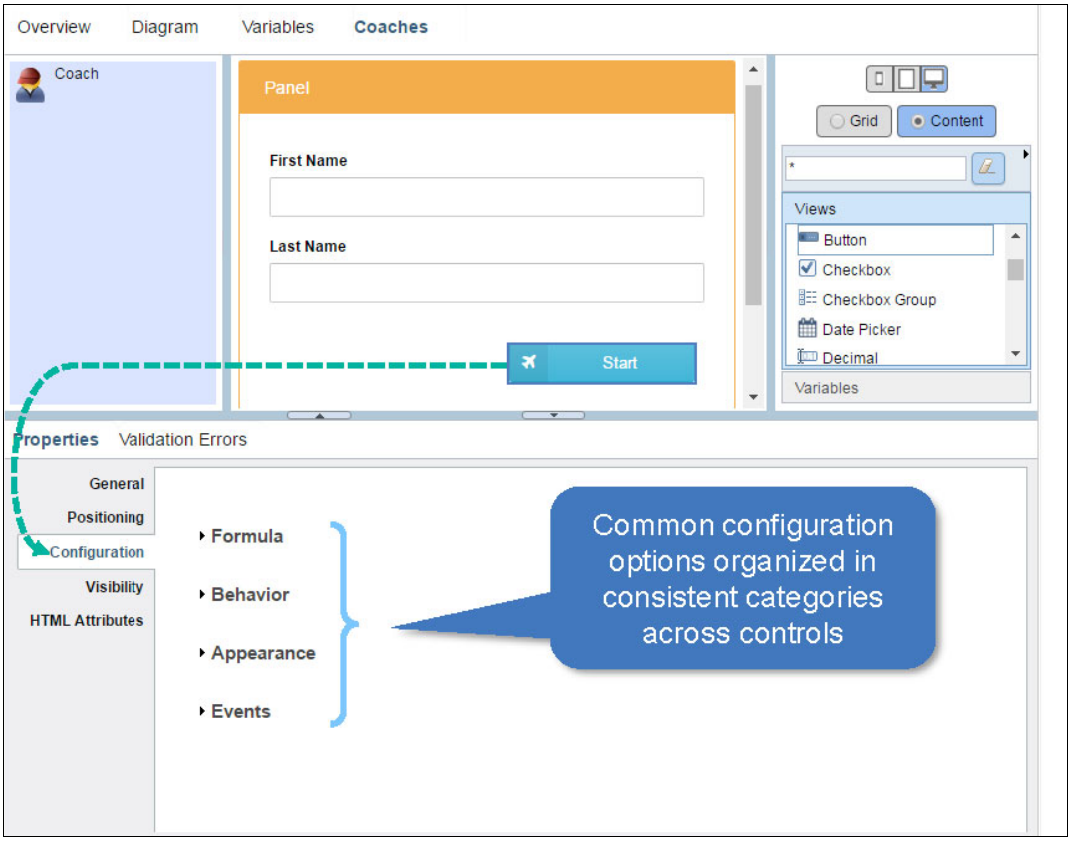


Figure 4-2 Extensive style configuration options example

4.3.2 Methods, events, and addressing

SPARK UI controls on a page or in views can access each other and call methods on other controls. They can also respond to events triggered programmatically or by user interactions (for example, button clicks, tabs changed, keys pressed, or data changed). The combination of event triggers, the ability to refer to other controls and to call various methods on those controls, provides a simple mechanism to create sophisticated behaviors.

Example 4-1 shows how a developer can change the text content of an Output Text control from a Button control on-click event.

Example 4-1 Addressing controls and calling methods

```
//Use button label to set content of the text control
text1.setText("Text set from button: " + button1.getText());
```

Example 4-1 is intuitive to UI developers because it is simple, familiar, and focuses on the problem. In that example, in the *on-click* event of a button can access another control and make it do things using calls to methods.

Methods

All SPARK controls have methods, such as *Getters*, *setters,* and various action methods. For example, these methods allow you to change the color of a button, set the title of a panel, expand a collapsible section, refresh a table backed by AJAX, make an image visible, and so on.

Example 4-2 shows how some methods abstract certain Coach Framework-specific constructs.

Example 4-2 How SPARK methods relate to Coach Framework methods

```
button1.setVisible(false)

…has precisely the same effect as
button1.context.options._metadata.visibility.set("value", "NONE")

Or
text1.setText("Text set from button: " + button1.getText())

…behaves exactly like
text1.context.binding.set("value", "Text set from button: " +
button1.context.binding.get("value"))
```

Other methods are specific to particular SPARK controls and have no counterpart in the Coach Framework, as shown in Example 4-3.

Example 4-3 Control methods unique to the Coach Framework SPARK UI extensions

```
table1.search(0, "Acme", false, true)
```

The method in Example 4-3 displays only rows in a table control where the first column (at index 0) contains the string `"Acme"`.

All SPARK controls provide documented methods and can be accessed from the SPARK support site. For information about a specific method, click the JS Doc link on the following web site:

http://ibm.biz/BPMSparkUIToolkitControls

Figure 4-3 shows methods for the Button control.

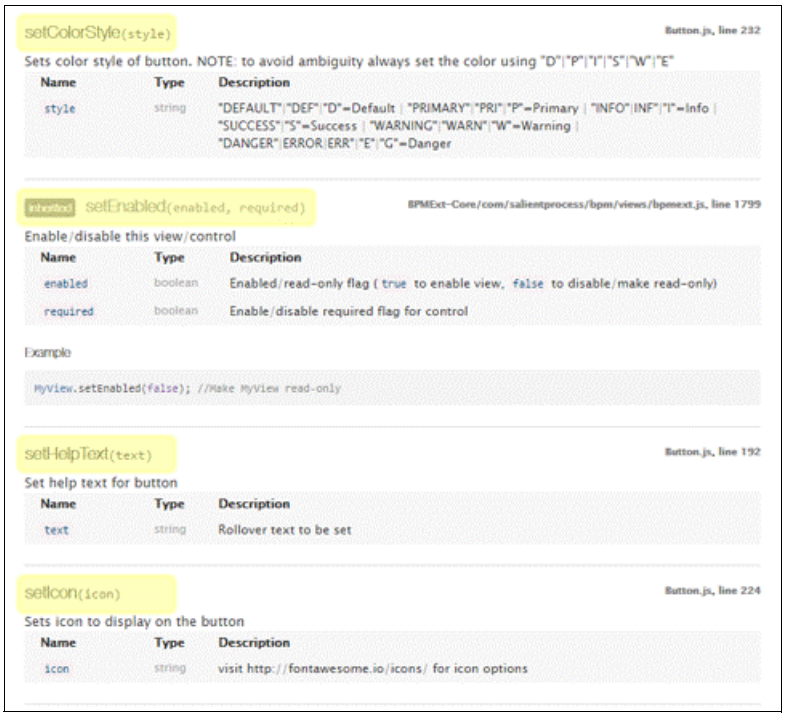


Figure 4-3 Online documentation for control methods

Common methods for all controls

Certain methods are common to all SPARK controls, Table 4-1 lists common SPARK control methods.

Table 4-1 Common SPARK control methods

Method	Description
`setVisible(visible, collapse)`	Sets view visibility. The collapse flag determines whether the space occupied by the hidden view is collapsed or displays as blank space. For controls that have no visual representation (such as the Event Subscription control).
`isVisible()`	Indicates whether a view is visible or not.
`setEnabled(enabled)`	Turns a view's editable state on or off (assuming the control supports such a state).
`isEnabled()`	Indicates whether a view is enabled or not.
`isBound()`	Indicates whether a view is bound to data or not.
`setData(data)`	Equivalent to `context.binding.set("value", val)`. Some controls provide more specialized methods such as `setText()` for a Text control or `setDate()` for a Date Picker control. Note that those methods are only aliases of `setData`.
`getData()`	Equivalent to `context.binding.get("value")`. Some controls provide more specialized methods such as `getText()` for a Text control or `getDate()` for a Date Picker control. Note that those methods are only aliases of `getData`.

Method	Description
`addClass(added, replaced)`	Adds, replaces, or removes a CSS class from the coach view top-level DOM element (referenced by `context.element`). *Added* is a string with zero or more space-delimited CSS class names to add to the view. *Replaced* is a string with zero or more space-delimited CSS class names to remove and replace with the added class names. To remove a CSS class from a view, specify two quotation marks (`""`) for *added* and the class names to remove for *replaced*.

Events

Events are key to interactivity in the SPARK UI Toolkit. They allow controls to respond to user, device, and various programmatic triggers.

The following actions are triggered by events:

- An alert displayed when a button is clicked
- Validation performed when an item in a drop-down list is selected
- An AJAX service fetching data when a collapsible panel is expanded
- A chart refreshed when a pop-up menu item is tapped
- A label updated with a new value every time a timer ticks
- A table that filters its content when a user types search text
- An output text displaying a result when an AJAX service returns

All SPARK control events are exposed in a similar way in the Events category of the control's configuration. Figure 4-4 provides an example of a button that shows an alert saying “Hello” and using the name of a person entered in the text control.

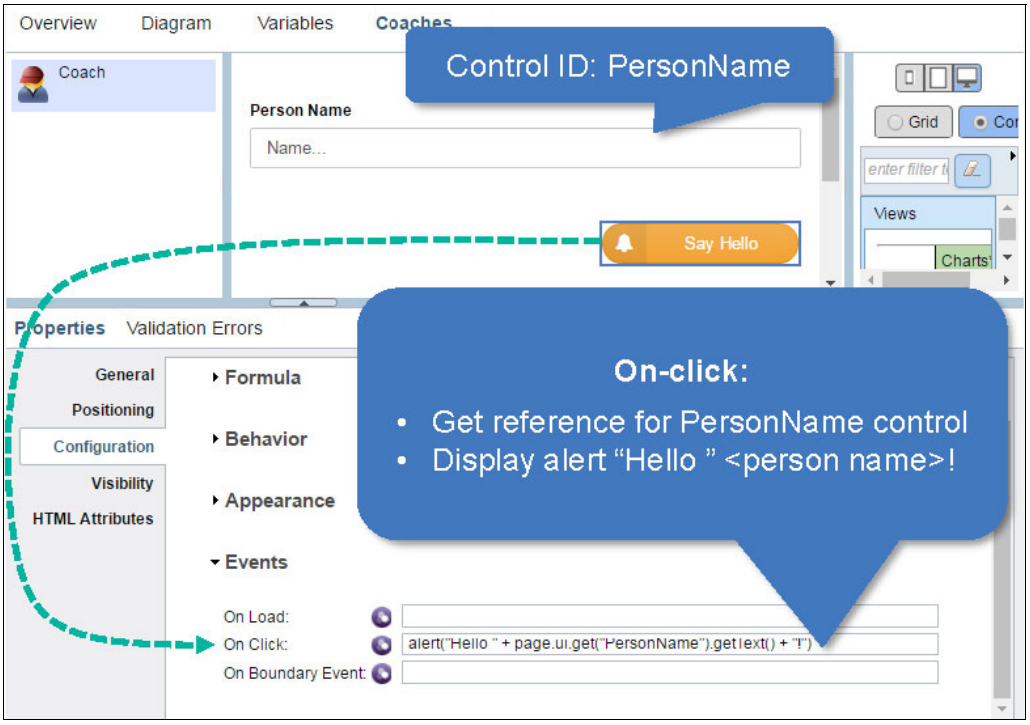


Figure 4-4 Specifying logic in a button click event

For a more compact and convenient notation in inline event logic, SPARK UI controls can be referred to in *shorthand* using the `${<control-id>}` notation. The inline event logic can be rewritten as the notation shown in Figure 4-5.

Figure 4-5 Shorthand notation for control references in inline event logic

TIP: Do not confuse the `${control-id}` reference lookup notation with the jQuery `$("<selector>")` notation. The SPARK UI Toolkit does not depend on jQuery, though it can coexist with it.

Note: The SPARK `${ }` notation is only valid when used in inline event logic. Proper syntax expects curly braces only and does not use quotes for the control-id reference lookup notation. The `${PersonName}` notation is valid, whereas `${"PersonName"}`, `$(PersonName)`, or `$("PersonName")` are not valid.

Events and Coach Framework boundary events

Buttons, icons, and other controls emit boundary events when clicked or otherwise activated. For most of those controls, the events fired before emitting the boundary event, such as on button click, can inhibit the boundary event by explicitly returning `false`.

In Figure 4-6, the boundary event inhibitor pattern provides a convenient way to add confirmation before navigating away from a coach.

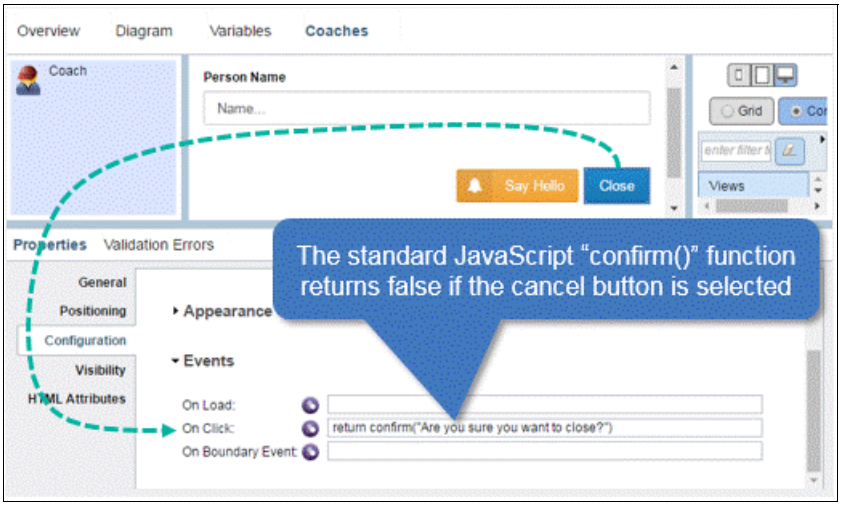


Figure 4-6 Boundary event inhibitor pattern (design time)

Figure 4-7 shows the resulting runtime behavior.

Figure 4-7 Boundary event inhibitor pattern (runtime behavior)

Event context variables

Sometimes context is important when handling events. For example, a Text control might need to restrict the length of the content or the input of certain characters or values. To do this the input event logic examines the attempted input, passed in the on input event as a context variable, before allowing or not allowing the new potential content.

Figure 4-8 illustrates how to implement this scenario.

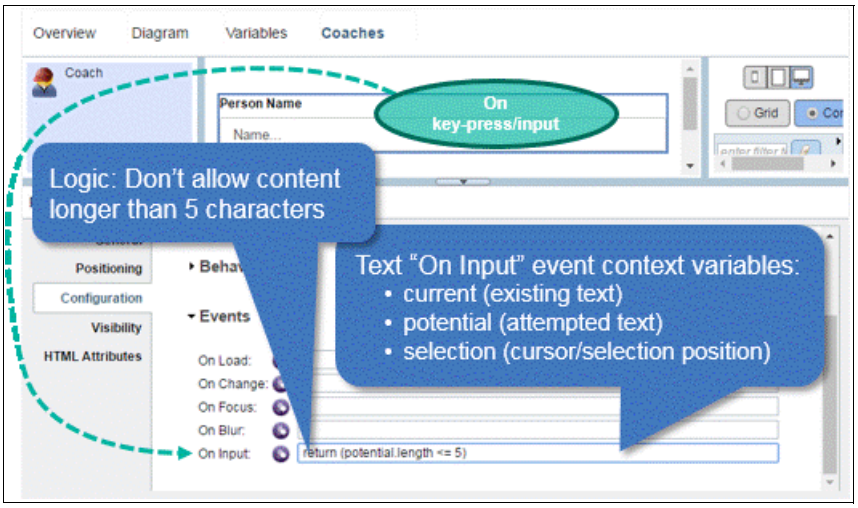


Figure 4-8 Text "On Input" event using context variable

Table 4-2 shows the context variables that the Text > On Input event provides to the inline event logic.

Table 4-2 Content variables for the Text > On Input event

Variable Name	Description
`me`	Similar to the *this* concept: Refers to the control that emitted the event.
`view`	Refers to the control's first parent view that is not only decorative. Often this means the composite coach view that contains the control.
`current`	Text content in the control before the change from the input.
`potential`	Attempted content. This becomes the new text content unless the logic returns false.
`selection`	Location of the cursor in text control or boundaries of the selection if the text was selected and replaced by new content.

The JS Doc for each control describes the methods and the events supported by the control and associated context variables as appropriate. See the JS Doc links for each control in the online SPARK control reference, which is available at:

https://support.salientprocess.com/spark-ui-controls/

Note: The `me` context variable is available in all events for all controls. It is a convenient reference to the control that emitted the event. The `view` context variable is also similarly available and points to the view containing the control.

Because these variables are common to all controls, they are not explicitly mentioned in the JSDoc as context variables.

Invoking non-inline event logic

Inline events are simple and convenient. If, however, IBM Process Designer only allows a single line for those events, specifying a lot of inline logic can be cumbersome.

Because inline event logic is only JavaScript at run time, functions can easily be called from an event handler. In Figure 4-9 on page 165, the `calculate()` function, which is defined in a script block at the page level, is called on a button click.

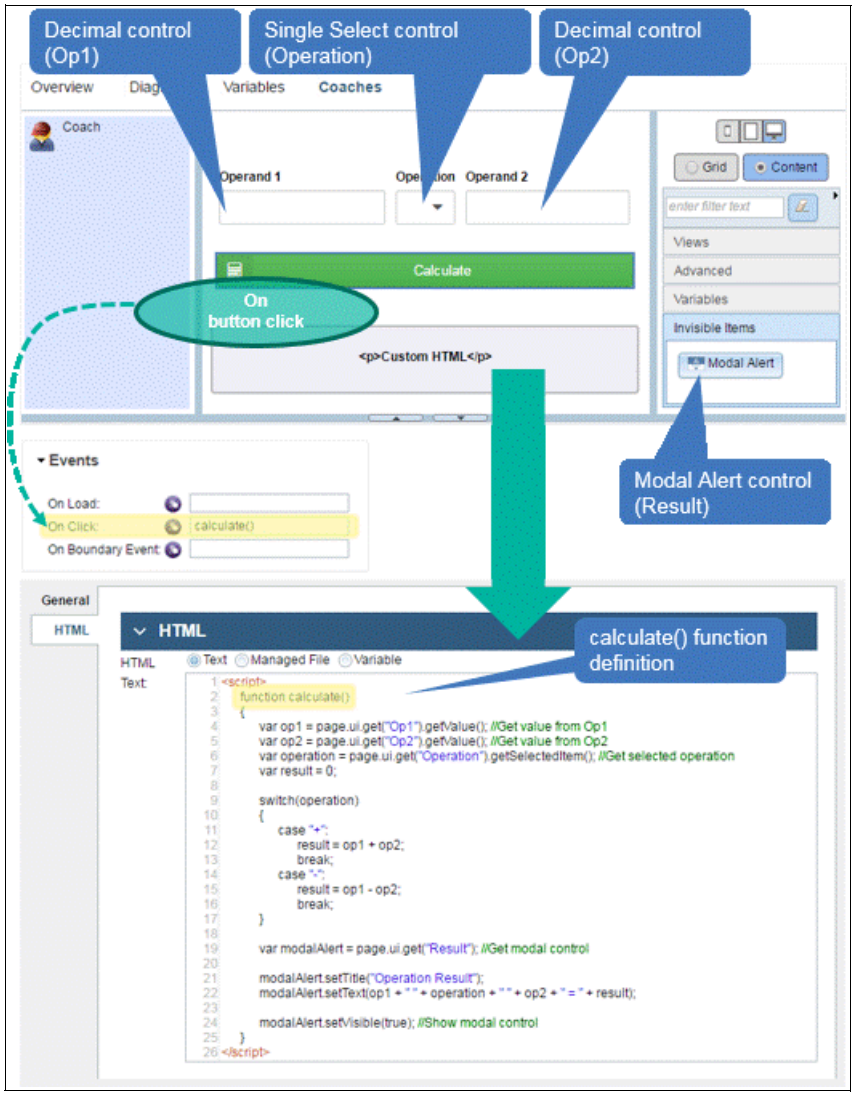


Figure 4-9 Calling a page-level function from an event handler

Note: There is no limitation to the globally-defined functions and globally-accessible objects that can be called from an inline event handler. If a function requires parameters, they can be passed in regular JavaScript manner. Context event variables can also be passed to externally-defined functions.

A key benefit that a reusable coach view must include is the ability to encapsulate not only the child views laid out on the containing view's canvas, but also their behavior. The SPARK UI Toolkit allows a composite coach view to host the business logic that is invoked by the various controls that it contains. This ability creates a self-contained, fully encapsulated custom reusable coach view made from smaller parts.

Hint: Full coach view encapsulation, which increases reusability and maintainability, is a significant benefit of the SPARK UI Toolkit.

Figure 4-10 shows how the Calculate button now calls a `calculate()` function defined in the containing `[view.calculate()]` composite coach view. It is located in the Inline JavaScript section of the coach view and not at the page level.

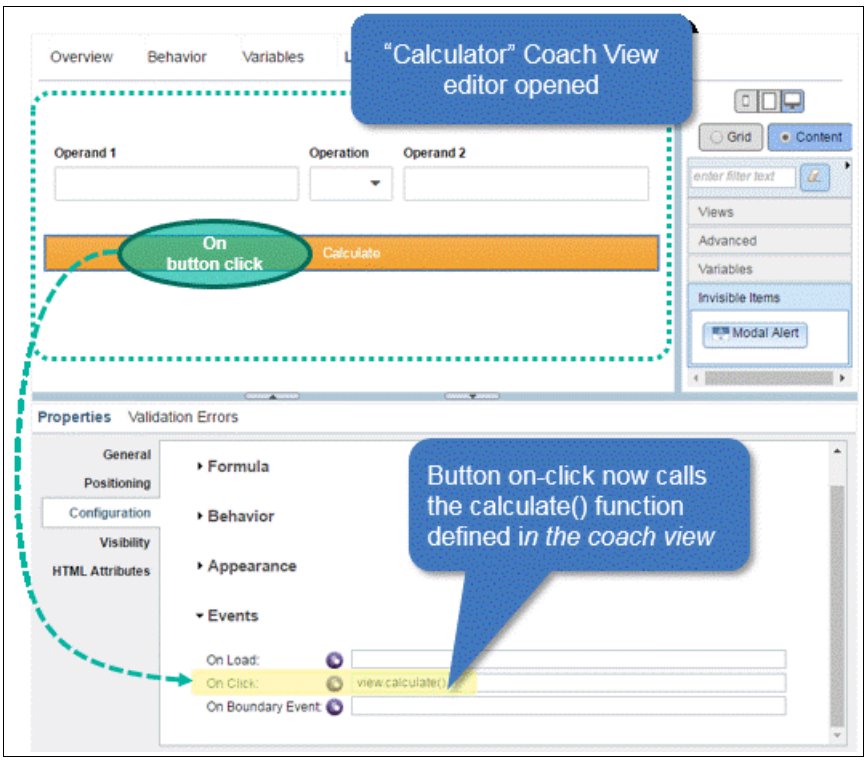


Figure 4-10 Calling a composite coach view-level function from an event handler

Figure 4-11 shows the definition of the Calculator composite coach view's `calculate()` function.

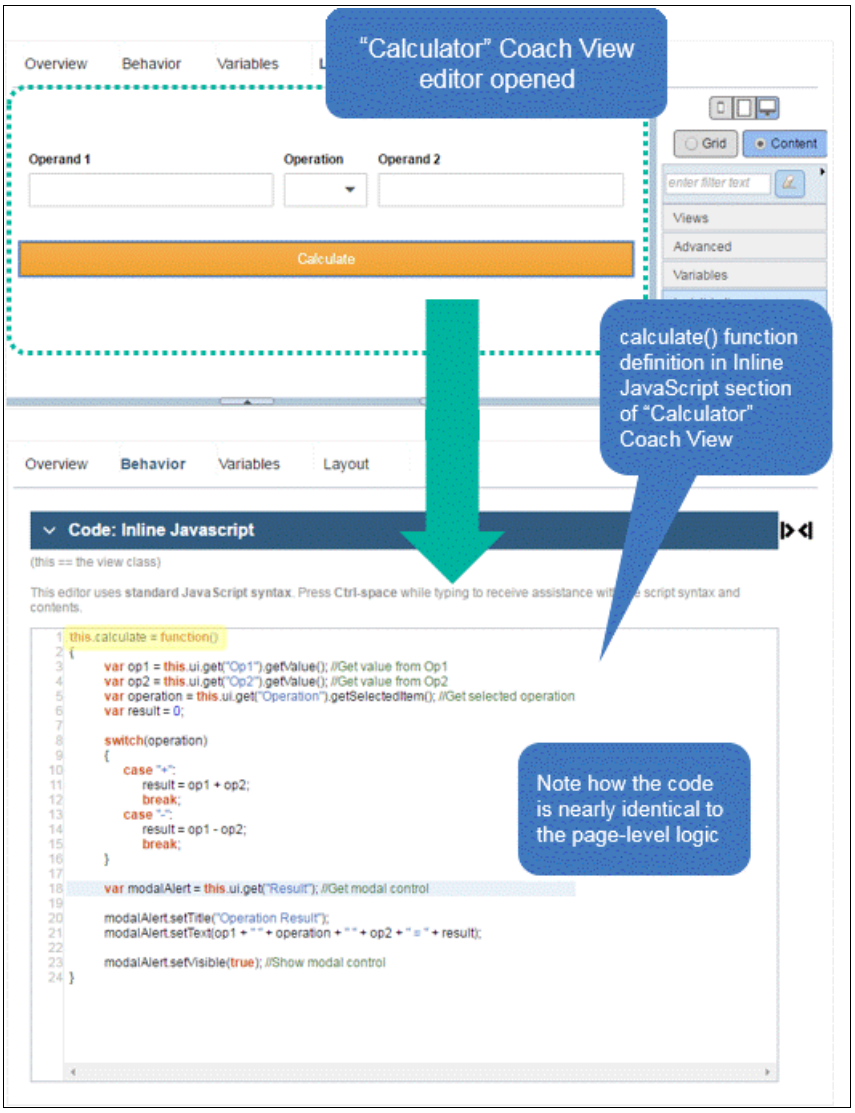


Figure 4-11 Defining a method at the composite coach view level

Note: In a simple step, nonreusable code at the page level is made reusable. The Calculator coach view relies on no external scripts or components to perform its work.

This is the fastest and most intuitive general approach to building reusable coach views.

Client-side script-based event logic

As described in "Events" on page 161, SPARK provides various ways of handling business logic in response to events. The most portable approach is to encapsulate the business logic in a composite coach view, as illustrated in Figure 4-11.

If, however, encapsulating logic is not important, use client-side scripts and boundary events to achieve a similar effect. This is done in a conventional Coach Framework manner and by using a part of the programming model extended by SPARK, as shown in Figure 4-12 on page 168.

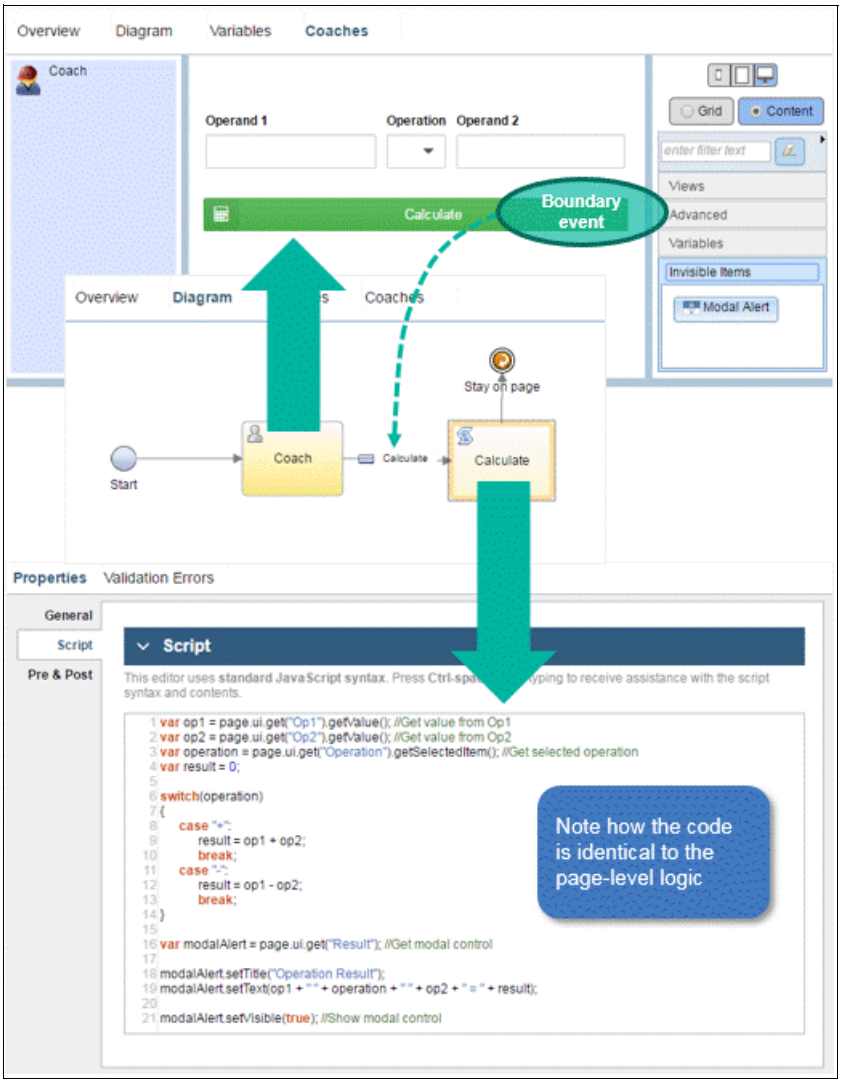


Figure 4-12 Using the SPARK programming model in a client-side coach

Firing boundary events from any event handler

Figure 4-12 shows how SPARK can make use of client-side scripts in the same way as the classic Coach Framework-based approach. Not all controls fire boundary events, however, and even for those that do, a boundary event cannot always be fired by the interaction the developer needs.

SPARK overcomes this limitation by providing a way to fire boundary events for any kind of event-triggered interaction. This is done through the Navigation Event control.

Figure 4-13 on page 169 shows an adapted version of the Calculator scenario with results computed using a Navigation Event control instead of a Button. In this scenario, the calculation is triggered every time a change occurs in the operands or the operation.

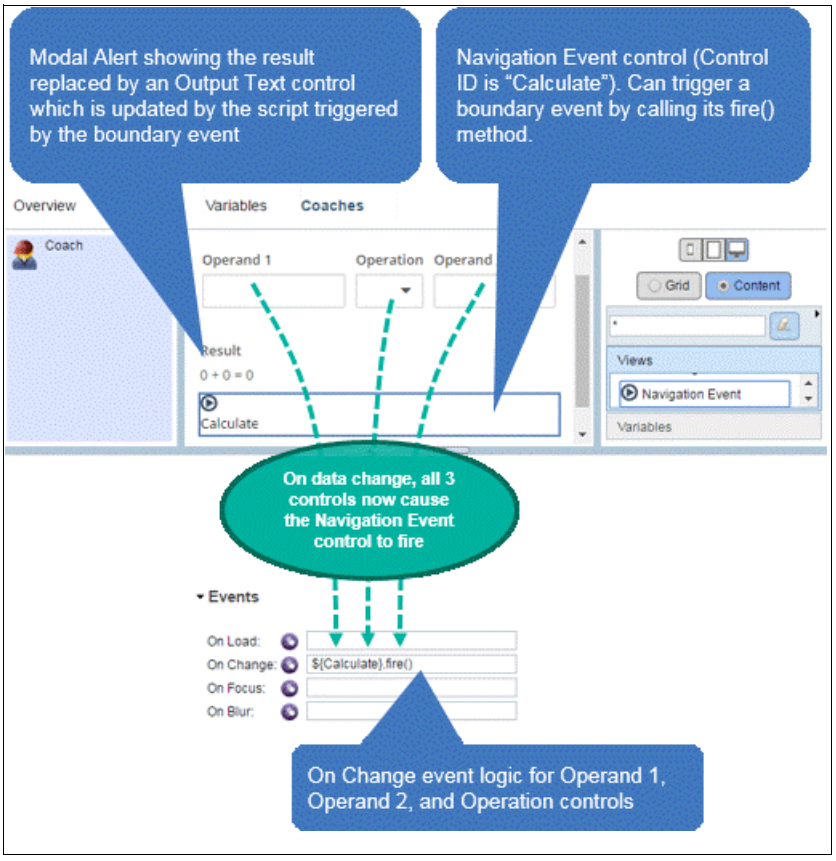


Figure 4-13 Firing boundary events from SPARK event handlers

Note: SPARK allows you to use many boundary events to execute business logic, as shown in Figure 4-13. However, boundary events can clutter the CSHS flow and are often not the best way to achieve deep reusability. Handling events at the coach-view level is more logical and easier to maintain.

You can be more productive with a less *mixed* SPARK approach; this approach results in a more streamlined solution.

Events: Summary

Handlers for a multitude of events are available on SPARK controls. Logic is run from any of those event handlers by calling methods on controls.

Event logic can be inlined or a function defined globally. Or, event logic can be called in a control's containing composite view.

Functions called in the parent composite coach view are defined in the view's Inline JavaScript section (see the coach view's Behavior tab in IBM Process Designer), as shown in Example 4-4.

Example 4-4 Event-callable function defined in a composite coach view

```
this.myFunction = function(<parameters if needed>) {
 /* Business logic... */
}
```

Example 4-5 shows how myFunction can be invoked from an event handler of the child control.

Example 4-5 Calling a function defined in a composite coach view from a control's event handler

```
view.myFunction(<parameters if needed>)
```

Controls can be referenced from event handlers using the following commands:

- The **page.ui.get(<*control-id*>)** command is at the page level or in client-side scripts.
- The **this.ui.get(<*control-id*>)** command is in functions defined in coach views (Inline JavaScript) to refer to the child views contained in the composite coach view.
- The **${<*control-id*>}** command is used only if specifying *inline* event logic. The **${ }** notation is translated to a real JavaScript control reference at run time).

Using the **${<control-id>}** command is a convenient way to refer to controls in inline event logic. The command works consistently at the page level and in a coach view. It is recommended that you use this command.

Lastly, your practice might be to create Human Service diagrams with dense wiring and client and server scripts for business logic. However, the SPARK UI event pattern, combined with pertinent SPARK control methods, helps minimize clutter, increases reusability, and focuses the developer on solving the business problem.

Tip: Using SPARK controls without taking advantage of the Coach Framework programming extensions contributed by SPARK forgoes a significant productivity and simplification advantage provided by the toolkit. When you use the SPARK UI Toolkit it is important to exploit its extended programming model.

Control referencing

Understanding how control referencing works is an important part of using the SPARK UI Toolkit effectively.

All examples of control referencing noted thus far (such as using the **${<control-id>}** commands, **page.ui.get("<control-id>")** or **view.ui.get("<control-id>")**) are straightforward. If you use SPARK and follow standard component encapsulation practices, you can be productive working with this simple addressing (in addition to working within repeating controls in tables, which is covered in 4.7, “Working tabular and repeating data” on page 211).

Tip: It is important to understand the structure and implications of SPARK addressing. This understanding helps you debug BPM UIs and exploit the programming model in creative ways to solve complex requirements.

Control referencing works like a directory tree in the SPARK UI Toolkit. However, control referencing is a view tree. The *address* of a view can include not only a control's ID but also indicators, such as *"/"*, "**..**", "[<index>]", that help navigate the view tree like a directory structure using relative or absolute addressing.

Figure 4-14 shows the *Simple Calculator* coach composition.

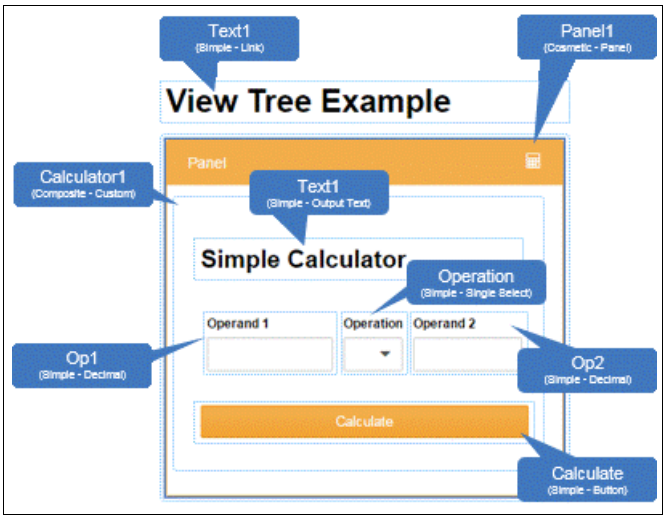


Figure 4-14 Composition of simple and composite coach views

The UI is represented as a tree, as shown in Figure 4-15.

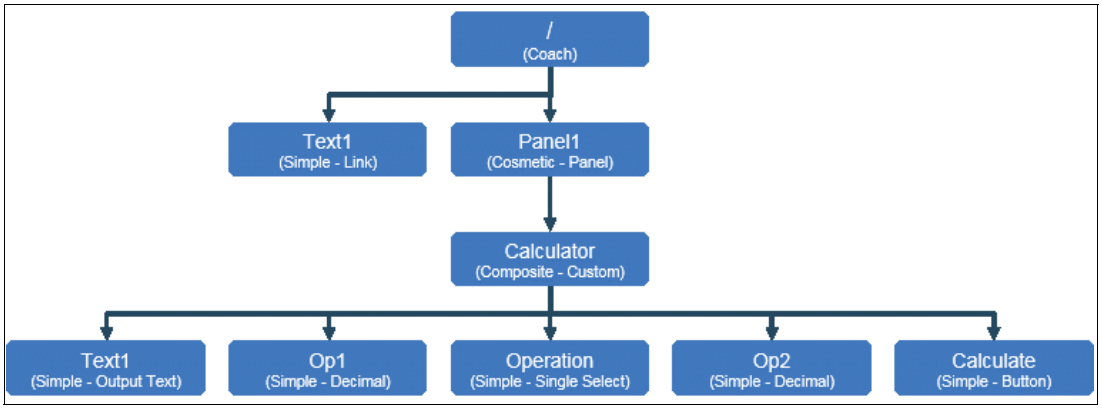


Figure 4-15 Control tree representation of a coach composition

Addressing in SPARK works as noted in the following list:

- A parent view that contains children controls can access its *immediate* children using the command:

  ```
  <parent view ref>.ui.get("<child view's control id>")
  ```

- A view that needs to access a sibling can do so in two ways:
  - Directly:

    ```
    <view ref>.ui.getSibling("<sibling view's control id>");
    ```

  - Or through its parent view:

    ```
    var parent = <view ref>.ui.getParent();
    parent.ui.get("<child view's control id>")
    ```

Recap: A parent (composite) view (view1) can access an immediate child with the **`view1.ui.get("<child control-id>")`** command. A view (view2) can access a sibling using the **`view2.ui.getSibling("<child control-id>")`** command.

Understanding inline event handlers and ${<control-id>} references

When the inline logic of a control's event handler refers to a control ID, for example `${Text1}`, the query is always done from the context of the parent view (or the context of the coach if there is no parent view), and not from the event's own emitting view.

At run time, the `${Text1}` reference from the event handler's inline logic translates to `me.ui.getParent().ui.get("Text1")`.

Note: An inline event handler, though associated with a control, runs in the context of the control's parent composite view (or the coach if there is no containing composite view). Use the *me* context variable to refer to the control emitting the event and the *view* context variable to refer to the parent composite view.

Special case for cosmetic controls

If the coach, or coach view composition, is arranged like a tree with parents and siblings (and descendants) the addressing scheme can break by moving controls around to different levels of the tree.

For example, after having created logic between a Button and a Text control (`Id: Text1`), the reference to Text1 control breaks if moved in a Panel control (`Id: Panel1`) for cosmetic reasons and the reference must changed from `${Text1}` to `${Panel1/Text1}`.

To prevent such problems, most container controls in SPARK whose purpose is to group or arrange or wrap around other controls are purposely not considered in the addressing hierarchy. Examples include panels, layouts, tab section, wells, and input groups. This flattens the view hierarchy, depending on how many cosmetic-only controls are present in a UI and how much they are nested. It also solves the problem of potential reference breakage from visual control rearrangements.

Figure 4-16 shows the coach view tree from Figure 4-15 on page 171 when adjusted to account for SPARK's handling of cosmetic coach views.

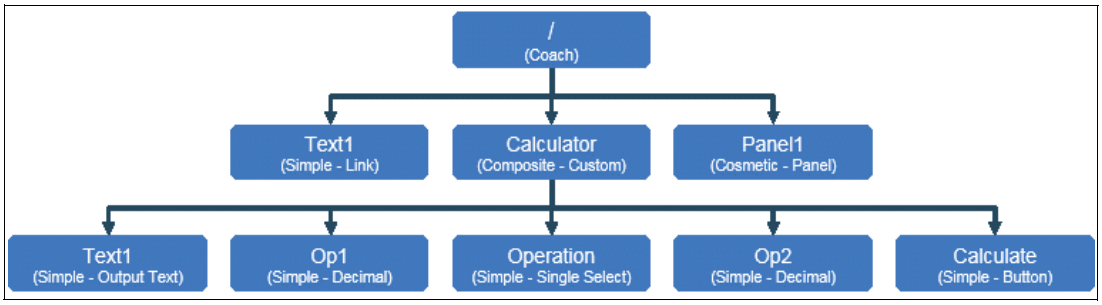


Figure 4-16 Control tree representation adjusted for cosmetic-only controls

Table 4-3 provides the list of controls that are considered cosmetic for SPARK UI addressing.

Table 4-3 List of cosmetic SPARK controls (not inserted in a control's addressing path)

Control	Notes
Caption Box	Caption Box is a container that wraps around a control to add a label or caption around it.
Collapsible Panel	
Deferred Section	

Control	Notes
Horizontal Layout	Unless bound to a list, which makes it a repeating container.
Input Group	Input group is a container that wraps around a control to add a button or icon to the left or right of the control.
Modal Section	
Page Layout Row	Deprecated after IBM BPM 8.5.0.1.
Page Layout Column	Deprecated after IBM BPM 8.5.0.1.
Page Layout Cell	Deprecated after IBM BPM 8.5.0.1.
Panel	
Panel Header	
Panel Footer	
Popup Menu	Popup menu is a container that wraps around a control to add a pop-up menu to it.
Stack	Stack is like a tab section without any tabs or decorations around it. It only show one pane at a time.
Status Box	Status Box is a container that wraps around a control to add a status message bubble under it.
Tab Section	
Table Layout	Deprecated after IBM BPM 8.5.6.
Table Layout Row	Deprecated after IBM BPM 8.5.6.
Table Layout Cell	Deprecated after IBM BPM 8.5.6.
Tooltip	Tooltip is a container that wraps around a control to add a tooltip to it.
Vertical Layout	Unless bound to a list, which makes it a repeating container.
Well	

Note: The control ID of Horizontal and Vertical Layout containers becomes part of the address of the child controls they contain if the layouts are bound to a list. A layout is bound to a list if they layout is configured as repeating containers.

Accessing controls in repeating containers

The Table control in the SPARK UI Toolkit is an example of a repeating control. At design time, only the first row is visually modeled. For example, a Table control (Id: Table1) might contain a Text control (Id: Text1) and a Button control (Id: Button1).

In the most use cases, there are three consistent patterns that control referencing needs to support. The following examples refer to a Table control, but control addressing must behave similarly for any repeating container:

- From outside of a table, a control needs to access a control that is in a table by control ID and by row index.

- From inside of a table, a control in a table column needs to access a control in another column but in the same row.
- From inside of a table, a control in a row needs to update a control outside of the table.

Figure 4-17 shows the inline logic of an on-click event to update the Text control in the fourth row of the table.

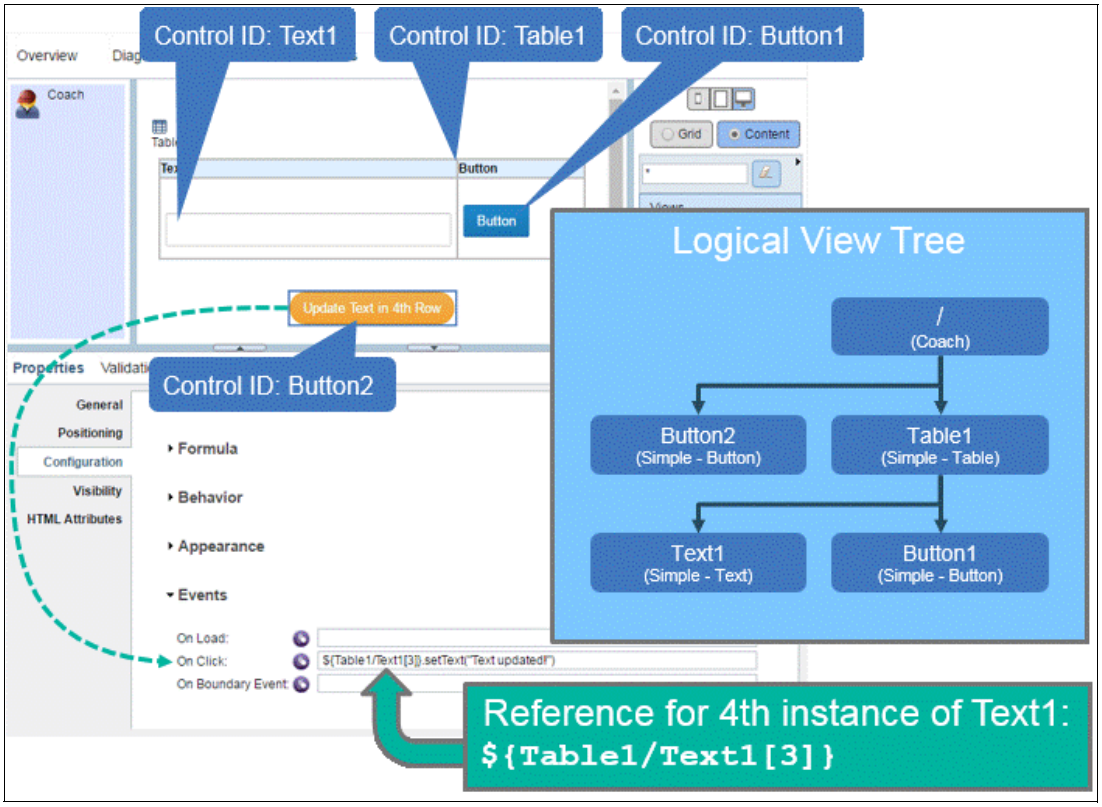


Figure 4-17 Event referring to nth repeating Text control in table

The example in Figure 4-18 shows how a control can access another one in the same row.

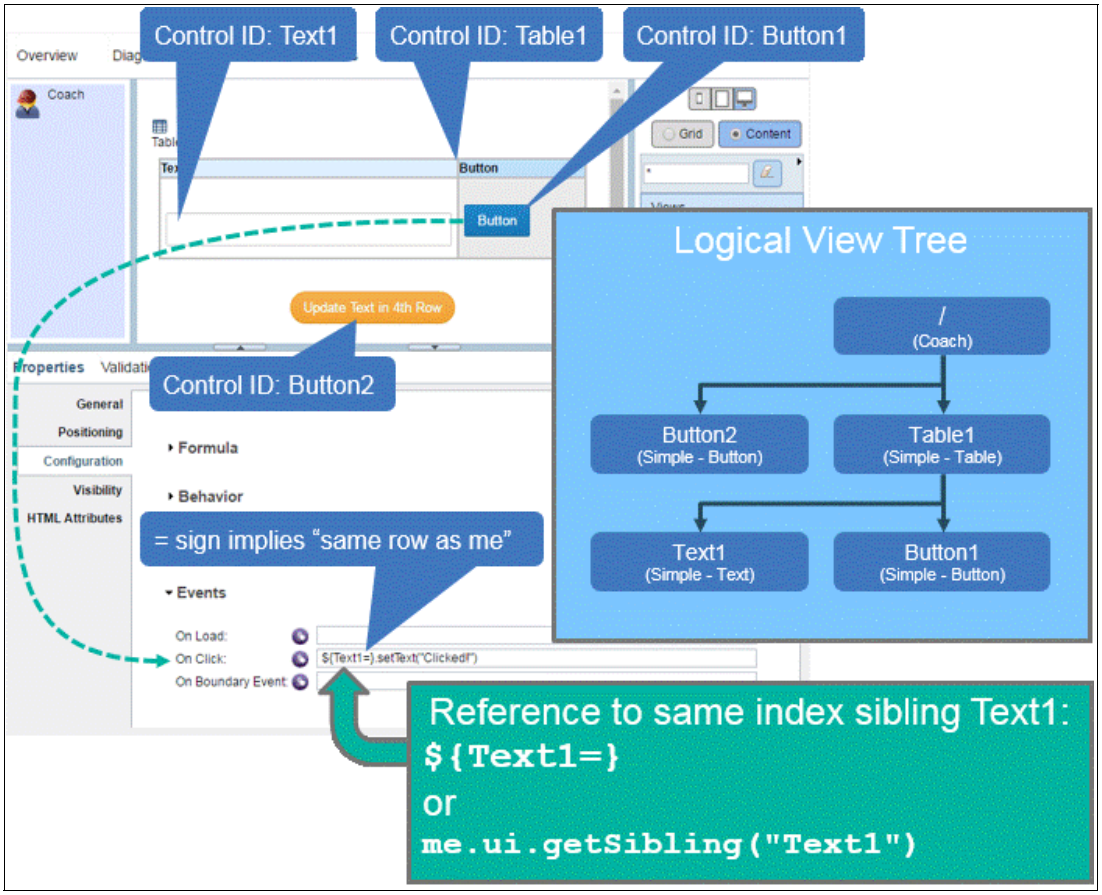


Figure 4-18 Event referring to Text control in table in same row as control emitting event

The runtime behavior of the examples in Figure 4-17 and Figure 4-18 is shown in Figure 4-19 on page 176.

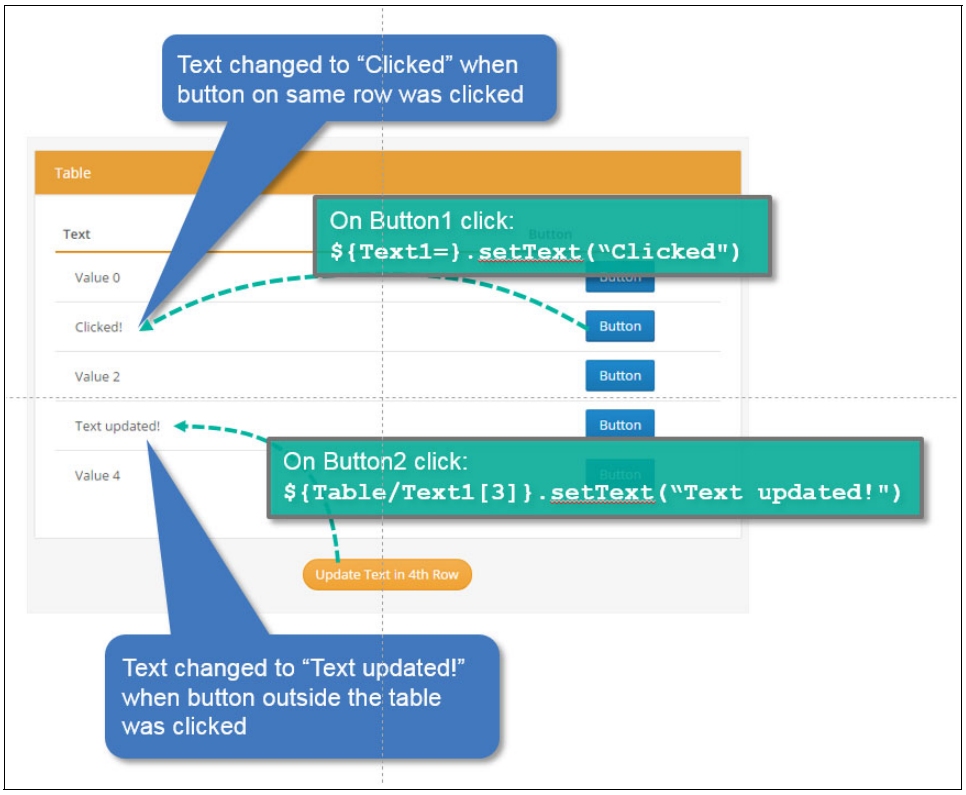


Figure 4-19 Control addressing and events with repeating table, runtime behavior

Note: Achieving the same behavior, as shown in Figure 4-19, without SPARK addressing using classic Coach Framework capabilities requires a more complicated and cumbersome approach. This approach uses wrapper composite coach views.

The third example in Figure 4-20 on page 177 shows how a control in a table can refer to another control outside the table using relative addressing.

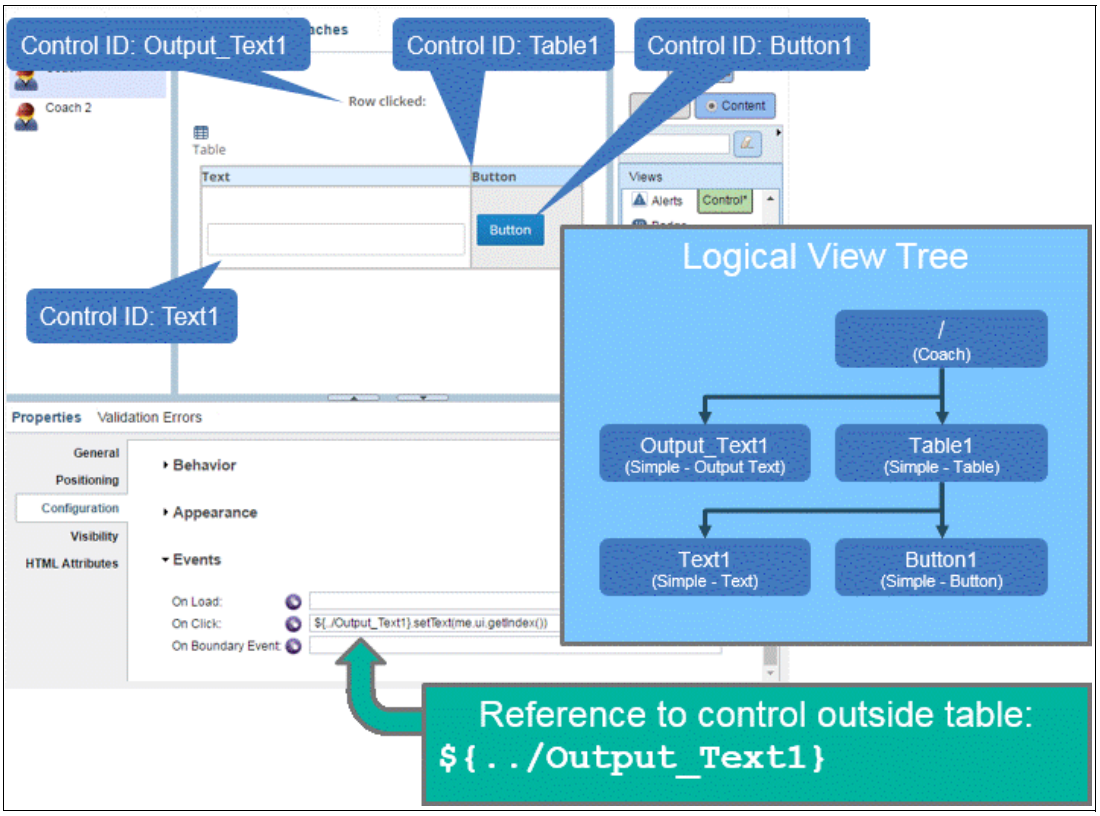


Figure 4-20 Event referring to Output Text control outside of table

Figure 4-21 shows the runtime result for the third example in Figure 4-20.

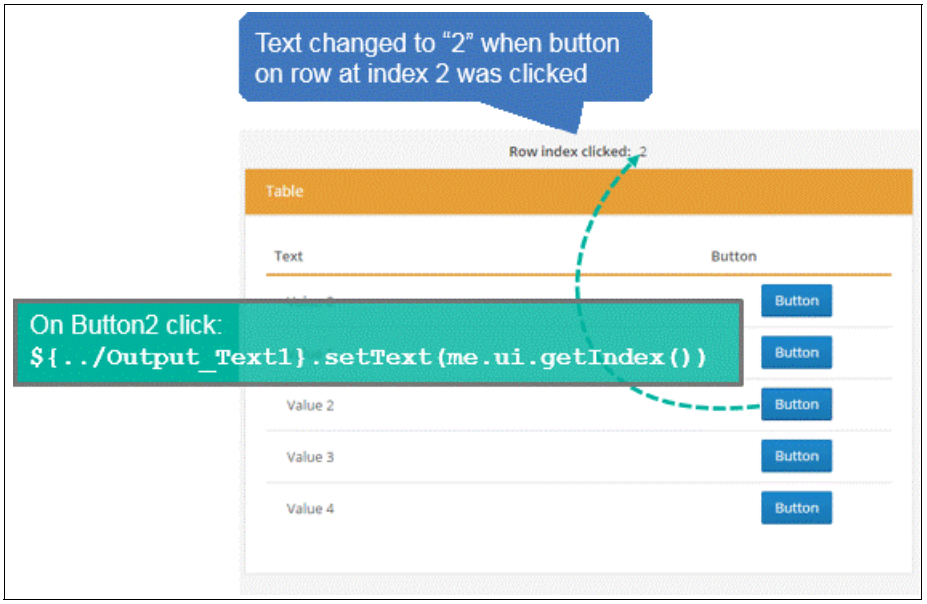


Figure 4-21 Control addressing from inside to outside of repeating table, runtime behavior

Tip: In the event handler of a repeating control, the **`me.ui.getIndex()`** command provides an easy way to determine the row or list index of the control whose event fired.

Tip: Using relative addressing is a better practice than using absolute addressing, especially within a composite coach view. Absolute addressing often requires the logic of a composite coach view to know details about the naming and structure of its parent. That logic can be altered by other UI developers and break absolute references.

More information about tables and repeating containers is covered later in this chapter (see 4.7, “Working tabular and repeating data” on page 211). This section addresses only the topic of typical addressing usage with repeating content.

4.3.3 Optional data binding

Data binding to controls is a valuable and convenient capability. It allows controls bound to data either through data binding or through configuration options to automatically synchronize their state based on the bound business data and vice versa. This capability is provided by the Coach Framework ready for use. The majority of SPARK UI controls also support this capability.

Most SPARK UI controls, however, work almost identically whether they are bound to business data or not. They do not need to be bound to data to be functional. This allows the UI developer to only use business data structures for *business data purposes*. There is no need for data holder structures (Business Objects) whose primary purpose is often to only back UI control states, even when those controls do not represent a meaningful business or process data concept. This can significantly reduce solution clutter and allows reusable coach views to have less external dependencies to function properly.

When a SPARK UI control is not bound to data, all its methods work as though it were bound to data, but no data synchronization occurs. In either case, all SPARK UI controls support the methods shown in Table 4-4.

Table 4-4 Data-related methods for SPARK controls

Method name	Description
`getData()`	Same as `context.binding.get("value").`
`setData(val)`	Same as `context.binding.set("value", val).`
`isBound()`	Indicates if the control has a real binding from IBM BPM or a synthetic binding from SPARK.

Note on aliases: Many controls provide more intuitive aliases to the **`getData`** or **`setData`** commands, for example **`getText`** or **`setText`** for the Text control, **`getDate`** or **`setDate`** for the Date Picker control, and so on. Regardless, the **`getData`** and **`setData`** commands always work on controls.

Tip: Only use data bindings to update or react to changes in legitimate business data that is part of the human service.

Data change events without a data binding

Many SPARK controls provide support for a change event. The change event works the same way even if the control is not bound to data, as shown in Figure 4-22.

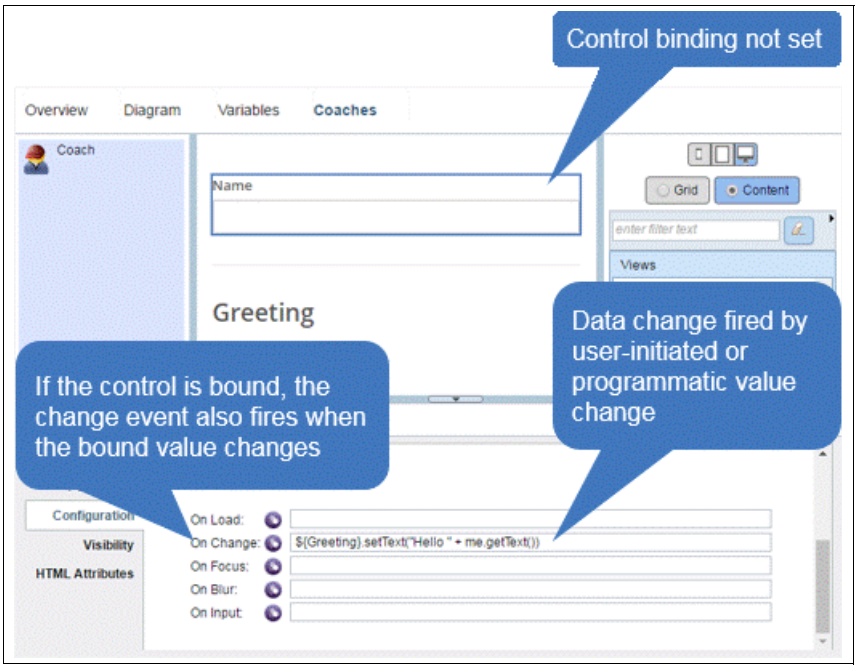


Figure 4-22 Data change events for bound or unbound controls

4.3.4 Validation

All SPARK UI control types that support validation visually reflect an invalid state. However, SPARK controls that support validation do not strictly depend on the conventional Coach Framework-based validation approach described in 2.7, “Adding validation to a coach” on page 39. However, the controls behave as expected with conventional validation.

The SPARK UI Toolkit and its controls provide support for combinations of the following types of validation behaviors:

1. Incorrect input is prevented in the first place. In this case, the control is never in an invalid state.
2. A control allows invalid input but flags the issue during typing.
3. A control allows invalid input but flags the issue after it loses focus.
4. Some controls contain invalid input, but submission is prevented until errors are resolved.
5. Some controls contain invalid input, and issues are flagged after submission.

Client-side validation is well-suited for cases 1, 2, 3, and 4, assuming the validation needed does not rely on server-side rules. Server-side validation, including AJAX-based validation, can also work well for cases 3, 4, and 5.

Client-side validation

Client-side validation means that logic can be triggered to examine the content of one or more controls as content changes. Then validation errors can be flagged on one or more controls in the composite view containing the controls or on the entire coach.

Figure 4-23 shows how the simplest kind of client-side validation can give user feedback as content is typed or as the control loses focus.

Figure 4-23 Validation error reported on a Text control

Figure 4-24 shows how this kind of behavior can be modeled with a regular expression on the Text control.

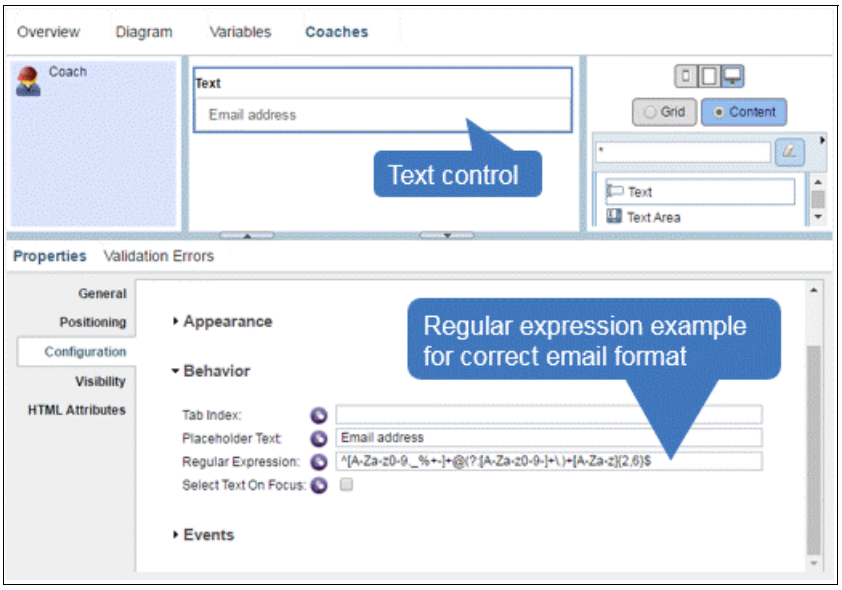


Figure 4-24 Example of validation through regular expression on Text control

Note: In the context of text validation, a regular expression is a character pattern that defines the allowable format, characters, numbers, or symbols that can be entered in the field. For more information about regular expressions, see the regular expression website:

`https://en.wikipedia.org/wiki/Regular_expression`

Other controls such as Masked Text provide build-in capabilities to restrict typed content through input masks, as shown in Figure 4-25 on page 181.

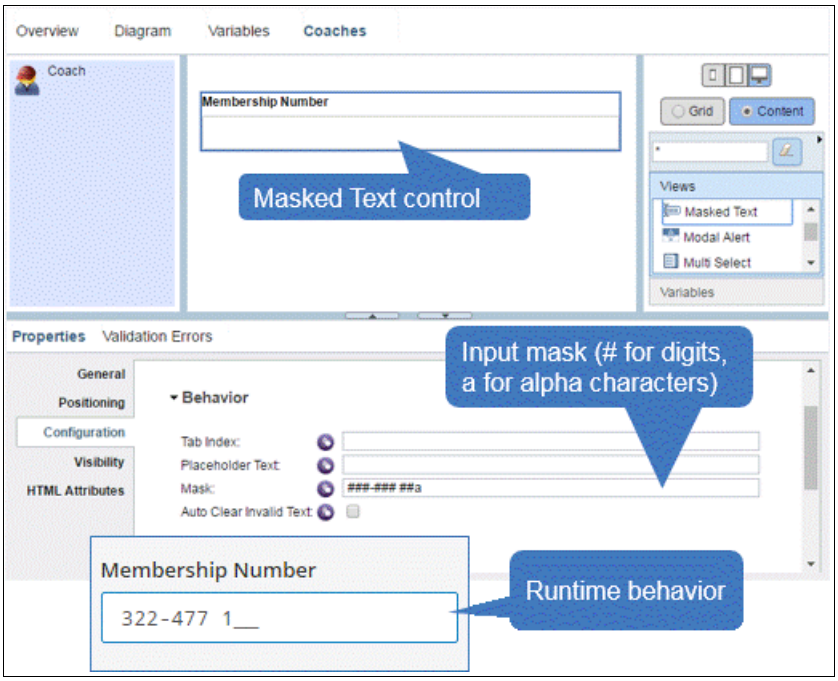


Figure 4-25 Example of validation enforcement with Masked Text control

Programmatic validation

Although configuration-time validation can be useful, the most flexible type of validation is programmatic. Table 4-5 lists the four methods provided by the extended SPARK programming model to assist with client-side validation behavior.

Table 4-5 Client-side validation-related methods for SPARK

Method name	Description

`<view>.setValid(`
`  flag,`
`  errorMessage`
`)`
	Sets the client-side validation state of a control. If not valid, the error shown in the validation error tooltip can be specified.

`<view>.isValid()`	Queries if a control is in a valid state.

`bpmext.ui.getInvalidViews(`
`  fromView`
`)`
	Retrieves a list of invalid views (optionally under a particular view). If there are no validation errors, the method returns an empty list or array.

`bpmext.ui.getRequiredViews(`
`  onlyEmtpy,`
`  fromView`
`)`
	Retrieves a list of required views (optionally under a particular view). The onlyEmpty flag filters out views whose binding data is set. If no matching views are found, the method returns an empty list or array.

The `setValid()` method can be used to set or unset the valid state on any view that supports the concept of a valid state. In Figure 4-26 on page 182, logic runs when the Decimal control value changes and shows an error if the validation test fails.

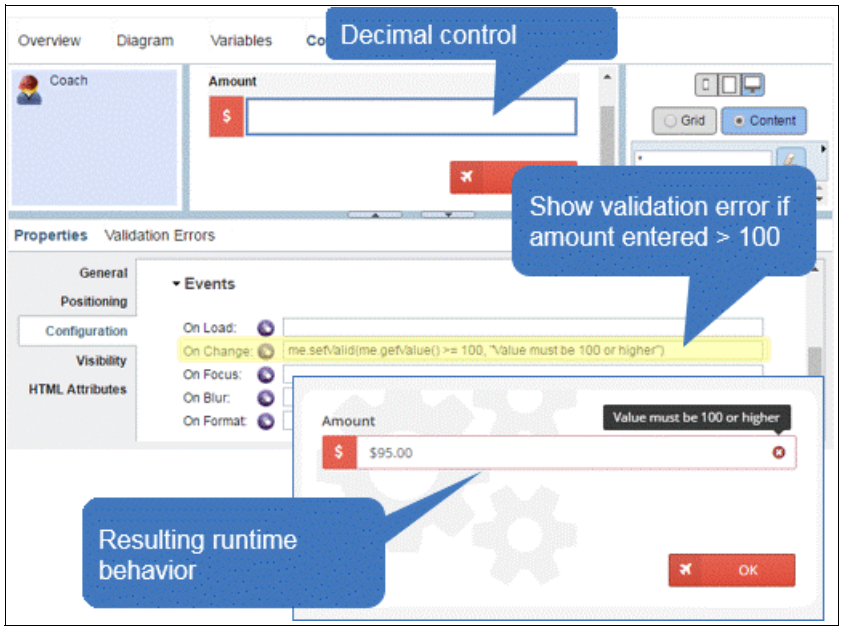


Figure 4-26 Using the setValid() method for client-side programmatic validation

Note: The decision for where validation logic is run (that is, on the client or on the server) must be carefully evaluated and made by the solution architect. The SPARK UI Toolkit provides streamlined support for both scenarios.

The next validation scenario is identical to the previous except that the validation logic runs in an AJAX service. The AJAX service invocation is done through the *Service Call* SPARK control.

Instead of calling the validation logic locally, the *Amount* Decimal's *on change* event invokes the *Validation* Service Call control. The call control invokes the server-side validation logic. After the AJAX service returns, the *on result* or *on error* event of the Service Call control is asynchronously triggered. The event then sets the *Amount* Decimal's validation state accordingly. See the interaction sequence in Figure 4-27 on page 183.

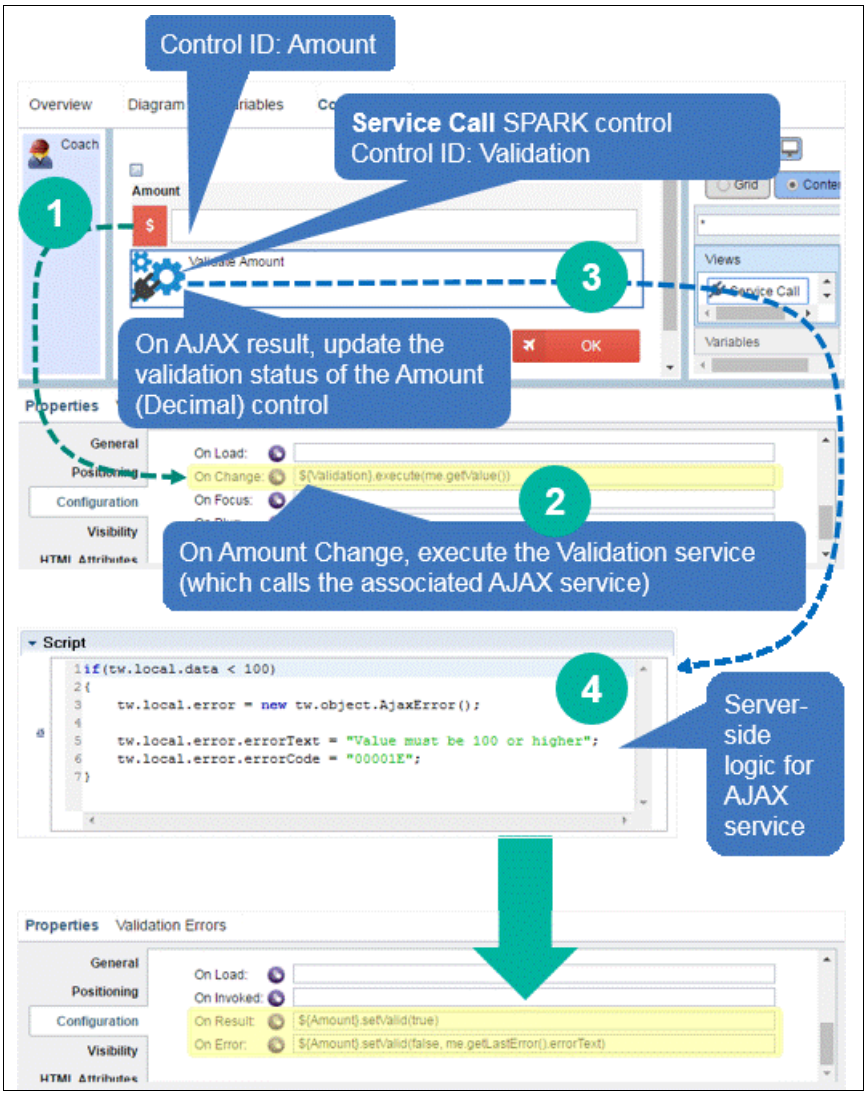


Figure 4-27 Using the setValid() method along with Service Call control and AJAX-based validation

Sequence recap:

1. Data changes in the Amount Decimal control, which triggers the On Change event.
2. The Decimal On Change handler calls the **`execute()`** command on the Validation Service Call control.
3. The Validation Service Call control invokes its associated AJAX service.
4. The associated AJAX service executes the validation logic (server-side) and returns an error if the amount < 100.
5. The Service Call control's event handlers call **`setValid(true|false)`** on the Amount Decimal control.

This scenario highlights another key benefit of the SPARK UI Toolkit: Service Call-based invocations do not rely on a human service diagram. This means that both client-side logic and server-side invocations can be fully encapsulated inside a view. This ability takes reusability a step further compared to a classic Coach Framework-based scenario.

Lastly, the scenario in Figure 4-28 explains how to use the **bpmext.ui.getInvalidViews()** command preventatively. This method easily prevents navigation if any control on a page or in its embedded views is in an invalid state.

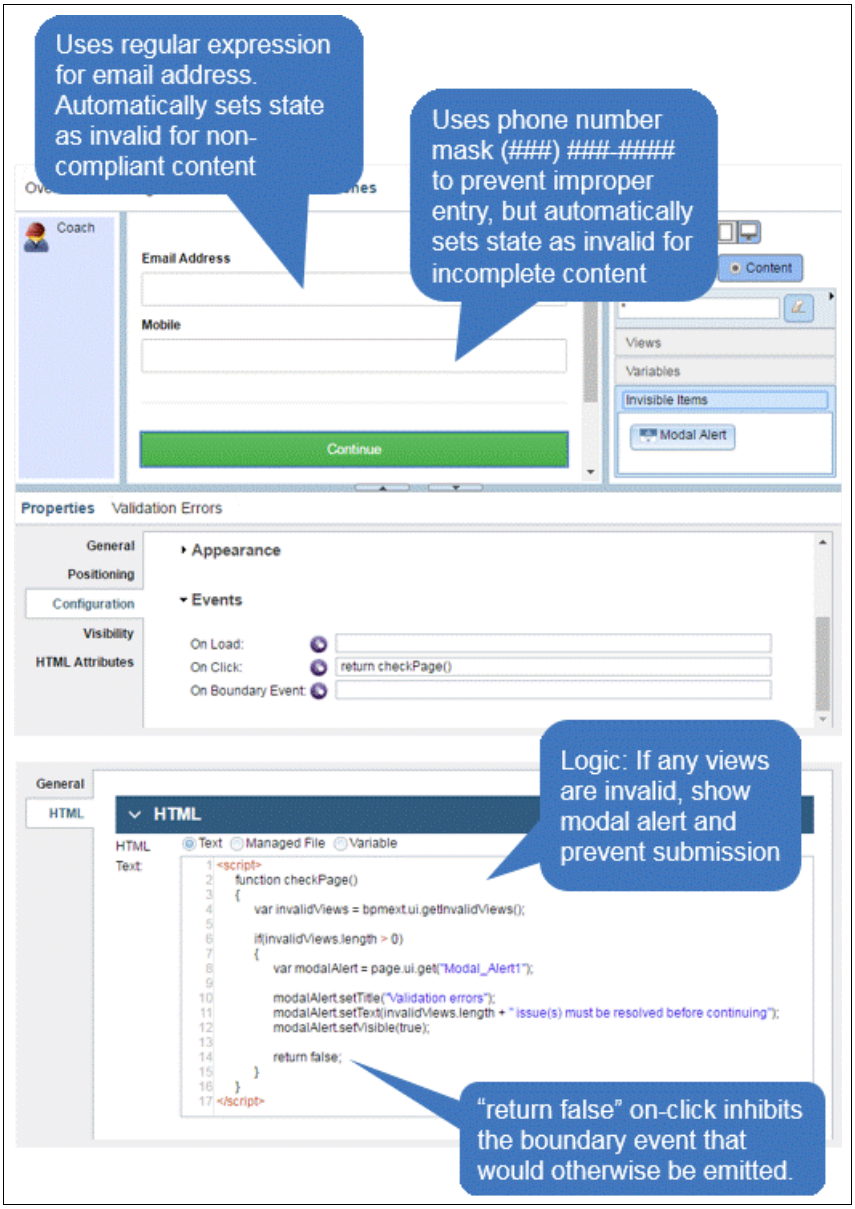


Figure 4-28 Using the bpmext.ui.getInvalidViews() method to inhibit coach navigation

Note: The same pattern shown for the **bpmext.ui.getInvalidViews()** command in Figure 4-29 on page 186 can be applied with the **bpmext.ui.getRequiredViews()** command to prevent navigation if not all required views are filled.

4.4 UI layout

The ability to configure layout behavior to control the following areas is essential to providing a naturally flowing user experience:

- Vertical or horizontal flowing
- Vertical and horizontal alignment of content (including justification, auto reflowing, or wrapping of content)

The ability for that behavior to adapt to different form factors is equally important and is covered in 4.6, “Responsiveness” on page 202.

As of IBM BPM 8.5.7, coach content can be laid out using a grid that is configurable. It is configurable at design time and controls layout behavior for coaches built in the IBM Process Designer Web Editor, as described in 2.10, “Designing a coach using the grid layout” on page 50.

Tip: Grid responsive behavior is always based on the width of the coach. Grids are best to use in a coach or for coach views that are always expected to take up the entire width of the coach.

SPARK UI layout controls provide sophisticated and consistent layout support for both the Web and Desktop Editors in IBM Process Designer. They also work well for coaches and coach views from IBM BPM 8.5.0.1 through IBM BPM 8.5.7.

Tip: The responsive behavior of SPARK UI layout controls is not relative to the coach page width. This makes SPARK UI layouts a good choice to lay out controls in a composite coach view (that is, a view that contains other controls).

4.4.1 Horizontal and vertical layouts

The core layout support provided in the SPARK UI Toolkit is contained in the two Horizontal Layout and Vertical Layout controls. The choice between horizontal and vertical layout is a design-time consideration. It is fully overrideable at run time (especially for responsive behavior, which is covered in “Layout controls: Responsive settings” on page 206. Both Horizontal Layout and Vertical Layout controls are backed by the same logic. The only difference is how IBM Process Designer displays the container controls at design time (using a horizontal or vertical layout).

Figure 4-29 on page 186 lists important layout and alignment configuration parameters available for the Horizontal and Vertical Layout controls.

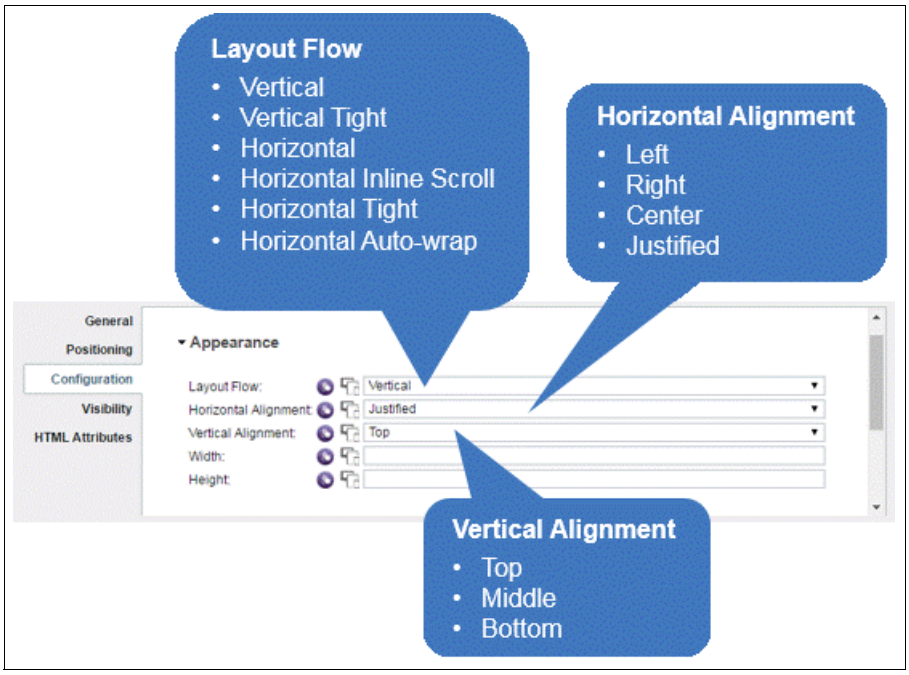


Figure 4-29 Layout and alignment configuration options for Horizontal and Vertical Layout controls

Horizontal layout flow and options

When using horizontal layout, all controls directly contained in the Layout control are displayed next to each other horizontally (see Figure 4-30).

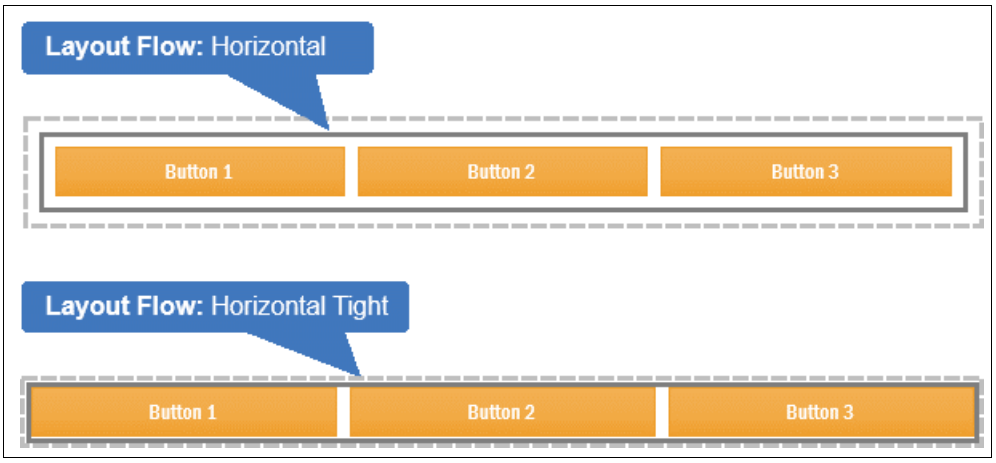


Figure 4-30 Horizontal layout modes with a default Justified horizontal alignment

By default, a margin is included around the layout, whether horizontal or vertical. Often, however, the UI developer might not want a new margin added with each new level of nesting, especially when a layout is nested inside of another. The *Horizontal Tight* layout option removes the margin to allow for a more esthetically pleasing layout appearance in such cases, as shown in Figure 4-30.

Left, right, center horizontal alignment

Figure 4-31 shows how layouts configured in a horizontal layout mode can align their controls left, right, or center. Additional horizontal layout configuration options also allow content to scroll horizontally or auto-wrap.

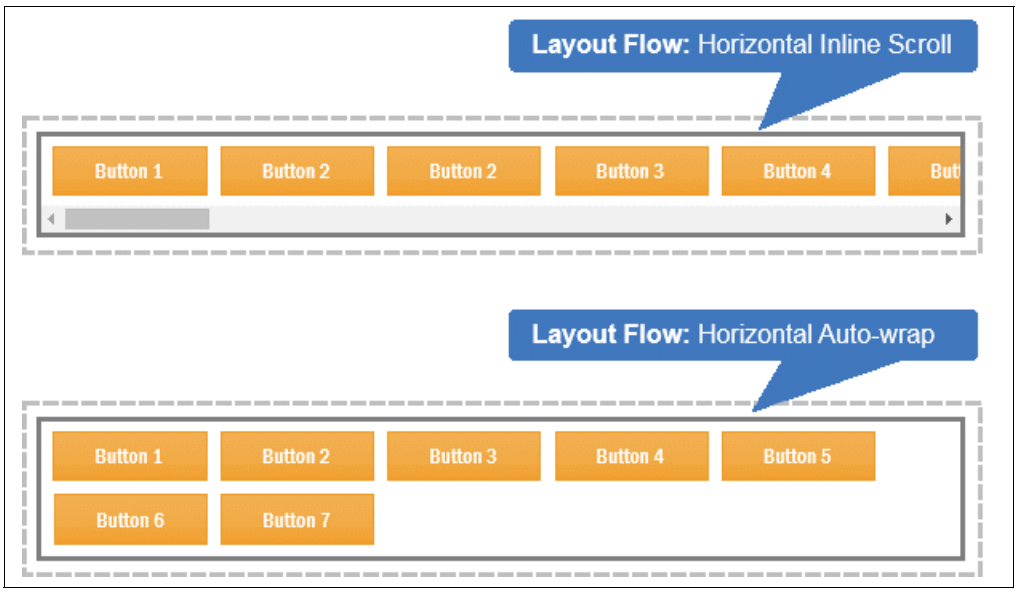


Figure 4-31 Horizontal layout modes with left horizontal alignment

For inline scroll, content appears properly aligned until it is compressed enough to cause horizontal scrolling. For auto-wrapping, alignment also works predictably and the wrapped content also aligns correctly (left, right, or center).

Note: When using the Horizontal Inline Scroll layout option, use controls that vertically expand (for example, a Popup Menu, Tooltip, and Status Box). Use these controls with caution because the expanded content might appear vertically cut off (that is, the scrolling portion of the browser clips vertically overflowing content).

Justified alignment

Justified alignment is the default alignment mode unless otherwise configured through the Horizontal Alignment option. The contained controls in a justified-aligned layout do not wrap overflowing content or scroll to show the overflow.

Instead, the width of the child controls can become elastic so that the entire row of controls takes up the width of the layout, such as table cells in a row. Contained controls can have a width configured in % or unit-based width (for example, 120 px or 2.9 em). The browser auto-adjusts the width of any contained control that does not have a width configuration option specified.

The width of controls contained in a layout can reach a point where they can be compressed no further. Figure 4-32 on page 188 shows how controls, buttons in this example, overflow unless responsiveness settings override this behavior.

Figure 4-32 Horizontal overflow and justified horizontal alignment

Note: Justified is not a valid horizontal alignment option for the Inline Scroll and Auto-wrap layout options.

Vertical layout flow and options

When using vertical layout, all controls directly contained in the Layout control are stacked vertically; they flow top to bottom. See Figure 4-33.

Figure 4-33 Vertical layout mode with justified horizontal alignment

The Left, Center, and Right horizontal alignment options also apply for vertically flowing layout content and the stacked control horizontal align. Figure 4-34 shows the center-aligned example.

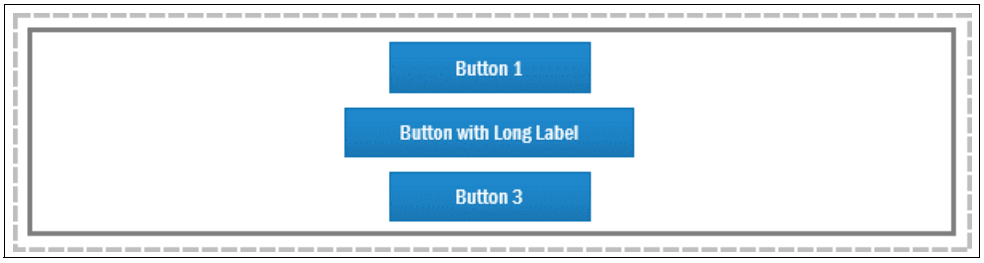


Figure 4-34 Vertical layout mode with Center horizontal alignment

Width and height

By default, unless the Width configuration option is set, a layout control takes up the entire width of its container, whether in a coach view or a coach. As shown in the Horizontal Inline Scroll example in Figure 4-31 on page 187, a Layout control with a Horizontal Inline Scroll layout flow scrolls its child content. This happens as soon as the combined (compressed) widths of all child controls exceeds the configured width of the Layout control.

In general, the widths of immediate child controls contained in the layout are configured by setting the Width option on each child control. Child control widths do not need to be set unless the browser's layout behavior needs to be overridden. In the case of Justified alignment (see “Justified alignment” on page 187), child controls act like table cells, with one cell for Vertical layout and multiple cells for Horizontal layout. The widths of justified child controls behave like the widths of cells in an HTML table.

Note: Width-layout behavior with the SPARK UI Toolkit does not use the 12 columns layout model (except for Page Layout controls, which are deprecated after IBM BPM 8.5.0.1). SPARK allows any width specifications and any number of columns. To use a 12 column model with responsiveness based only on coach width, use the Grid in IBM Process Designer's Web Editor, otherwise use a SPARK UI Layout control.

Unless the height of a layout is configured, the Layout control expands vertically to accommodate showing its child content. If the height is configured and the content vertically exceeds the height configured, then the layout automatically allows vertical scrolling. Figure 4-35 shows a vertically scrolling layout.

Figure 4-35 Vertically scrolling layout

Setting the Height of a Layout control works predictably with any unit (for example 150 px or 6.2 em). However, heights set using percentages (%) only work if all parent elements (or layouts or views) of the Layout control have a height specified until either of the following is encountered:

- A height is specified in units (not using %).
- The coach body element is reached in the DOM hierarchy and has either a percentage (%) or a unit-based height specification.

This is a limitation of specifying heights using CSS-based styling.

Vertical alignment

The ability to configure a Layout control's Vertical Alignment option matters in two situations:

- If the height of the layout is configured and is greater than the vertical space taken by its content, whether the layout flows horizontally or vertically.
- The height of child content flowing horizontally is uneven.

Figure 4-36 on page 190 shows the first of the listed situations.

Figure 4-36 Layout height greater than height of contained controls

Figure 4-37 shows the second listed situation.

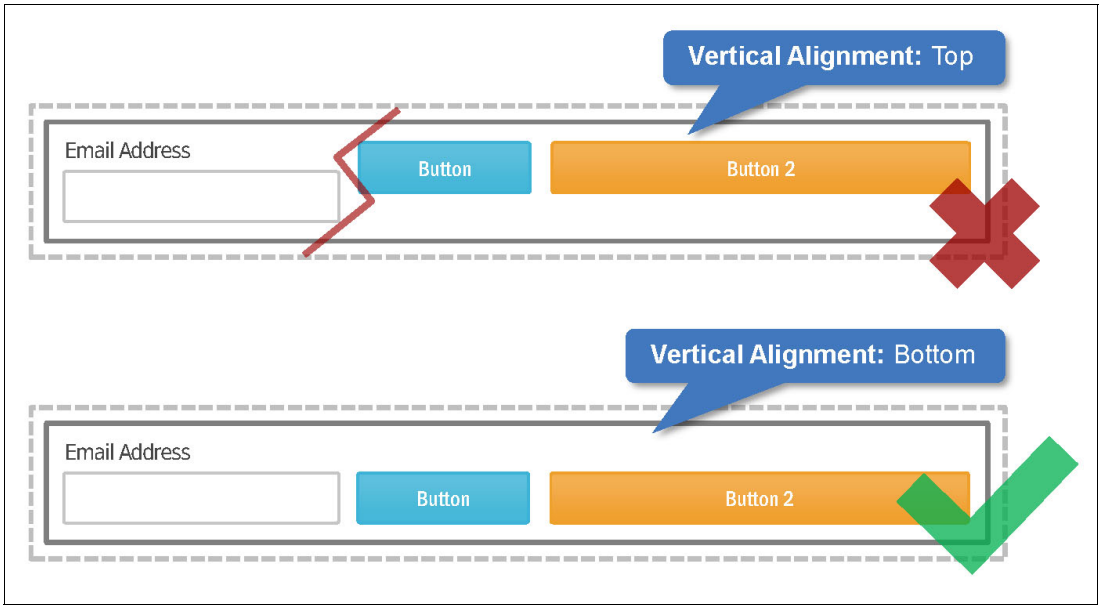


Figure 4-37 Uneven heights for controls in horizontally flowing layout

Nesting layouts

Horizontal or vertical layout controls must be nested to achieve a variety of layout behaviors. Layouts can contain other nested layouts with their own Layout Flow, Horizontal Alignment, Vertical Alignment, Width, and Height configuration options.

These capabilities, especially when combined with responsive behavior support, provide powerful and flexible options for BPM UI layout at the coach or coach view level.

The Tight (Vertical or Horizontal) Layout Flow options are most useful in nested situations where certain parts of the child layout need to align with the edges of the parent layout. Figure 4-38 on page 191 shows the layout difference of using a nested layout with and without the Tight option.

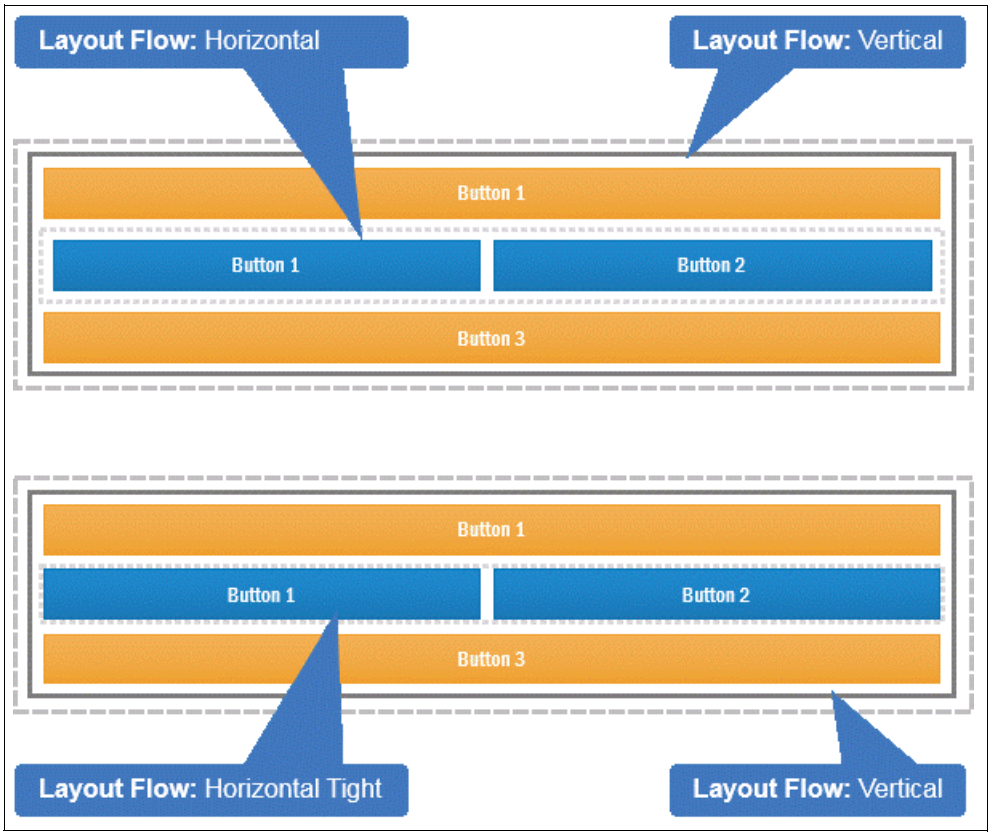


Figure 4-38 Using the Tight layout flow option for nested layouts

4.4.2 Tabbing or stacking UI content

The SPARK UI Toolkit provides a Tab control that shows one pane at a time. It also provides a Stack control for arranging panes of content that only display one at a time.

Tab Section control

Figure 4-39 on page 192 shows an example of the design-time and runtime behavior and appearance of the Tab Section.

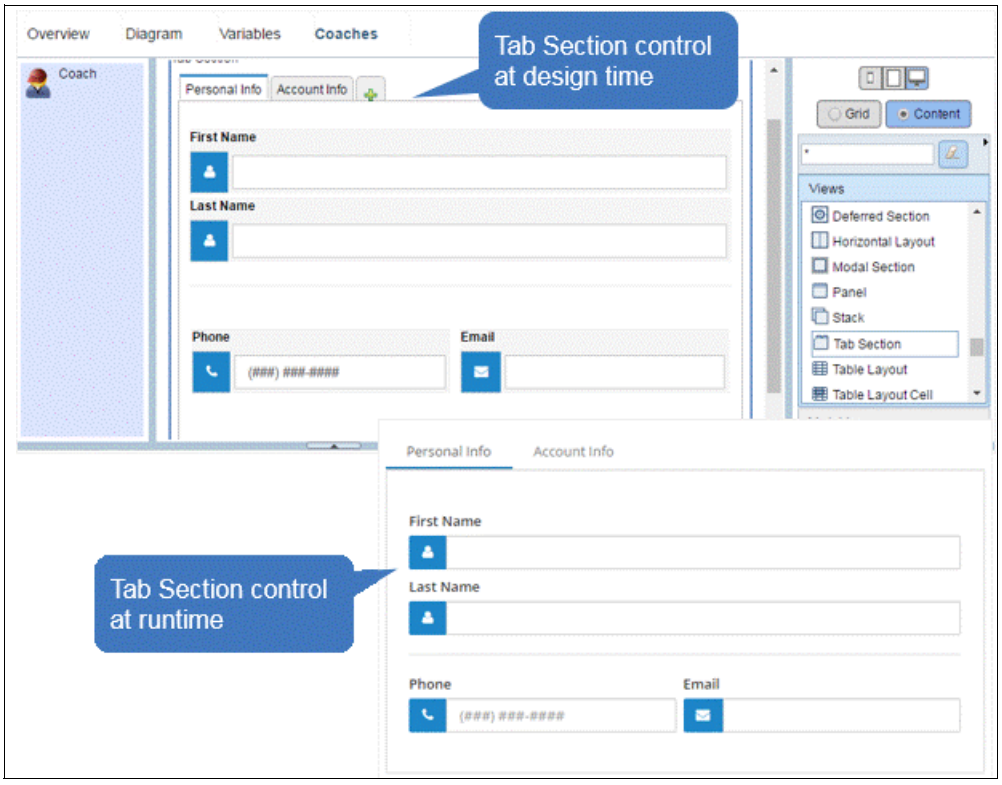


Figure 4-39 Tab Section control (design time and run time)

The Tab Section control provides a Tab Changed event that gets fired when the current tab is changed, either by the user or programmatically. The binding data, if the control is bound, contains the 0-based integer index of the tab currently showing. The UI developer can also change the current tab programmatically using the **`setCurrenPane()`** command.

Note: A 0-based tab index means that the first tab is at index 0, the second at index 1, and so on.

Stack

The Stack control is similar in behavior to a Tab Section control but without any visual decorations; it has no visible tabs or borders.

4.4.3 Displaying modal content

At times content needs to be displayed modally, in such a way as to prevent input in other parts of the UI. In those cases, the Modal Section provides a flexible option to do so for any control. An example of this is a Panel for modal dialog-like behavior, such as a Well containing a Progress Bar control.

Note: A Well is a simple *cosmetic* container that can be colored or left blank and given an optional background icon. See 4.4.4, “Wells” on page 195 for more information about this topic.

The Modal Section acts only as a container. The control the UI developer inserts in the containment box at design time is shown modally, against a darker backdrop, at run time. Figure 4-40 shows a Modal Section containing a Well and a Progress Bar control.

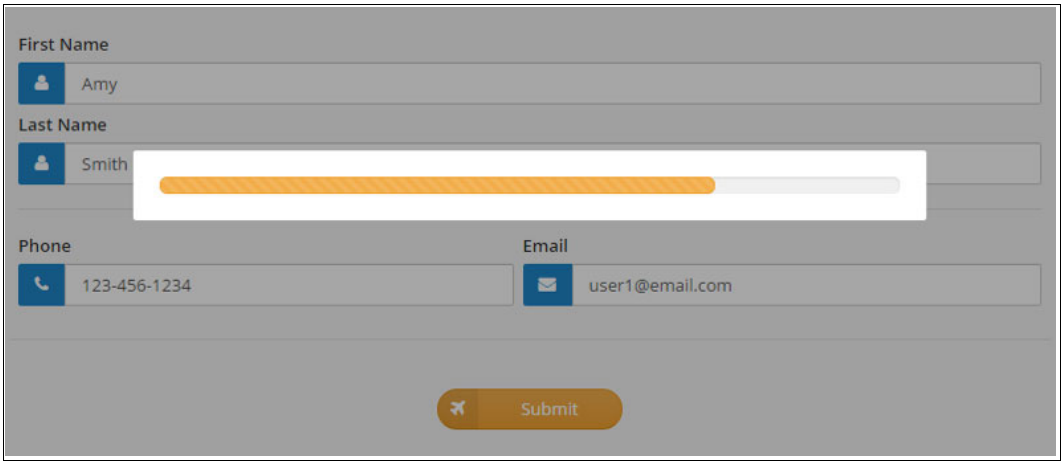


Figure 4-40 Modal Section displayed at run time with a Progress Bar

Figure 4-41 shows the same Modal Section containing a Well and a Progress Bar in IBM Process Designer's Web Editor.

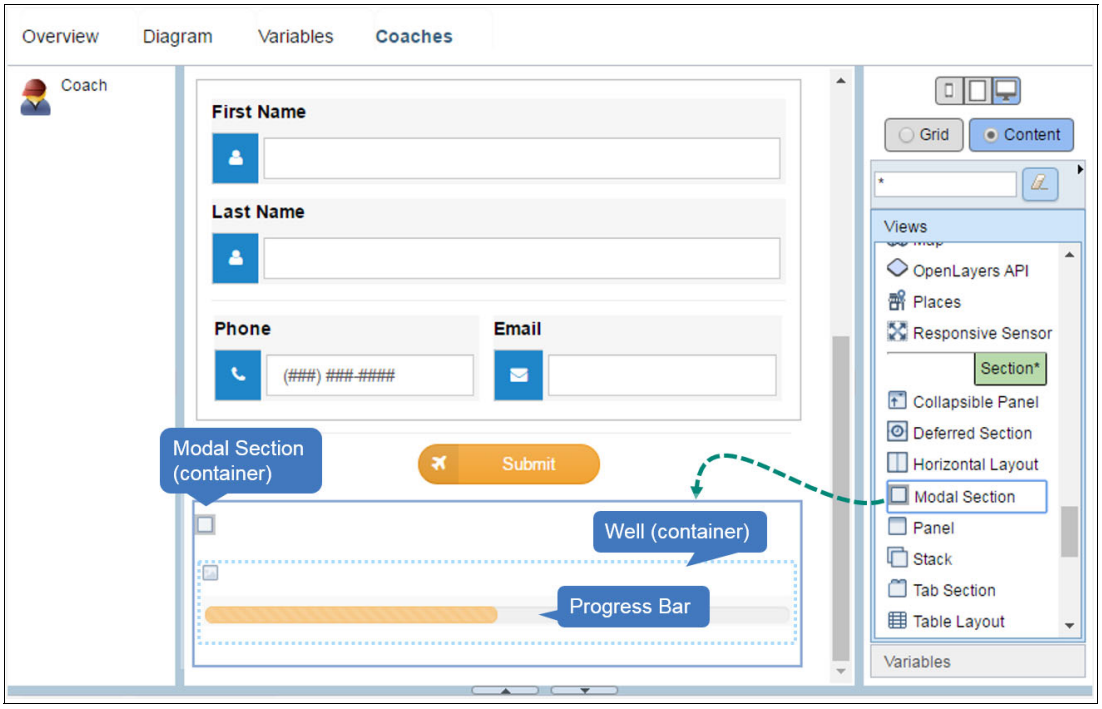


Figure 4-41 Modal Section in IBM Process Designer Web Editor

Showing and hiding modal sections

The display of modal sections is based on the standard Coach Framework visibility setting and can also be controlled using the **`setVisible()`** SPARK UI command. For this reason, a Modal Section control's visibility setting must be set to NONE initially, otherwise the section is displayed as the coach opens. The visibility of the *Modal Alert* control works the same way.

After being displayed, the Modal Section can be closed in two ways:

- By calling the **`setVisible(false)`** command on the Modal Section control.
- By clicking in the darkened area of the section (assuming the *Close On Click* configuration option has been set at design time), as illustrated in Figure 4-42.

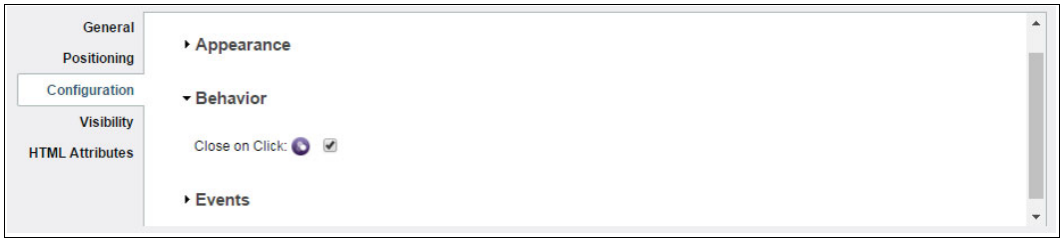


Figure 4-42 Close on Click configuration option for Modal Section

Lastly, the *On Close* event on a Modal Section control is available to take programmatic actions when closing and fires when the Modal Section closes from user action or programmatically (from the **`<Modal Section control ref>.setVisible(false)`** command).

Displaying dialogs modally

Modal sections can be used equally to display modal dialogs. Figure 4-43 on page 195 shows a Panel displayed as a modal dialog (meaning in a Modal Section) and the simple logic to open and close it.

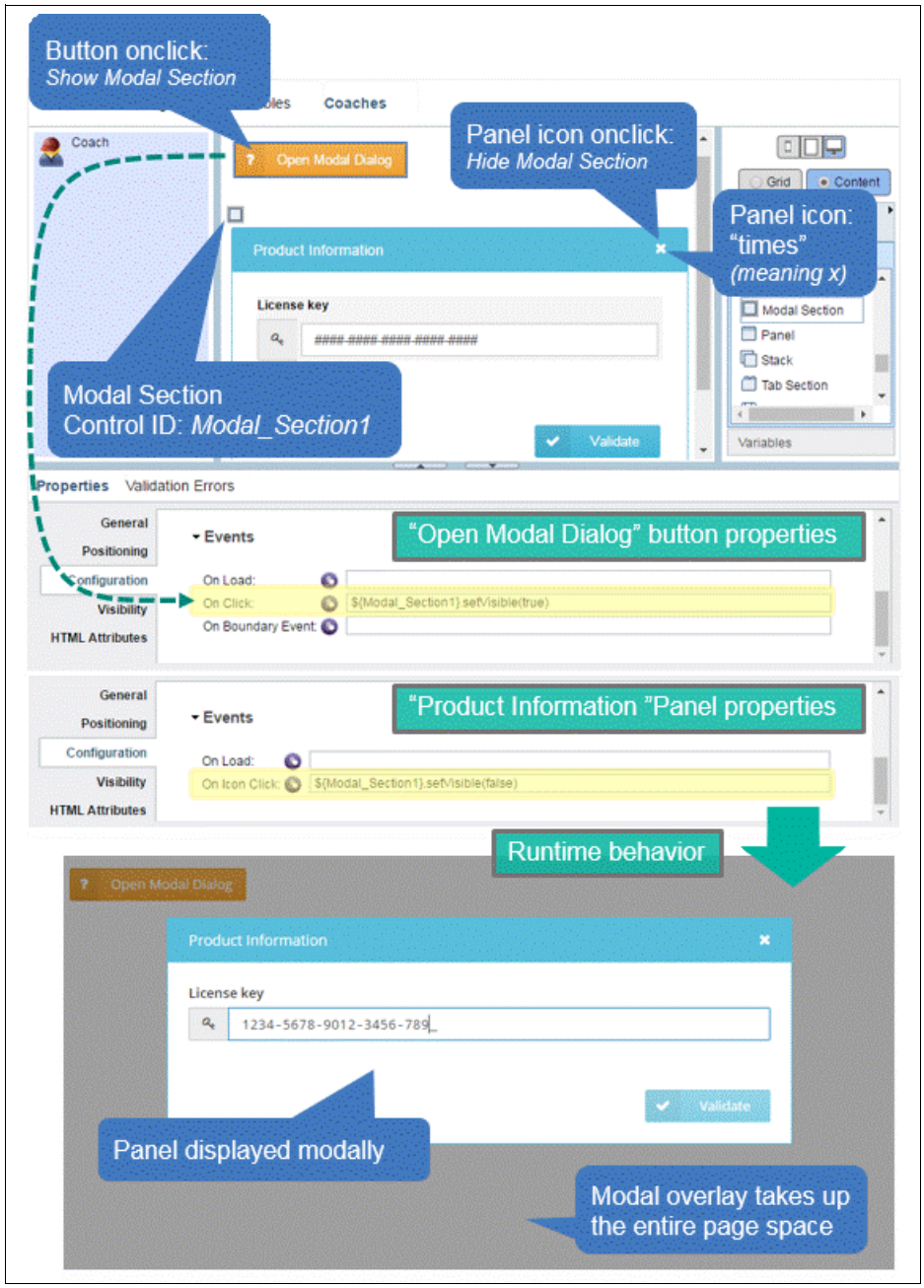


Figure 4-43 Panel control displayed in a Modal Section as a modal dialog box

4.4.4 Wells

Wells provide a simple way to display content in an area that can be colored and can include an icon in its background. The purpose of Wells is limited to making UIs more visually appealing and organized by separating parts of the UI from their background using different coloring. Wells are appropriate to use by themselves or inside Layout controls or a Grid.

Figure 4-44 shows a Well configured to display a bank icon in its background, using the INFO color style, containing a Donut Chart control.

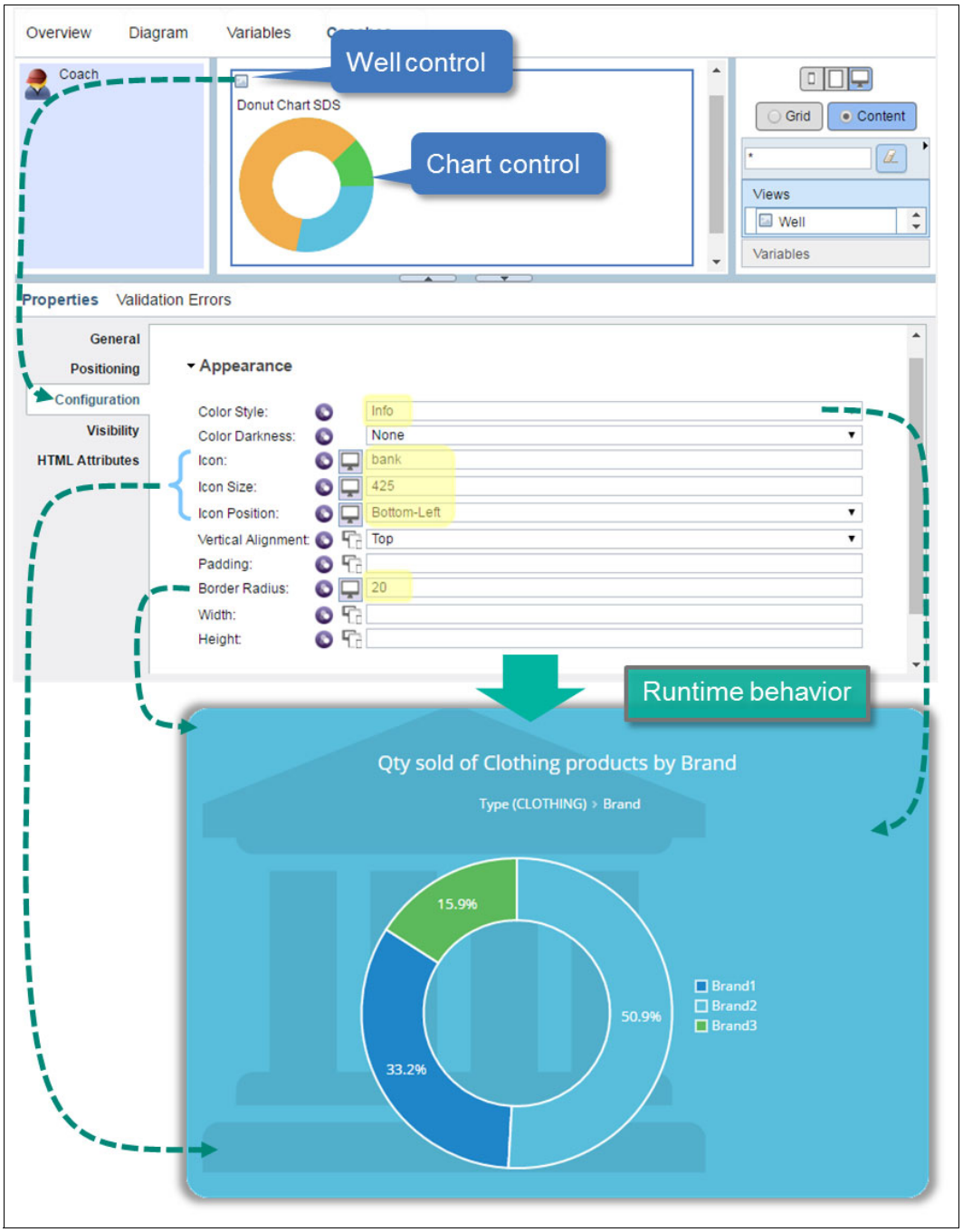


Figure 4-44 Configuration and runtime behavior of a Well control containing a Chart

4.5 Calling AJAX services

AJAX services under the Coach Framework are often called directly or indirectly from the diagram of a human service. This option is simple to use but provides low reusability because it does not allow the service invocation to be encapsulated inside a coach view.

Another classic option, for coach views only, is to create a *Service* type configuration option and then to call the service through context options using the **<serviceName>(serviceArgs)** command. This option, however, can expose a detail about the coach view as a configuration option. In some cases, this detail must remain internal to, or encapsulated in, the view. It is also a fairly technical exercise.

The SPARK UI Toolkit includes a control that offers streamlined and encapsulated interaction capabilities that work the same way in coaches and in coach views. They also work in a manner that is consistent with how SPARK events work.

4.5.1 Service Call control

The Service Call control is in the SPARK UI Toolkit palette (shown in Figure 4-45) under the *System* category.

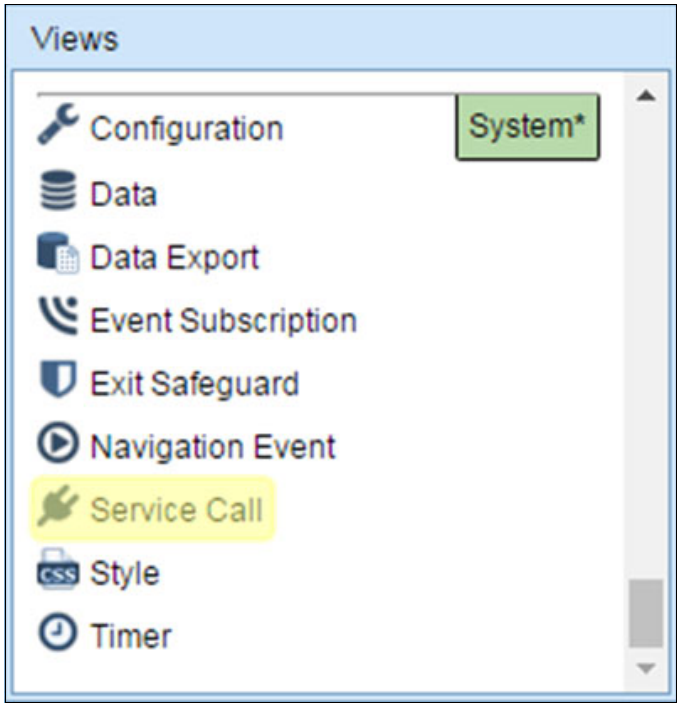


Figure 4-45 Control palette showing Service Call control

Configuration

The Service Call control is configured and works in the following manner:

- A Service Call control must always be associated with a IBM BPM AJAX service, as shown in the configuration example of Figure 4-46.

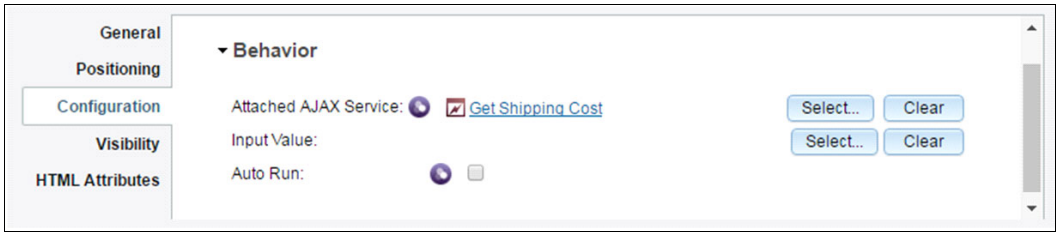


Figure 4-46 Service call associated with IBM BPM AJAX service

- The service call control provides the **`execute()`** command to trigger the invocation of the associated AJAX service. The **`execute()`** command accepts a single input that can be a simple or complex type, such as following inputs:
  - `${Service_Call1}.execute("abc")`: Used to specify a string as input.
  - `${Service_Call1}.execute({"prop1": "value1", "prop2": 2})`: Used to specify an object with a prop1 string property and a prop2 integer property.
- If the single parameter for the **`execute()`** command is omitted, the value associated with the *Input Value* configuration option is used.
- The control **`setInputData()`** command provides a programmer-friendly way of updating the value of the Input Value configuration option.
- If the *Auto Run* configuration option is checked, the Service Call control calls **`execute()`**. This is done automatically every time the value associated with the Input Value configuration option changes and when the control is first loaded.
- The Service Call control can show a visual progress indicator (see Figure 4-47) that is displayed when the AJAX invocation is in progress.

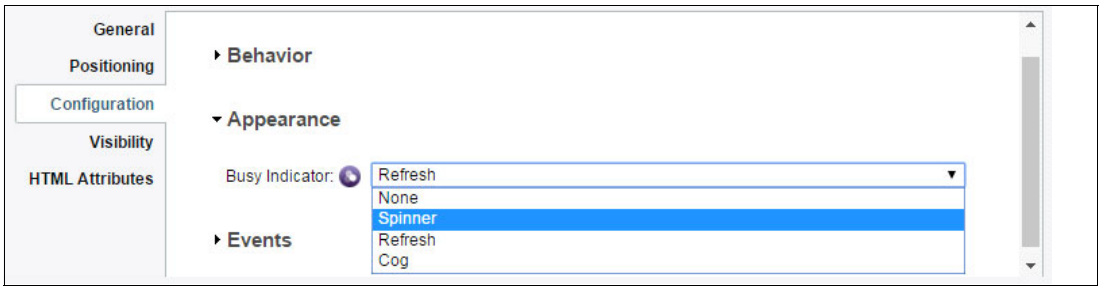


Figure 4-47 Service Call busy indicator configuration

Invocation sequence

Figure 4-48 on page 199 shows the invocation sequence required for the Single Select control containing shipping modes to use the Service Call control to trigger a shipping cost query. This query in turn sets the retrieved cost in a Decimal control.

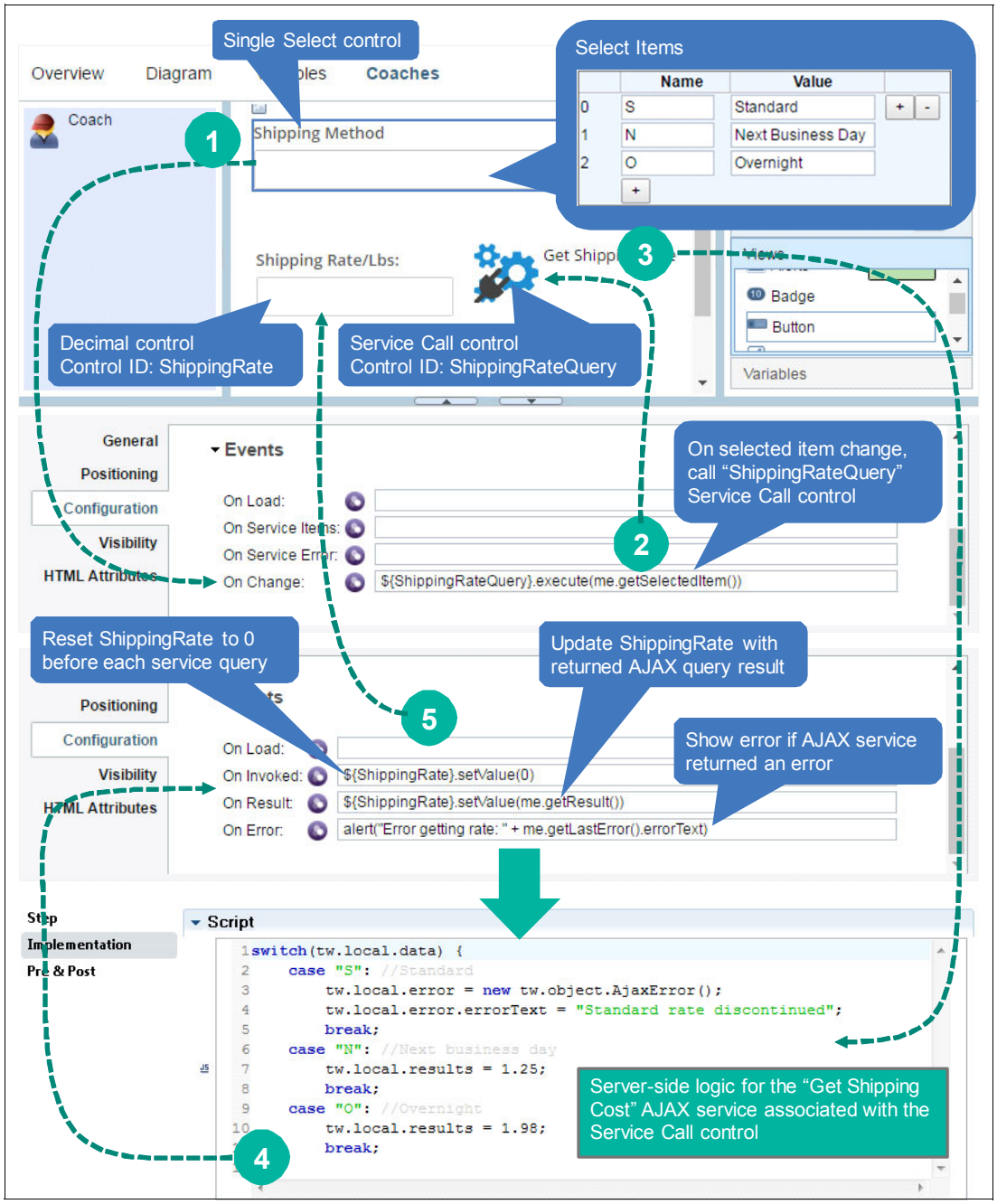


Figure 4-48 Usage example for the Service Call control

Sequence recap:

1. Selected item changes in the Single Select control, which triggers the On Change event.
2. The Single Select On Change handler calls the **`execute()`** command on the ShippingRateQuery Service Call control.
3. The ShippingRateQuery Service Call control invokes its associated AJAX service.
4. The associated AJAX service executes its server-side logic and returns the shipping cost (or an error for standard shipping mode).
5. The Service Call control's event handlers call the **`setValue(<returned AJAX result>)`** command on the Shipping Rate Decimal control.

Figure 4-49 shows the resulting behavior at run time for a successful invocation for the given sequence, logic, and configuration noted in the previous sequence recap.

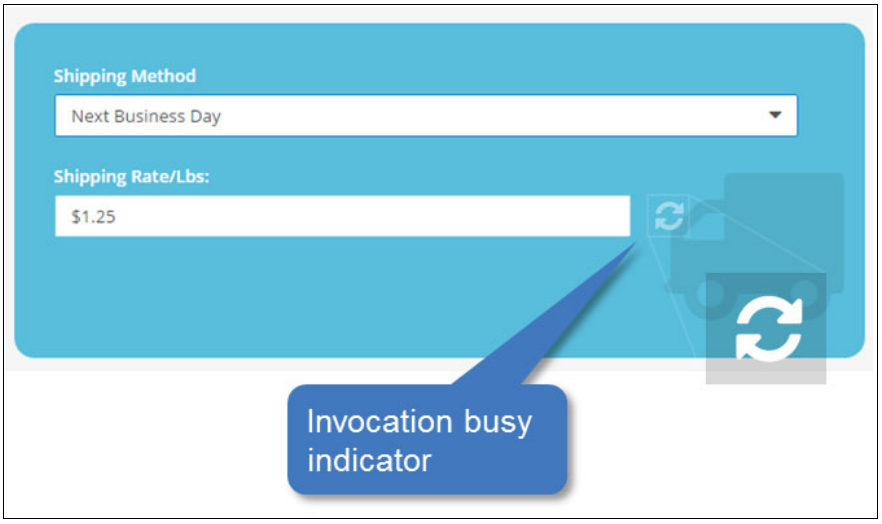


Figure 4-49 Runtime example for Service Call control, successful invocation

Figure 4-50 on page 201 shows a failed invocation.

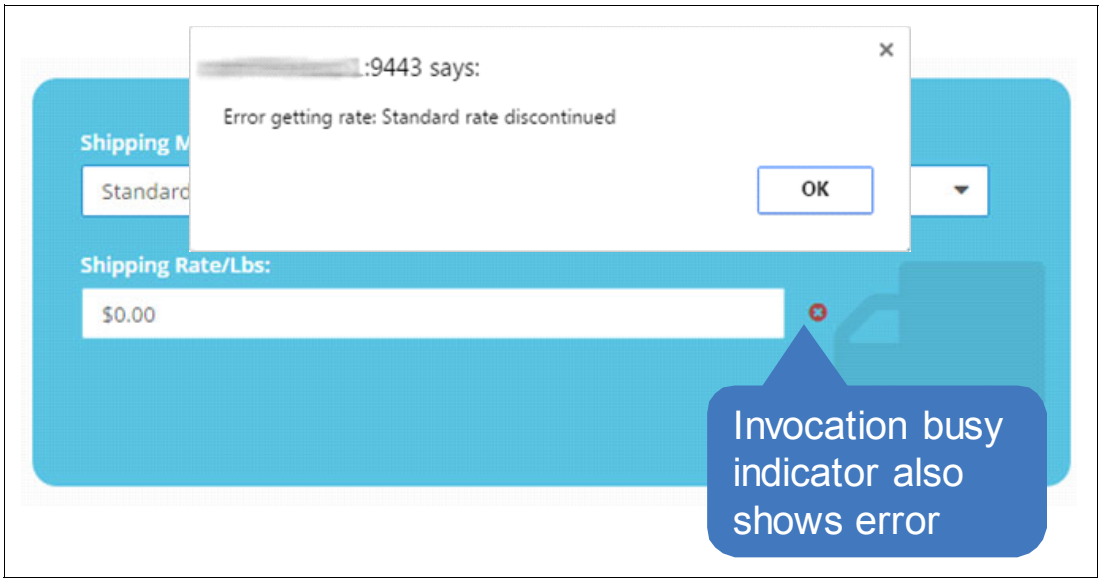


Figure 4-50 Runtime example for Service Call control, AJAX service error

Handling server logic errors

AJAX service logic error information is accessible from the Service Call control through the **`getLastError()`** command, which contains an error object with the following properties:

- `errorText`
- `errorCode`

When the On Error Service Call event fires, the **`getLastError().errorText`** and **`getLastError().errorCode`** commands are used to retrieve the message. The message is retrieved from the AJAX service logic that is associated with the error.

Reporting errors from AJAX service logic

Errors in AJAX service logic can be reported in three ways inside the AJAX Service associated with the Service Call control:

- By initializing t**`w.local.error`** to new t**`w.object.AjaxError()`** and setting the errorText and errorCode property of the error. With this approach, an error event or exception is not thrown. When the AJAX invocation returns, the Service Call control automatically activates its On Error event.
- By using an Error end event. In this case, the error code and the data from the error mapping on the Error Event map go to both errorCode and errorText.
- By throwing a server-side JavaScript error, for example by throwing a new Error (“Invalid account number”). The exception Java text, including the location of the offending statement, is mapped to the errorText. The value of errorCode, in this case, is determined by IBM BPM, not by the service developer.

4.6 Responsiveness

Responsiveness in the SPARK UI Toolkit works on the following two levels:

- Classic (IBM BPM-based) responsiveness: The basic responsive behavior exposed in the IBM Process Designer Web Editor for controls that provide adaptive properties.
- Enhanced Web Component-based responsiveness: The SPARK responsive capabilities that behave consistently since IBM BPM 8.5.0.1 work with both IBM Process Designer Web and Desktop Editors. They allow greater flexibility than coach form factor-based responsive triggers.

4.6.1 Classic responsiveness

Most SPARK UI Toolkit controls provide adaptive configuration options that can be exploited through IBM Process Designer's Web Editor.

Figure 4-51 shows an example of the Note control *Label Style* adaptive property. This property allows IBM Process Designer Web Editor to specify different configuration option values across the three IBM BPM-supported screens or form factors (Small, Medium, and Large).

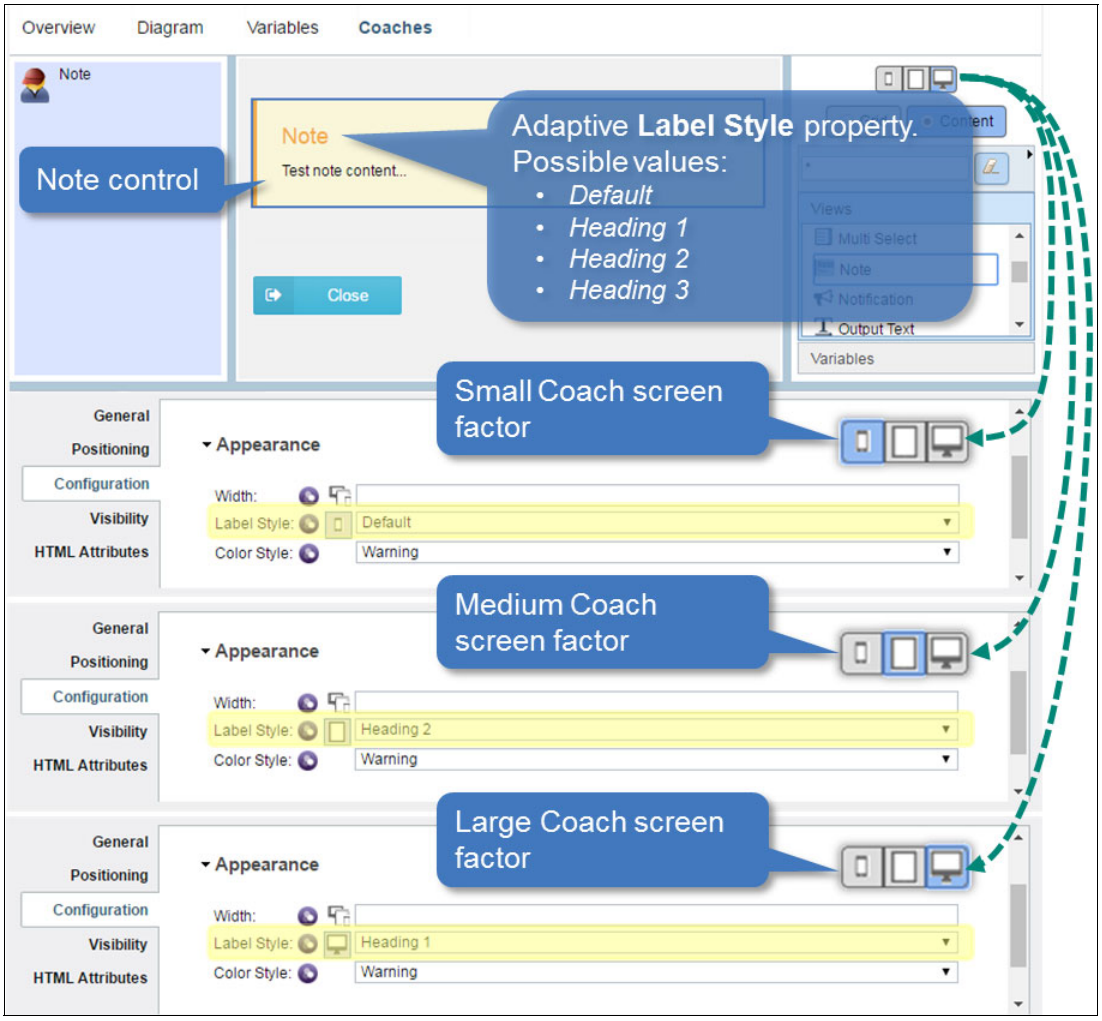


Figure 4-51 SPARK control adaptive property example

At run time, the control auto-adjusts the configuration option value of its adaptive property based on the Coach Framework's screen factor-based trigger. Figure 4-52 shows the runtime example for the previously-configured Note control.

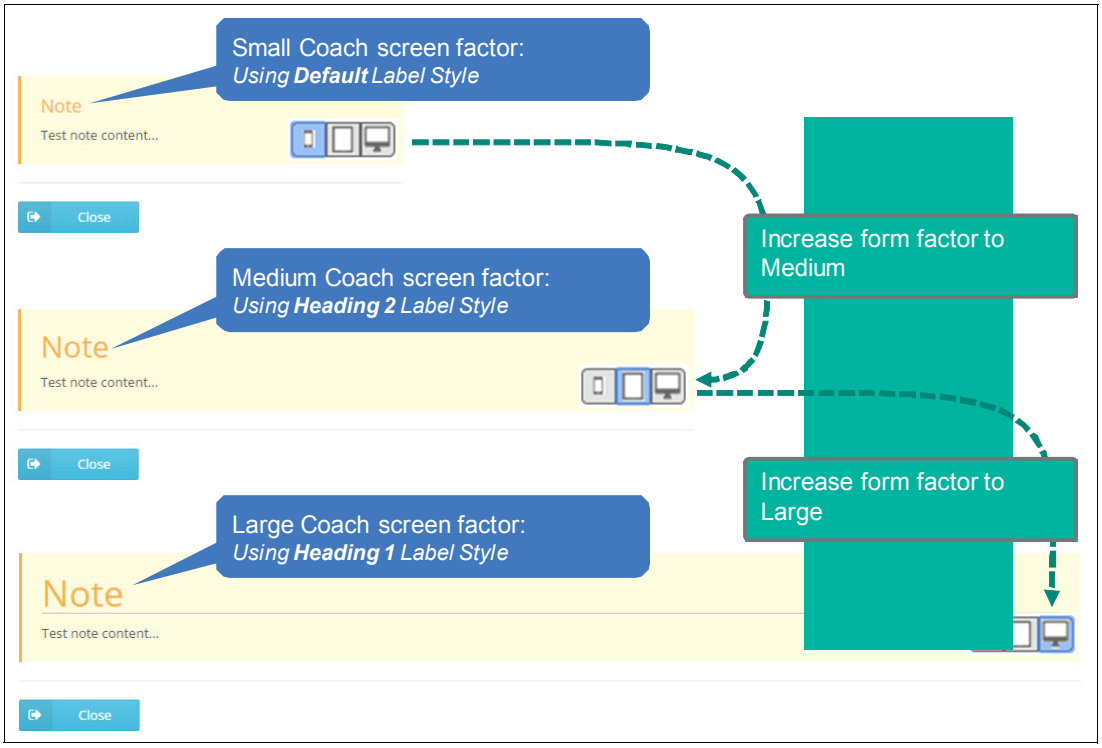


Figure 4-52 IBM Process Designer Web Editor's adaptive properties, runtime behavior

Using this approach (which only works with IBM BPM 8.5.5 and higher and requires using Process Designer Web Editor), a large number of SPARK UI controls provide adaptive properties. These properties can be configured, as appropriate, to provide this type of behavior.

4.6.2 Enhanced Web Component-based responsiveness

By combining layouts and the Responsive Sensor control, the SPARK UI Toolkit provides responsive behavior benefits. Those benefits are as follows, in addition to classic responsiveness support in IBM BPM:

- Responsive behavior is provided and works consistently across product versions from IBM BPM 8.5.0.1.
- Coach views can predictably determine their responsive behavior even when they do not take up the entire width of the coach in which they are placed.

Coach views with SPARK behave like *Web Components*. As such, they do not depend on the overall coach form factor from the IBM Process Designer web editor to trigger layout changes. Layout changes can instead be triggered by the size change of the coach view itself, independently from the coach that contains it.

SPARK UI responsiveness relies on the following three controls:

- Responsive Sensor
- Horizontal Layout
- Vertical Layout

Responsive Sensor control

The Responsive Sensor is a container control. It wraps itself around its contained child controls, usually a Horizontal or Vertical Layout, and acts as a size-sensing rubber band. It is configured with *box factors* that help determine when the sensor activates responsive behavior in its contained Layout controls.

Figure 4-53 shows an example of the Responsive Sensor control at design time.

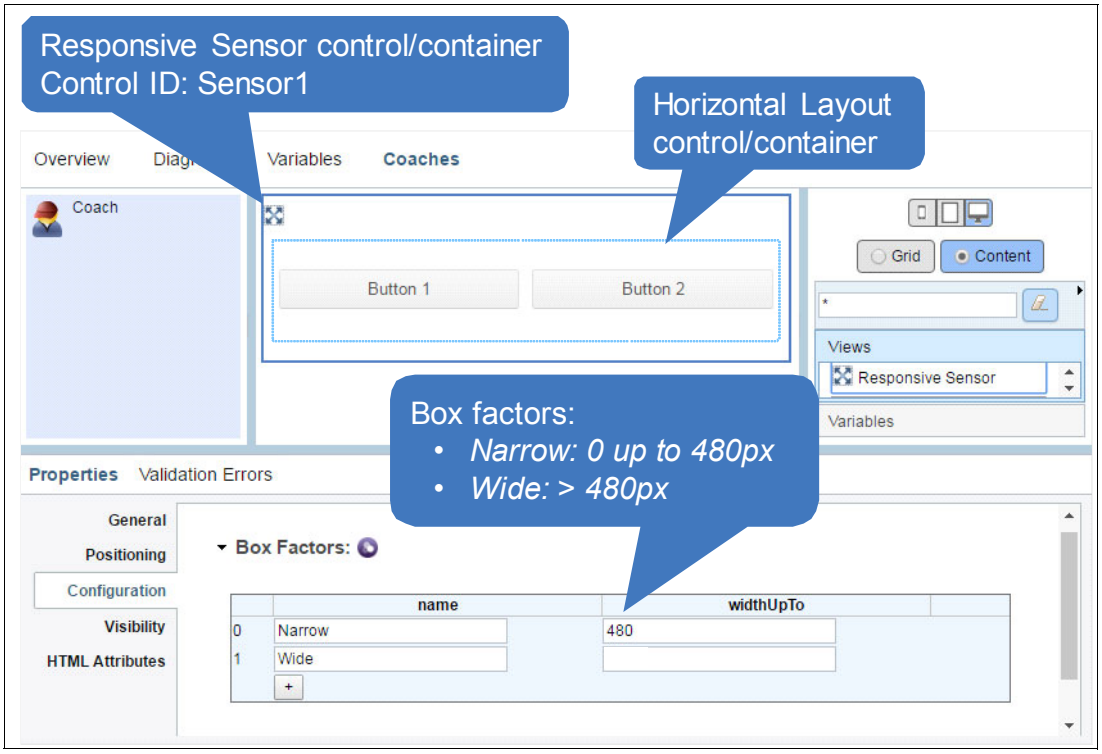


Figure 4-53 Response Sensor and box factor configuration

After box factors are specified, a Layout control (Horizontal in Figure 4-53) inside the Responsive Sensor specifies layout-related behavior. The behavior is based on one or more of the box factors configured. In Figure 4-54 on page 205, the layout is configured with the following options by default (settings in Appearance category):

- Layout Flow: Horizontal
- Horizontal Alignment: Justified

The settings in the Responsive category override the default configuration. For example, if Sensor1's box factor name is *Narrow*, then the Layout Flow is Vertical (option labeled as childLayout). Figure 4-54 on page 205 provides the full design-time configuration.

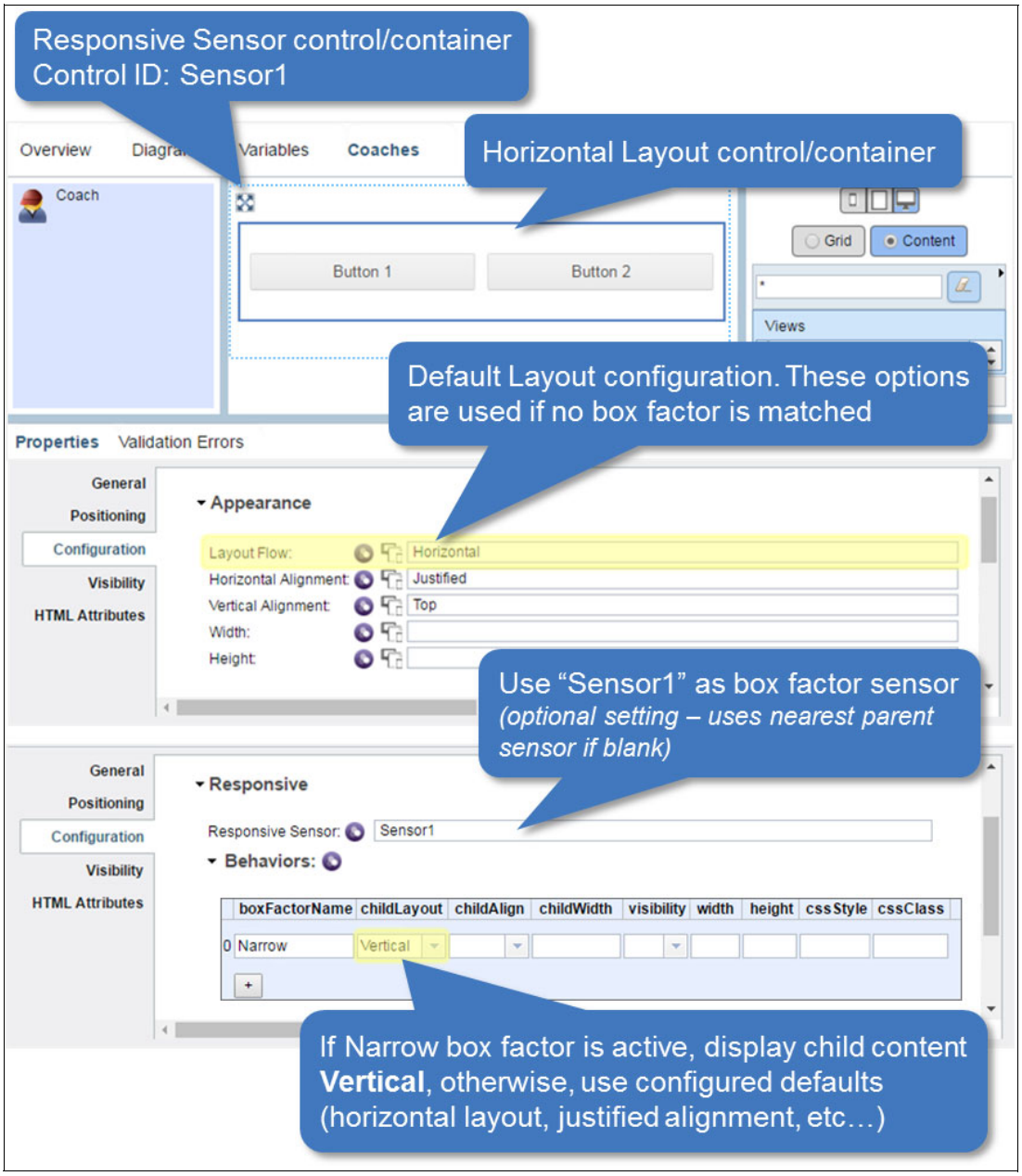


Figure 4-54 Responsive behavior overriding the default layout configuration

Note: The name of the responsive sensor used by the Layout control (see the **Responsive** → **Responsive Sensor configuration** option) is not a mandatory setting. If it is not provided, the Layout control listens to its nearest Responsive Sensor parent.

Box factor change event

At times it can be useful to take programmatic action whenever a box factor is activated. The Responsive Sensor control provides a special event (**Events** → **On Responsive Boundary**) that fires every time a box factor changes, including the first time a box factor is activated.

The **Events** → **On Responsive Boundary** event handler provides an event context variable. At run time, the variable contains the properties shown in Table 4-6.

Table 4-6 Responsive Sensor's on Responsive Boundary event handler context

Event context variable property	Description
boxFactor	The name of the Responsive Sensor box factor that became active
lowerBound	The lower bound in pixels for the current box factor
upperBound	The upper bound in pixels for the current box factor
Width	The current width in pixels that triggered the box factor change

Example 4-6 shows the code in the *On Responsive Boundary* event handler of a Responsive Sensor control. This code logs the current box factor name. It is logged to the web browser console when a responsive boundary is crossed.

Example 4-6 On Responsive Boundary event handler code sample

```
console.log("Active box factor: " + event.boxFactor
```

Reminder: The On Responsive Boundary event does not fire every time a resize occurs. It only fires when a new box factor becomes active.

Layout controls: Responsive settings

When a box factor is active and a Horizontal or Vertical Layout's responsive behaviors option uses that box factor, any *Behaviors* configuration setting corresponding to the box factor entry is applied. Figure 4-55 shows a configuration setting example.

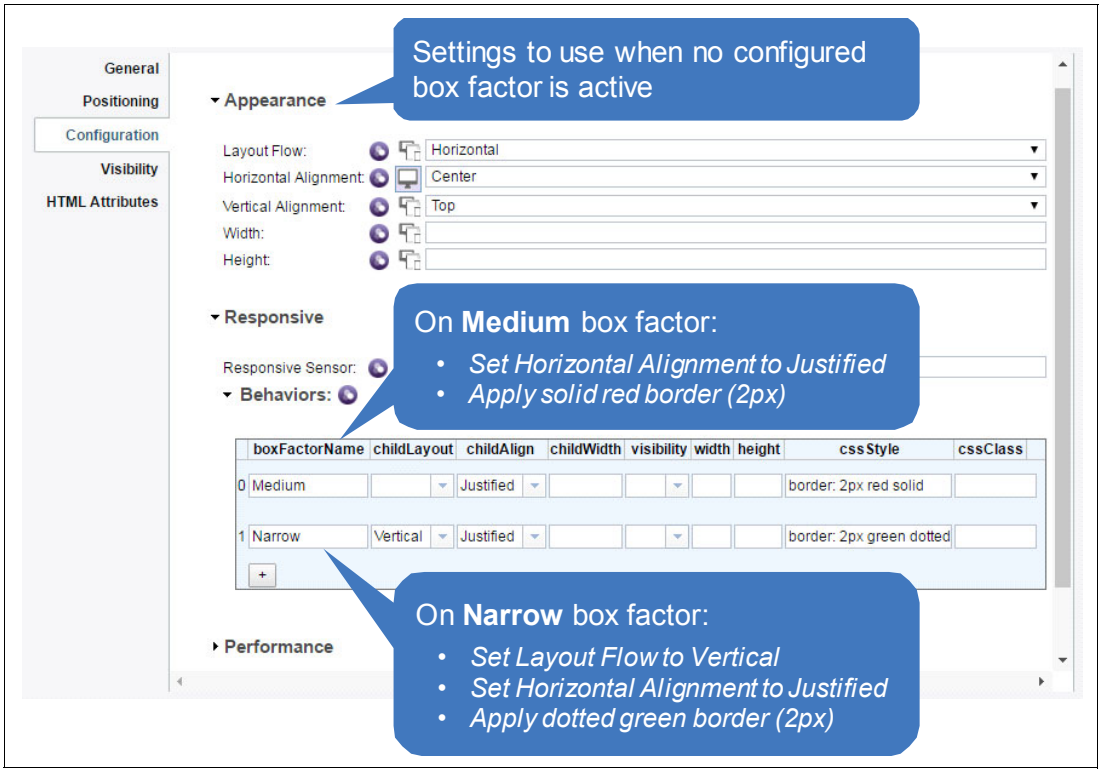


Figure 4-55 Layout configuration with responsive settings

Note: The example in Figure 4-55 assumes a Responsive Sensor contains the Layout and the Responsive Sensor is at least configured with the box factors named *Medium* and *Narrow*.

Table 4-7 provides descriptions of the Behaviors configuration settings for a Layout control.

Table 4-7 Responsive configuration settings reference

Setting	Description
boxFactorName	The name of the Responsive Sensor box factor that activates the configuration for the row of settings.
childLayout	Same as **Appearance** → **Layout Flow**: The Layout Flow applied to the child content of the Layout control (Horizontal, Horizontal Inline Scroll, Horizontal Tight, Horizontal Auto-wrap, Vertical, and Vertical Tight).
childAlign	Same as **Appearance** → **Horizontal Alignment**: The Horizontal Alignment applied to the child content of the Layout control (Justified, Left, Center, and Right).
childWidth	Widths of child controls contained in this Layout: If one width is specified (for example 100%), all children controls have that width. If two widths are specified (for example 64% and 33%), the first child has the first width, the second child has the second width, the third child has the first width, the fourth child has the second width, and so on. As many widths as needed can be specified; they are applied as per the child width algorithm described. They must be separated by a space. Units can be, for example, %, px, or em.
width	The Width of the Layout control (relative to its container if using %).
height	The Height of the Layout control (relative to its container if using %).
cssStyle	CSS settings to be applied to the `<div>` element representing the Layout control This is the same div as the `context.element` div reference from the Coach Framework API.
cssClass	CSS class to be added to the `<div>` element representing the Layout control This is the same div as the `context.element` div reference in the Coach Framework API.

Note: For any **Responsive** → **Behaviors** setting left blank, given a particular box factor name), the configuration specified under the Appearance configuration category applies.

4.6.3 Coach view responsiveness

This section helps the BPM UI developer decide when to use the Responsive Sensor in coach views in the following situations:

- Instead of relying on the built-in web editor grid in IBM Process Designer.
- Instead of using adaptive properties based on form (coach) factors.

Responsive behavior using built-in IBM BPM capabilities

Figure 4-56 on page 208 shows two composite coach views (Contact Information and Billing Information) stacked vertically on a coach. The responsive behavior of the views is based on

adaptive properties that adjust the arrangement of input controls as the form factor changes from Medium to Small.

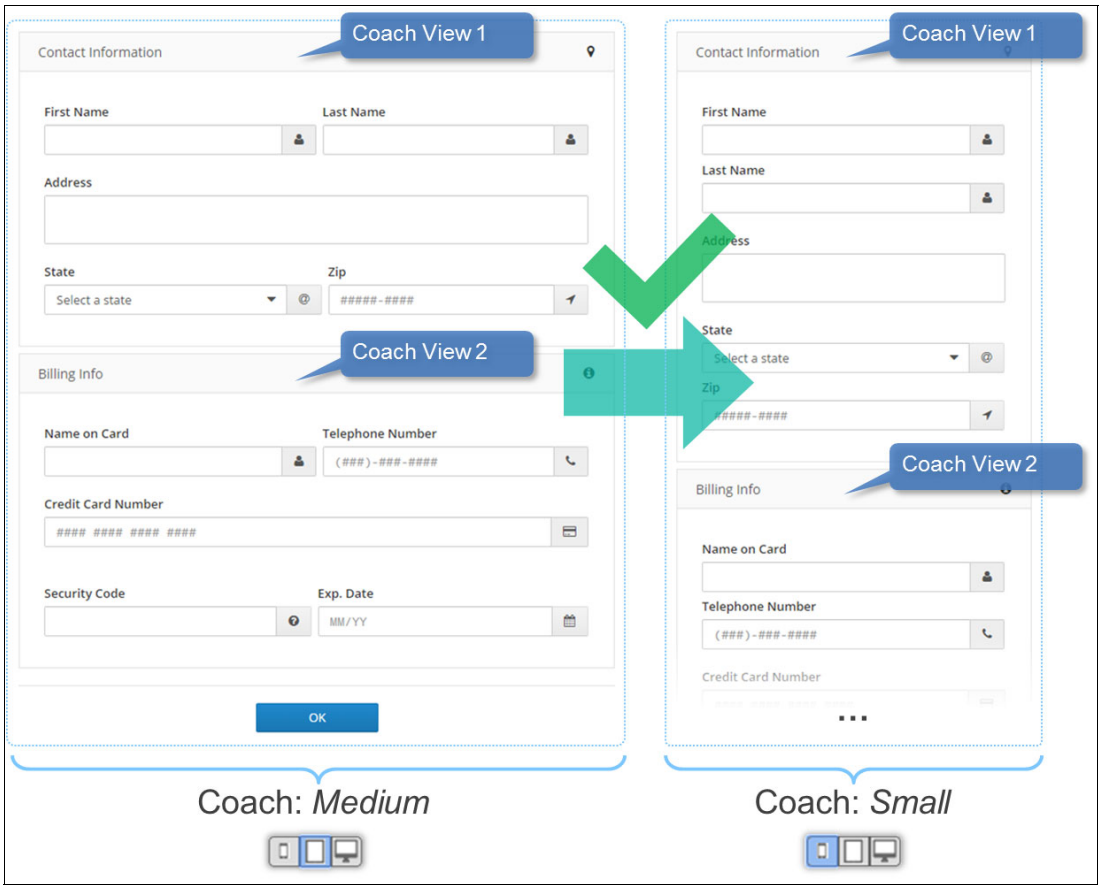


Figure 4-56 IBM BPM-based responsiveness, correct behavior

The form factor-based use case, shown in Figure 4-56, works equally well using the following options:

- The IBM Process Designer web editor grid
- The layout or alignment-related adaptive properties of the SPARK Layout controls
- The Responsive Sensor

However, when a UI developer expects that a composite coach view might be placed on a coach UI in such a way that it does not take up the entire width of the page, there are benefits with other options. The Responsive Sensor control, used inside the coach view, is the best option to predictably control the coach view content's layout behavior.

Without a Responsive Sensor, when the two coach views are laid out horizontally side by side, the layout cannot accurately determine when to reflow. This is because each coach view takes up 50% of the coach's horizontal space; a coach view is twice as narrow by the time the coach-driven form factor takes effect. Figure 4-57 on page 209 shows the *Error! Reference source not found* error.

Figure 4-57 Coach width-based responsiveness with side-by-side composite coach view

Responsive behavior using SPARK UI capabilities

When using a Responsive Sensor, the width of a coach is not considered unless the Responsive Sensor is configured as wide as the page. This means that coach view responsiveness is an aspect that is controlled by, and encapsulated in, the coach view. The same example shown in Figure 4-57 is provided for contrast in Figure 4-58 on page 210.

Figure 4-58 shows a Responsive Sensor and a Horizontal Layout configured with a single Responsive > Behaviors configuration option for a narrow (≤ 420 px) form factor.

Figure 4-58 SPARK UI-based responsiveness

4.7 Working tabular and repeating data

The SPARK UI Toolkit provides several ways of displaying repeating data. Repeating data can be displayed as follows:

- From IBM BPM, through human service data represented as local variables
- From an AJAX service as a list of simple types, string or number; or of complex types defined in IBM Process Designer as Business Objects

4.7.1 Table and Service Data Table controls

The two tabular data controls provided by the SPARK UI toolkit are as follows:

- The Table control, for working with IBM BPM human service data
- The Service Data Table control, for displaying data returned by a backing AJAX service

Table control

The Table control can be bound to a list of complex objects. It is a high-performance control to display data in cells as coach views or using other cell-rendering options. Figure 4-59 provides key feature and capability highlights of the SPARK UI Table.

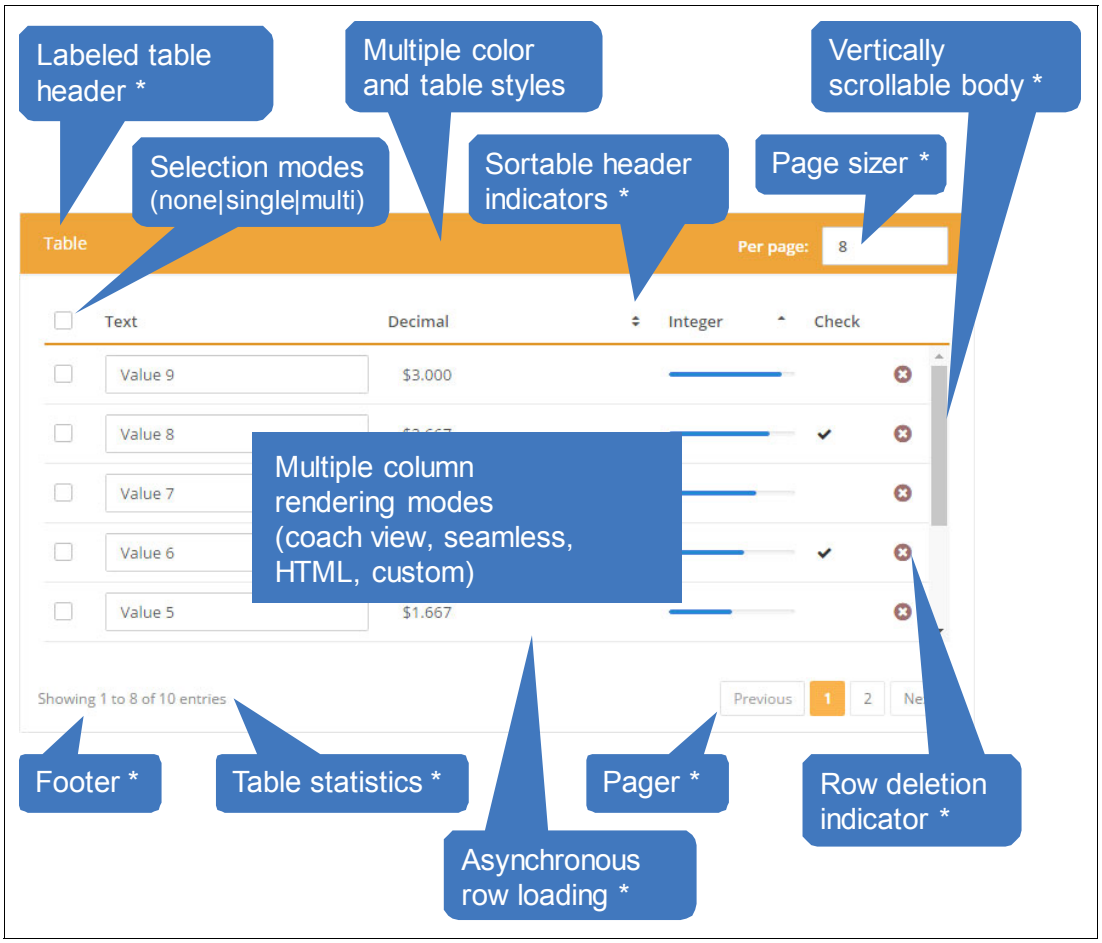


Figure 4-59 High-level Table control options (denotes optionally enabled features)*

To configure the columns in a table, various controls must be added to the columns of the table by dragging and dropping from the palette. When needed, the individual controls for each column can be bound to `<list>.currentItem.<property>` as required, as shown in Figure 4-60.

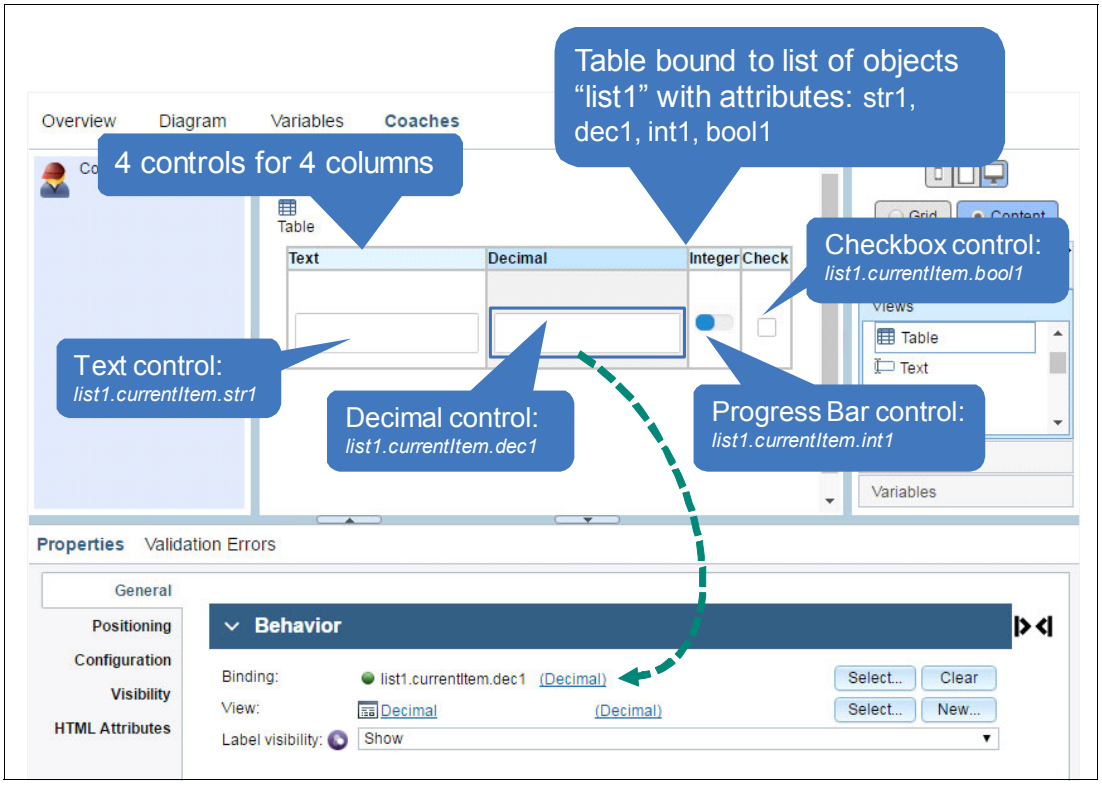


Figure 4-60 Binding table and column controls to BPM data

Note: Tables can only be bound to lists of complex types. For lists of simple types (such as strings, numbers, and so on) use a Layout control instead of a Table. Horizontal and Vertical Layouts can be bound to lists of simple types and, thus, become repeating containers, as described in 4.8.1, "Layouts and repeating data" on page 223.

Column configuration

By default, a table with no column configuration renders all columns containing a control as a coach view. Figure 4-61 on page 213 shows other rendering options available from the *Columns* table configuration option.

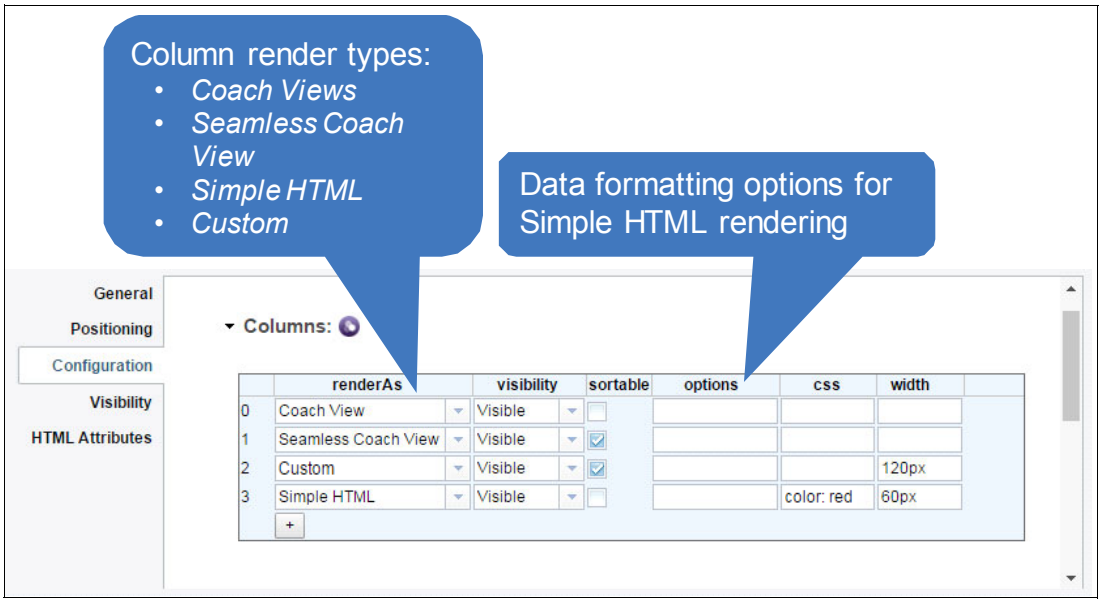


Figure 4-61 Table column configuration options

To fully configure columns in a table, a configuration entry must be explicitly added for each column mapping to a control in the table. The first control corresponds to column configuration entry 0, the second to entry 1, and so on.

When displaying in a table, the borders on an *input* type coach view control such as Text, Integer, Decimal, and Date Picker are distracting and add unnecessary clutter. The Seamless coach view option renders the coach views but automatically removes the border styling on input controls, as illustrated in Figure 4-62 on page 214.

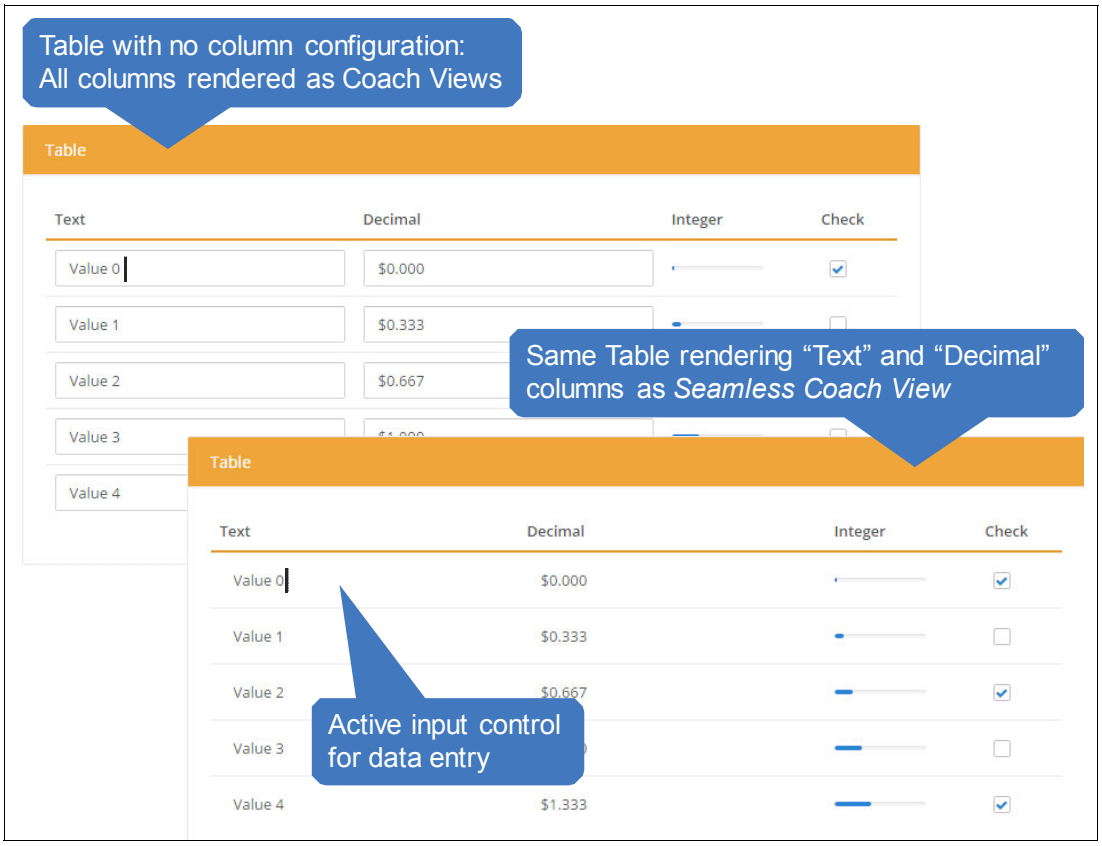


Figure 4-62 Comparison between coach view and Seamless coach view rendering options

Note: The Table control respects tab index on data entry. A control in the Table that is rendered as a coach view can be tabbed to without using a mouse click or an explicit finger tap. A currently active input control in the table shows the cursor and is styled with a slight grey background, as shown in Figure 4-62.

Simple HTML column rendering

Coach views are always more expensive to render than a simple HTML Document Object Model (DOM) element. This is because they incur the management and lifecycle overhead of the Coach Framework. When Table data shown in a column can be displayed statically, the UI developer might consider using Simple HTML rendering.

With Simple HTML rendering, the control that is dragged and dropped in the Table column at design time is *still necessary* because it gives the column a hint about how the data is rendered. For example, a Checkbox control might hint to the Simple HTML rendering mode that a boolean true value is rendered as a *check*, not as the text *true*.

At run time, instead of the table cell containing a live coach view, it contains a lightweight HTML representation. For example, a simple check mark (for a true value) is rendered in a cell in place of an actual live Checkbox coach view. Figure 4-63 on page 215 shows how Simple HTML rendering (for all columns in this example) is minimally configured and how it is rendered at run time.

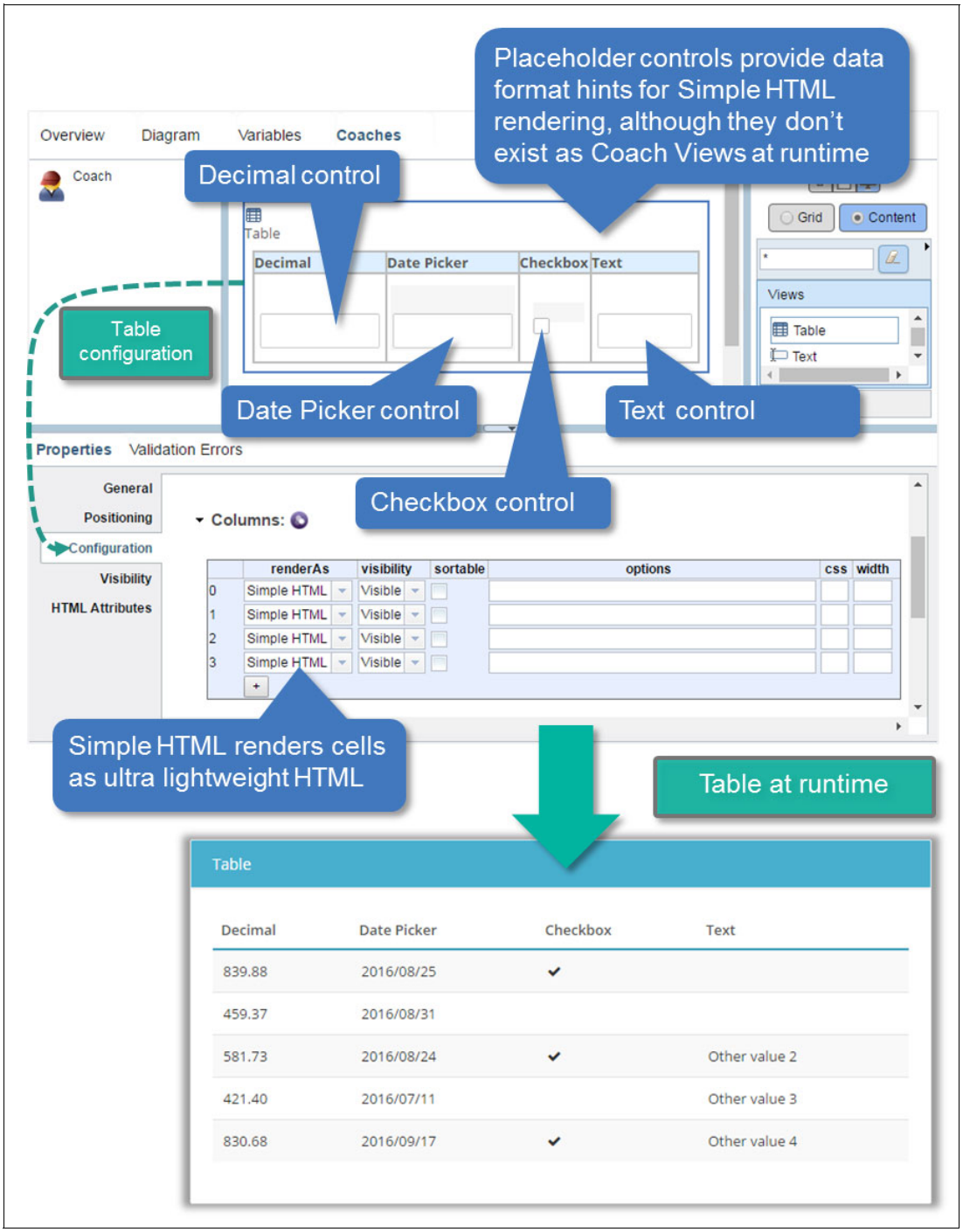


Figure 4-63 Basic Simple HTML column rendering for a Table (no format options specified)

Figure 4-64 on page 216 uses the same example as in Figure 4-63. However, Figure 4-64 on page 216 shows how data formatting options in Simple HTML rendering can be specified to influence the rendering of cell data.

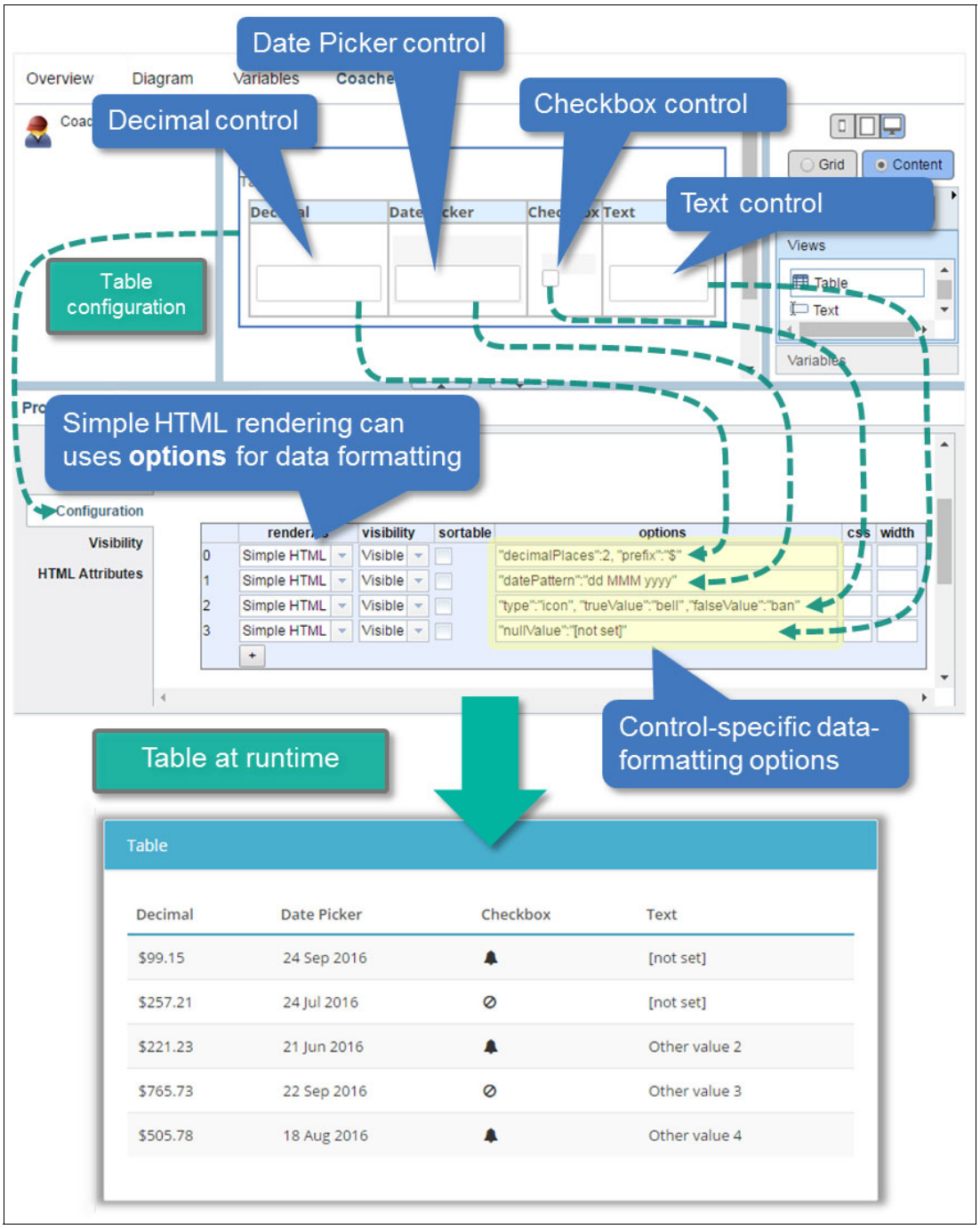


Figure 4-64 Simple HTML column rendering for a Table with data formatting options

Note: Do *not* omit quotes around the option keywords or they do not load properly. For example, `"dateFormat": "MMM yyyy"` works, but `dateFormat: "MMM yyyy"` does not work.

Table 4-8 on page 217 summarizes the supported data formatting options if Simple HTML or Custom rendering is used for a Table column (that is, the column containing the appropriate associated *hint* control).

Table 4-8 Simple HTML and Custom format options

Hint Control	Possible Options	Notes

Date Picker	`"datePattern": "<format spec>"`
	
	For example:
	`"datePattern":"MM/dd/yyyy HH:mm:ss"`
		Quick reference:
		► yy or yyyy for year (for example, 16 or 2016)
		► MM or MMM or MMMM for month (for example, 06 or Jun or June)
		► dd for date, meaning day, in month (for example, 14)
		► EEE or EEEE for day of week (for example, Fri or Friday)
		► z for time zone (for example, Mountain Daylight Time)
		► hh for 0 - 12 hour in day (for example, 08)
		► a for AM/PM indicator (for example, PM)
		► HH for 0 - 24 hour in day (for example, 20)
		► mm for minutes (for example, 54)
		► ss for seconds (for example, 42)
		
		Refer to `dojo/date/locale::format()` for a complete reference: https://dojotoolkit.org/reference-guide/1.10/dojo/date/locale/format.html

Checkbox	`"type": "<"check"|"icon"|"text">`
	`"trueValue": "<text or icon name if true>"`
	`"falseValue": "<text or icon name if false>"`
	
	For example:
	`"type":"text",`
	`"trueValue":"Y",`
	`"falseValue":"N"`
	Or
	`"type":"icon",`
	`"trueValue":"thumbs-up",`
	`"falseValue":"thumbs-down"`
		If the type is `"check"`, no other options are needed. A true value renders as a check and false renders as blank.
		If the type is `"icon"`, any icon name available in SPARK does not work. For example `"times"`, `"check-circle"`, `"battery-full"`, `"comment"`, and other types.

Decimal	`"decimalPlaces": <integer>`
	`"decimalSeparator": "<text>"`
	`"thousandSeparator": "<text>"`
	`"prefix": "<text>"`
	`"postfix": "<text>"`
	
	For example, for USD currency:
	`"decimalPlaces": 2,`
	`"decimalSeparator": ".",`
	`"thousandSeparator": ",",`
	`"prefix": "$"`
		A decimal can be formatted as an integer by specifying zero decimal places.

Integer	`"thousandSeparator":"<text>"`	
Text	`"nullValue": "<text>"`	Text to specify (if any) to display null or undefined values. The text content specified must be HTML-escaped. For example, specify `"<not set>"` for `<not set>`.

4.8 Searching content

Rows in the Table control can be searched either by matching content in a specific column or across all rows. The Table control does not provide a search text field ready for use. Instead, a simple and flexible method is provided to allow searching in different ways.

The Table **`search()`** command allows the following parameters:

- `columnIndex (0-based integer)`: Only match content in a particular column. If null, matches across all columns.
- `searchText (text)`: Search expression to matched.
- `caseInsensitive (boolean)`: If true, performs a case insensitive search.
- `regularExpression (boolean)`: If true, consider `searchText` as a regular expression.

Figure 4-65 on page 219 shows a Text control used for a simple search.

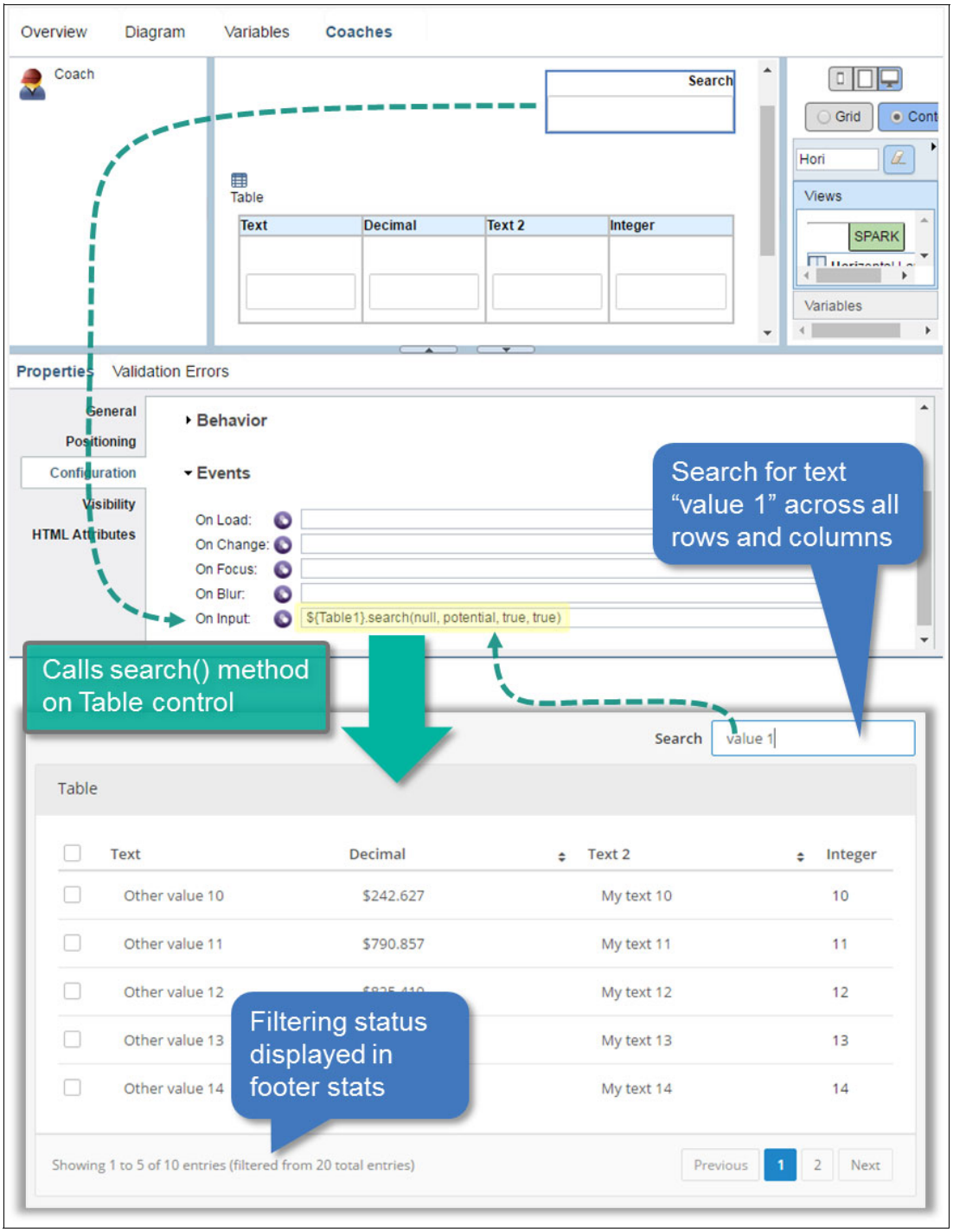


Figure 4-65 Table-wide text search example with Text search field

Note: Use the **`clearSearch()`** command to remove any filter on the table set through the **`search()`** command.

Service Data Table control

The Service Data table control retrieves data from an AJAX service instead of from human service data.

Because coach views are tied to human service data, the Service Data Table cannot render cells as coach views. They can render them only as Simple HTML and Custom. Instead of using a list-based data binding as its data feed, the Service Data Table control includes an AJAX service configuration option as its data source, as shown in Figure 4-66.

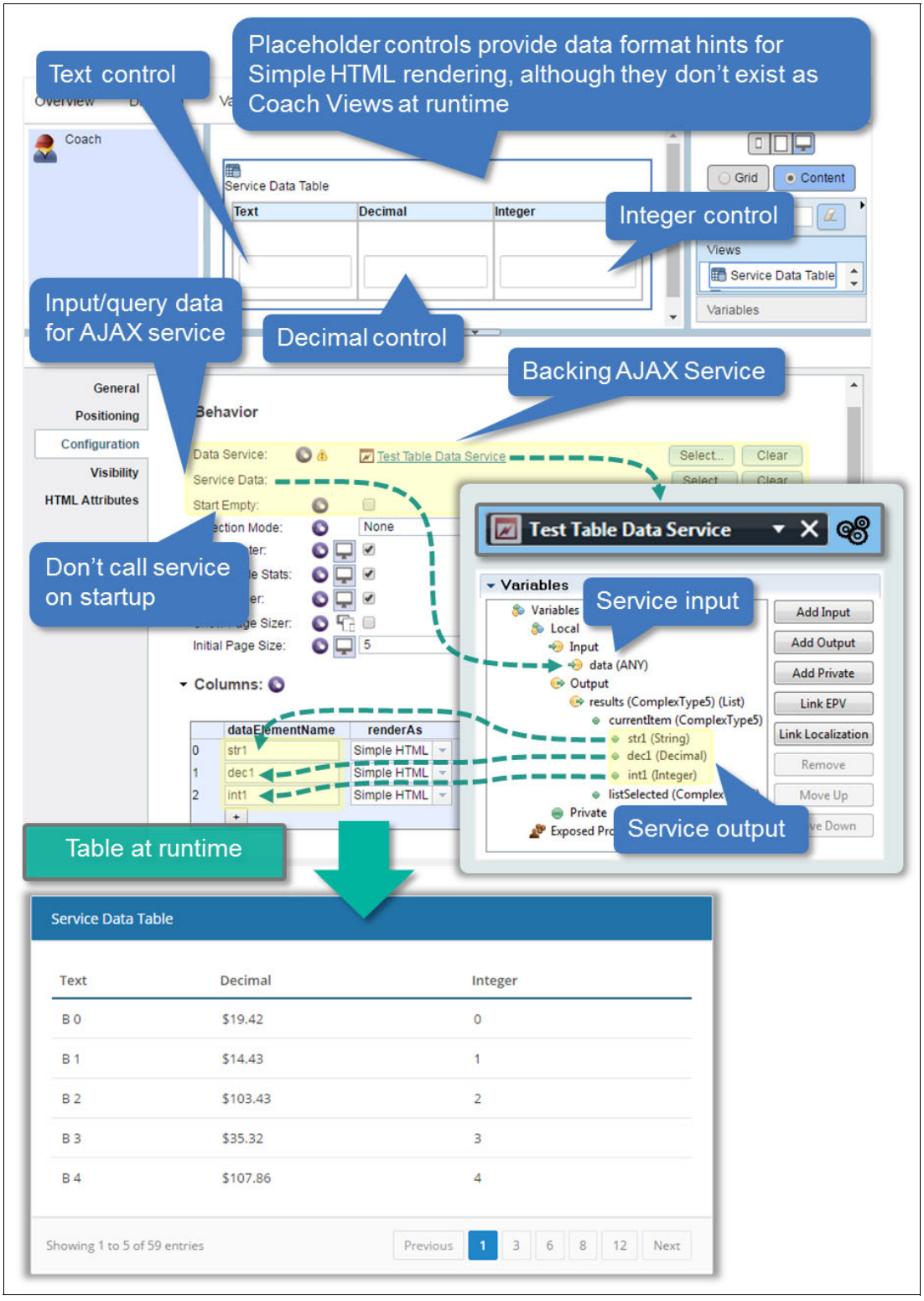


Figure 4-66 Service Data Table control configuration and attached AJAX service

Column configuration

Unlike the Table control, the Service Data Table must be configured for each column it displays. Column configuration includes the mapping of the data returned by the service and the type of rendering to use for the column, as shown in Figure 4-67.

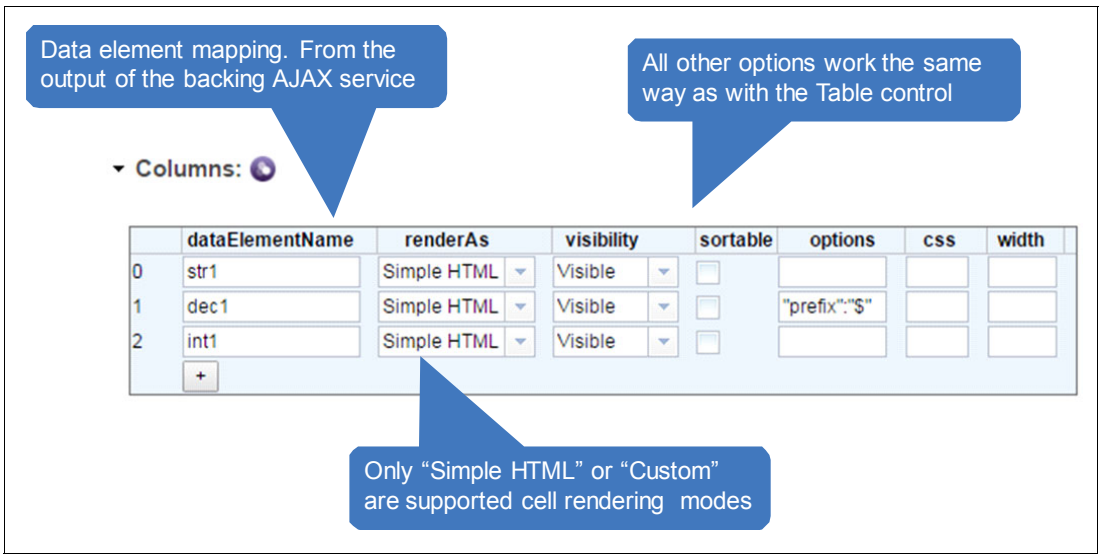


Figure 4-67 Service Data Table column configuration options

Querying and re-querying data

A Service Data Table might need to re-query its associated AJAX service with the same or with new input data to refresh its content or to issue a new query. Figure 4-68 on page 222 shows how this is done by calling the **`setQueryData()`** command (passing in new input data), and then by calling the **`refresh()`** command.

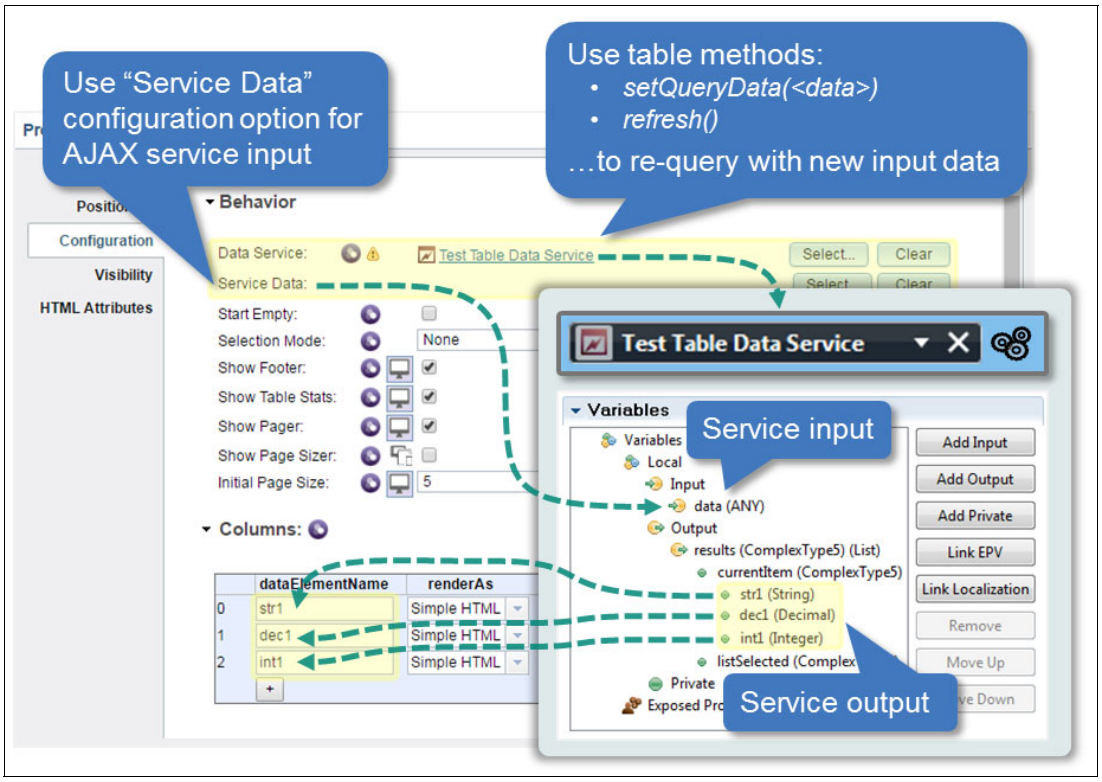


Figure 4-68 Service Data Table AJAX service re-query

If the associated AJAX service accepts a simple data type for input, such as an integer, the **`setQueryData()`** command can be invoked, as shown in Example 4-7.

Example 4-7 Invoking Service Data Table: setQueryData() with simple input

```
${Table1}.setQueryData(1200);
```

Example 4-8 shows how to pass a complex type as input to the AJAX query.

Example 4-8 Invoking Service Data Table:setQueryData() with complex input

```
${Table1}.setQueryData({"custName":"Smith", "accType":"C"});
```

Lastly, after setting the query data, the **`refresh()`** command must be called to invoke the AJAX service with the new input data. See Figure 4-69 on page 223.

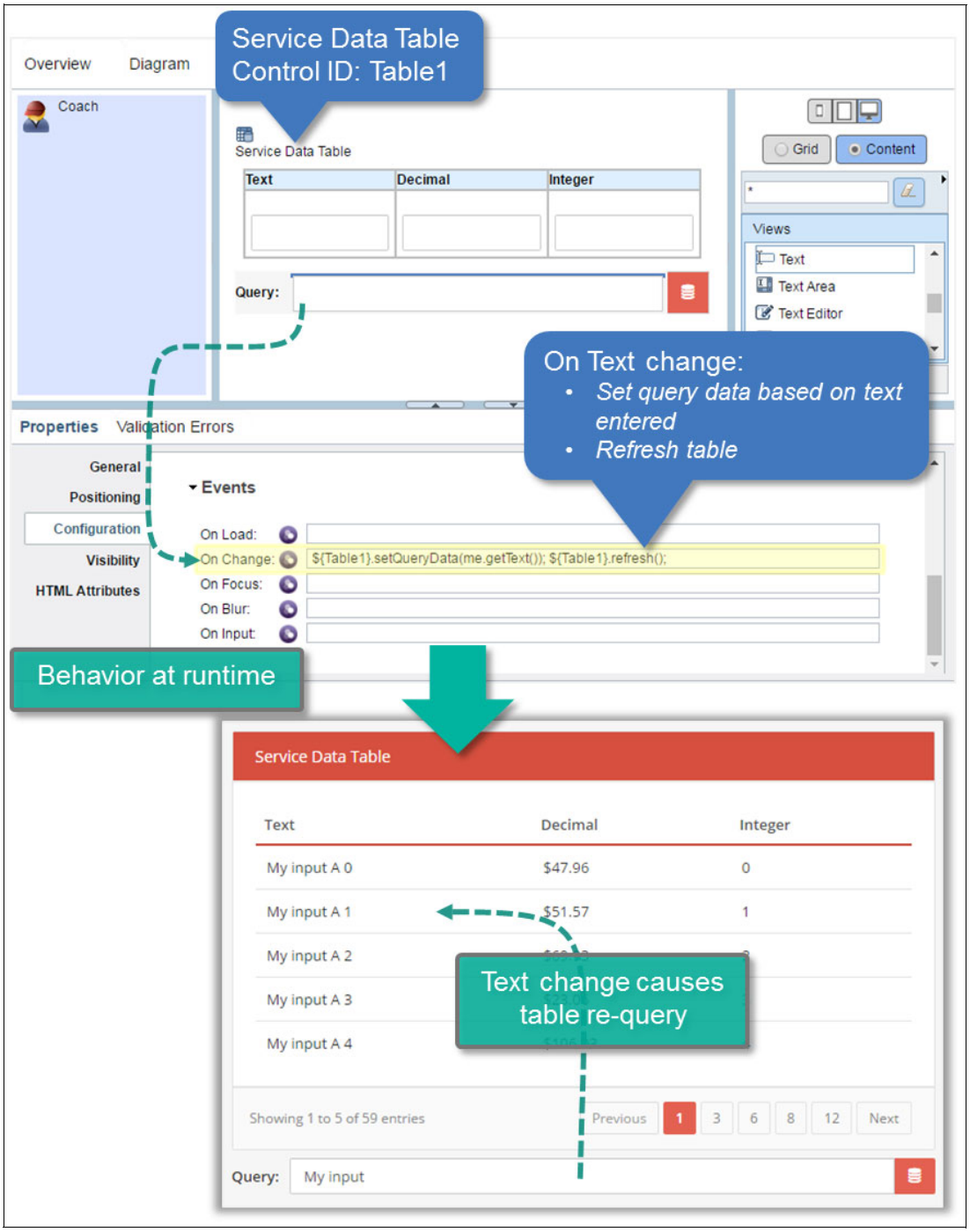


Figure 4-69 Example of Service Data Table re-query based around text update

4.8.1 Layouts and repeating data

When layouts are bound to a list, the coach views contained in the Layout that are bound to the current list item ("`currentItem`") are automatically repeated at run time. Figure 4-70 on page 224 shows that the Layout is bound to a list of strings containing user IDs.

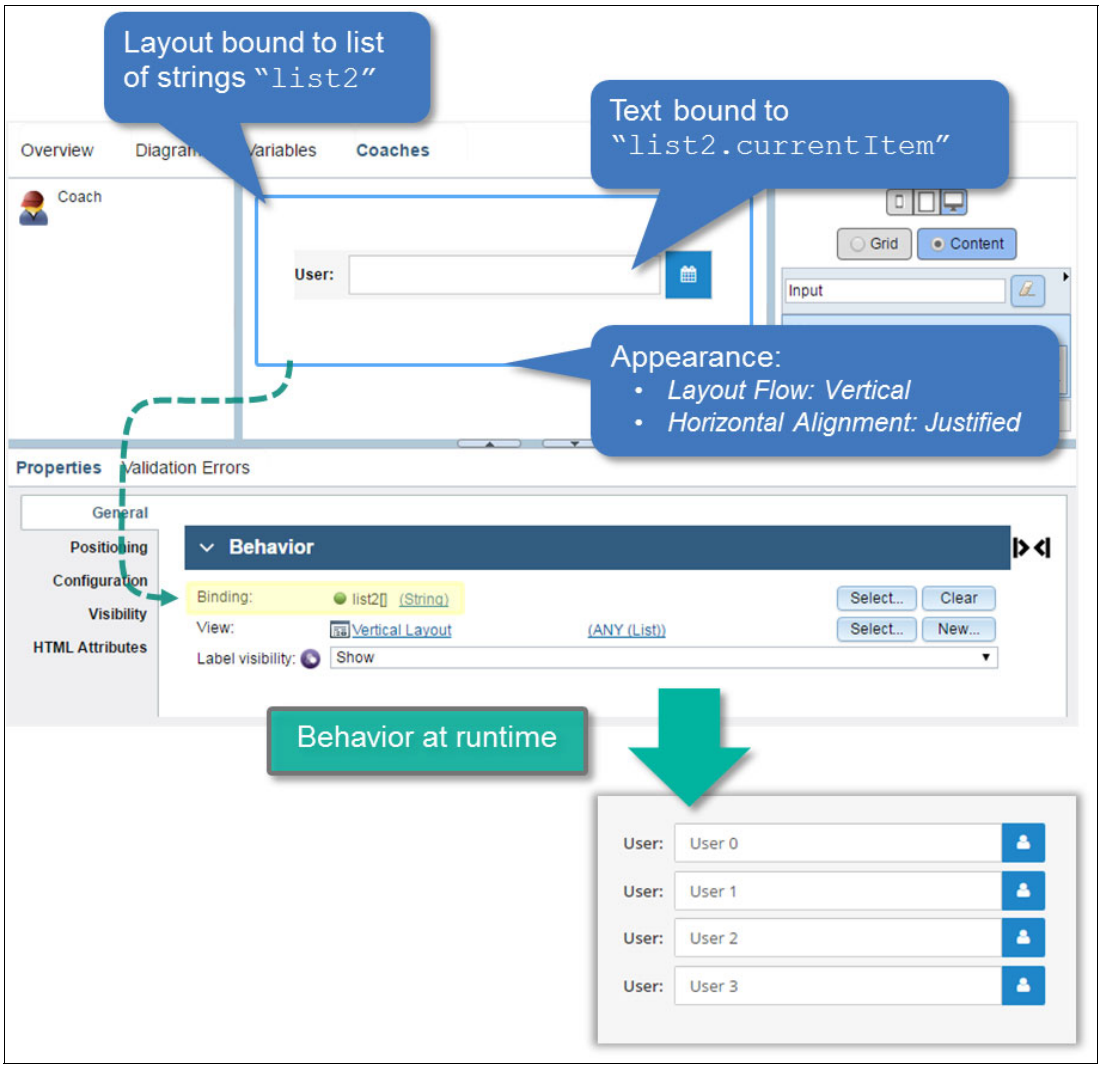


Figure 4-70 Repeating behavior of Layouts when bound to lists

Layout flow and alignment with repeating content

The Layout options in the following list, in addition to other options, are applied to the repeating content just as they are to any other child content:

- Layout flow
- Horizontal alignment
- Responsive behaviors

In Figure 4-71, the same layout as in Figure 4-70 displays user IDs using Layout Flow *Horizontal Auto-wrap* and Horizontal Alignment *Right*.

Figure 4-71 Layout flow and alignment applied to repeating Layout content

Performance options

As with the Table control, the Layouts control provides asynchronous loading capabilities. This allows large lists to display without disrupting the user experience. To engage this display, use the following configuration options in the Performance configuration category:

- Async Loading (same behavior as Table control option)
- Async Batch Size (same behavior as Table control option)

Note: Contained coach views can only be rendered as *coach views*. No Seamless coach views, Simple HTML, or Custom rendering options are currently available for a Layout.

The Horizontal/Vertical Layout control also provides methods to efficiently append and remove list elements instead of causing the reload of the entire list. Two of the main commands follow:

- **appendElement(<simple or complex object>)**
- **removeElement(<0-based index>)**

4.9 Formulas

Formulas are a simplifying and streamlining feature of the SPARK UI Toolkit. They can significantly reduce the need for creating code to maintain computational relationships between controls on a form.

Formulas provide support for simple expressions that prescribe how the value of a control is the result of an operation involving other controls. For example, the `${Total}` Decimal control is the sum of the `${Sales}` Decimal control and the `${TaxAmount}` Decimal control.

When the value of a control can be computed through a formula, the control provides a special option in the Formula category of its configuration options. Figure 4-72 on page 226 shows a Decimal control's configuration example.

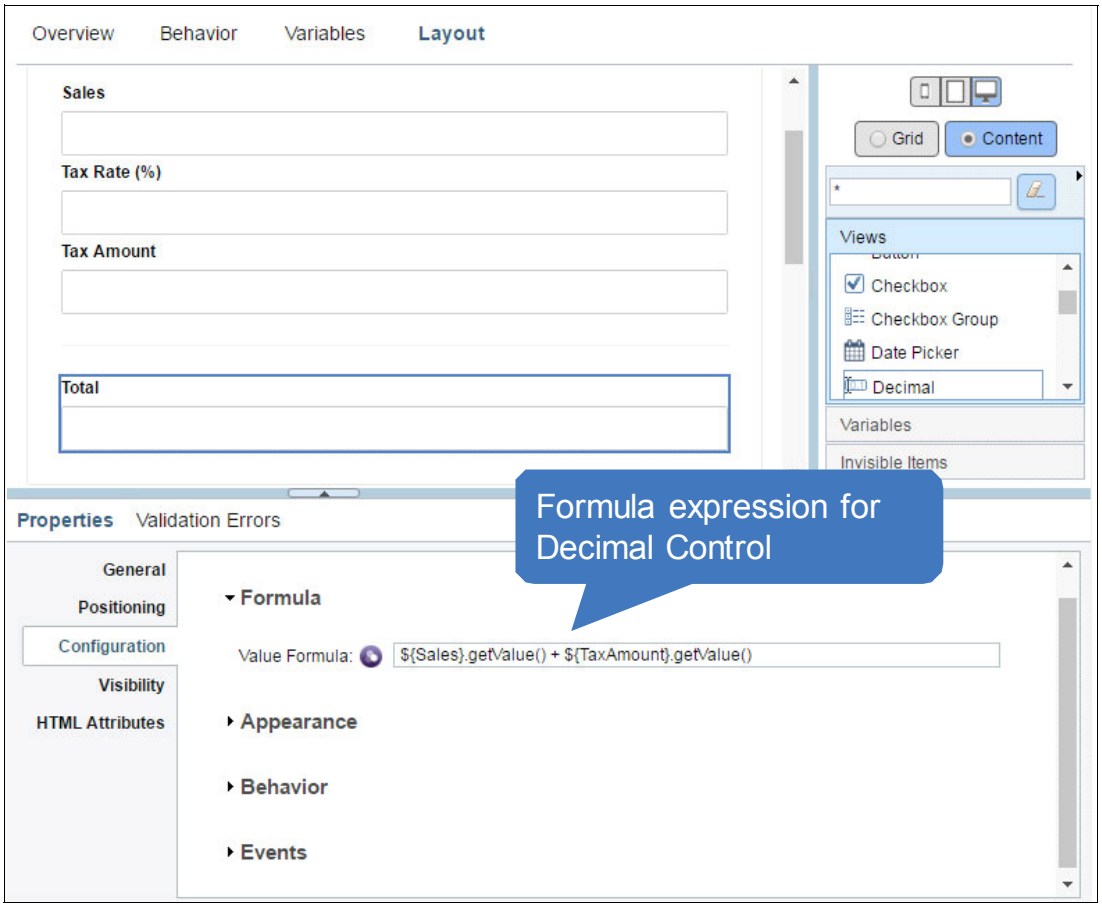


Figure 4-72 Formula configuration option example

After the formula is specified, the behavior of the control regarding the computation of its value does not require maintenance. Any updates to the values of any of the controls that are referenced in the formula causes the expression to recalculate automatically.

When a formula references a control whose value is itself computed through a formula, the cascade of updates occurs efficiently and in the correct order.

Note: As is usually the case for SPARK controls, data binding for controls is optional even when using formulas. If a control computed through a formula has a data binding, the bound value is automatically updated every time the formula result changes.

4.9.1 Syntax for control references

The same syntax and addressing behavior that works with event handlers works with formulas, and the notations seen previously work as expected. The following list notes some examples:

- A *Total* Decimal control refers to a sibling *Sales* control using the `${Sales}` reference.
- A *LineTotal* control in a table refers to a *Quantity* control in its same row as `${Quantity=}` (the equal sign meaning the same row as *me*).
- A *Price* control in a Table refers to a *Margin* control that is a sibling to the table containing Price as `${../Margin}`. This is because Margin is one level up in the view tree from Price.

Note: The `${ }` syntax is an optional convenience and can be replaced in event handlers with the less compact `page.ui.get("<control-id>")` or `view.ui.get("<control-id>)`. However, the compact notation is not optional with formulas.

The `${ }` notation (or the more compact `@{ }` notation explained in "Compact references") is mandatory in formulas. Otherwise, the result of the formula is not updated after its initial value.

Compact references

Because formulas are meant to be as simple to use as possible, SPARK supports a more compact notation than the `${ }` reference syntax. The expressions in Example 4-9 are fully equivalent and resolve to the same logic at run time. The last expression is visibly simpler to use than the first.

Example 4-9 Compactness comparison between ${ } and @{ } notations

```
${Sales}.getValue() + ${TaxAmount}.getValue()
…is equivalent to:

@{Sales} + @{TaxAmount}
```

The `@{ }` notation shortcuts the various implied control-specific methods calls such as those noted in the following list:

- `getText()` for a Text control
- `getValue()` for a Decimal or Integer control
- `getProgress()` for a Progress bar

The `@{ }` notation requires less programming skills to express.

That said, the formula expression becomes a JavaScript expression at run time. Thus, the sophistication of a formula is constrained only by the following items:

- The formula must be a single JavaScript statement (but ternary operations such as `<condition> ? <statement 1> : <statement 2>` are allowed).
- At run time, the formula, after references have been translated to JavaScript, is placed inside a `return( )` statement.

For example, if the result of the formula `"@{Sales} + @{TaxAmount}"` must be fixed to two decimals, Example 4-10 shows how it can be altered.

Example 4-10 Using JavaScript in formulas

```
(@{Sales} + @{TaxAmount}).toFixed(2)
```

Note: The `@{ }` compact notation only works in formulas and does not work in event handlers.

4.9.2 Formulas referencing specific controls

Formulas between scalar (non-repeating) controls need to reference the controls involved in the calculation and add needed logic in the expression to produce the result. Figure 4-73 on page 228 computes the TaxAmount control's value.

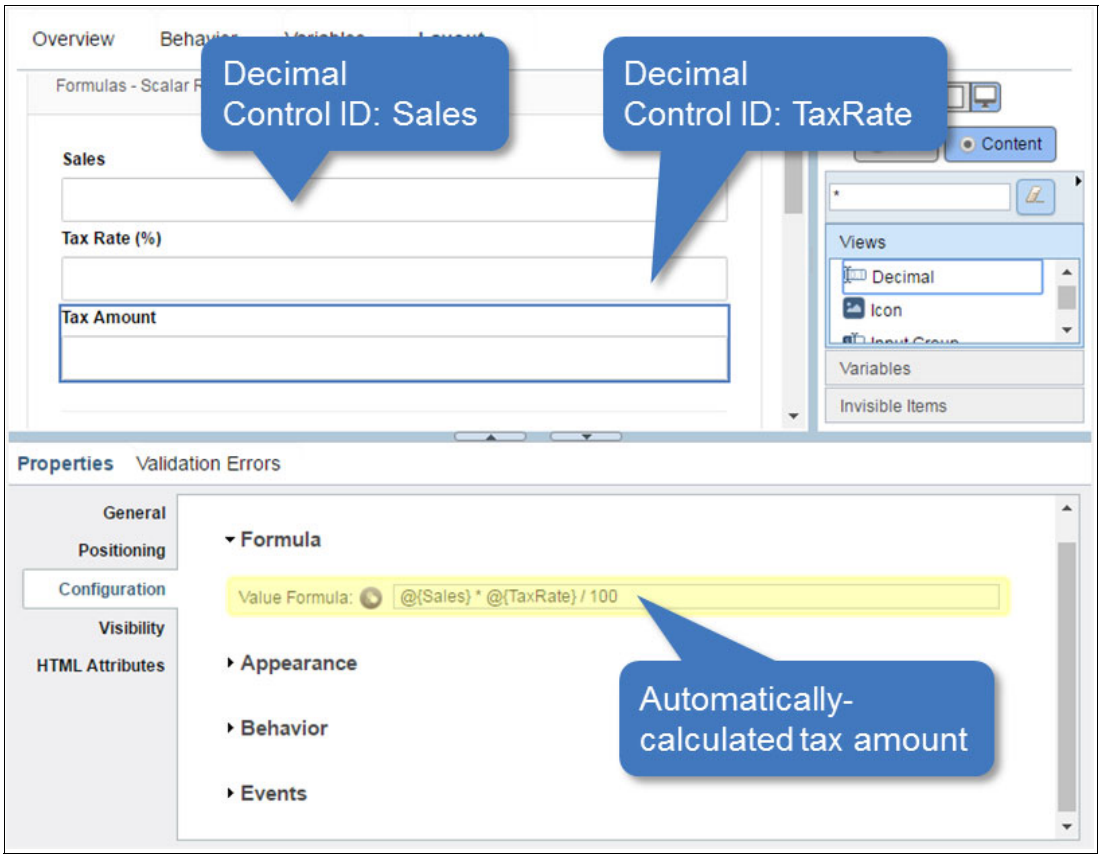


Figure 4-73 Formula expression with two specific control references

References between controls in a table

A special case of control-to-control references involves a control in a table row referencing another control in the same row. Figure 4-74 on page 229 shows how the *LineTotal* Decimal control references its peer (meaning same row, same index) controls. The *Price* and *Quantity* peer controls are referenced through the `@{<control-id>=}` notation.

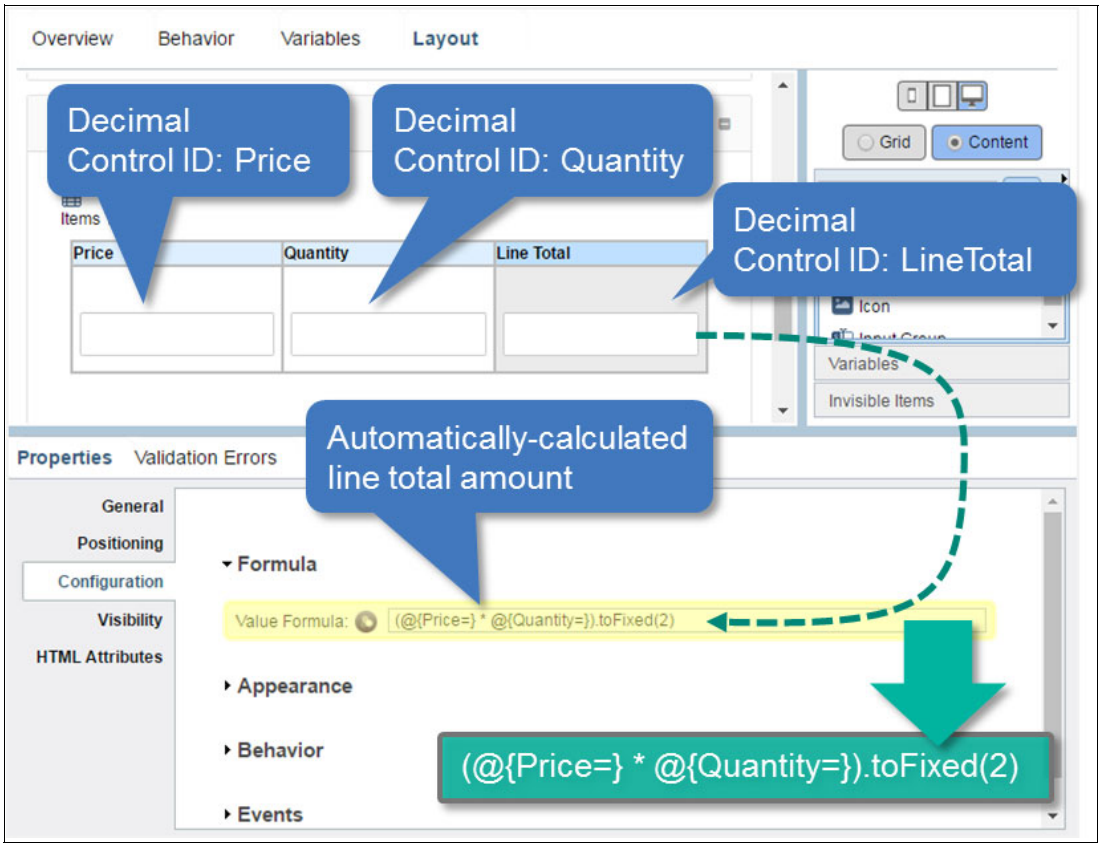


Figure 4-74 Same-row referencing in formulas

4.9.3 Formulas for control aggregates

Formulas can also be used to compute results from repeating controls in a table. For example, all repeating instances of a *LineTotal* control in a table can be summed into a grand total.

Example 4-11 shows how the `@{ }` notation allows formulas to refer to all controls named `<control-id>`.

Example 4-11 Using the aggregate reference notation in formulas

```
@{ItemsTable/LineTotal*}
```

The asterisk (`*`) symbol at the end of the reference in Example 4-11 indicates that all controls named `LineTotal` are under the Table control named `ItemsTable`.

Functions used with aggregate references

The functions currently available to work with aggregate references are noted in the following list:

- `SUM( )`
- `COUNT( )`
- `MIN( )`
- `MAX( )`
- `AVG( )`

Because those functions work with aggregate references, they are referred to as aggregate functions in the SPARK UI Toolkit.

Figure 4-75 shows the use of the `SUM()` function combined with the aggregate notation for all controls named `LineTotal` under the Table named `ItemsTable`.

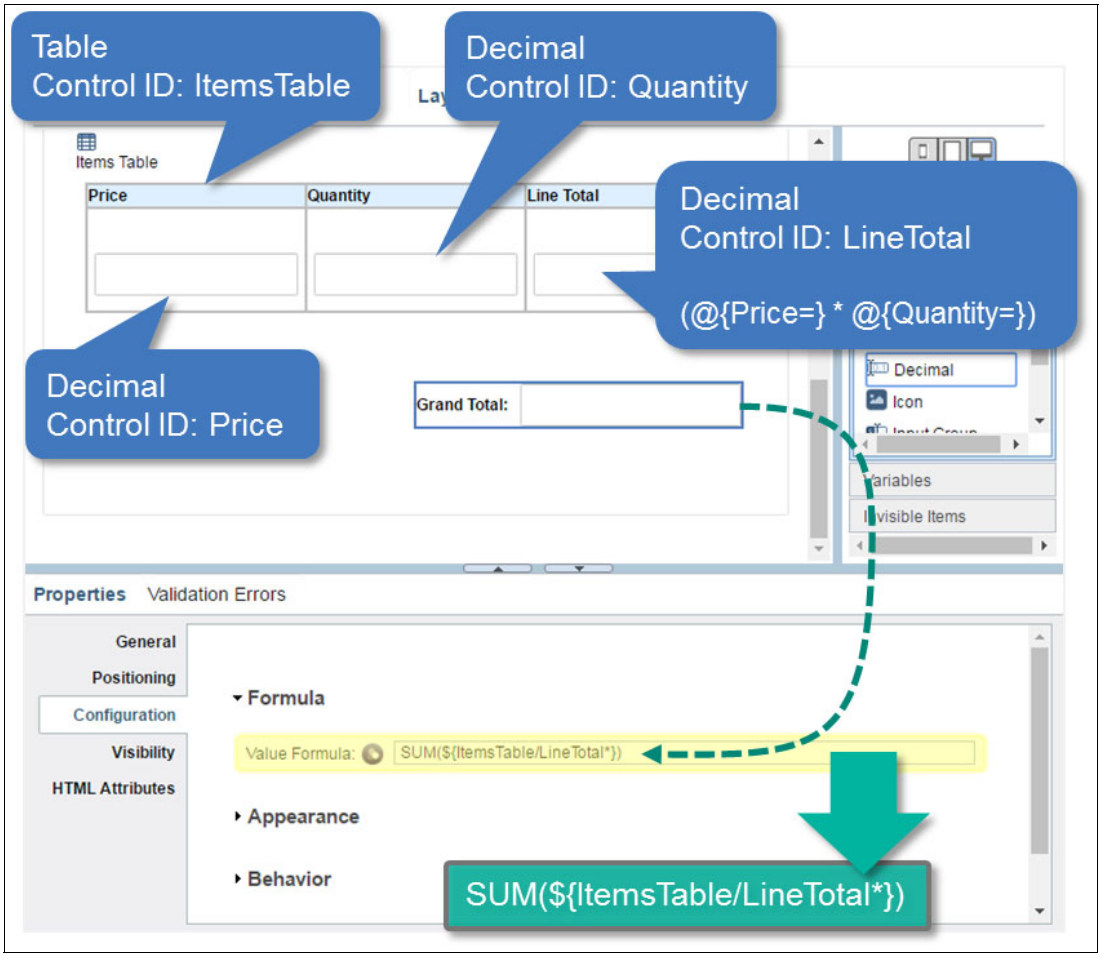


Figure 4-75 Formulas referencing a control aggregate with the @{<control-id>} notation*

As is the case for all other formulas, value changes in any field directly or indirectly referenced by the `SUM()` function causes the formula using the `SUM()` function to be recalculated.

Figure 4-76 on page 231 shows the runtime behavior for the UI composition.

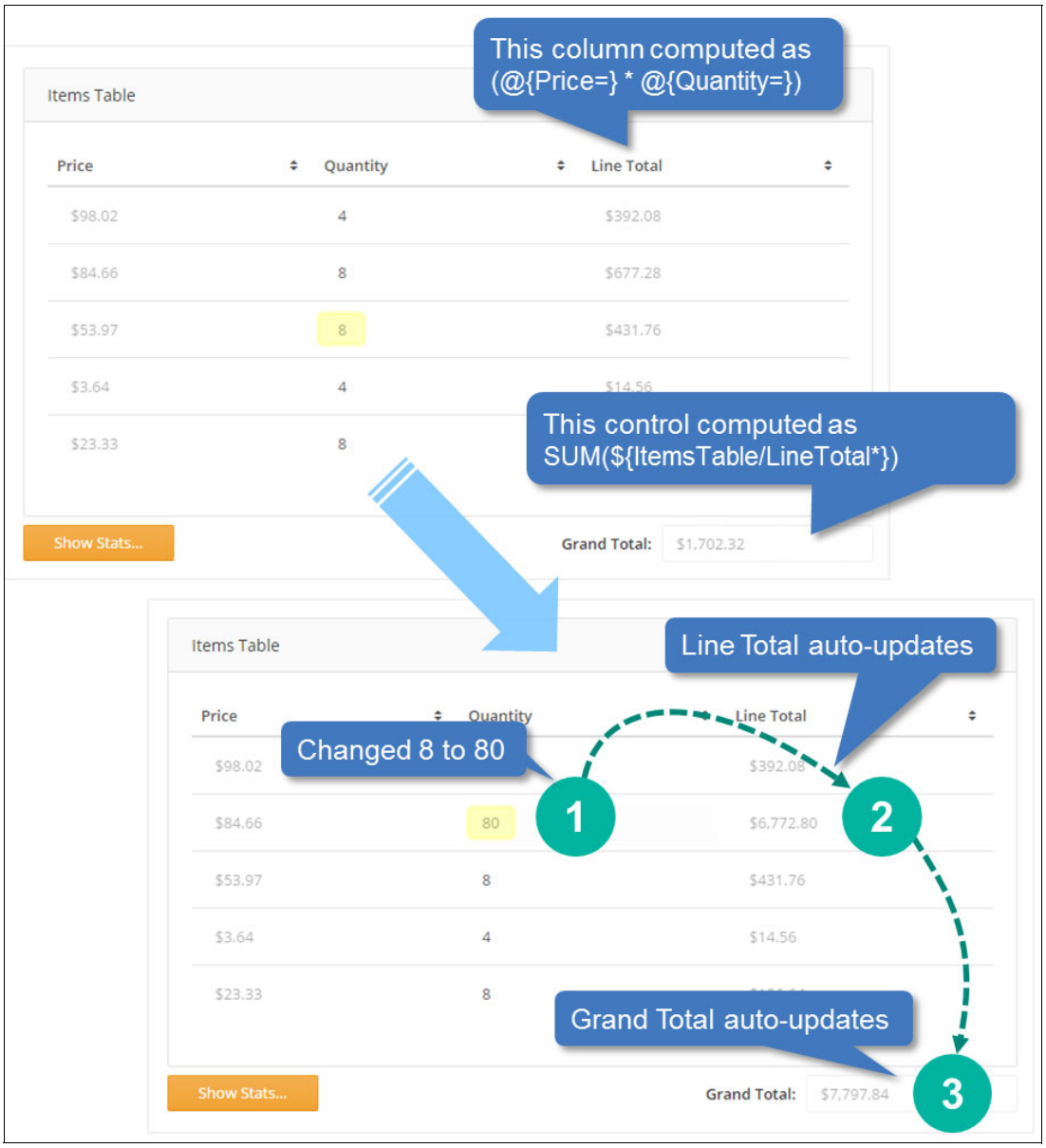


Figure 4-76 Runtime behavior of aggregate references and functions

4.9.4 Working with data directly

So far, all formula examples given have depended on controls on a coach or in a coach view. This works well as long as the controls are in existence, even if they are hidden. Formulas are also able to handle delayed instantiation of controls. Those results are automatically updated as the controls come into existence, even asynchronously.

There are specific situations, such as using paging with tables, when a portion of the controls in the table might never come into existence. For example, the user might never display a particular table page. In such cases, the formula does not have access to all the controls it needs to consider for the computation to show the expected result.

To illustrate the use case, Figure 4-77 on page 232 shows a grand total computation based on a `@{ItemsTable/LineTotal*}` formula with a paging table. The illustration shows before and after the second page is displayed.

Figure 4-77 Limitations of @{<control-ref>} formula notation with paging tables*

For such cases, the SPARK formula framework provides support for a special notation. This notation works directly with the data backing the Table control instead of relying on control references inside the table. This special notation must be used in conjunction with an aggregate function, as shown in Example 4-12.

Example 4-12 Direct data reference in a Table control

```
SUM(${ItemsTable}, FOR_EACH{#{lineTotal}})
```

It is assumed in Example 4-12 that the ItemTable control is bound to a list of objects containing the following attributes:

- Price
- Quantity
- LineTotal

With such an approach, the total is always correct. The formula does not depend on the existence of the controls in the table and directly uses the table's backing data.

The `FOR_EACH` expression part, however, provides one more option. Because the SPARK UI Toolkit does not make data binding mandatory, you might expect a Line Total control to have no backing data because its value is derived through client-side calculation. In such a case, the UI developer needs the sum of price x quantity calculated for each row.

Accordingly, the expression in Example 4-13 is supported.

Example 4-13 Using FOR_EACH { } to compute sub-expressions in data-backed formulas

```
SUM(${ItemsTable}, FOR_EACH{#{price} * #{quantity}})
```

Note: The content of the `FOR_EACH{ }` clause can be any valid JavaScript expression that returns a result.

Tip: Use the proper syntax with data-backed expressions. Note where parentheses are used and when curly braces are used. They are *not* interchangeable.

4.9.5 Formulas for non-numerical computations

Formulas do not need to produce numerical results. They work just as well with string data, or other types of data. For example formulas can also be used to set the following items:

- The text value of an Output Text control
- The title of a Panel or Collapsible Panel control
- The text value of a Note control
- The content of a Tooltip control
- The data for a Data control
- The text content of a QR Code control to be graphically represented
- The label of a Button control

Figure 4-78 on page 234 shows a formula used to set the title of a Panel control to the following sequence:

- `Hello` and the name entered in the Text control
- `Hello user`, if no text is entered

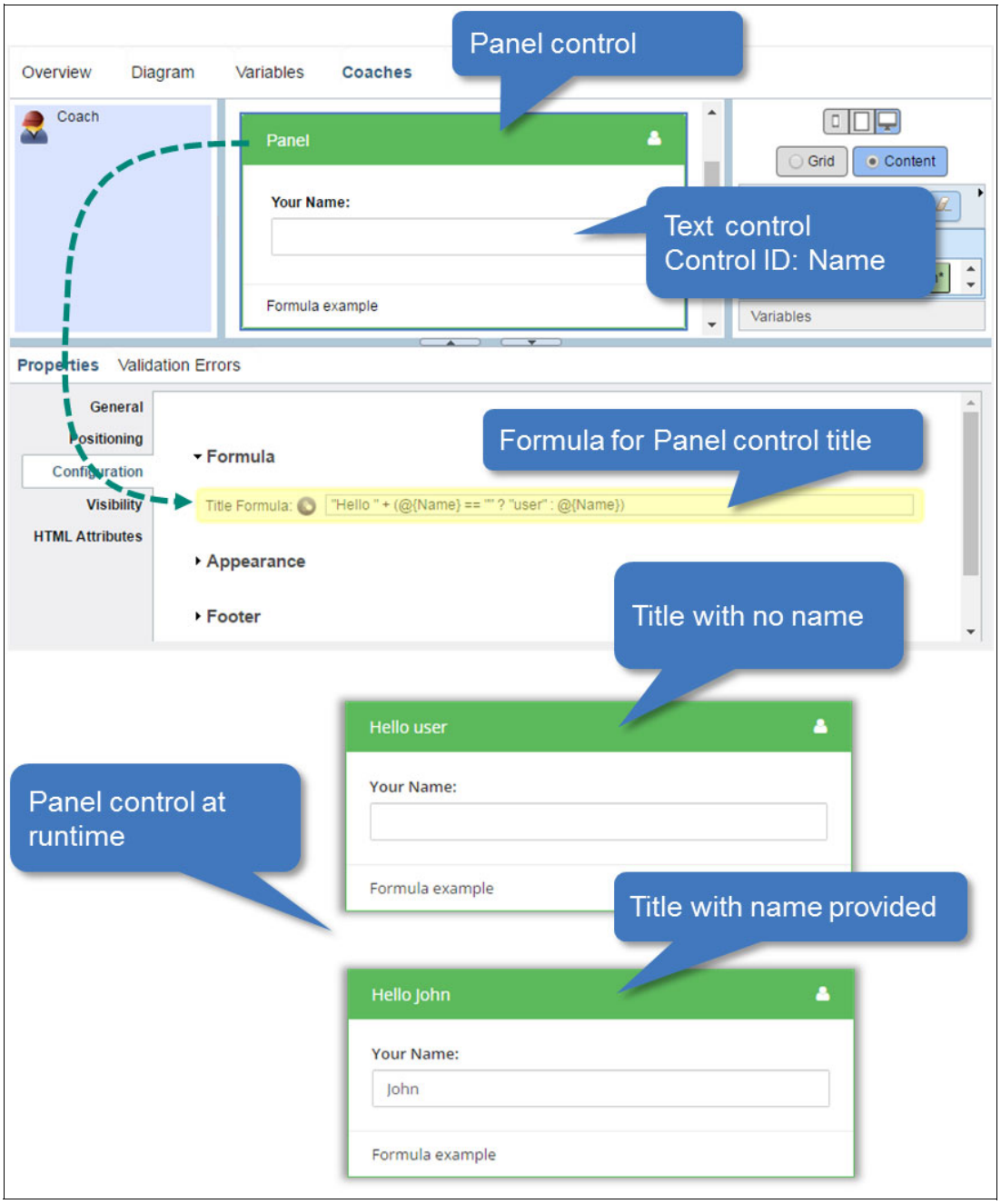


Figure 4-78 Using formulas for non-numerical computations

4.9.6 Formulas for control-value initialization

A convenient use of formulas, especially for controls that are not bound to data, is for control value initialization. Figure 4-79 on page 235 shows the value of an Output Text control initialized to a specific value.

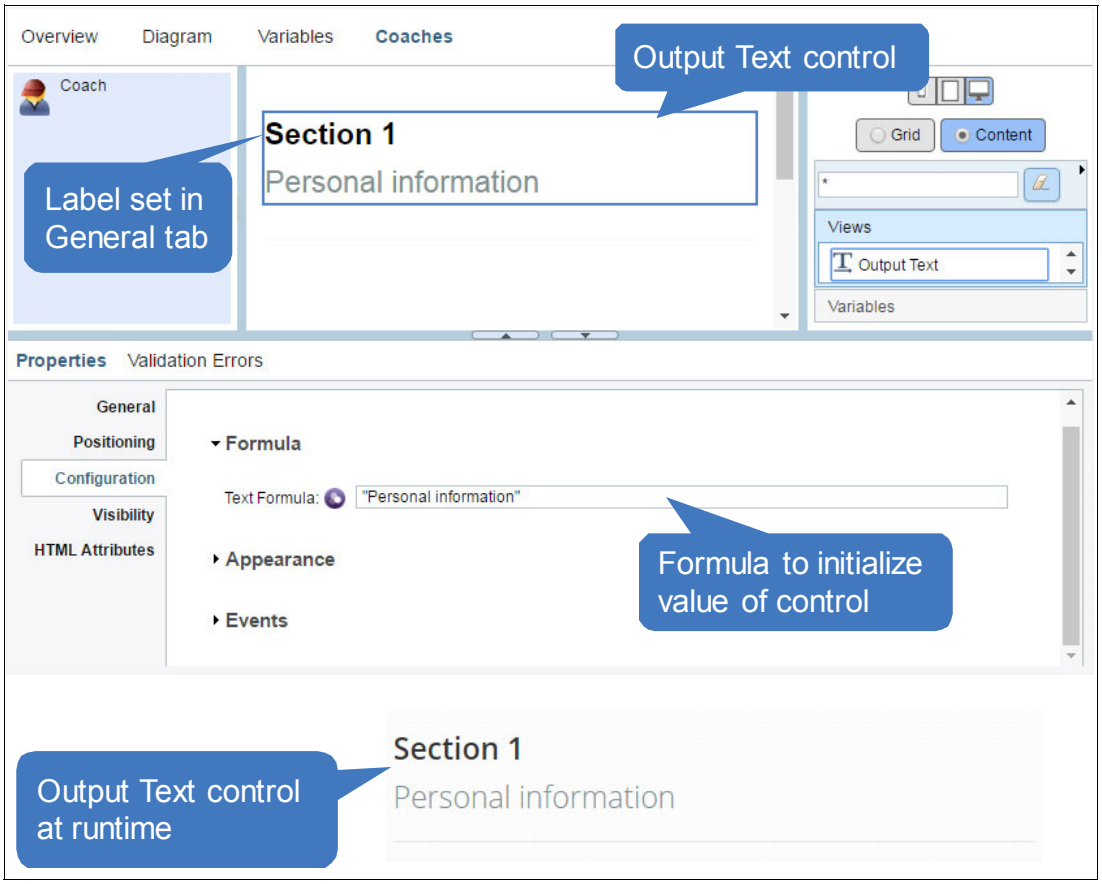


Figure 4-79 Initializing unbound control data with formulas

In general, formulas do not need to contain references to other controls; a simple JavaScript expression works just as well. However, the content of the formula expression is JavaScript and, therefore, a string specified for initialization *must* be contained in quotes.

4.10 Reporting and analytics

The ability to visualize data in charts, to render tabular data, and to provide analytics capabilities for BPM and business data in general is built into the SPARK UI Toolkit. That process is done through three key enabling aspects:

- Controls: Seven chart types represented as seven different controls
- Drill-down framework: An XML-configurable and customizable drill-down framework that automatically gives charts drill-up and drill-down capabilities
- AJAX service-backed tables: Service Data Table control that can be combined with charts and the events they provide to zoom in on data represented in charts

This section covers each key aspect of the SPARK UI reporting and analytic features.

4.10.1 Charts

SPARK UI Charts are highly configurable and interactive controls that can represent data from a backing AJAX service or from human service data. The SPARK UI Toolkit includes seven chart controls that can be categorized in three broad classifications:

- Composition: Pie Chart, Donut Chart
- Comparison or Evolution: Bar Chart, Area Chart, Line Chart, and Step Charts
- Profiling: Multi-purpose chart

Figure 4-80 on page 237 provides a sampling of many of the charts types included in the SPARK UI toolkit.

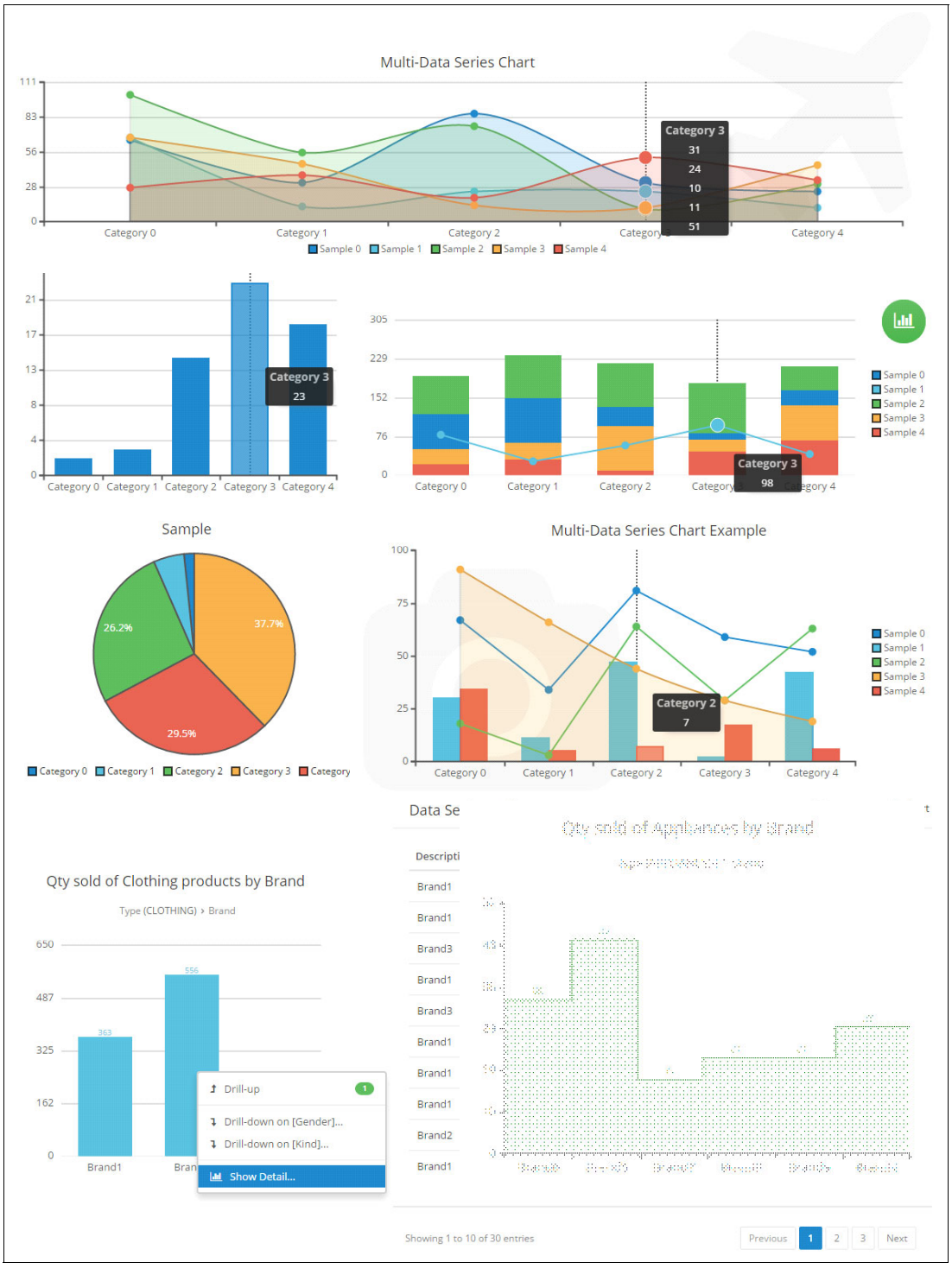


Figure 4-80 Sampling of SPARK UI Toolkit charts

Connecting charts to data

Charts can visualize data retrieved through a configured associated AJAX service or, less commonly, through data from the human service. Figure 4-81 on page 238 shows a chart control retrieving data through an AJAX service.

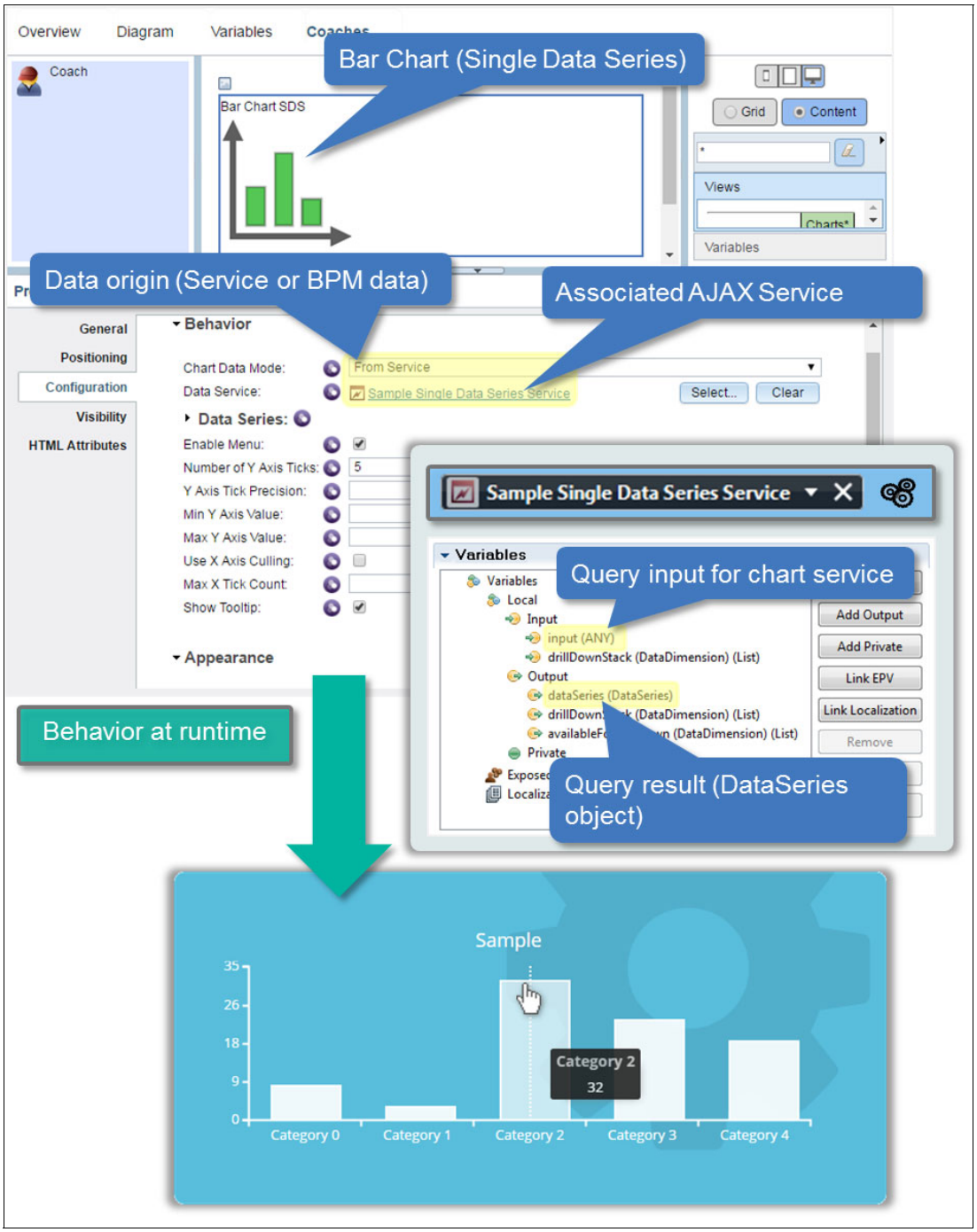


Figure 4-81 Bar Chart backed by AJAX service

Note: The Chart AJAX service developer only needs to work with the *input* input variable and the *dataSeries* output variable. All other inputs and outputs are automatically populated by the Chart control or by the Drill Down service.

Figure 4-82 on page 239 shows the AJAX service sample code used to populate the bar chart example in Figure 4-81.

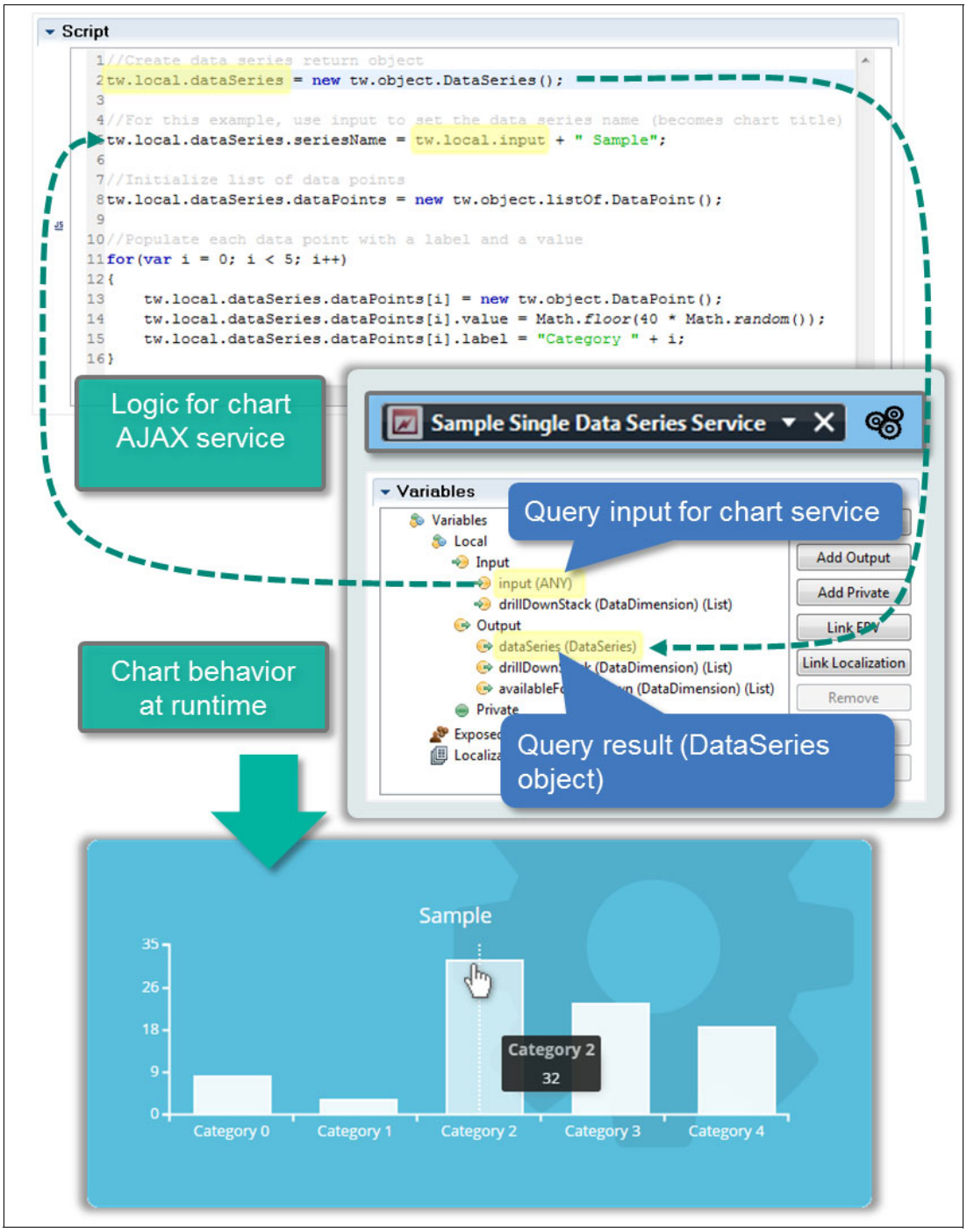


Figure 4-82 Sample AJAX service code in context of Bar Chart

In summary, to make a single data series chart work, the UI developer must complete the following steps:

- Place a chart on the coach or coach view canvas.
- Configure the chart to use an AJAX service; create a chart service based on the AJAX service named `Default Chart Single Data Series Service`.
- Create logic in the AJAX service to populate the `tw.local.dataSeries` object.
- Configure the chart's visual appearance, if required.

- Configure the chart's behavior (such as bounds for the X or Y axis, or both, data tooltips), if required.
- Playback the chart and adjust the appearance and behavior, if required.

Working with charts programmatically

Charts provide a number of methods to influence behavior, visual representation, and data query or re-query.

Figure 4-83 shows how to influence what a chart displays based on input data that is used in the chart's AJAX query. The illustration shows a media consumption chart that adjusts its data based on the region selected.

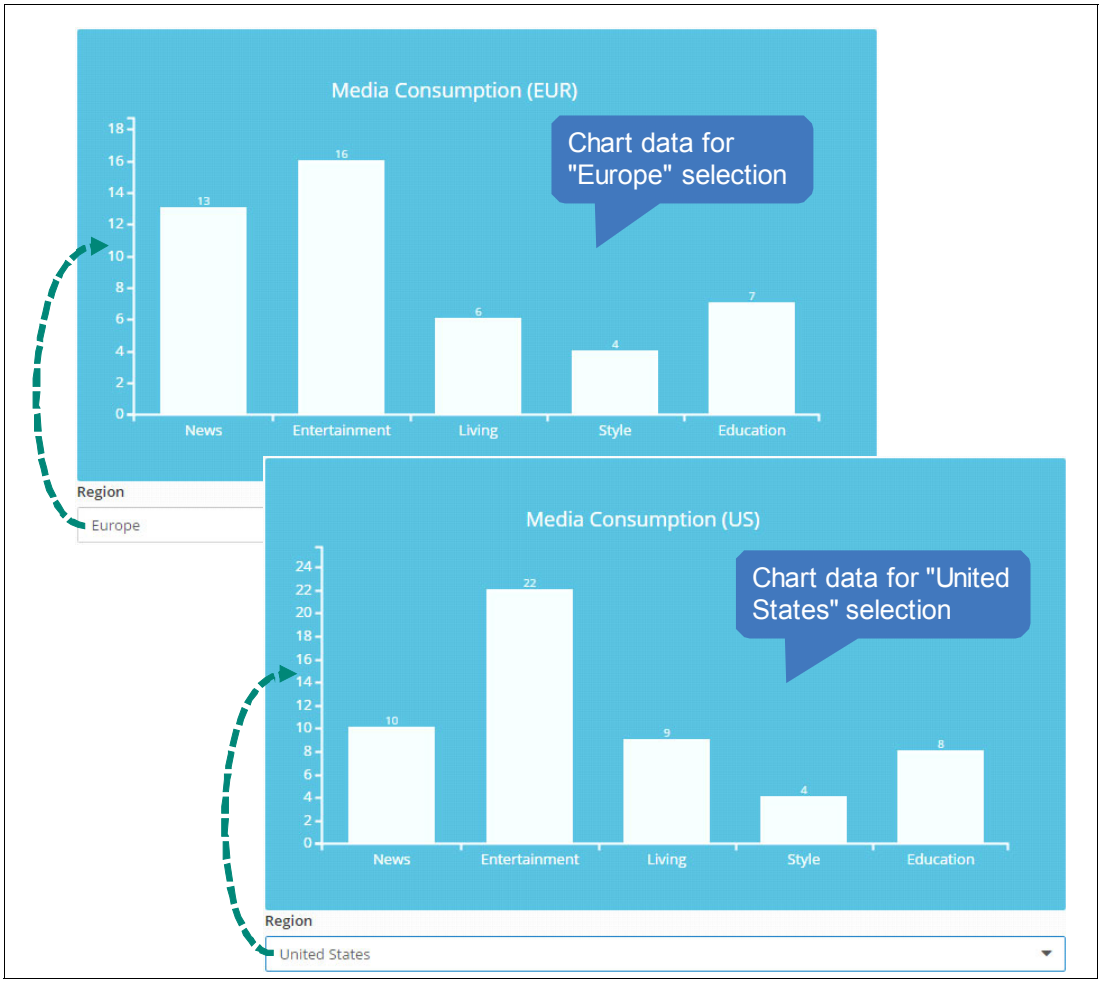


Figure 4-83 Example media consumption data represented in bar chart based on region

The behavior in Figure 4-83 is modeled with the following options:

- A Bar Chart SDS control.
- A Single Select control containing regions (for example Europe or United States).
- An AJAX service backing the chart. The bulk of the logic is in the AJAX service performing the actual query. This is usually the case and is not specific to using the SPARK UI Toolkit.

On the client side, the Single Select *On Change* event needs to set the chart's query data to the current region selected. It then refreshes the chart, as shown in Figure 4-84.

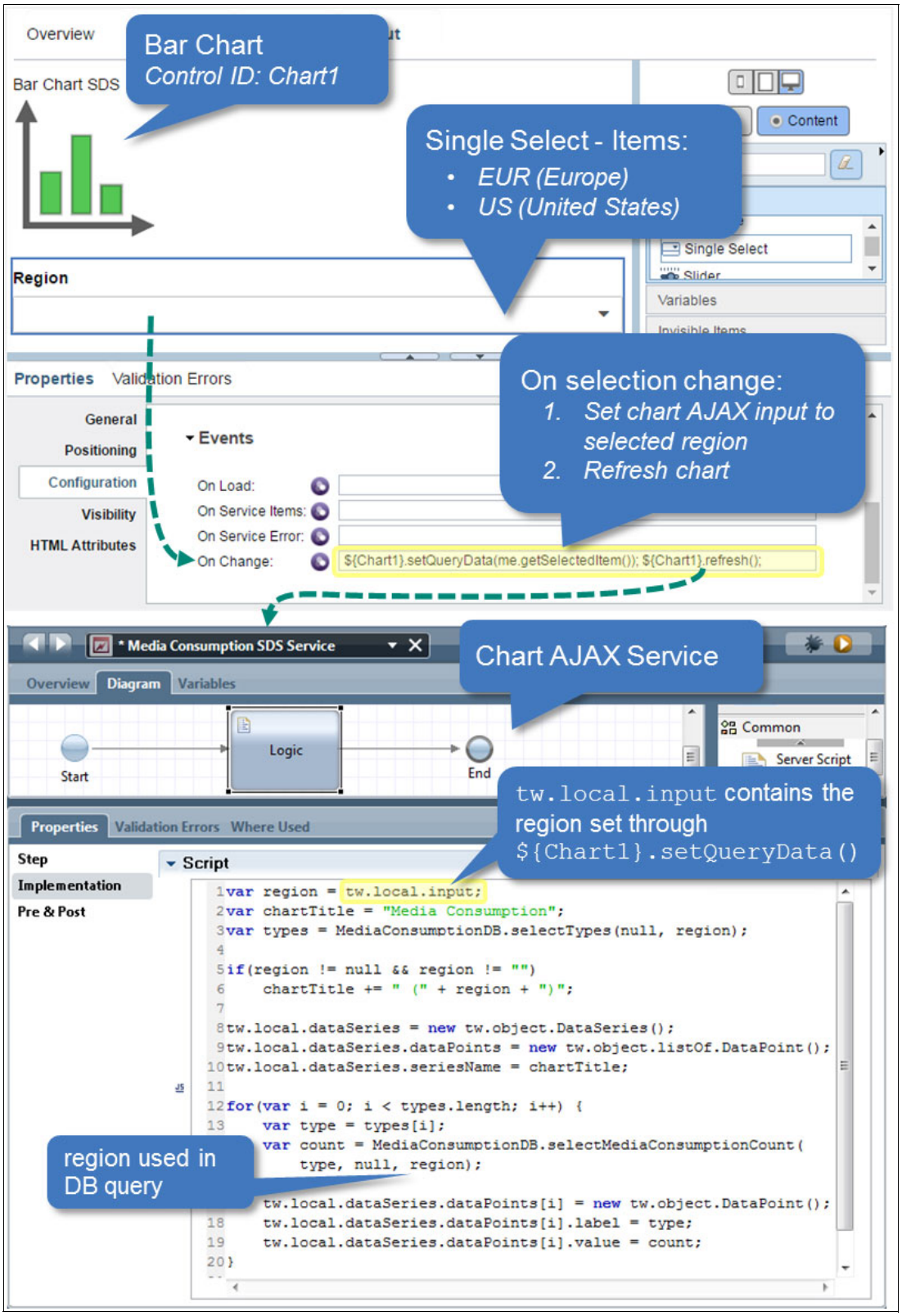


Figure 4-84 Modeling and development detail for refreshing charts after setting input data

In addition to influencing query behavior, charts provide a number of methods to control visual and data representation behavior.

In Table 4-9, several charts methods are successively called to show, hide, disable, and enable group data series. The methods also annotate the charts with labeled horizontal and vertical lines. Figure 4-86 on page 243 shows matching illustrations of the chart's behaviors. The chart starts in the state depicted in Figure 4-85.

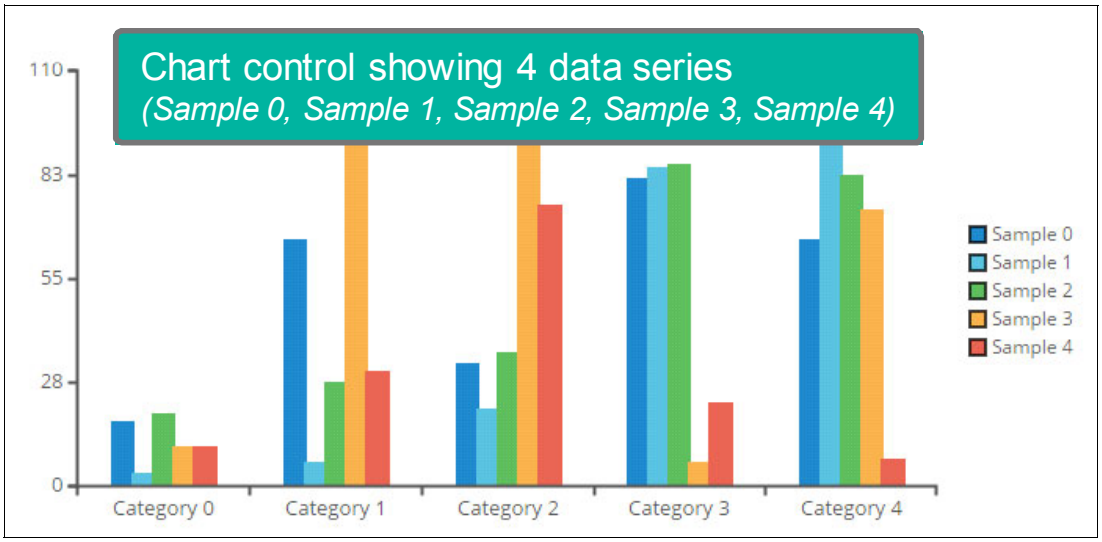


Figure 4-85 Initial chart state before transformation sequence

Table 4-9 shows the sequence of chart data representation and annotation method calls.

Table 4-9 Sequence of chart data representation and annotation method calls

Step	Logic	Step	Logic
1	<Initial chart state>	9	chart.removeVerticalLine(); chart.transform("area-spline", "Sample 3");
2	chart.groupSeries(true, ["Sample 0", "Sample 1", "Sample 2", "Sample 3"]);	10	chart.transform("line", "Sample 2");
3	chart.addVerticalLine(2, "Line 2"); chart.groupSeries(true, ["Sample 2", "Sample 3", "Sample 4"]);	11	chart.showSeries();
4	chart.groupSeries(true);	12	chart.focusSeries(["Sample 0", "Sample 1", "Sample 2"]);
5	chart.defocusSeries(["Sample 0", "Sample 3"]);	13	chart.addHorizontalLine(60, "Line 1");
6	chart.groupSeries(false);	14	chart.groupSeries(true); chart.transform("bar");
7	chart.transform("line", "Sample 0");	15	chart.focusSeries(["Sample 0", "Sample 1", "Sample 2", "Sample 3", "Sample 4"]);
8	chart.hideSeries(["Sample 1", "Sample 4"]);	16	chart.transform("spline");

Figure 4-86 shows the matching successive runtime representations of the charts (for each of the 16 calls from Table 4-9 on page 242).

Figure 4-86 Chart transformation sequence using 16 successive chart method calls

Events

Charts can respond to the following events:

- Lifecycle events (after load)
- User events (when clicking or tapping the chart)
- Backing AJAX service events (when the data series is returned from the service)

Table 4-10 lists events common to chart controls.

Table 4-10 Chart control events

Event	Description
On Load	Fired immediately after a chart is loaded, but it is not necessarily populated with data at that time. Useful to take initializing steps such as adding a menu with static menu items for the chart. See “Menus” for more information.
On Refreshed	Fired every time a chart has loaded and rendered new data from its associated AJAX service, or when the chart loads its bound data if not using an AJAX service.
On Click	Fired when the user clicks or taps a data point in the chart. The data point that is selected can be retrieved through the chart's **`getSelectedDataPoint()`** command. The On Click event is also the best place to add context menu items. See “Menus” for more information.
On Menu Action	Fired when a menu item is clicked (see “Menus” for how to create context menu items). The action context variable provides action.name to determine what menu item is clicked.

Menus

Menus are automatically displayed on charts when the following occurs:

- The **Behavior** → **Enable Menu configuration** option is set on the chart.
- A data point (showing as a bar in a Bar Chart, a slice in the Pie Chart, and so on) is clicked by a user.
- There is at least one menu item defined for the chart, through the **`addStaticMenuAction()`** and **`addTempMenuAction()`** commands, or because the chart's drill-down behavior is active.

Figure 4-87 on page 245 shows the addition of two menu actions through the **`addStaticMenuAction()`** command to a Bar Chart. Menu items are associated with action names; the action name is then checked in the *On Menu Action* event to help decide what to do when a menu item is clicked.

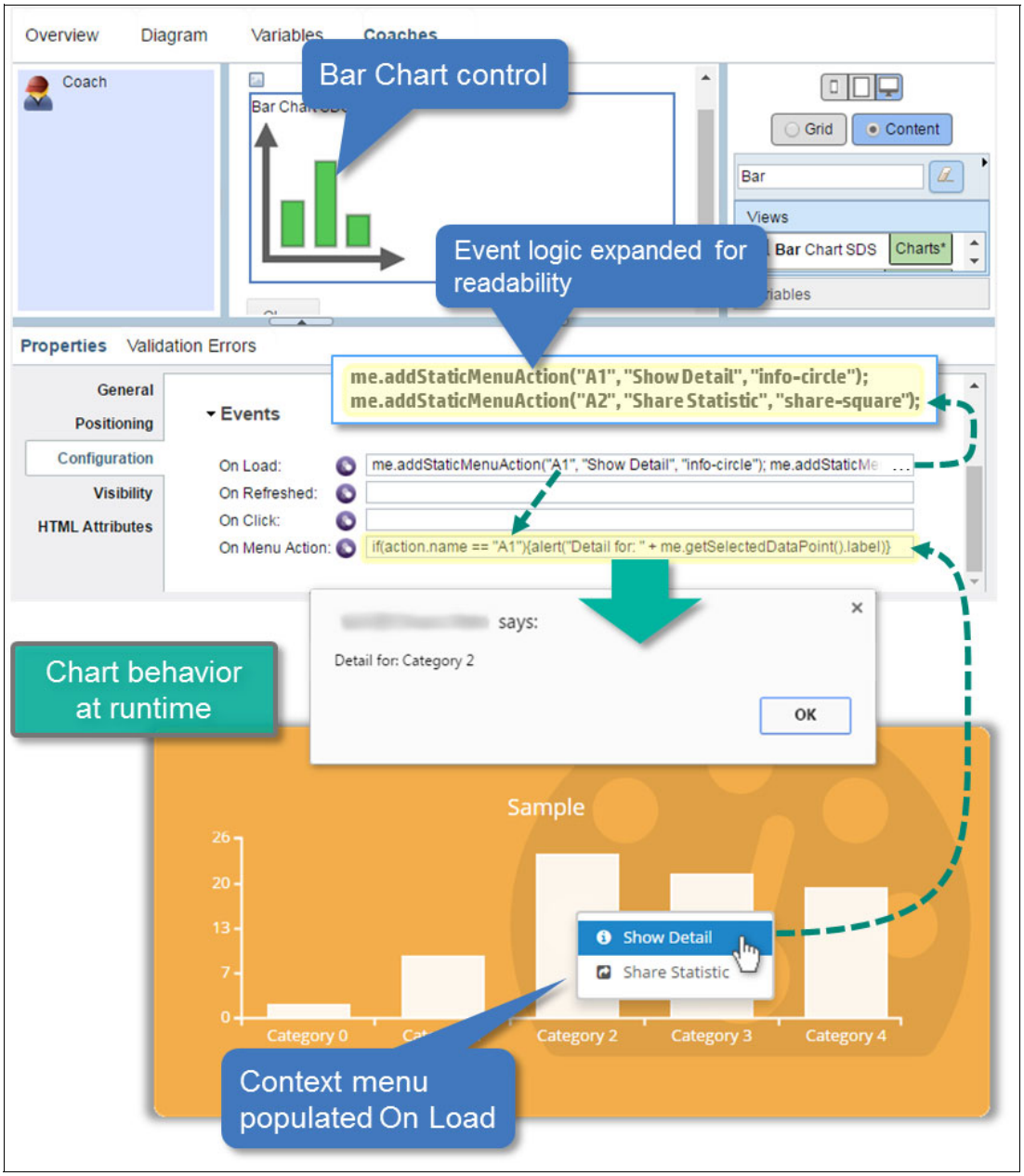


Figure 4-87 Adding menu items to a chart and taking action on item clicked

Sometimes it is useful to add a menu item that is only valid given the context of the actual *click* of the chart. Figure 4-88 on page 246 shows how to create a temporary menu item that is only shown when a bar labeled *Category 0* is clicked. Figure 4-88 shows how the **`getSelectedDataPoint()`** chart command is used in the On Click event. The method examines what the user has clicked. In this example, the label of the select data point is checked.

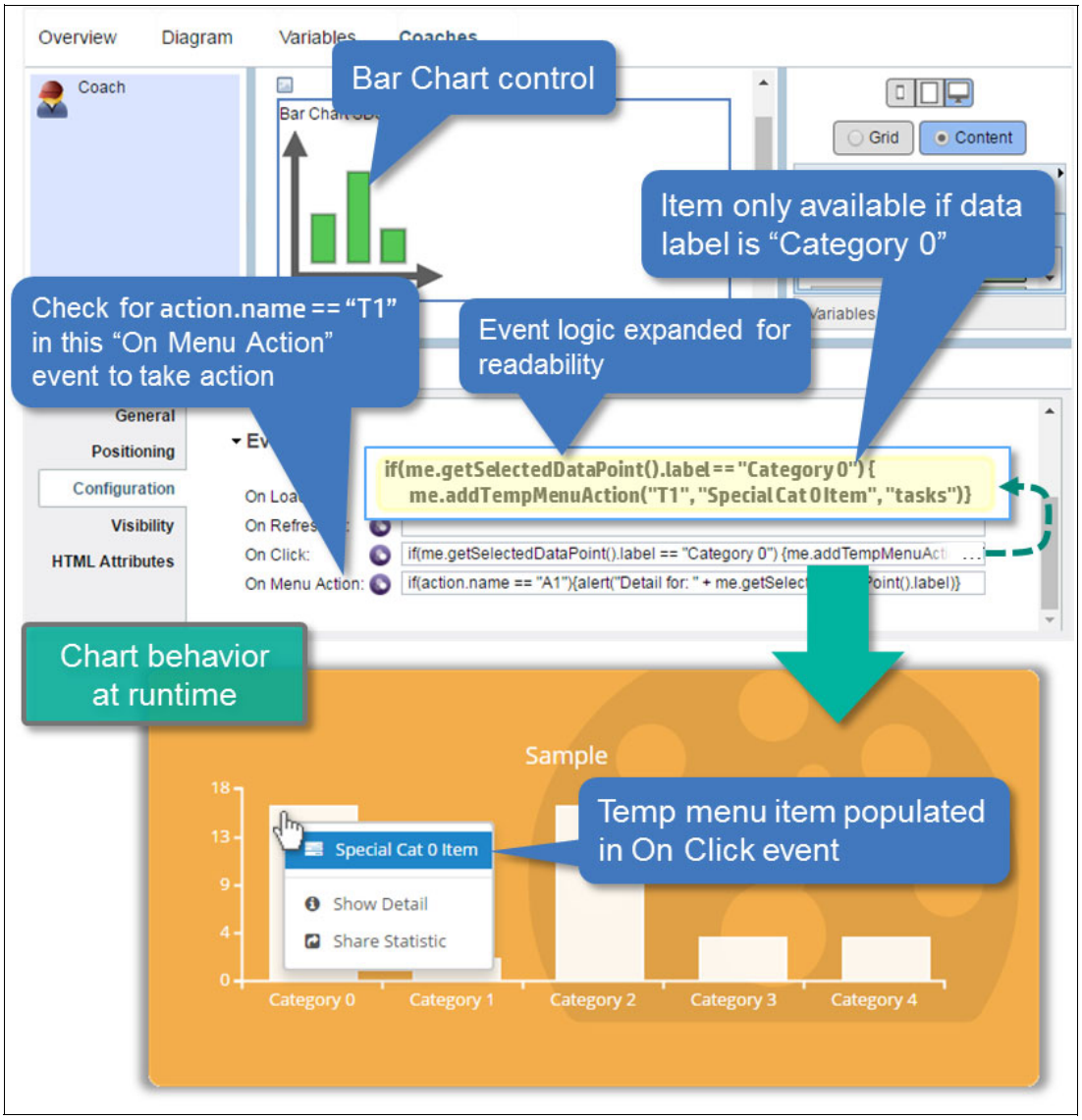


Figure 4-88 Using temporary menu items for context-specific chart clicks

Note: Temporary menu items are best created in the Chart's On Click event. They disappear as soon as the menu closes. Temporary items are listed in a separate (top) section of the context menu.

4.10.2 Working with tables and charts

Reporting requirements often include both displaying summary data and providing details regarding some aspects of a chart in tabular form. Chart controls and the Service Data Table control can work together to provide such an experience. In Figure 4-89 on page 247, the user taps in the category of a chart (for example News). This requests more detailed data on News media consumption to be displayed in a table under the chart.

Figure 4-89 Runtime example of chart and table controls working together

The example in Figure 4-89 relies on a chart menu-item click to the following actions:

- Request a table query corresponding to the selected data point label (*News* in this example)
- Set the title of the collapsed panel containing the table to *Details* - <data point label>

After the AJAX query for the table returns, the collapsed panel expands, triggered by the table's *On Service Data Loaded* event. The populated table is revealed.

Figure 4-90 on page 248 shows the design-time detail for this interaction.

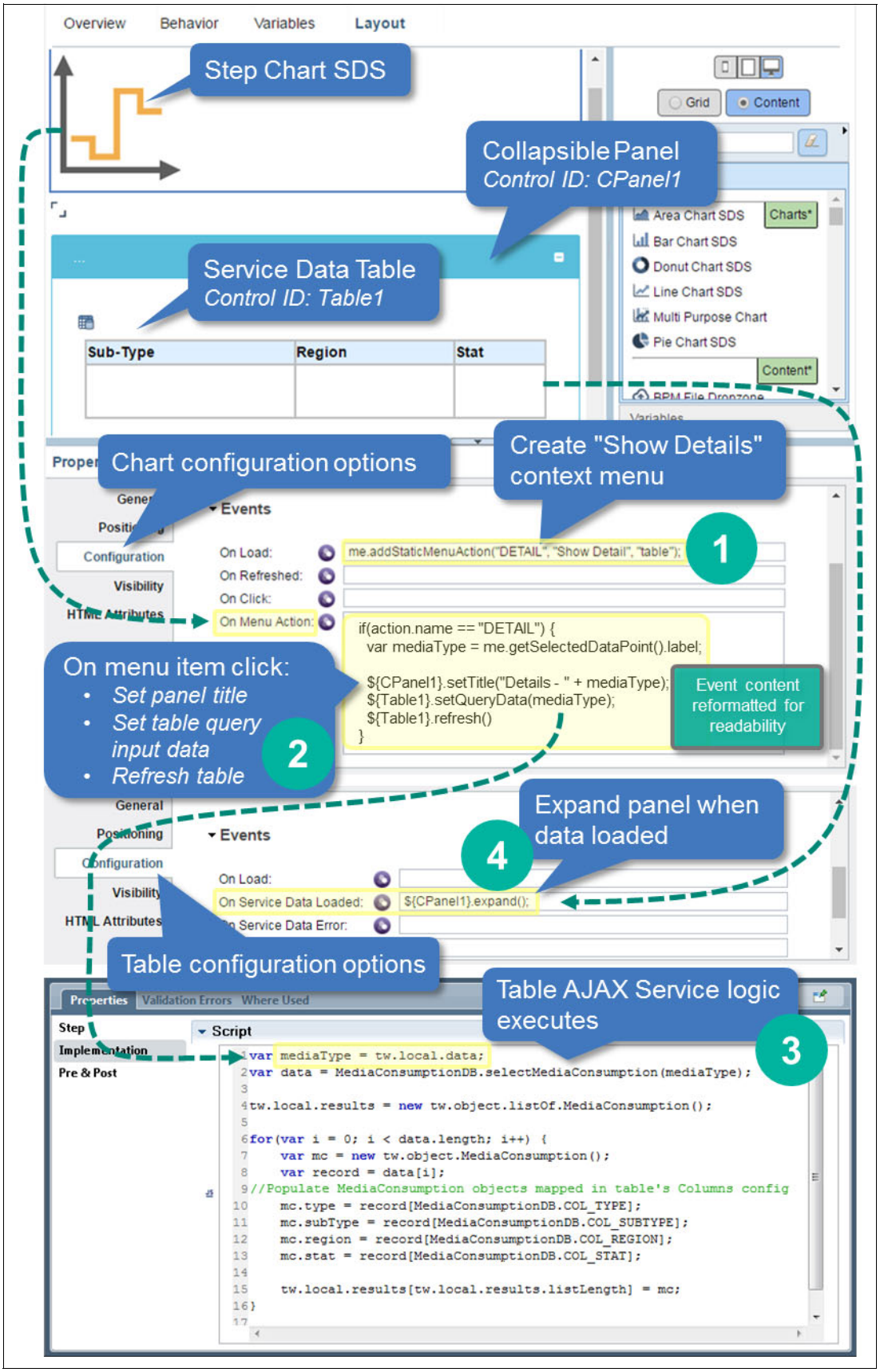


Figure 4-90 Design-time composition for chart, table, panel interaction

4.11 Solutions for common patterns and requirements

This section provides guidance for using SPARK UI-specific capabilities to implement common requirements when building BPM UIs using the SPARK UI Toolkit.

4.11.1 Working with large forms

User Experience (UX) guidelines suggest limiting the amount of content presented to a user at the same time. There are also other reasons to mitigate the performance impact of loading a large number of controls on a single coach. An example is in an extreme case when a web browser freezes or becomes unresponsive for several seconds.

An effective performance impact mitigation approach can include a combination of some or all of the following suggestions:

- If displaying data in a Table control, change the rendering option of one or more table columns to *Simple HTML*. Doing so can drastically reduce the cost of rendering the table. The tradeoff is that those columns cannot be edited or updated because they are not coach views at run time.
- For Table, Service Data Table, or (Horizontal and Vertical) Layout controls, the following loading options can completely eliminate a browser freezing problem. They do so by loading content asynchronously and by still providing an option to fine-tune the overall content load time:
  - **Configuration** → **Performance** → **Async Loading**: Select to enable.
  - **Configuration** → **Performance** → **Async Batch Size**: Number of rows (or controls for a Layout) loaded at the same time as part of an asynchronous batch. The larger the batch, the faster the overall load time, at the cost of the browser acting less responsive.
- To suspend the instantiation of one of more controls until explicitly requested, use a Deferred Section control, such as in the following examples:
  - A Tab Section control containing Deferred Sections directly under each of its tabs might delay the instantiation of the content of each tab until the particular tab is selected. Use the `lazyLoad()` command on the deferred section control when the On Tab Changed event is fired, or over several seconds, in the background.
  - A Table control containing potentially large coach views on each row might instead display a few light columns. This can include a button to request the display of the heavy coach view only if the button is clicked. The heavy coach view is contained in a Deferred Section that in turn only loads the processing-intensive content of the corresponding row when or if requested.

Because the Table and Service Data Table controls were described in 4.7.1, “Table and Service Data Table controls” on page 211, only the scenarios using Deferred Sections are provided in this section.

Tab Section with on-demand content instantiation

Figure 4-91 on page 250 shows how a Tab Section and a Deferred Section under each tab can work together. They can delay the loading of coach views until the tab containing them is selected.

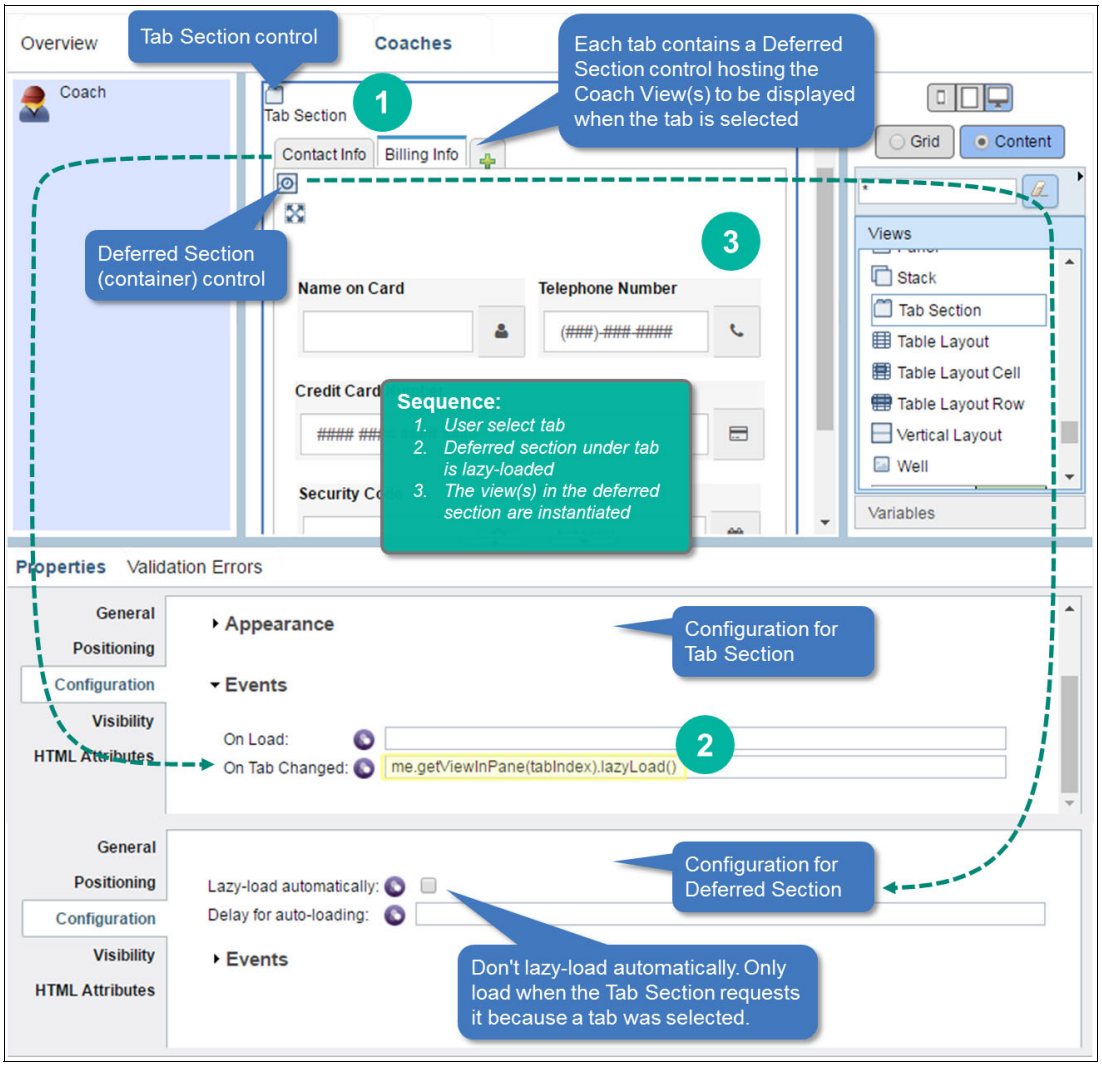


Figure 4-91 Example of deferred content loading in a Tab control

Except for the template HTML that the Coach Framework inserts into the page, the Billing Info view does not exist until the Deferred Section lazy-loads it.

Note: The Billing Info coach view is small (containing approximately 10 controls) in the example shown in Figure 4-91. Thus, using a Deferred Section in a production scenario is excessive. It is also not a problem because a Deferred Section has a small footprint. However, the larger the view (for example 40 or 50+ contained controls), the greater the benefit.

Table with on-demand content instantiation

Figure 4-91 on page 250 shows how a table that otherwise needs to display a lot of content on each row can instead display a few light controls per row (including a deferred section). It only loads the full coach view in the row when the user clicks the **View** button to show more detail. Figure 4-92 on page 251 shows how the in-table, defer-loaded coach view scenario behaves at run time.

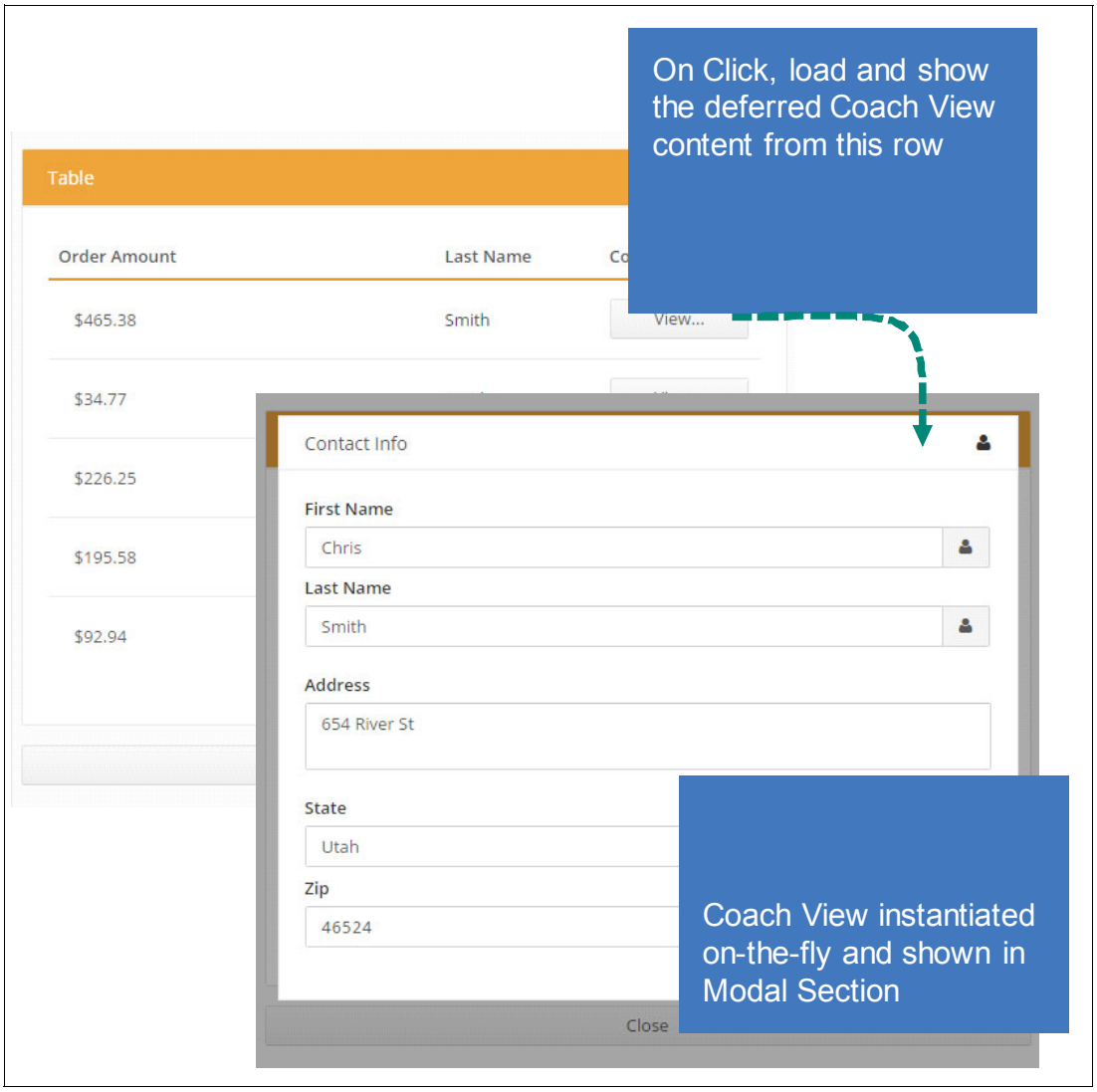


Figure 4-92 Deferred or on-demand loading of coach views in table rows

The advantage of this approach is reduced cost. The performance of a Table that might otherwise be expensive to render (the larger the views contained on each row the more expensive) can be managed. Only the cost of instantiation is incurred if or when the user requests to see more detail. The scenario can be composed as shown in Figure 4-93 on page 252.

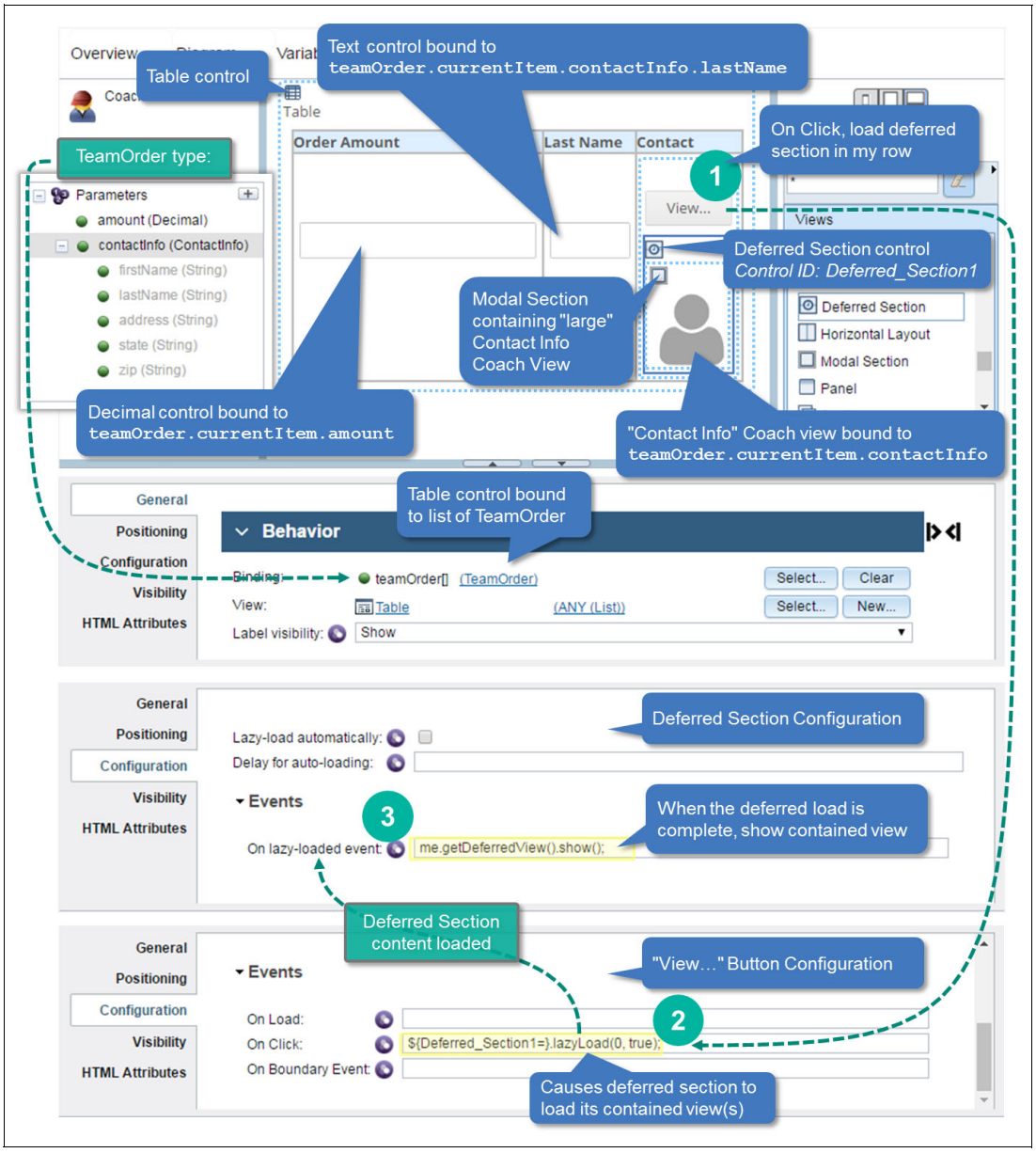


Figure 4-93 Modeling on-demand loading of coach views in table rows

In the composition shown in Figure 4-93, the coach view is contained inside a Modal Section.

Both the Modal Section and the Contact Info coach view are contained under the Deferred Section. Therefore, even the instantiation overhead of the Modal Section is deferred until the user requests to load the content. In fact, it is the Modal Section that is lazy-loaded. The Coach Framework loads the view contained in the Modal Section when the Modal Section instance comes into existence.

Note: The Deferred Section's **`lazyLoad()`** command takes two parameters. The first parameter is a delay in milliseconds to wait before the lazy-loading takes place. The second parameter is for convenience. Calling the **`lazyLoad()`** command on Deferred Sections that have content that is already-loaded does not load it again. However, if the second parameter is set to true, it fires the On Lazy-Loaded event again so the logic it contains can execute again.

4.11.2 Server-side paging

Client-side paging loads all records in browser memory but only displays a portion of the in-memory data set page by page. Server-side paging only retrieves in browser memory a small subset of a potentially large data set. Server-side paging then displays in the table the entire data subset that is in memory.

Server-side paging is a common requirement for the Service Data Table control, but it works differently from the built-in client-side paging feature of the table controls.

The paging behavior is effectively emulated by tracking a cursor or offset in the remote data set and requesting a batch of records. The page size is essentially the maximum batch size (that is, the maximum number of records to be returned in the query).

For example, with a 250 remote-record data set and a page size of 100 the following occurs:

- A query of up to 100 records at offset zero returns the first page of 100 records.
- A query of up to 100 records at offset 100 returns the second page of 100 records.
- A query of up to 100 records at offset 200 returns the last page containing the remaining 50 records.

Note: Server-side paging is most easily achieved with a Service Data Table control because it is the only table control capable of fetching data through an AJAX service.

Note 2: Another more complex and less reusable pattern can be implemented through a Table control with fetched human service data. The data is fetched progressively through diagram logic and *Stay On Page* interactions. The pattern presented in Figure 4-94 is simpler and can be easily reused and encapsulated.

Figure 4-94 uses a Service Data Table to query completely artificial data for Media Consumption volumes. The runtime behavior for the sample server-side paging scenario to be implemented is also shown in Figure 4-94.

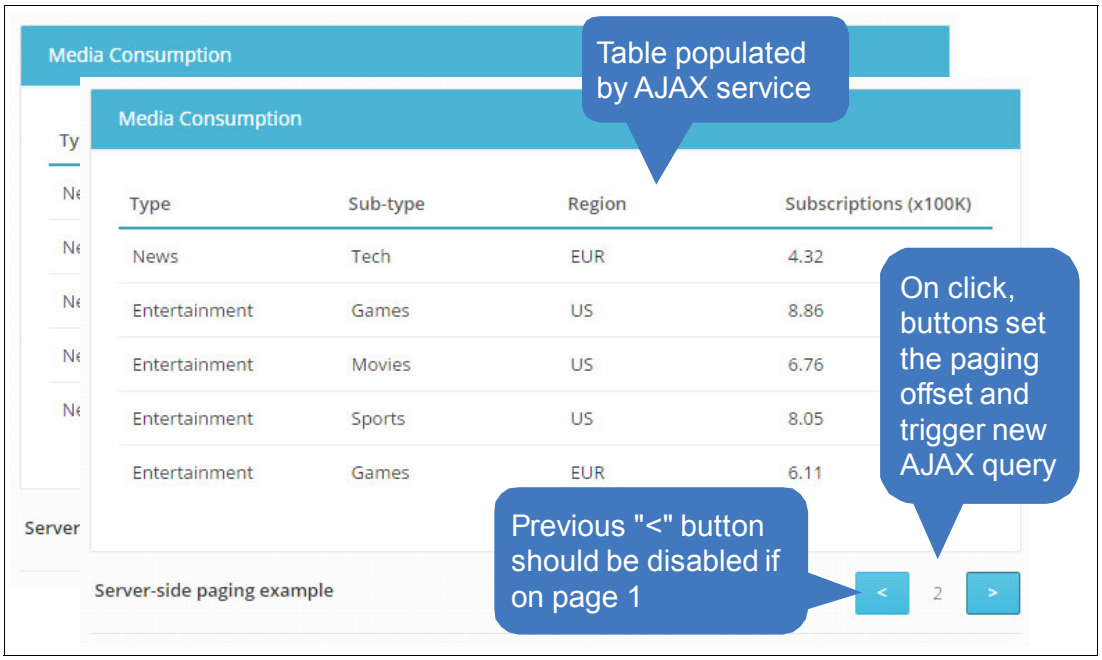


Figure 4-94 Runtime behavior for server-side paging Media Consumption query scenario

The Service Data Table-based pattern has the following two halves:

- The client side, with the Service Data Table control and the logic required to set up the query (including page size and offset). Then the table is refreshed.
- The server side, which includes the table's AJAX Service, with its associated query and resulting data.

Client-side implementation

The sample scenario implementation shown in this section uses a coach view that contains the Service Data Table. The paging navigation controls (*previous* and *next* Button controls, and an Output Text control) are used to show the current page number.

Note: Using a coach view to contain all the controls and associated business logic is not mandatory for the server-side paging pattern. However, it helps keep the implementation self-contained and reusable.

Overall structure

Figure 4-95 on page 255 shows the overall structure and the logic defined for the *Server Side Paging Table Example* coach view.

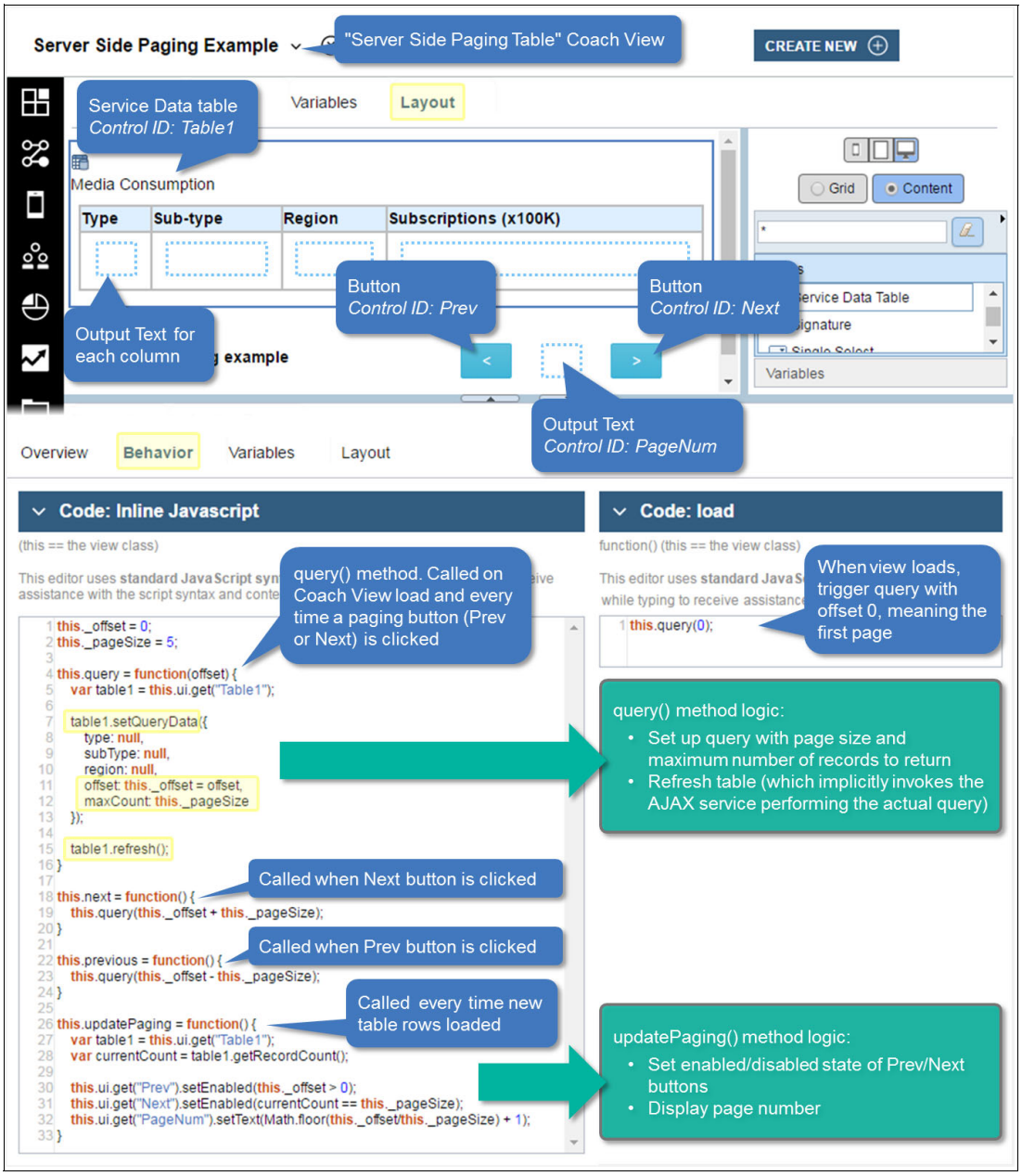


Figure 4-95 Structure and logic for the Server-Side Paging Table Example coach view

Note: The page size in the example provided is arbitrarily set to 5. In a more comprehensive example, the page size can be set from a configuration option or from a *Page Size* Integer control in the coach view.

Service Data Table configuration

The Service Data Table is configured with the *Media Consumption Query* AJAX service described in “Server-side implementation” on page 257. Figure 4-96 on page 256 shows key details of the Service Data Table configuration used for the example.

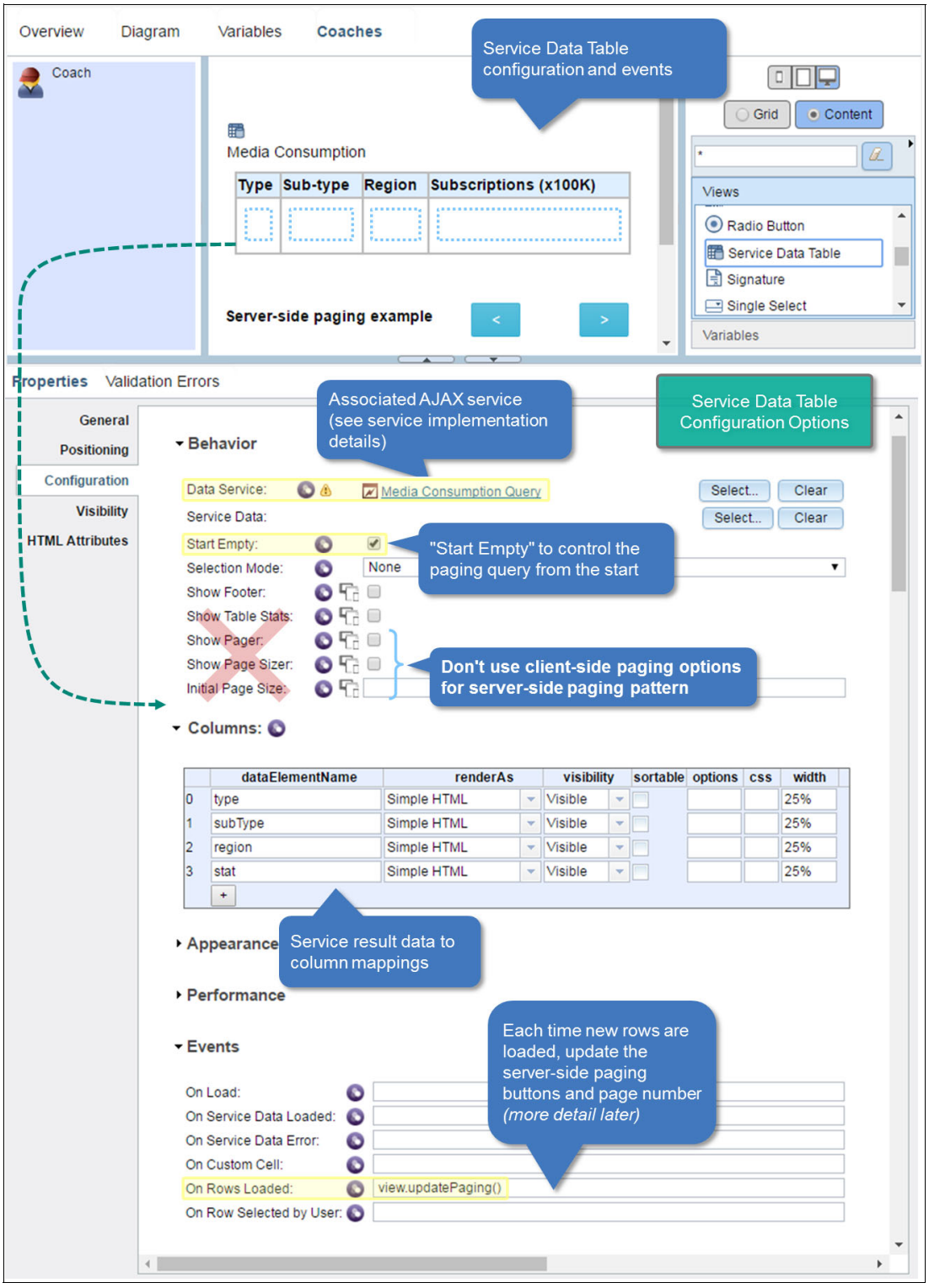


Figure 4-96 Service Data Table configuration for server-side paging example

By default, the Service Data Table control automatically invokes its associated AJAX service at load time for the table. In this example, however, the initial automatic query is prevented through the *Start Empty* setting. The initial query is instead done explicitly and with the desired initial offset (0) at load time for the *Server Side Paging Example* coach view. This is

done by calling the **`this.query(0)`** command from the view's load handler (see load logic in Figure 4-95 on page 255).

Paging navigation buttons

Because the Service Data Table pager widget in the table footer can only be used for client-side paging, the footer and pager widget are not shown. Instead the server-side paging pattern uses controls such as Buttons or Icons for the user to page forward or backward.

Figure 4-97 shows the two paging buttons (labeled "<" and ">") that are used to trigger the AJAX re-query with a new offset. This is done by invoking the **`previous()`** and **`next()`** coach view commands from the On Click event of the "<" and ">" buttons, respectively.

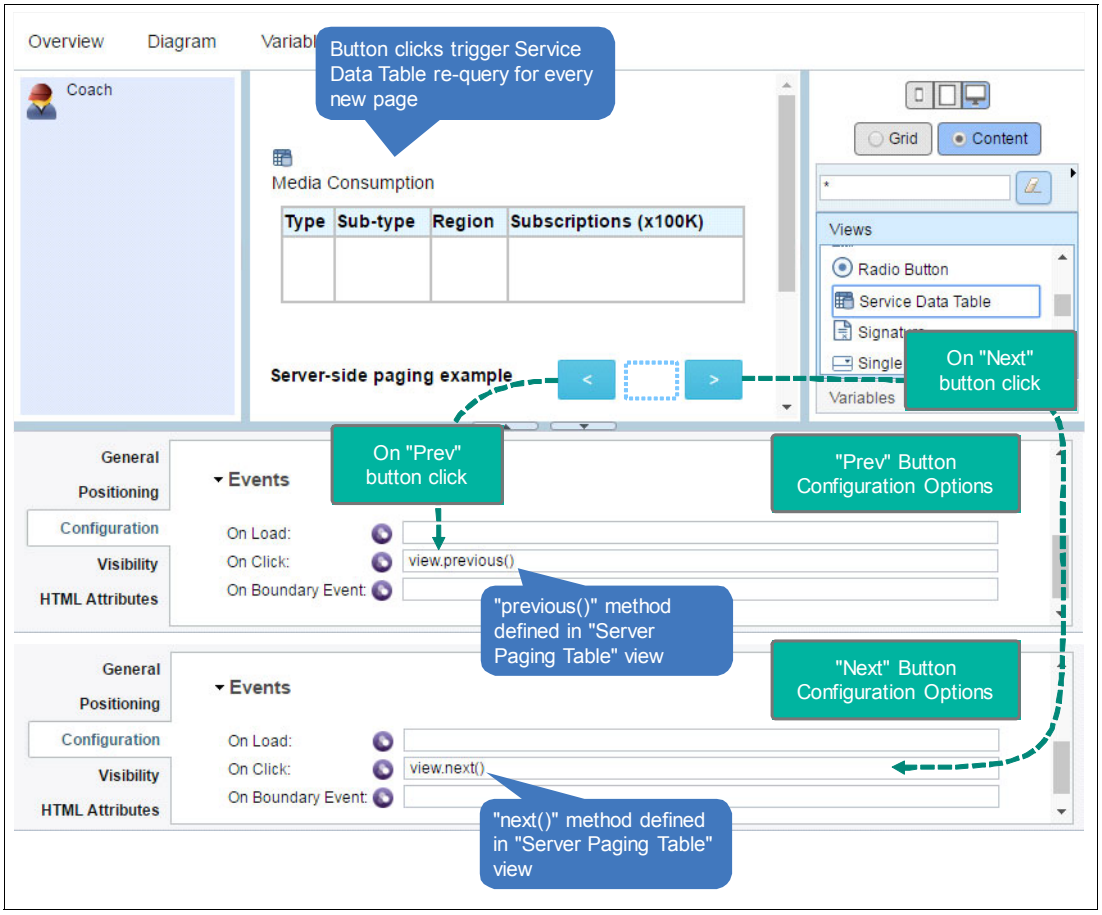


Figure 4-97 Server-side paging navigation through buttons

Server-side implementation

The template for the AJAX service associated with the Service Data Table does not prescribe inputs used for server-side paging. It only allows for input of one data element of type `ANY`. To combine server-side paging parameters along with other query data, create a *MediaConsumptionQuery* complex type. Then add the following server-side paging-related attributes (in addition to other business case-specific attributes):

- `offset` (Integer)
- `maxCount` (Integer)

The offset and maximum record count are then used in the database query to determine the *page* (that is, the slice of records) for which to return results. Figure 4-98 on page 258 shows the server-side logic for the AJAX service.

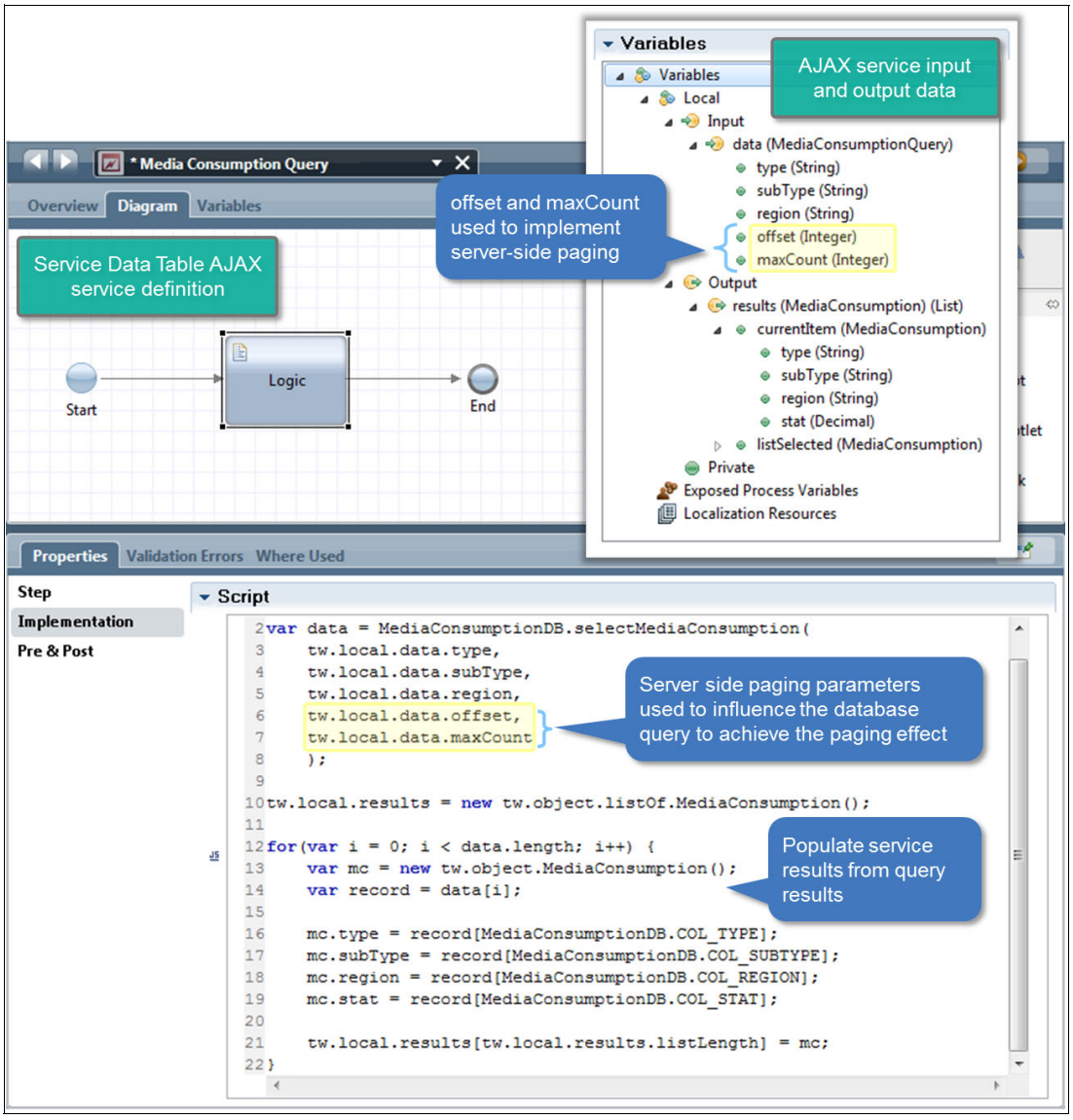


Figure 4-98 Server-side paging AJAX service implementation

Pattern summary

The following list recaps the pattern-specific aspects of server-side paging with a Service Data Table control:

1. Create an AJAX service associated with the Service Data Table.
2. Ensure the input data for the AJAX service includes attributes for an offset and a maximum record count.
3. Implement the AJAX Service query logic to take the server-side paging parameters into account.
4. Ensure that client-side paging aspects are turned off for the Service Data Table configuration.
5. Add controls (such as Buttons, Icons, or Links) that can be used to attach table re-query logic to on-click events to allow previous page and next page navigation.
6. Implement simple client-side logic to manage the offset specified whenever a re-query is triggered.

Note: The example provided is purposefully simplistic. Additional sophistication can include retrieving the full record count up front to display the current page and the maximum number of pages available. More variability to control the page size and other query and sorting criteria can easily be introduced by extending the presented pattern.

4.12 More on the SPARK UI Toolkit

From a BPM UI developer's perspective, the SPARK UI Toolkit represents a deliberate compromise between simplicity and flexibility.

The benefit of this approach is that the level of UI development expertise required to be effective is low after you understand the core SPARK UI patterns. At the same time, the toolkit provides many extension and customization capabilities to implement advanced and complex production requirements and scenarios.

Additional content on the SPARK UI toolkit is available and provides a solid supplement to this chapter. The following topics are noted so that you can explore additional aspects of the toolkit in context of the foregoing material:

- Controls: Creating simple and composite coach views with SPARK capabilities
- Tables: Custom rendering and performance options
- Charts: Implementing drill-down behaviors
- Useful tips on debugging, deciding when to use diagram-based or encapsulated control logic

For more information, see the following website:

http://ibm.biz/SalientProcessToolkits

4.13 Conclusion

In this chapter you have learned how the SPARK UI Toolkit provides a fully featured responsive set of controls. These controls can be used to build modern user interfaces for desktop and mobile devices. These tools provide a great user experience and accelerates developer productivity up to four times compared with traditional techniques. The SPARK UI Toolkit extends the Coach Framework by adding the following benefits:

- An addressing and event framework that simplifies how to reference controls and react to user interactions
- Simplified client-side validation and enabling coach authors to express formulas

SPARK controls are designed to perform well when used with large data sets, using lazy loading and server-side paging techniques. Charting controls are visually impressive and provide drill-down capability that can be used to create custom dashboards, which are covered in 5.2.4, “Extend Process Portal with custom dashboards” on page 277.

The following information about IBM Process Portal is described in 5.2, “Modifying the Process Portal user experience” on page 272:

- How coaches are used to define custom user interfaces other than for task completion UI
- How the process portal can be extended and customized

5

IBM Process Portal

IBM Process Portal 8.5.7[1] is implemented entirely from coaches and coach views, which illustrates how you can build sophisticated UIs using the Coach Framework. It provides the primary UI that business users can use to perform their work when using process solutions developed using IBM Business Process Manager (BPM).

This chapter outlines Process Portal features from the business user perspective and shows how process authors can extend Process Portal to provide custom UIs that enable business users to complete their work. It also describes ways in which the user experience can be modified and explains how to use Process Portal with IBM Process Federation Server to enable IT administrators to manage the BPM infrastructure without significantly impacting the business user experience.

This chapter includes the following topics:

- Process Portal features
- Modifying the Process Portal user experience
- Securely accessing IBM Process Portal from a mobile device
- Process Federation Server
- Conclusion

[1] You might want to view the *Getting Started with Process Portal in IBM BPM 8.5.7* video (http://ibm.biz/IBM_Process_Portal_857_Video) or refer to Process Portal information available in IBM Knowledge Center (http://ibm.biz/IBM_Process_Portal_857).

5.1 Process Portal features

Two primary types of business users use Process Portal to perform their work:

- Knowledge workers
- Team leaders

Knowledge workers use Process Portal to do the following activities:

- Discover tasks they need to perform.
- Execute tasks.
- Access relevant information used to perform their work.
- Collaborate with other colleagues.

Team leaders use Process Portal to:

- Monitor and manage the work that is being performed by their team members.
- Report on and analyze the performance of business operations.
- Perform their own tasks.

This section describes the standard features of Process Portal that are available to knowledge workers and team leaders to perform their work. Customizations and extensions to the standard features are covered in subsequent sections.

5.1.1 Mobile enablement

With IBM BPM 8.5.7, you can access Process Portal from either your desktop or a mobile device, as illustrated in Figure 5-1 on page 263. Mobile access enables field workers to participate in processes at the time of need rather than having to wait until they return to the office to complete work. Avoiding such delays can help to improve the overall performance of business operations.

Figure 5-1 Process Portal accessed from a desktop, tablet, or smartphone

The Process Portal UI is responsive. So the layout and behavior adapts, depending on the available screen size (or form factor) of the chosen device. Process Portal is accessed via a web browser using the following basic URL format:

`https://<host>:<port>/ResponsivePortal`

where *<host>* and *<port>* are specific to your IBM BPM deployment.

5.1.2 Process Portal layout

The Process Portal UI is divided into the following areas, as shown in Figure 5-2 on page 264:

- The *main menu* on the left side is used to manage the user profile and launch dashboards and processes.
- The *context menu* on the right side provides access to social features and the process instance stream.
- The *central area* displays content, including dashboards and task completion coaches.

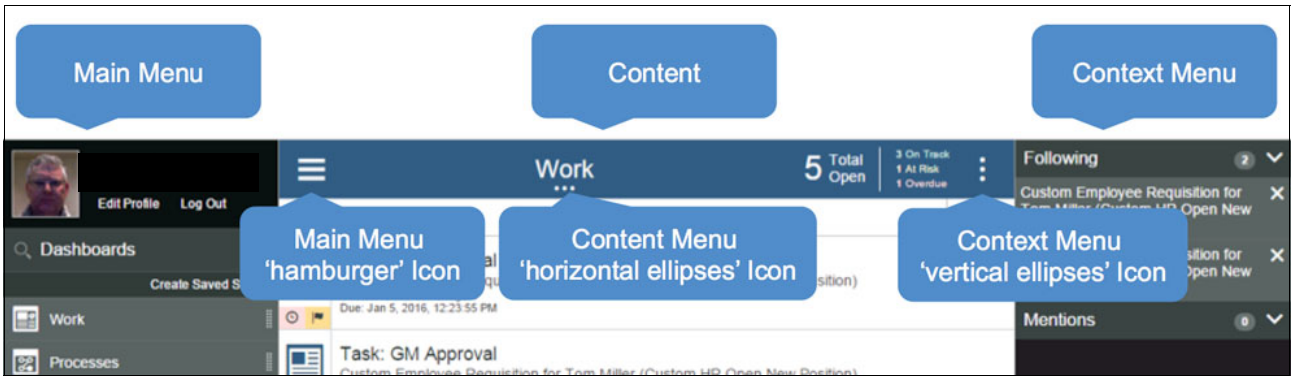


Figure 5-2 Process Portal layout

On a smartphone, both the main menu and context menu are minimized by default. Click the *hamburger* icon on the top left to view the main menu. Similarly, click the *vertical ellipses* icon on the top right to view the context menu. On a tablet only the context menu is minimized by default. The context menu is sticky. When minimized, it remains minimized the next time you log on.

To open the Content Menu, click the *horizontal ellipses* icon or anywhere in the Content Header. The Content Menu provides an option to open the current content in a new window, which enables you to perform multiple activities simultaneously.

5.1.3 Discovering tasks with the Work dashboard

After you log in to Process Portal, you are presented with tasks within the Work dashboard, as illustrated in Figure 5-3. You see only the tasks that you are authorized to access.

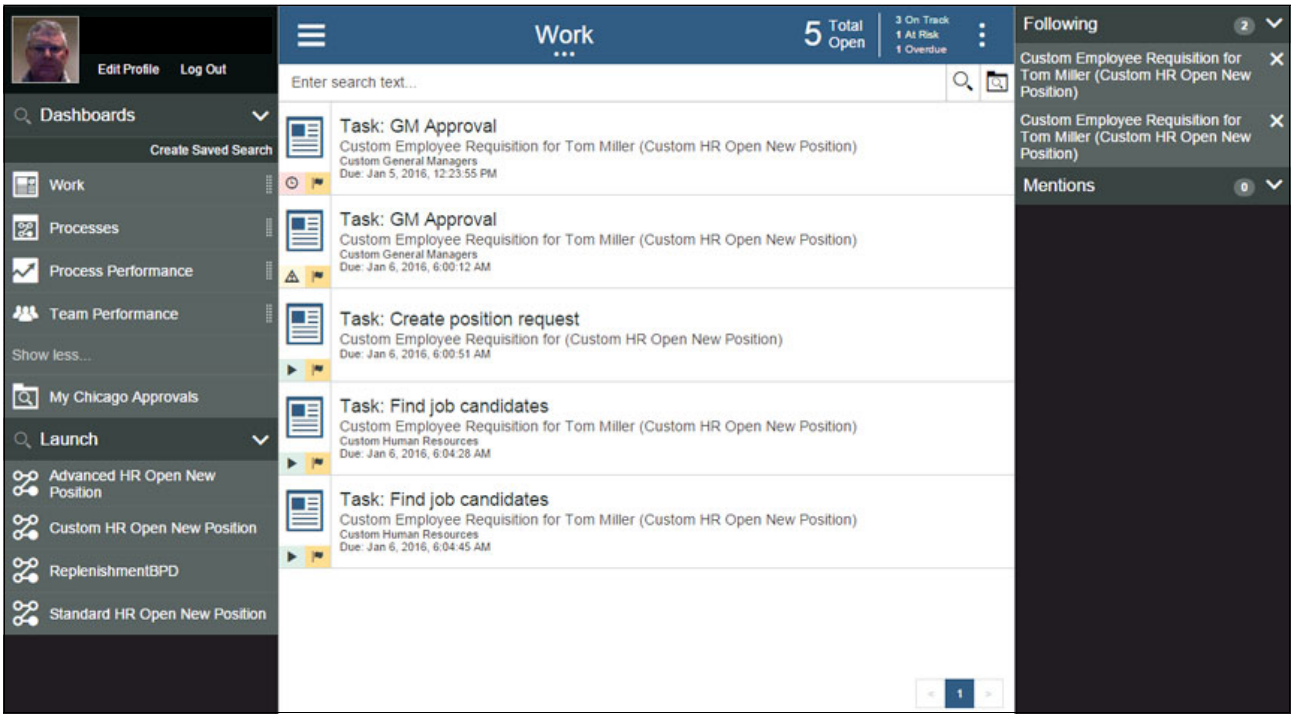


Figure 5-3 Work dashboard displays tasks using the List View

By default tasks display as a list, but you can change this view to a tabular layout by selecting the Table View option, as illustrated in Figure 5-4.

Each task within the List View includes basic information about the task, but you can view business data and actions that can be performed on the task by selecting the task icon. Within the Table View, you can select which columns to display. To order tasks, click the column headers.

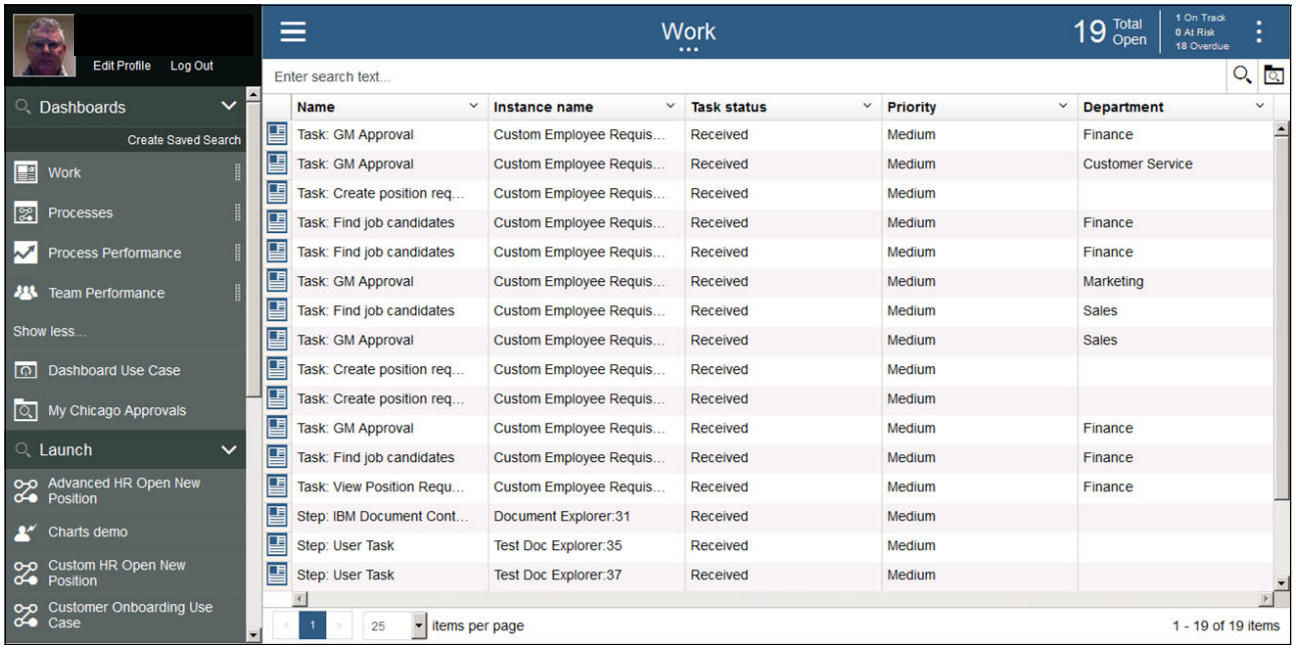


Figure 5-4 The Work Dashboard displaying tasks using the Table View

5.1.4 Locating tasks using filters and saved searches

You can quickly filter tasks in either view by entering search text at the top of the Work dashboard. The resulting task list includes only those tasks where the search text matches one or more process or business attributes that are associated with the task.

If you want a more precise search, you can define a *Saved Search*. Figure 5-5 on page 266 illustrates a Saved Search definition. You can either create a Saved Search from scratch or base one on an existing Saved Search as a starting point.

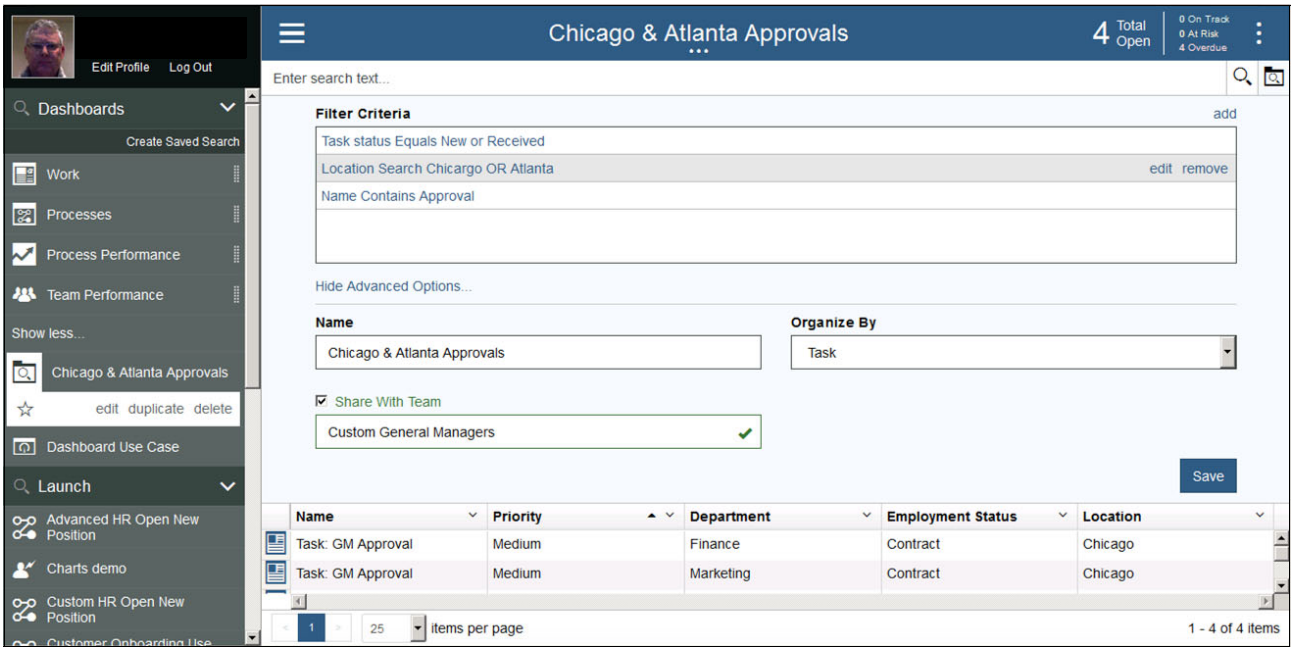


Figure 5-5 Saved Search definition

Within a Saved Search definition, you can specify a number of filter criteria, where each filter matches a specific process or business attribute, such as *Name contains Approvals*, where the Name of the task contains the word Approvals. The resulting task list includes only those tasks that match all the filter criteria.

In the illustrated example, the three filters are combined using a logical `AND` operator. However, within a specific filter, you can specify a search that matches any of the specified search text, separated by an `OR` operator. For example, `Location Search Chicago OR Atlanta` matches tasks where the Location business field is either Chicago or Atlanta.

When defining a Saved Search, the resulting task list displays at the bottom of the Saved Search dashboard. So you can test the Saved Search as it is being defined.

As the names suggests, you can give a Saved Search a name and save it for later use. However, you can also define a Saved Search to locate tasks without saving it. Individuals can create Saved Search for their personal use. Then, those Saved Searches display within the list of dashboards in the main menu. You can also share a Saved Search with a team or everyone within the organization.

When selecting a Saved Search, the resulting task list is displayed and managed in exactly the same way as the Work dashboard. That is, you can filter and view tasks in either List View or Table View, and you can add, remove, and order columns. In fact, the Work dashboard is actually implemented as a Saved Search.

5.1.5 Executing tasks

After you locate a task that you want to work on, you can launch the task by selecting it. Click the task so that the task can be claimed, and the task completion coach displays within the Process Portal window. When that happens, both the main menu and context menu are minimized to provide the maximum screen area for you to complete your work. The *hamburger* and *vertical eclipse* menu icons are visible on the task header so that you can access the main menu and context menu if needed while you complete you task.

When the task is complete or postponed, the Work dashboard displays to prompt you to select another task.

5.1.6 Working with processes and cases

BPM 8.5.7 no longer includes a distinction between *processes* and *cases*. Within Process Portal, these elements are collectively referred to as *processes*. Conceptually, cases are processes that are more unstructured and rely on the knowledge worker to determine the order to perform the activities and to determine when the case is complete. Process Portal enables business users to work with structured processes, unstructured cases, and hybrid processes that fall somewhere in between (that is, structured processes with ad-hoc activities and ad-hoc activities that call structured sub-processes).

The processes that you are authorized to start are listed in the launch list at the bottom of the main menu. When you select a process, a new process instance starts. A *Launch UI Coach* displays if you need to provide business data to define the context of the process. This concept is particularly important for cases that do not include a structured flow and only comprise a set of ad-hoc activities.

You use the *process dashboard*, illustrated in Figure 5-6, to view active or completed process instances. You can locate specific instances by specifying a search filter that matches on the process instance name. The icon to the left of each process instance name indicates whether it is on track, at risk or overdue. The *star* icon to the right of the instance name is used to follow the process. (See 5.1.9, “Social features” on page 269 for more details about social features.)

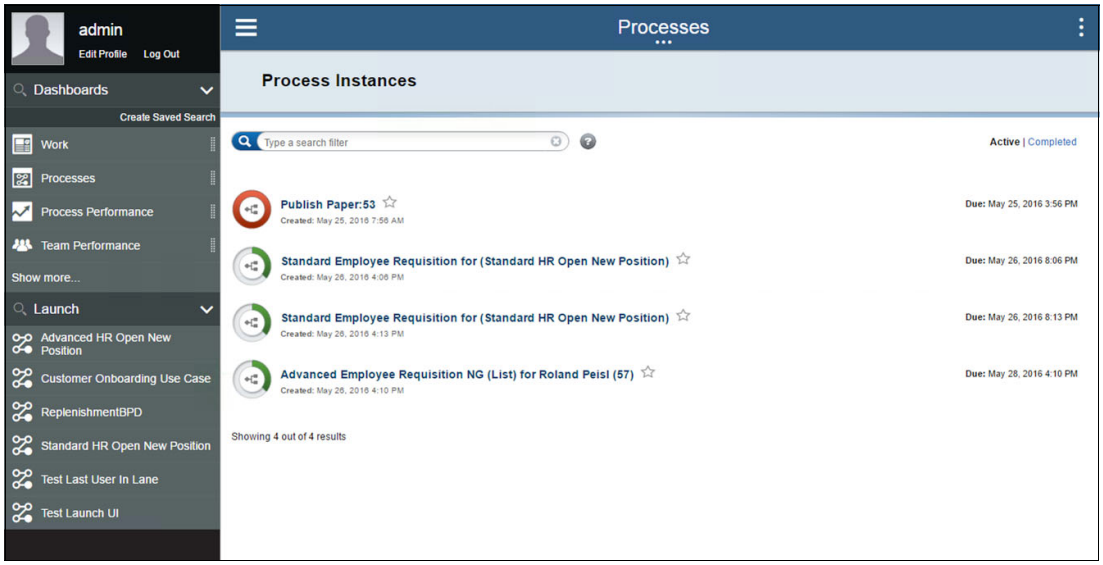


Figure 5-6 Process dashboard listing all active process instances

You can view the details of a specific process instance by selecting the process instance name from the process instance list.

Figure 5-7 on page 268 illustrates the Process Instance UI within the Process dashboard. The window is divided into the following main panes:

- The *data* section displays business data that is associated with the instance.
- The *documents* section lists any documents associated with the instance.

- The *tasks* section lists any active tasks that the user is authorized to claim.
- *Streams* provide a history of activities and collaborations.
- The *activities* section displays ad-hoc activities that the user is authorized to access.

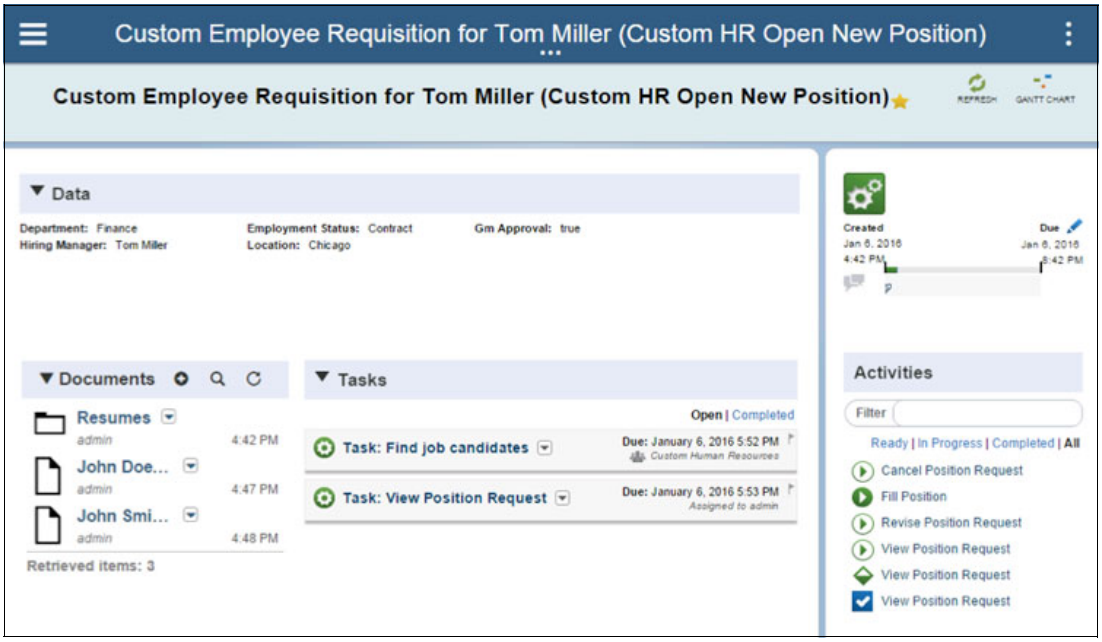


Figure 5-7 Process dashboard showing process instance details

Table 5-1 describes the various ad-hoc activity states.

Table 5-1 Ad-hoc activity states

Icon	State	Description	Configuration
	Ready	Activity to be triggered manually by the user. Goes to Working state when triggered.	Manual, Required
	Ready-Optional	Activity to be triggered manually by the user, but does not have to be triggered to complete the process. Goes to Working state when triggered.	Manual, Optional
	Working	Activity has been triggered but is not yet completed. Goes to Completed state once complete.	Manual/Automatic Required/Optional
	Completed	Activity completed successfully.	Manual/Automatic Required/Optional
	Waiting	Activity waiting for a precondition to be satisfied. After the precondition is met, it goes either to the Ready state for manual activities or the Working state for automatic activities.	Manual/Automatic Required
	Waiting-Optional	Activity is waiting for a precondition to be satisfied but does not have to be triggered to complete the process. After the precondition is satisfied, it goes either to the Ready state for manual activities or the Working state for automatic activities.	Manual/Automatic Optional

Icon	State	Description	Configuration
	Failed	Activity did not complete successfully. If restarted, activities resume their original state.	Manual/Automatic Required/Optional

From the Process Instance UI, you can view a projected path analysis data by clicking the Gantt Chart icon on the right side of the header section, which is beyond the scope of this book. However, this process is discussed in the Process Performance video, and a link to that video is provided in the next section.

5.1.7 Performance dashboards

IBM BPM 8.5.0 introduced Process Performance and team performance dashboards. The performance dashboards are available to all users and are now accessed from the main menu in IBM Process Portal 8.5.7, along with all the other dashboards.

The performance dashboards enable team leaders to monitor the performance of business operations and take appropriate actions. The functionality provided by these dashboards goes beyond the scope of this book. However, you can find more information in the following videos:

- http://ibm.biz/BPM85_Process_Performance_Video
- http://ibm.biz/BPM85_Team_Performance_Video

5.1.8 Custom dashboards

Process authors create custom dashboards to extend Process Portal to provide business users with access to information and facilities that help them perform their work. Custom dashboards can be used for a variety of reasons, including:

- Retrieve and update business data from an external system of record.
- Report on process metrics such as SLAs and KPIs.
- Report on business data.
- Add new portal capabilities, such as Out of Office.
- Replace product dashboards, such as the Work dashboard with a Get Next Task dashboard.

Custom dashboards are exposed to specific teams and are accessed from the main menu in IBM Process Portal 8.5.7, along with all the other dashboards. For more information about how to process authors can build custom dashboards, refer to 5.2.4, “Extend Process Portal with custom dashboards” on page 277.

5.1.9 Social features

Process Portal includes a number of social capabilities that allow business users to collaborate when performing their work. These capabilities are available using the context menu.

Follow

To *follow* a process instance, click the *star* (favorites) icon positioned immediately after the name of a process instance in various places where they are referenced within the standard Process Portal dashboards, as illustrated in Figure 5-8. The instances that you are following appear at the top of the context menu.

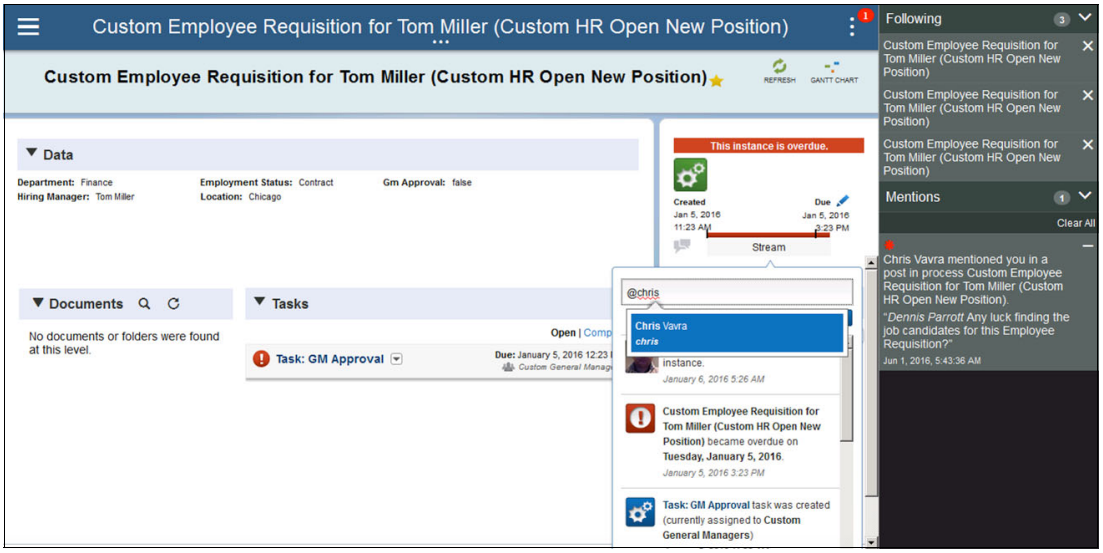


Figure 5-8 Following and Mentions on the context menu

Stream

The *Stream* provides a history of the activities and communications that are associated with a specific process instance. You can view the Stream either from the Process dashboard (see (Figure 5-8) or from within the context menu when working on a task (see Figure 5-9).

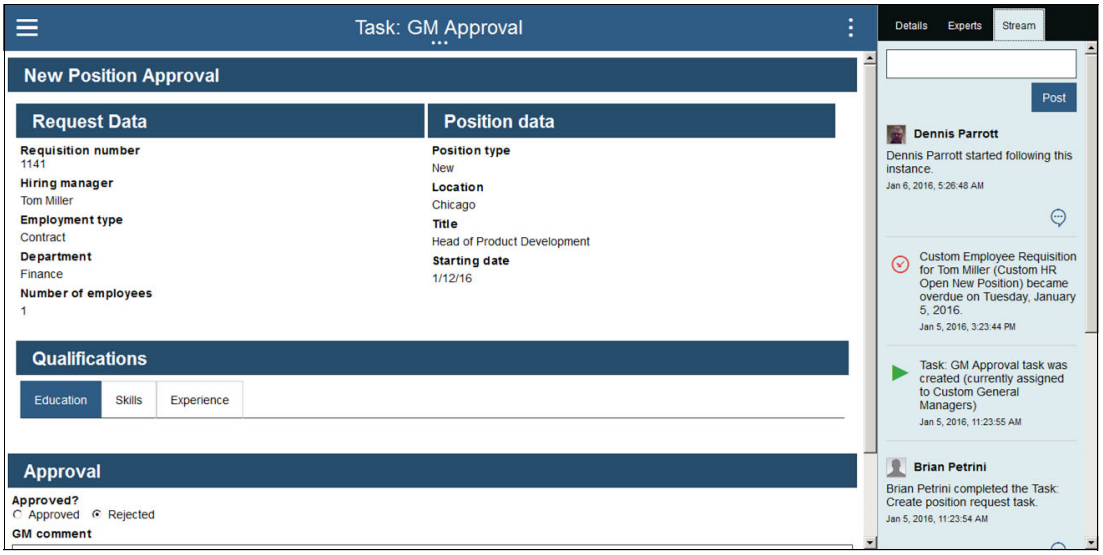


Figure 5-9 The Stream shown on the context menu when performing a task

Mentions

Business users can collaborate by messaging each other using *Mentions*. Figure 5-8 shows how to post a message using the message box at the top of the Stream. A message can contain one or more at signs (@) to refer to colleagues. After the message is posted,

everyone referenced within the message receives a notification that indicates that they have been mentioned. The number in the red circle next to the context menu icon indicates the number of mentions that are pending. Any pending messages are listed in the context menu, as illustrated in Figure 5-8 on page 270. When read, the message disappears from the context menu, but there is a record of the conversation in the Stream so that the person referenced can refer back to it as required.

Experts

Business users can collaborate with *Experts* when performing tasks. Experts display in the context menu, as shown in Figure 5-10. Experts are identified in the following ways:

- *Experienced Users* are individuals who have worked on similar tasks in the past. These user are automatically identified by the system.
- *Subject Matter Experts* are individuals from teams who are explicitly assigned to activities at design time by the process author.

Expert team membership can be managed at run time using the BPM Admin console.

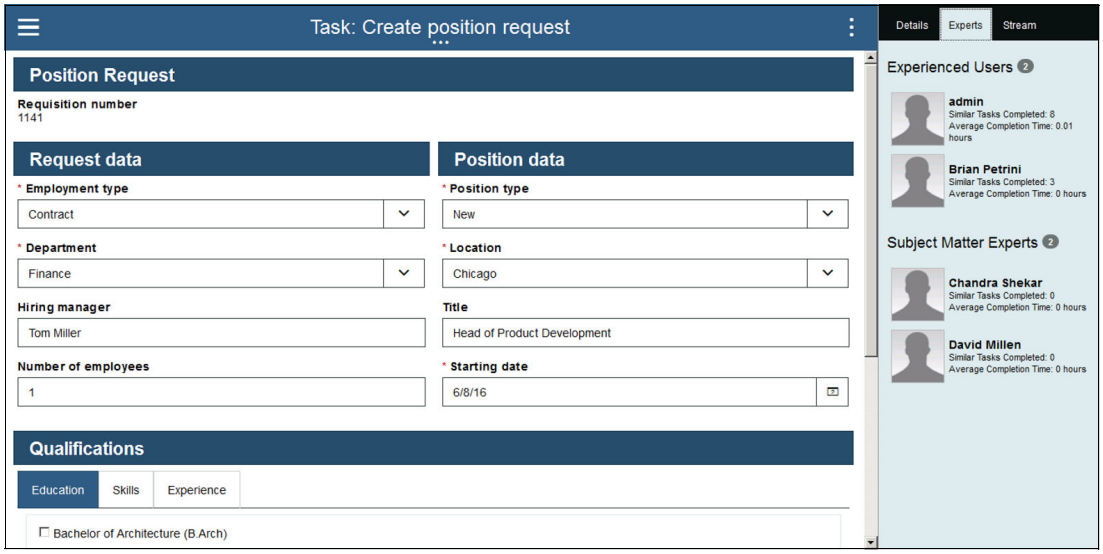


Figure 5-10 Collaborating with Experts when performing a task

Details

The last tab on the context menu, shown in Figure 5-11, provides general process instance details that can be useful when performing the task.

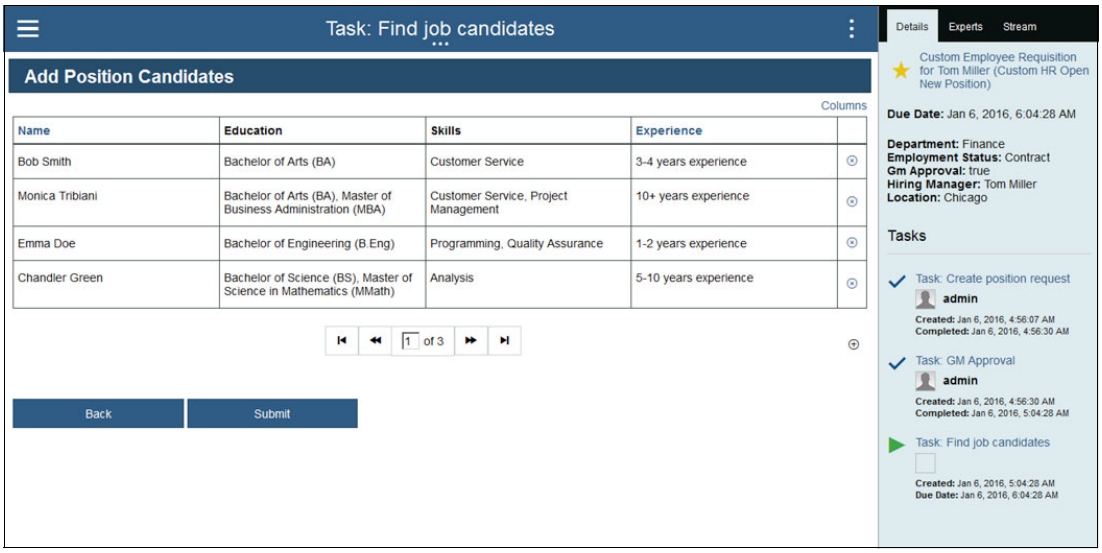


Figure 5-11 Viewing general process details when performing a task

5.2 Modifying the Process Portal user experience

The IBM BPM product team has received many requests for enhancements to make modifications and add new features to Process Portal. IBM has delivered many of these enhancements, but each user has specific requirements. So Process Portal 8.5.7 provides several options to modify the user experience:

- Personalization
- Configuration
- Styling
- Custom Dashboards
- Customization

Personalization is done by the business users themselves and impacts only each user's personal user experience. General configuration options and portal styling are controlled by the Process Administrator and impacts everybody's user experience. Custom Dashboards are defined by process authors and can be exposed to specific teams to provide custom UIs to enable business users to complete work. Finally, if all else fails, you can make customizations to Process Portal or even build a custom portal.

The sections that follow explains each of these methods.

5.2.1 Personalize the user experience

You can personalize Process Portal by making changes to the layout, updating personal preferences within the user profile, or defining you own Saved Searches.

Sticky modifications

We included previously an example of layout personalization by showing or minimizing the context menu. This option is sticky; so it remains open or closed when the user next logs on. Other layout changes that are sticky include:

- Task list

  The task list includes a number of sticky personalization settings:

  - Display tasks in either List View or Table View
  - Add or remove columns to the Table View
  - Specify the order that tasks are displayed

- Favorite dashboards

  You can also configure dashboards as favorites by clicking the *star* icon within the dashboard menu, as shown in Figure 5-12. Your favorite dashboards are always visible, whereas other dashboards are shown only by clicking **show more...** or when searching for a dashboard.

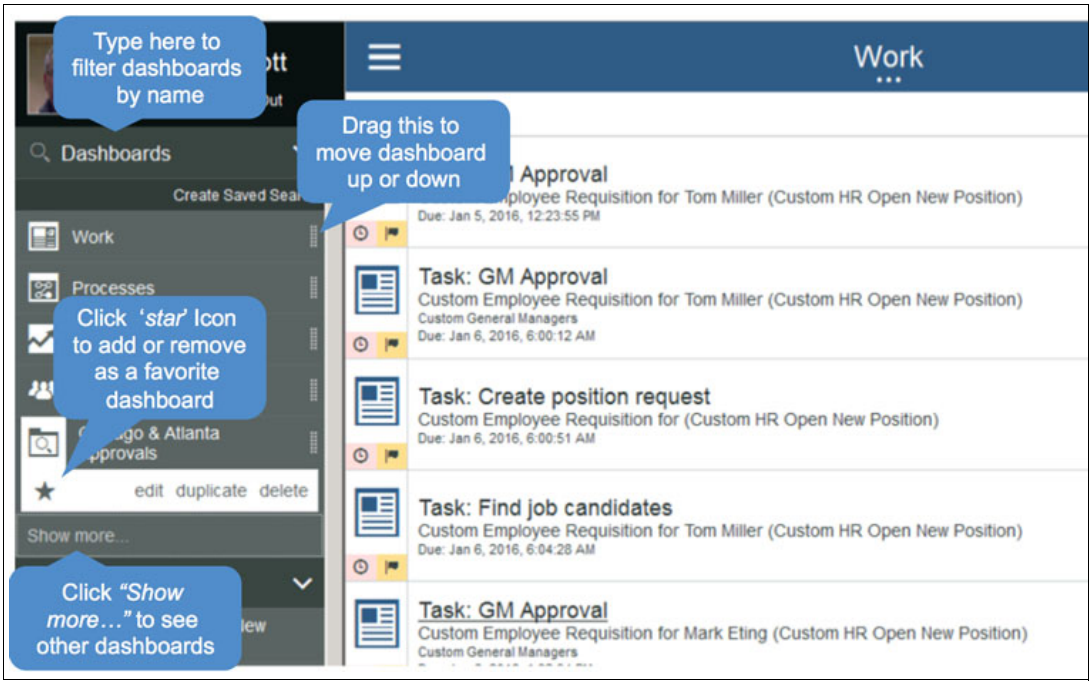


Figure 5-12 Dashboard favorites

- Replacing the Work dashboard

  The default dashboard displays each time you log in. Initially the Work dashboard is the default, but you can effectively replace the Work dashboard with an alternative by making another dashboard the default instead. For example, knowledge workers might choose to display their favorite Saved Search by default, whereas team leaders might choose to display the Team Performance dashboard by default. You can move favorite dashboards by dragging them to the required location using the shaded area to the right of the dashboard's name. Moving a favorite to the top of the list makes it the default dashboard.

Note: To hide the Work dashboard use the `com.ibm.bpm.portal.hideWorkDashboard` configuration option using the Mashups Config Service, as detailed in 5.2.2, “Configure Process Portal” on page 274.

User profile

Within your profile, you can configure your personal Portal Preferences, as shown in Figure 5-13. In the profile, you can set language and locale settings, indicate whether tasks are opened in a new window, and specify various notification options.

Note: The language and locale settings defined in the user profile affects the content in both Process Portal and the task completion UI (where localizations are defined within the coaches).

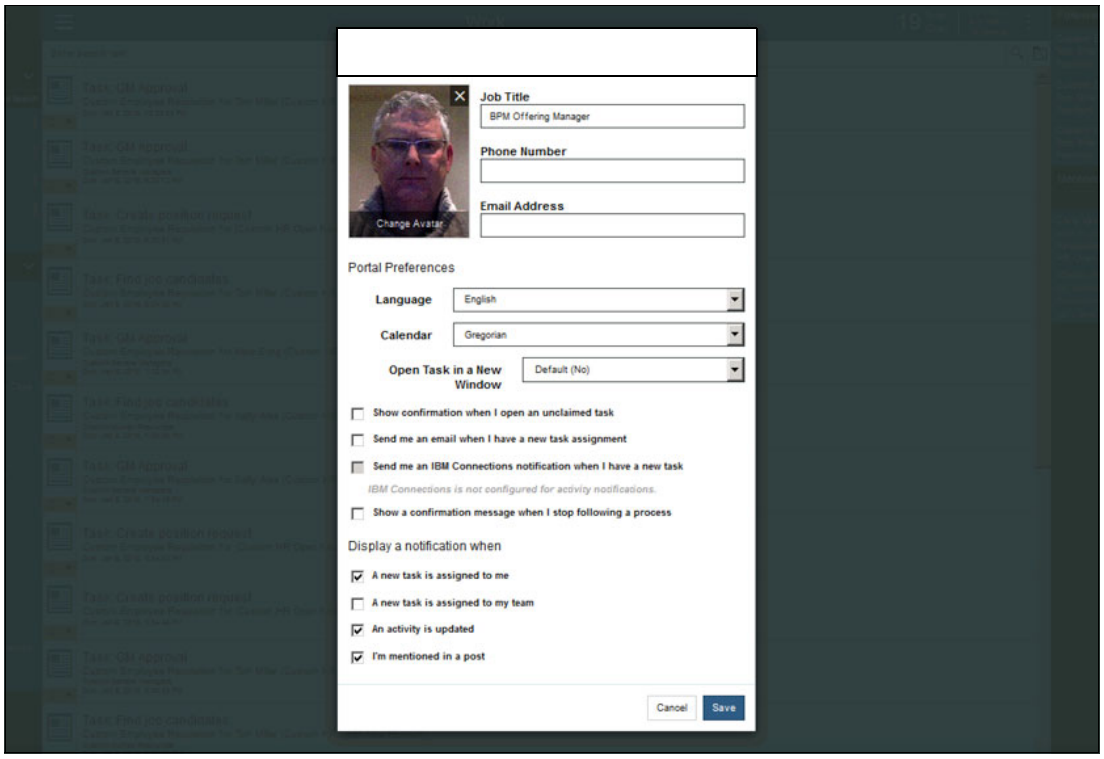


Figure 5-13 Setting personal preferences in the profile

5.2.2 Configure Process Portal

You can configure Process Portal by adding or changing options in either:

- The Mashups Config Service
- The `100Custom.xml` file[2]

Important: Changes that you make via these configuration methods are *not* affected by interim fixes or new IBM BPM releases.

[2] Consult the following IBM Knowledge Center article for information about how to update configuration options in the `100Custom.xml file`:
`http://ibm.biz/BPMProcessPortalConfig100CustomXML`

Configuration options in Mashups Config Service

The Mashups Config Service provides the following configuration options for Process Portal:

- Role-based task actions: Modify Task, Reassign, Due Date, Task Priority
- Refresh behavior
- Social Features: Following, Mentions, Experts, and Stream
- Hide Work dashboard
- Default start page (dashboard)
- Order of the dashboards
- Task-list format
- Browser behavior for opening tasks
- Time-tracking data for experts
- Support for resumable services
- HTML frame support
- Performance issues with snapshots in the Launch and Dashboards

Note: These configuration options define the default Process Portal behavior when the you first logs in. User personalization options take precedent. So some of these options can be overridden, such as Order of dashboards.

For more information about these configuration options, see IBM Knowledge Center:

`http://ibm.biz/BPMProcessPortalConfigMashups`

To demonstrate Process Portal configuration, the example shown in Figure 5-14 on page 276 switches off Process Portal social features. In this example, go to the WebSphere Application Server Console, add the `com.ibm.bpm.portal.disableSocial` property, and set the value to `following, mentions, experts, stream`.

Note: Setting the `com.ibm.bpm.portal.disableSocial` property value to `following, mentions, experts, stream` is the same as setting it to `all`.

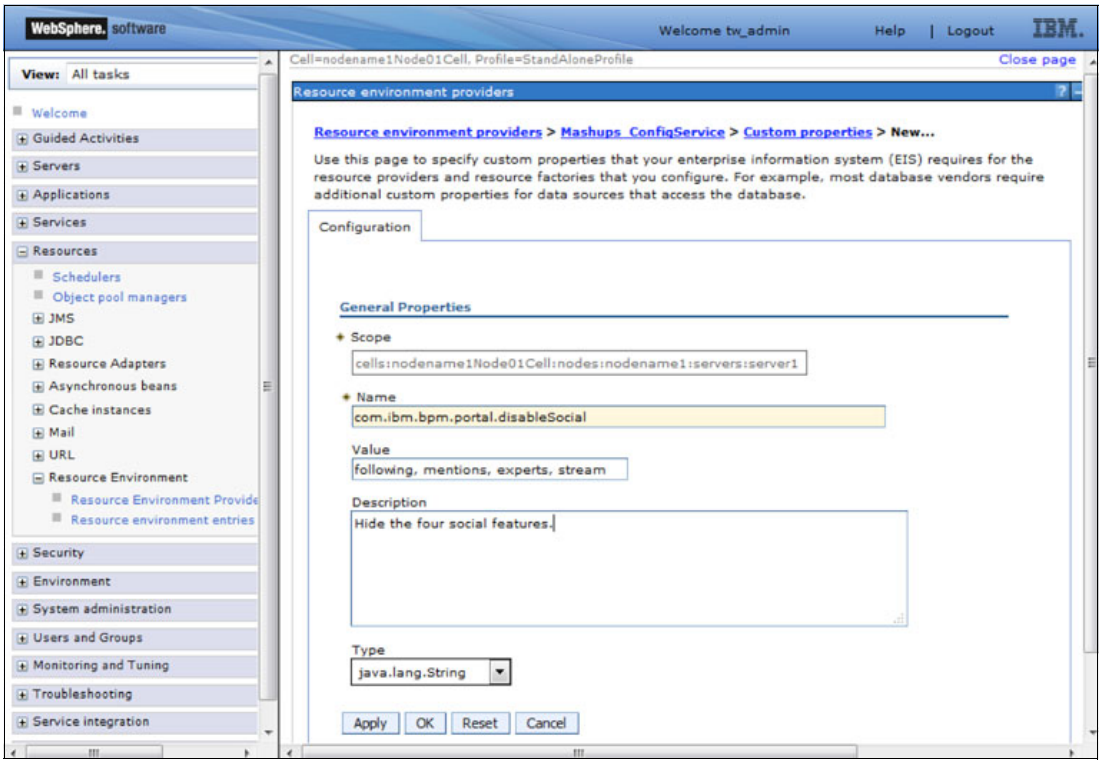


Figure 5-14 Switching off Process Portal social features using Mashups Config Service

After the BPM server restarts, the social features *Following* and *Mentions* no longer displays, as shown in Figure 5-15.

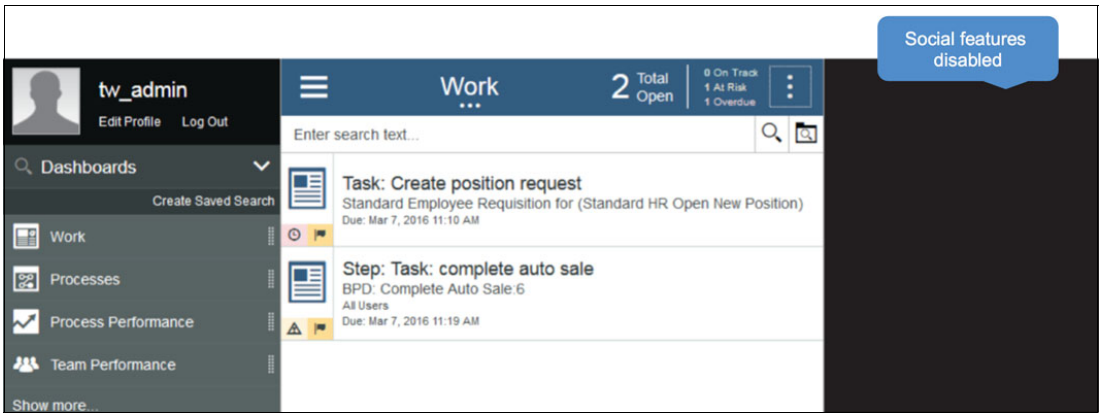


Figure 5-15 Process Portal with the social features disabled

Configuration options in 100Custom.xml

The `100Custom.xml` configuration file provides the following configuration options for Process Portal:

- Configuring Process Portal notification and refresh behavior
- Configuring the default duration for projected path discovery in Process Portal
- Disabling the display of time tracking information in Process Portal
- Configuring the default snapshot for Process Portal running on a Process Center server
- Opening services in Heritage Process Portal in a new browser window
- Configuring the My Team Performance dashboard for Heritage Process Portal (deprecated)

For more information about these configuration options, refer to IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/en/SSFPJS_8.5.7/com.ibm.wbpm.admin.doc/topics/cadm_portal_configproperties_100custom.html

5.2.3 Style Process Portal

Styling Process Portal is as simple as styling coaches, as detailed in Chapter 2, “Creating user interfaces with coaches” on page 9, because the Process Portal UI is completely constructed using coaches, except for the login window.

Process Portal process app uses the BPM Theme (from the System Data toolkit) by default. Process authors can either update the Process Portal process app to use a different theme, or BPM Administrators can dynamically update the theme at run time, as detailed in Chapter 2, “Creating user interfaces with coaches” on page 9.

5.2.4 Extend Process Portal with custom dashboards

You can use *custom dashboards* for a variety of reasons, as outlined in 5.1.8, “Custom dashboards” on page 269. This section provides an example *Custom Report Dashboard* to demonstrate how to create a custom dashboard.

An example Custom Report Dashboard

This example Custom Report Dashboard is a human service that is exposed as a dashboard, as illustrated in Figure 5-16. If related to a process, the dashboard typically is defined in the same process app as the process itself, whereas a general purpose dashboard can reside in a separate process app. A dashboard can be exposed either to all users or to a specific team. After the process app is deployed, the dashboard displays in the main menu of Process Portal for all members of the specified team.

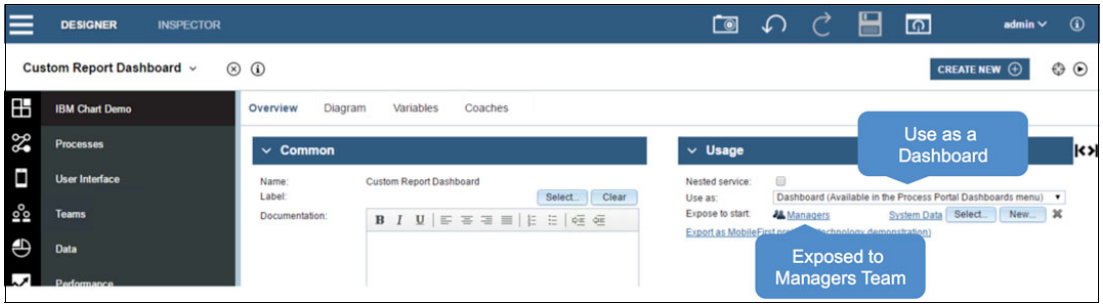


Figure 5-16 Example Custom Report Dashboard (Overview tab)

Figure 5-17 shows the human service flow for the Custom Report Dashboard. *Get Chart Data* in this example is a General System Service that retrieves the data for the dashboard.

Figure 5-17 Example Custom Report Dashboard (Diagram tab)

In reality, Get Chart Data is typically an Integration Service that retrieves data from a system of record or the BPM Performance Data Warehouse. Within this example, Get Chart Data constructs the chart data using Server Scripts, as illustrated in Figure 5-18.

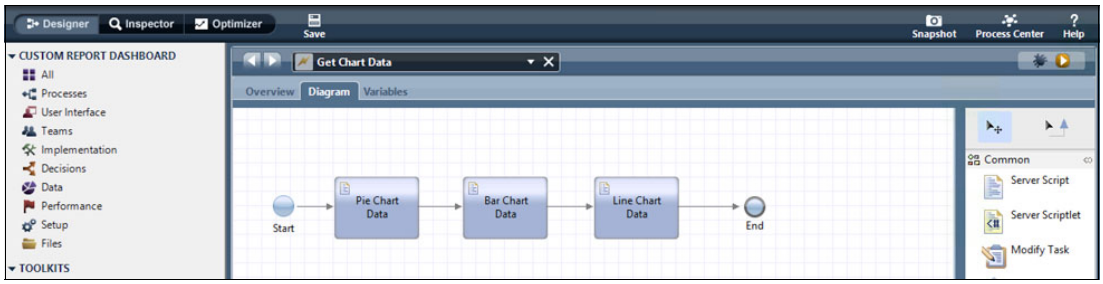


Figure 5-18 Example Get Chart Data (General System Service)

Example 5-1 shows the JavaScript that is used to provide the Bar Chart Data within this example. This example illustrates the format of the data that is required to populate a bar chart using the stock Chart control.

Example 5-1 Bar Chart Data

```
tw.local.barChartData = {
       plots: [
           {
               series: [
                   {
                       label: "Buy",
                       data: [
                           { name: "Apple", value: 4 },
                           { name: "Cherry", value: 3 },
                           { name: "Lemon", value: 1 }
                       ]
                   },
                   {
                       label: "Sell",
                       data: [
                           { name: "Apple", value: 5 },
                           { name: "Cherry", value: 2 },
                           { name: "Lemon", value: 4 }
                       ]
                   }
```

```
            ]
        },
        {
            series: [
                {
                    label: "Popularity",
                    data: [
                        { name: "Apple", value: 2 },
                        { name: "Cherry", value: 4 },
                        { name: "Lemon", value: 5 }
                    ]
                }
            ]
        }
    ]
};
```

The chart control is defined within the system dashboards toolkit, and you can use it to display the following chart types:

- Pie chart
- Bar chart (horizontal bar plot)
- Column chart (vertical bar plot)
- Line chart
- Area line chart
- Combo chart

The chart data is stored in private variables within the Custom Report Dashboard, as shown in the Variables Tab in Figure 5-19. The `ChartData` business object is a business object type that is also defined in the Dashboards toolkit.

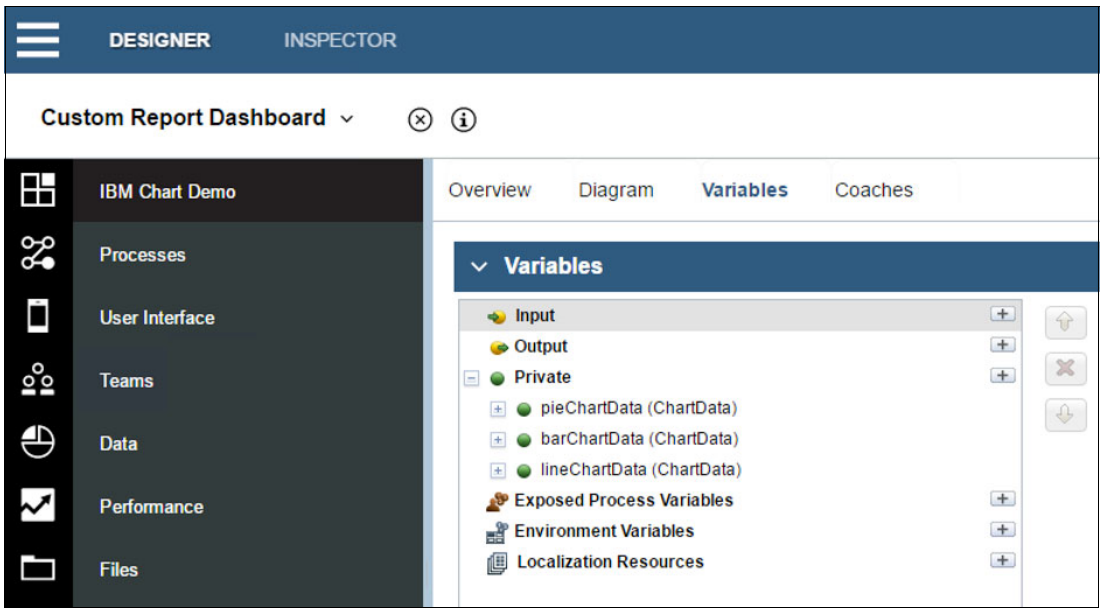


Figure 5-19 Custom Report Dashboard (Variables tab)

Figure 5-20 shows the coach that is used to display the dashboard. This example illustrates the six different chart types that are supported by the Chart control. The Properties tab shows how the Chart control is configured. Display options are used to define which type of chart to display for each set of plot data. The Chart control and its configuration options are detailed further in IBM Knowledge Center:

http://ibm.biz/BPMChartControl

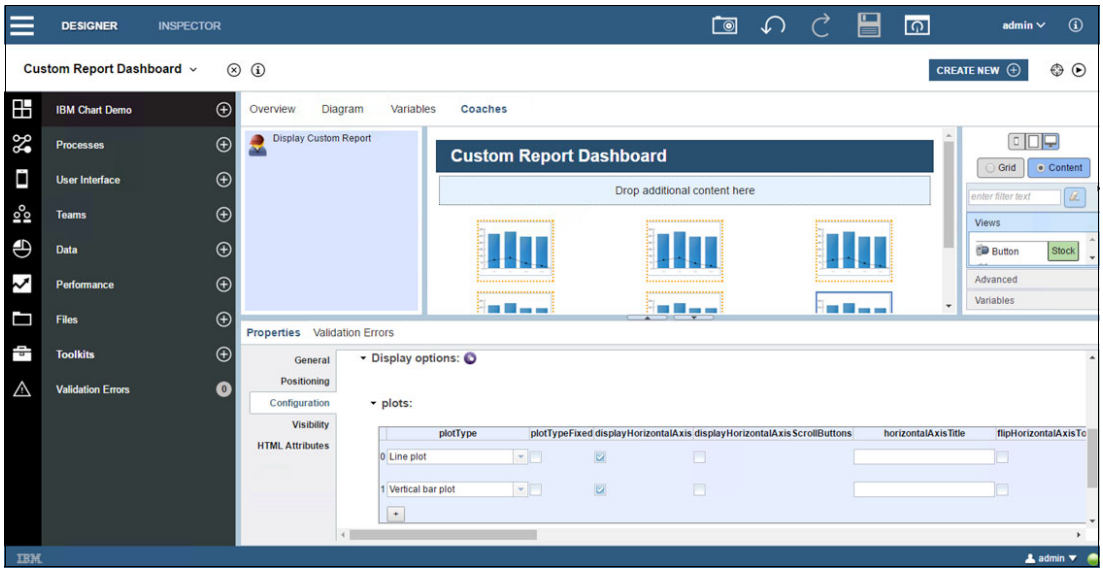


Figure 5-20 Custom Report Dashboard (Coaches tab)

Figure 5-21 shows the example Custom Report Dashboard when launched from Process Portal.

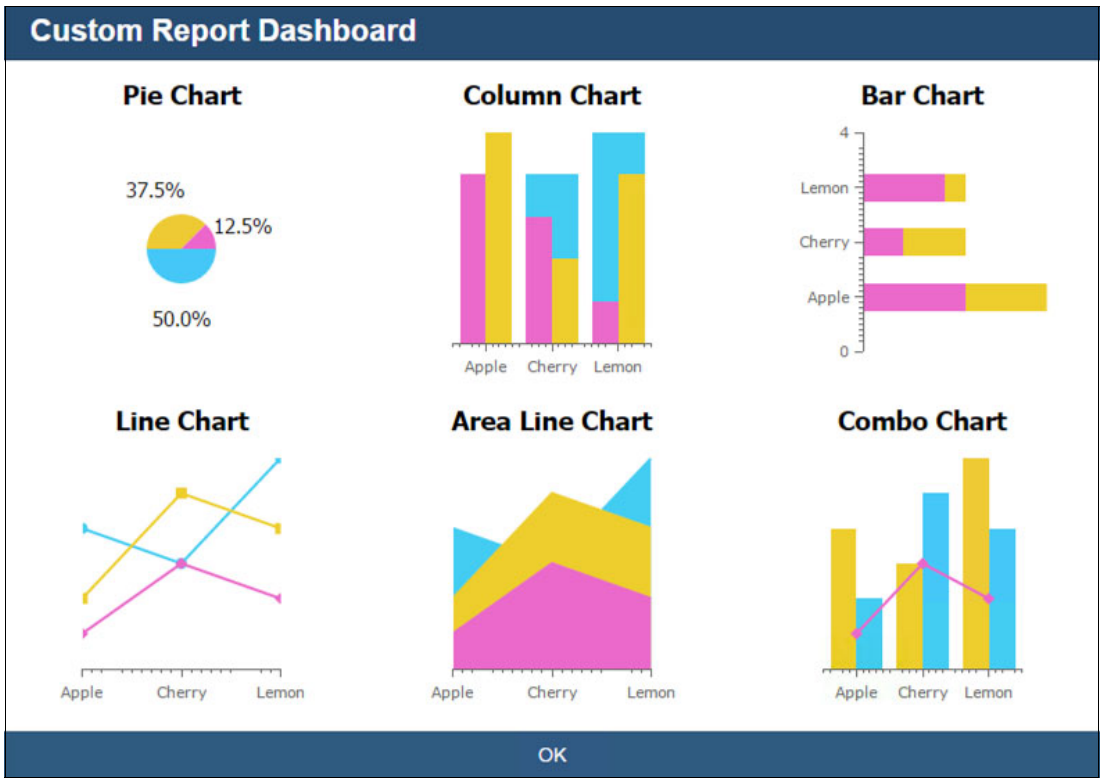


Figure 5-21 Custom Report Dashboard (Coaches tab)

The system Dashboards toolkit

The system Dashboards toolkit was originally created to provide the coach view implementation for the Heritage Process Portal. The majority of the controls are specific to the Heritage Process Portal implementation, but it also includes a number of generic controls that are useful for building dashboards and task completion UI.

The Dashboards include the following generic controls:

- Chart
- Chart with Time Selector
- Data
- Data Section
- Dialog
- Floating Layout Control
- Icon Button Control
- Refresh Button
- Refresh Controller
- Split Panes Section
- Two Column Section

Other coach views within the Dashboards toolkit, such as Task List, can be useful when building a custom portal, but these are superseded by the coach views in the Responsive Portal Components toolkit.

To learn more about dashboard controls, refer to IBM Knowledge Center:

http://ibm.biz/BPMDashboardToolkit

5.2.5 Define a Process Instance Launch UI

When launching a new process instance from Process Portal, it is a common requirement to present the user who started the instance with a form, so that the user can enter details providing the initial context for the process.

For structured Business Process Modeling Notation (BPMN) processes, you can define the process instance launch using a number of methods:

- Automatically assign the first task to the user by routing the task using Assign to: “Lane” with User Distribution: “Last User”.
- Create a custom dashboard that triggers a new process instance using an undercover agent (UCA) or JavaScript API.
- Create a custom Process Instance Launch UI.

The last option was introduced in IBM BPM 8.5.5 when the Basic Case capability was first introduced. At that time this option was the only option that worked for Basic Case. When the case capability was merged with structured processes in IBM BPM 8.5.7, this option became the preferred mechanism to accommodate this requirement for both process and cases.

Figure 5-22 on page 282 shows how to create a custom Process Instance Launch UI for a Process. This panel is available in the View tab for a Process definition. Click **New** to create a new client-side human service, or click **Select** to select an existing client-side human service.

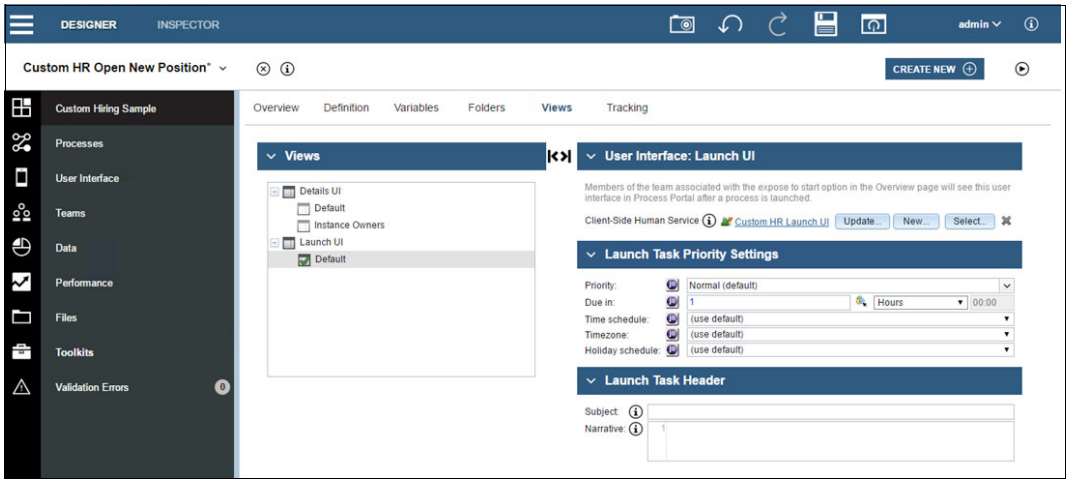


Figure 5-22 Create a Process Instance Launch UI for a process

Note: You cannot create a Process Instance Launch UI for Business Process Definition (BPD) in the Eclipse-based Process Designer. You can only create a Process Instance Launch UI for the new Process type that was introduced in IBM BPM 8.5.7 or a Basic Case type that was introduced in BPM 8.5.5 using the web-based Process Designer.

After entering the name of a new client-side human service, the client-side human service and coach is automatically generated, as illustrated in Figure 5-23 on page 283. The business objects within the process are used to construct the coach automatically. This method provides you with a starting point to customize as required. For example, if you decide to remove some of the business fields that do not need to be populated at the start of the process or layout the coach using the grid, it's entirely up to you.

Note: As the process is developed further, you can modify the business object definitions. You might need to revisit the Process Instance Launch UI (client-side human service and coach) to keep it aligned with the process by clicking **Update** in Figure 5-22.

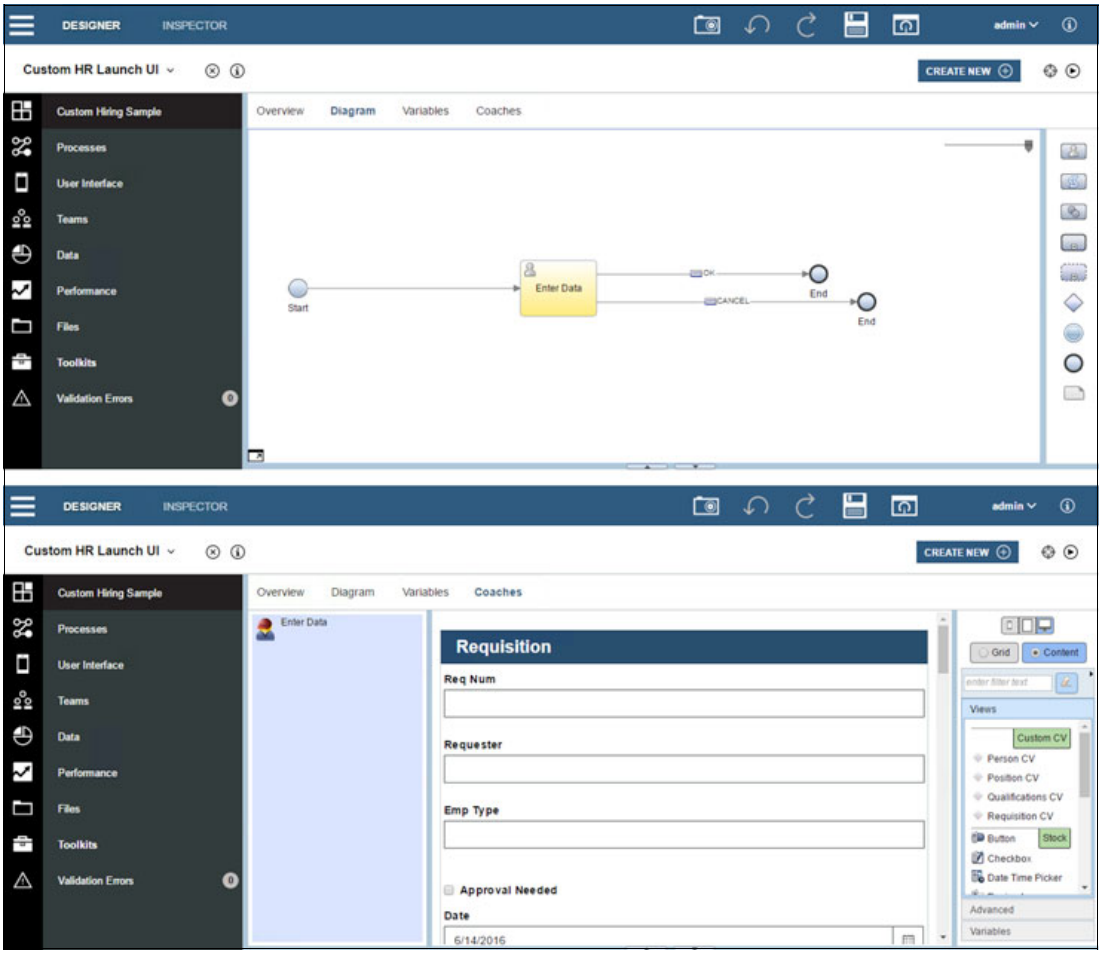


Figure 5-23 Auto-Generated client-side human service and coach for the Process Instance Launch UI

Note: The process should be exposed to a particular team for it to show up within the Process Portal launch list. The client-side human service is actually a Task service that executes in the background.

Now that the process has a Process Instance Launch UI, each time a new process instance is launched from Process Portal the Launch UI coach is displayed, as illustrated in Figure 5-24 on page 284.

The process instance actually starts immediately when launched from Process Portal. The Launch UI client-side human service is executed as a task and displays as a completed task within the process instance stream.

The generated Launch UI coach includes an OK button and a Cancel button by default, but you can rename these buttons if you choose. When you complete the coach, press **Cancel** to set the `cancelLaunch` variable within the client-side human service to `true`, and the process instance is terminated. Otherwise, the process instances continues to execute as expected.

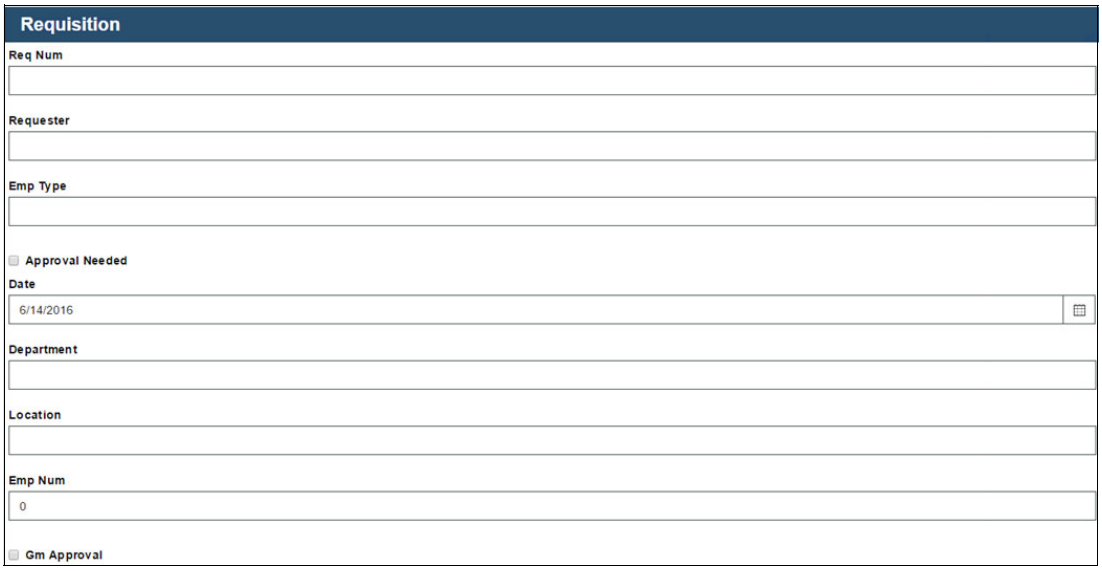


Figure 5-24 The Process Instance Launch UI displayed at run time from Process Portal

5.2.6 Define a custom Process Instance Details UI

All processes use the standard Process Instance Details UI by default, as illustrated in Figure 5-7 on page 268. Figure 5-25 shows the default Views configuration for a process.

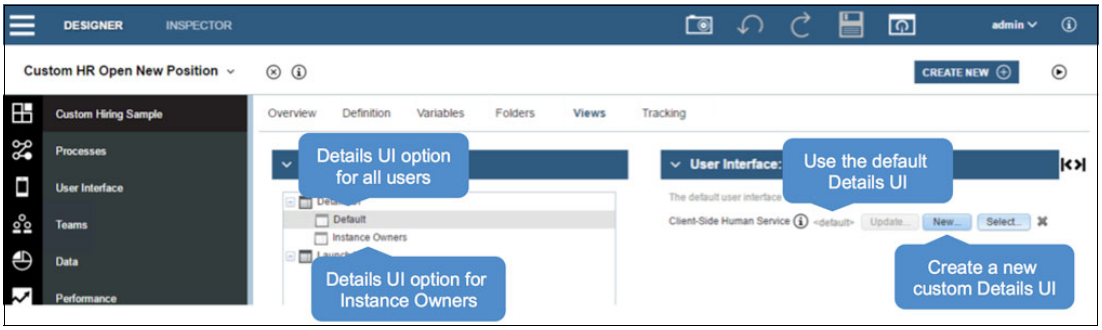


Figure 5-25 The default configuration for the Details UI within the process Views tab

A custom Process Instance Details UI is typically required when the process author wants to present additional information in the context of a specific process or to define an alternative layout. The process author can define a custom Process Instance Details UI for all users by highlighting the Default Details UI option and then clicking **New** to create a new client-side human service or **Select** to select an existing client-side human service.

Similarly, the process author can define a custom Process Instance Details UI specifically for the Instance Owners team by creating a client-side human service using the Instance Owners Details UI option.

Figure 5-26 shows the process Views configuration after a new custom Process Instance Details UI is created for all users.

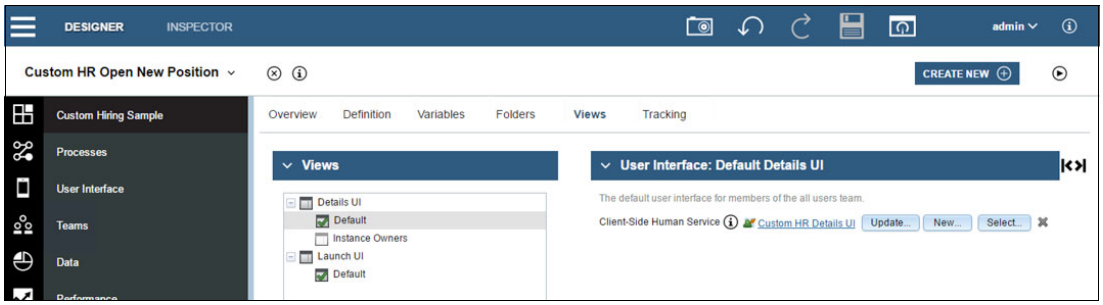


Figure 5-26 Custom Process Instance Details UI defined for All Users

When creating a new Process Instance Details UI, a new client-side human service is automatically generated, as illustrated in Figure 5-27. This approach provides a default implementation that is identical to the standard Process Instance Details UI, which can then be customized as required.

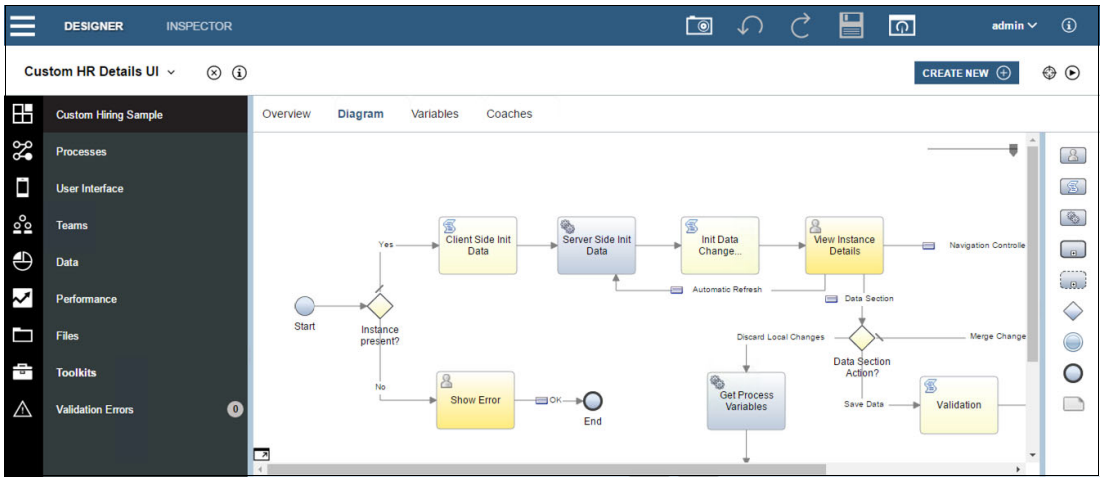


Figure 5-27 Auto-generated client-side human service baseline for the custom Process Instance Launch UI

The View Instance Details coach provides the main the UI for the custom Process Instance Details UI, as illustrated in Figure 5-28.

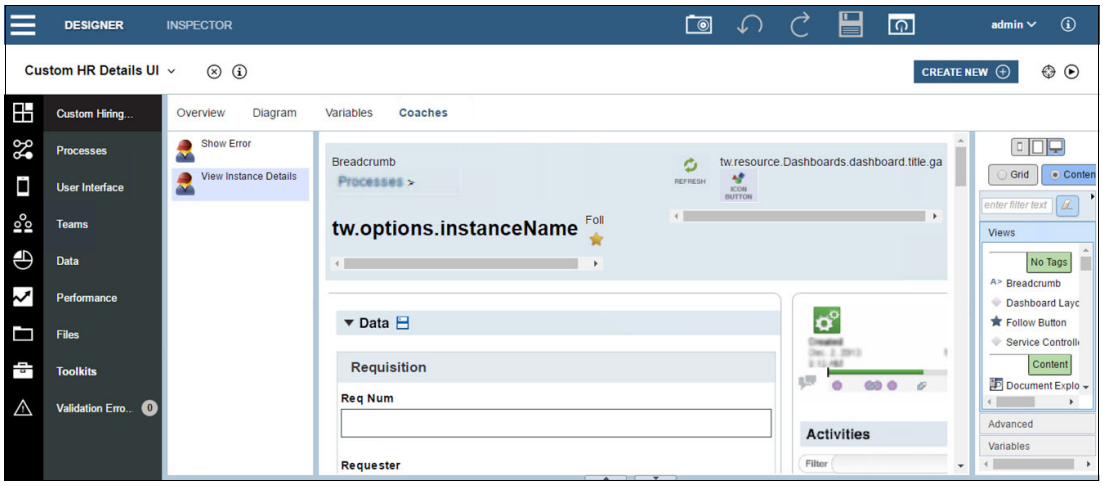


Figure 5-28 The View Instance Details coach for the custom Process Instance Details UI

5.2.7 Clone Process Portal

Process Portal, available until IBM BPM 7.5, was written in Java and was not intended to be customized. Nevertheless, many customers were determined to make modifications and needed to reverse engineer and update the code directly, which was extremely difficult and invalidated their product support.

In recognition of the desire for customers to make customizations, the new Process Portal in IBM BPM 8.0 was redeveloped using coaches to make it easier for customers to customize. However, the 20% of the implementation that was not coach-based ironically included the things that customer's typically wanted to customize. For this reason, in IBM Process Portal 8.5.7, Process Portal is implemented entirely from coaches except for the login page and heritage dashboards.

To avoid incurring in any support issues, IBM advise customers not to modify Process Portal directly. Instead it is better to create your own custom portal that can be based on a copy of the reference implementation that IBM provides ready to use. That way the original IBM Process Portal 8.5.7 always remains intact and any support issues experienced with the custom portal can easily be reproduced on the IBM implementation.

Important: The Custom Process Portal is considered to be custom code and is not supported directly.

This section details how to clone IBM Process Portal 8.5.7. After you create a clone, you can make customizations, as described in 5.2.8, "Customize Process Portal" on page 299. However, *strongly* consider using configuration options and custom dashboards before making any customizations.

Note: Because the Custom Process Portal is a clone of a specific snapshot of the IBM implementation, it will not benefit from any interim fixes or upgrades to Process Portal, unless you decide to take another clone and reapply your customizations.

IBM Process Portal 8.5.7 includes the following elements:

- The Responsive Portal Components toolkit
- The Process Portal process app
- The Responsive Portal enterprise archive (EAR) file

You can close each of these elements, as described in the following sections.

Tip: Avoid manually cloning these components for IBM BPM 8.5.7 and get started quickly by downloading the baseline Custom Process Portal that is already prepared. See Appendix A, “Additional material” on page 395 for instructions about how to download the `SG24-8355-ch5.zip` file.

Create a Custom Process Portal Toolkit

Use the BPM Process Center console to create a clone of the Responsive Portal Components toolkit, as illustrated in Figure 5-29. The clone is named *Process Portal Copy* and has the acronym *SYSRPC2* by default. You can rename the clone and the acronym. For the examples in this book, the clone is named *Custom Process Portal Toolkit*, but the acronym is left unchanged.

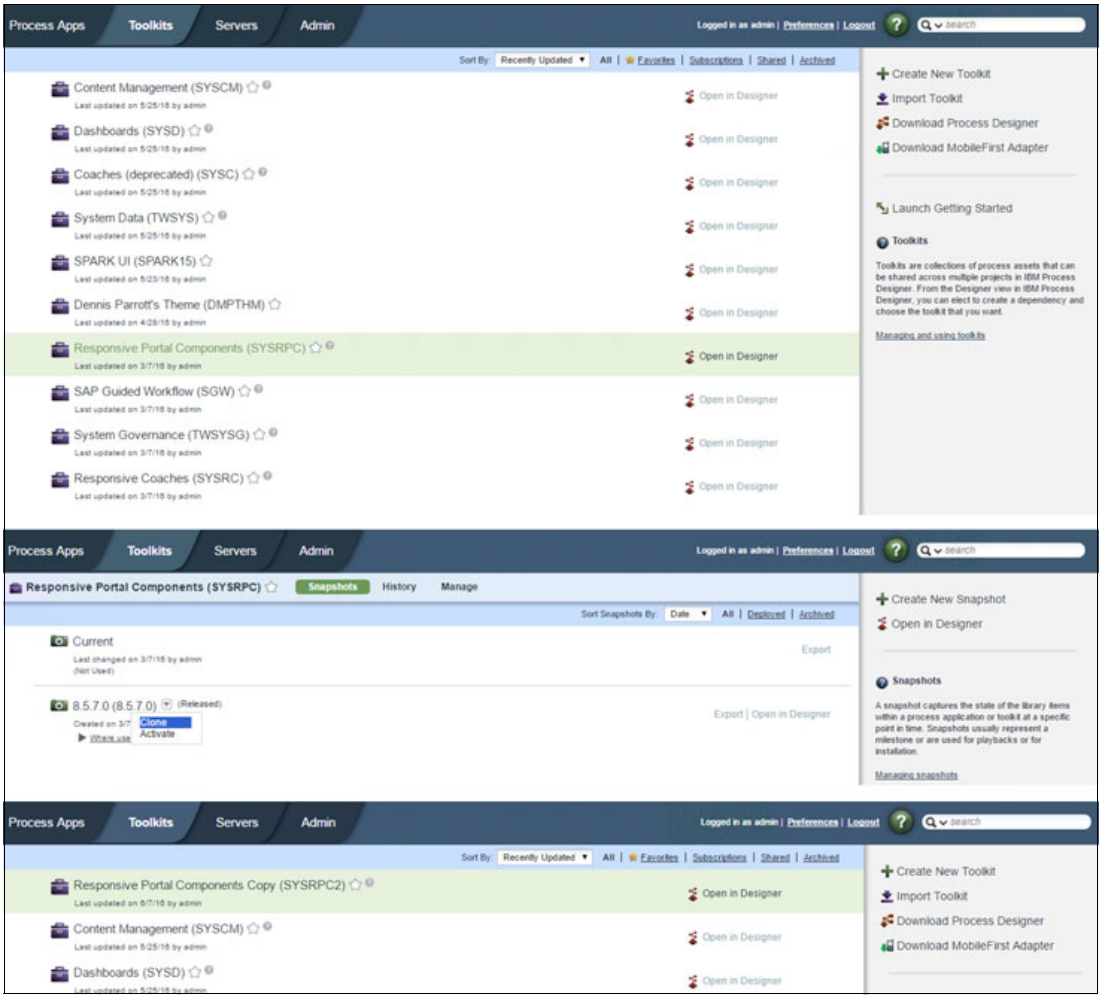


Figure 5-29 Cloning the Responsive Portal Components toolkit

Tip: Change the tag of each the coach view in the Custom Process Portal Toolkit to *Custom Portal*. Use Eclipse-based Process Designer to change these tags in bulk, as illustrated in Figure 5-30. Changing the tags is useful later to distinguish between coach views in the custom toolkit and the original toolkit. After updating the Custom Process Portal Toolkit, don't forget to take a new snapshot.

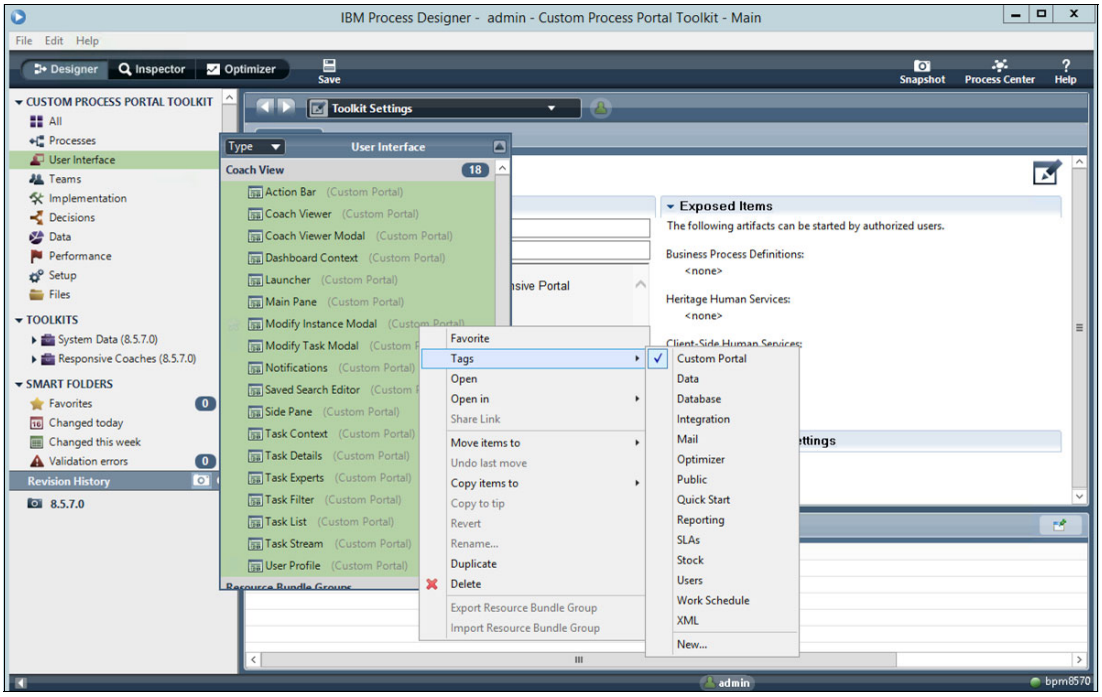


Figure 5-30 Update coach view tags in Custom Process Portal Toolkit

Now you need to update the toolkit references in the Custom Process Portal Toolkit, so that there are no dependencies on the original toolkit:

1. Open the Action Bar coach view in Process Designer, as illustrated in Figure 5-31 on page 289.
2. Update the JavaScript for the load event handler to reference the Custom Process Portal Toolkit acronym, as shown in Example 5-2.

   Example 5-2 Example to change toolkitAcronym

   ```
   window.dojoConfig.App.bpmPortalComponents = {
      toolkitAcronym: "SYSRPC2"
   };
   ```

   The `toolkitAcronym` is changed from **"**`SYSRPC`**"** to **"**`SYSRPC2`**"** so that it references the correct toolkit.
3. Repeat these steps for all remaining coach views within the Custom Process Portal Toolkit.

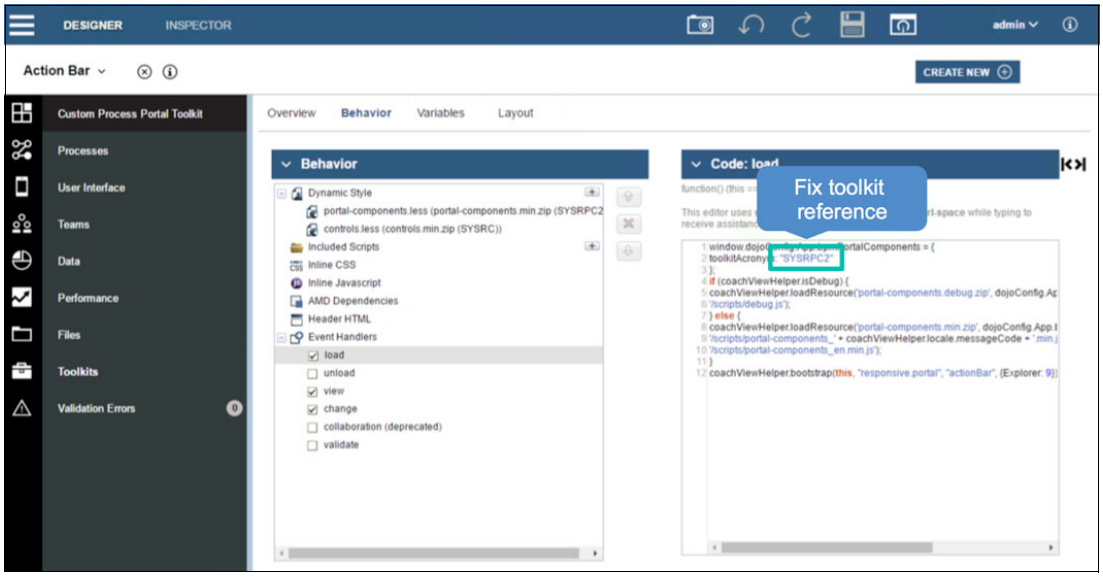


Figure 5-31 Update toolkit references in custom coach views

Create a Custom Process Portal process app

To create a Custom Process Portal process app:

1. Clone the Process Portal process app.
2. Add the Custom Process Portal Toolkit as a toolkit dependency.
3. Update toolkit references in Portal coach view load event handler.
4. Update the dynamic styles in the Portal and Work coach views.
5. Update data types to reference those defined in the custom toolkit.
6. Update the coach views to reference those defined in the custom toolkit.
7. Remove the Responsive Process Portal toolkit dependency.
8. Take a snapshot of the Custom Process Portal app.

These steps are detailed further in the following sections.

Clone the Process Portal process app

Use the BPM Process Center console to create a clone of the Process Portal process app, as illustrated in Figure 5-32 on page 290. The clone is named *Process Portal Copy* and has the acronym *SYSRP2* by default. You can choose to rename the process app. For the remainder of this book, the example for the cloned process app use the name *Custom Process Portal App*, but the acronym is left unchanged.

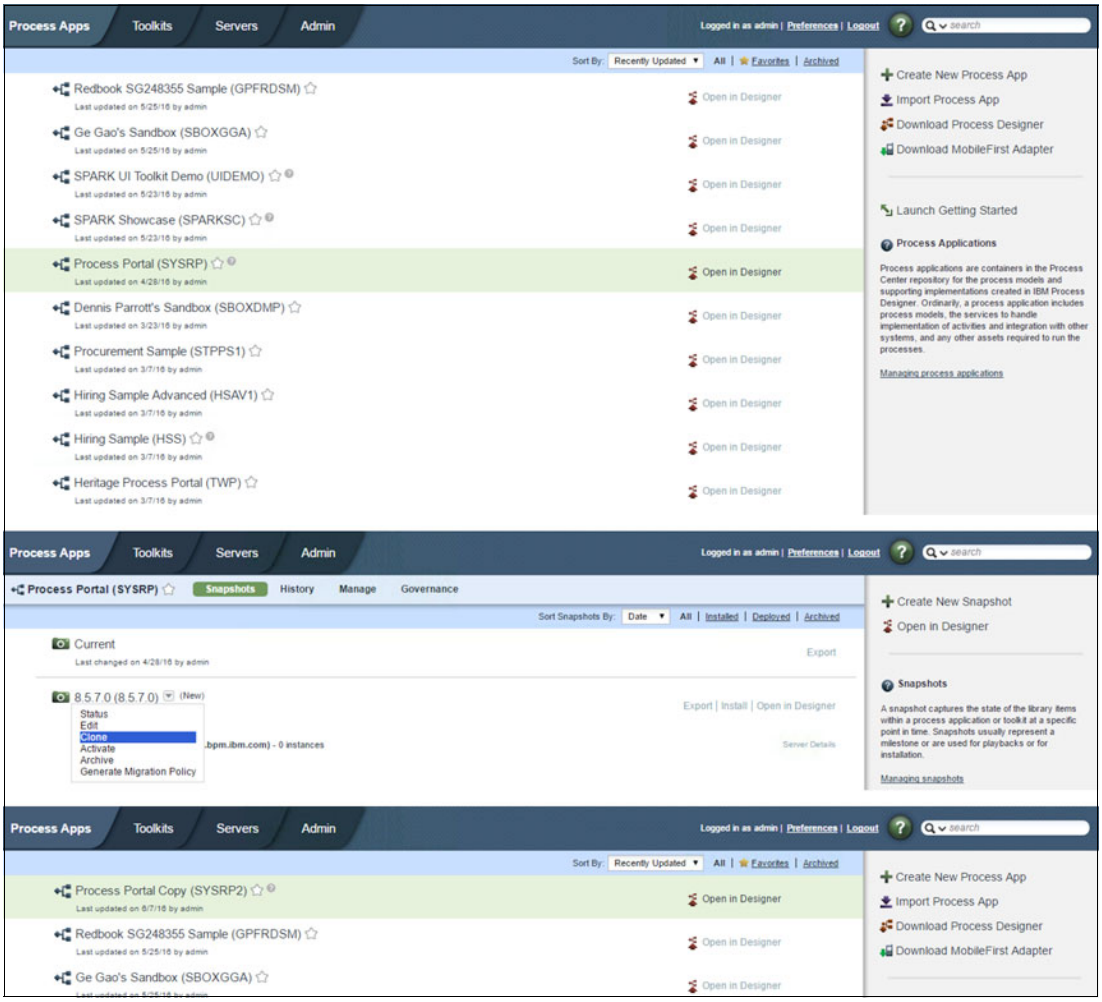


Figure 5-32 Cloning the Process Portal process app

Add the Custom Process Portal Toolkit as a toolkit dependency

Open the Custom Process Portal App in Process Designer. Then, add the Custom Process Portal Toolkit as a toolkit dependency, as illustrated in Figure 5-33 on page 291. Leave the Responsive Portal Components toolkit as a dependency, because the Custom Process Portal App still uses the coach views referenced in the Responsive Portal Components toolkit at this point.

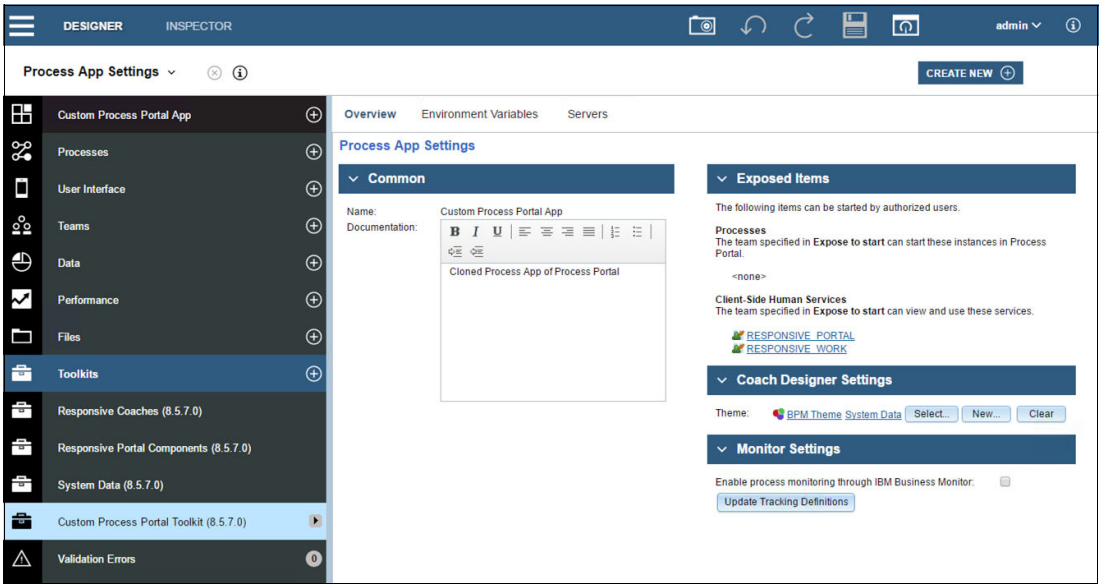


Figure 5-33 Adding the Custom Process Portal Toolkit as a toolkit dependency

Update toolkit references in Portal coach view load event handler

Open the Portal coach view. Then, update the JavaScript within the load event handler to reference the new custom process app and toolkit by updating the acronyms to `SYSRP2` and `SYSRPC2`, respectively, as illustrated in Figure 5-34.

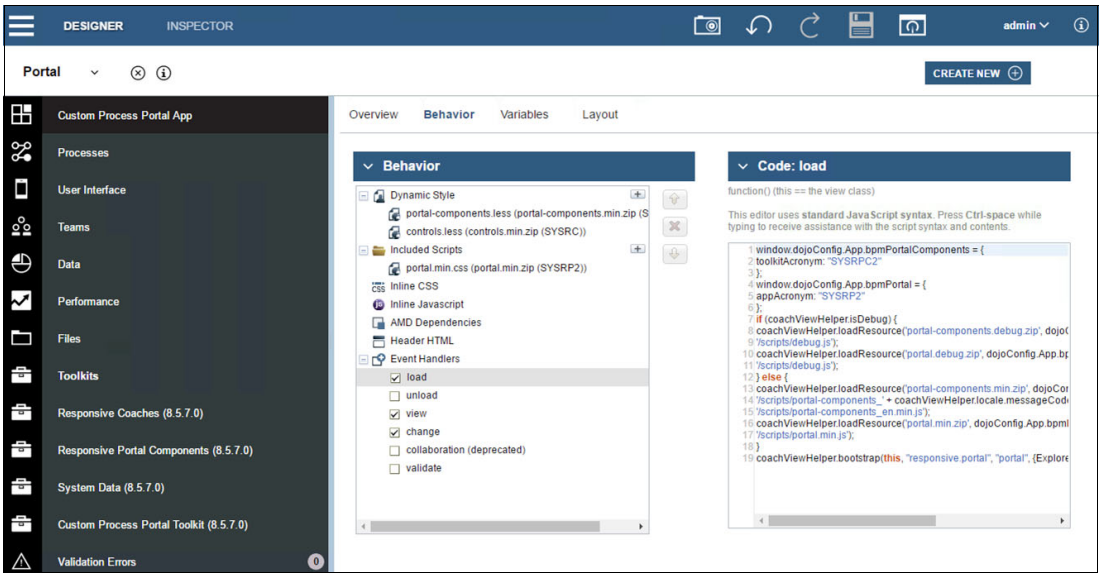


Figure 5-34 Updating Portal coach view load script to reference the custom process app and toolkit

Tip: Change the tag of each the coach view in the Custom Process Portal App to *Custom Portal*. Use Eclipse-based Process Designer to change these tags in bulk. Changing the name tags is useful later so that you can distinguish between coach views in the custom app and the original app and avoid making customizations to the original process app by mistake.

Update dynamic styles in Portal and Work coach views

Open the *Portal* coach view in Process Designer, select the Behavior tab, and update the Dynamic Styles by replacing the following line:

```
portal-components.less (portal-components.min.zip (SYSRPC))
```

with this line:

```
portal-components.less (portal-components.min.zip (SYSRPC2))
```

Be careful to retain the original order of the less files, as shown in Figure 5-35.

Repeat this process for the Work coach view.

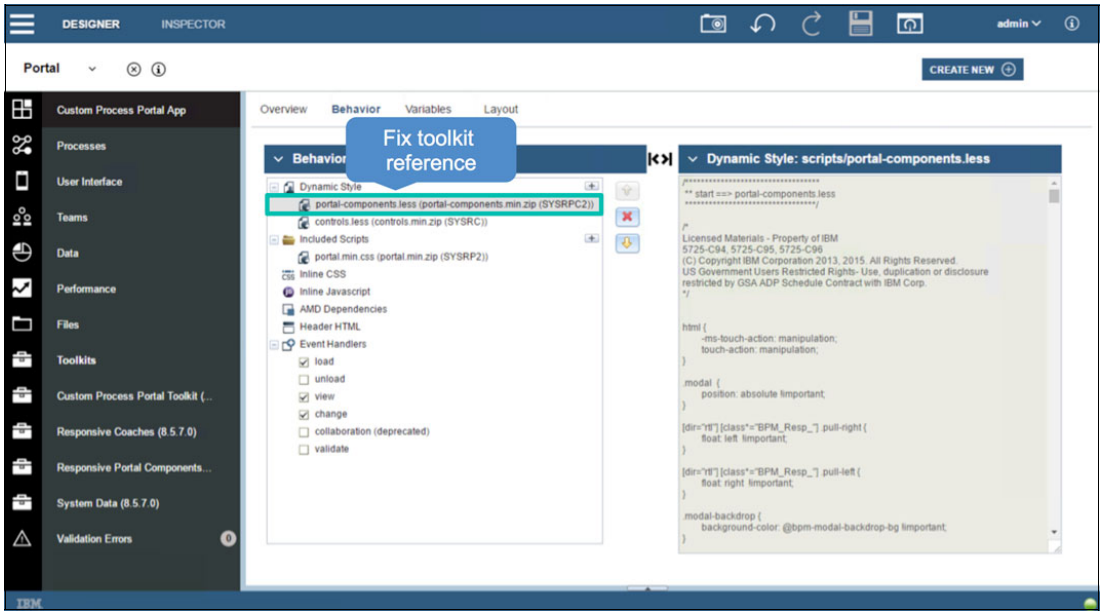


Figure 5-35 Updating Portal coach view to reference the portal-components.less file

Update data types to reference those defined in the custom toolkit

Open the Portal coach view in Process Designer. Then, select the Variables tab, as illustrated in Figure 5-36 on page 293. Update the data types for the following variables to reference the business objects that are defined in the custom toolkit:

- `currentTask` (Task)
- `currentDashboard` (Dashboard)
- `currentViewMode` (PortalViewMode)

Similarly, open the `RESPONSIVE_PORTAL` client side human service in Process Designer. Then, select the Variables tab, and update the data types for the following variables to reference the business objects defined in the custom toolkit:

- `currentTask` (Task)
- `currentDashboard` (Dashboard)
- `currentViewMode` (PortalViewMode)

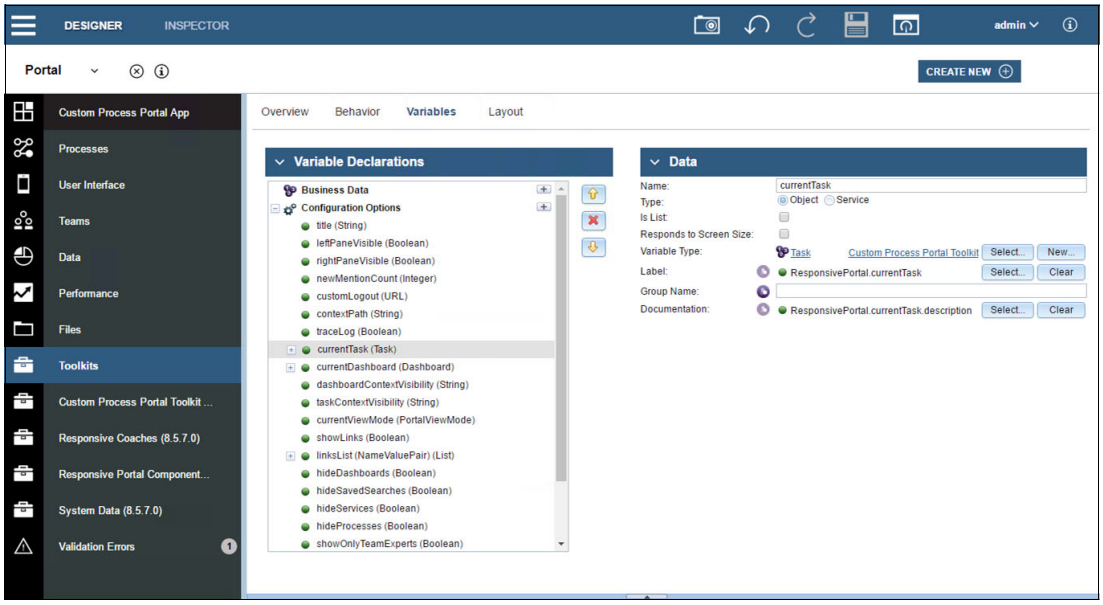


Figure 5-36 Updating the Portal coach view to reference data types in the custom toolkit

Update the coach views to reference views defined in the custom toolkit

To update coach views to references views in the custom toolkit:

1. Open the Portal coach view in Process Designer.
2. Go to the Layout tab, as illustrated in Figure 5-37 on page 294 and update the coach views to reference the following views defined in the custom toolkit:
   - Side Pane (main menu)
     - User Profile
     - Launcher
   - Main Pane
     - Action Bar
     - Coach Viewer
   - Side Pane (context menu)
     - Dashboard Context
     - Task Details
     - Task Experts
     - Task Stream
     - Task Context
   - Notifications

Note: The Side Pane and Main Pane coach views contain other coach views. For example, the Main Pane contains Action Bar and Coach Viewer. Notice that the Side Pane is used *twice*, once for the main menu and then again for the context menu.

Tip: To select Side Pane and Main Pane, click the vertical edge on the left or right side. To select Task Details, Task Experts, Task Stream, and Task Context, click each of the tab headings.

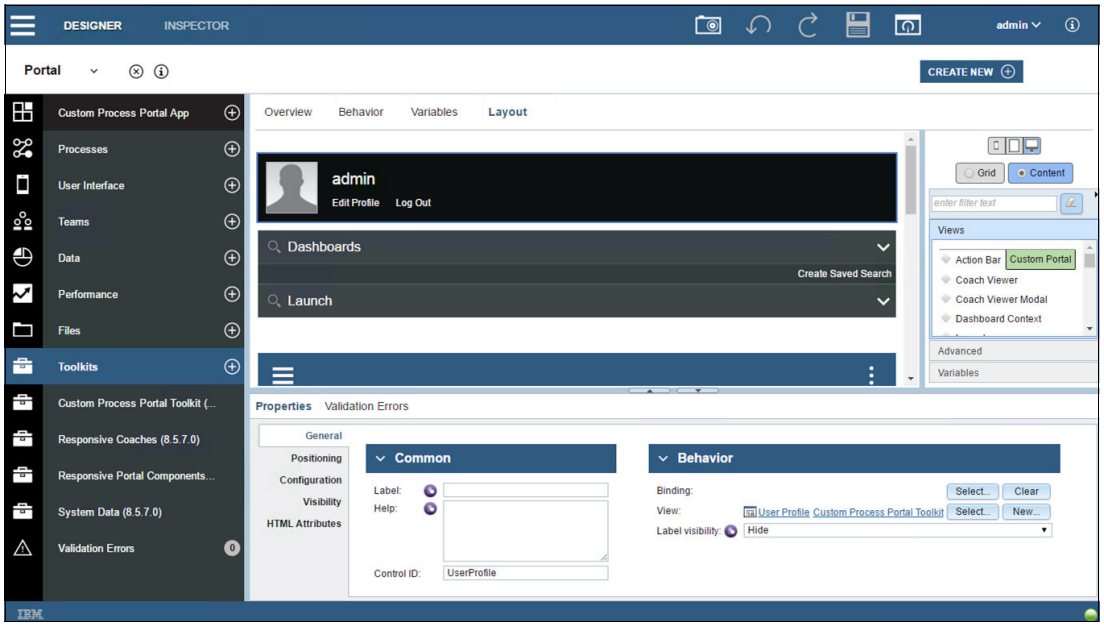


Figure 5-37 Updating the Portal coach view to reference coach views in the custom toolkit

3. Open the Work coach view in Process Designer, go to the Layout tab, and update the following coach views to reference the views defined in the custom toolkit:
   - Task Filter
   - Last List
   - Coach Viewer Modal
   - Modify Task Modal
   - Modify Instance Modal
   - Notifications

Remove the Responsive Process Portal toolkit dependency

After you update all the toolkit references, as detailed in the previous section, you can remove the Responsive Portal Components toolkit as a toolkit dependency for the Custom Process Portal app.

Tip: If you have validation errors within the Custom Process Portal App after removing the Responsive Portal Components toolkit dependency, you have missed some of the references.

Take a snapshot of the Custom Process Portal App

Take a snapshot of the Custom Process Portal App, so that you have a base-line implementation that you can use to create one or more custom portals.

Note: Creating a second custom portal app is much easier, because it can share the same Custom Process Portal Toolkit. You do not need to update the toolkit dependencies if you create the second custom portal app by cloning the Custom Process Portal App baseline snapshot.

Create a Custom Process Portal EAR file

To create your own Custom Process Portal EAR file you need to perform the following steps:

1. Clone and unpack the Process Portal EAR file.
2. Update the `web.xml` file.
3. Update the `application.xml` file.
4. Update `ibm-application-bnd.xmi` file.
5. Package and rename the Custom Process Portal EAR file.
6. Install the Custom Process Portal EAR file.
7. Start the Custom Process Portal EAR file.

These steps are detailed further in the following sections.

Clone and unpack the Process Portal EAR file

Use the WebSphere Application Server Admin console to export the `IBM_BPM_ResponsivePortal_SingleCluster` EAR file, as illustrated in Figure 5-38.

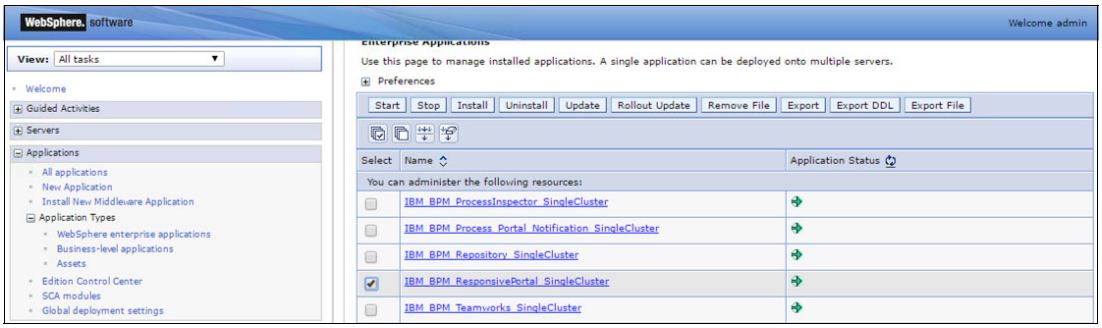


Figure 5-38 Exporting IBM Process Portal EAR file

You can use either the local file system or your favorite Java developer tools to unpack and make the following changes to the EAR file.

Tip: The EAR file and embedded web archive (WAR) files are compressed (`.zip`) files. So if these files are updated on the local file system, you can modify them by extracting the content, making the updates, and compressing them again.

Update the web.xml file

Update the following file as shown in Example 5-3:

`IBM_BPM_ResponsivePortal_SingleCluster.ear\responsivePortal.war\WEB-INF\web.xml`

Example 5-3 The web.xml file extract

```
<init-param>
   <param-name>appName</param-name>
   <param-value>SYSRP2</param-value>
</init-param>
<init-param>
   <param-name>serviceName</param-name>
   <param-value>RESPONSIVE_PORTAL</param-value>
</init-param>
```

The `appName` should match the acronym for your Custom Process Portal App (`SYSRP2`).

The `serviceName` should match the human service within your Custom Process Portal App that defines the custom portal user interface. For now, you can leave the name as `RESPONSIVE_PORTAL` so that you can reuse the existing implementation as a base line for the custom portal. However, later you might want to change this name if you want to build a custom portal from scratch.

Update the application.xml file

Update the following file as shown in Example 5-4:

`IBM_BPM_ResponsivePortal_SingleCluster.ear\META-INF\application.xml`

Example 5-4 The application.xml file extract

```
<application id="CustomPortal_ID">

      <display-name>Custom Portal</display-name>

      <description>Custom Portal</description>

      <module id="WebModule_2084909887354">

         <web>

            <web-uri>responsivePortal.war</web-uri>

            <context-root>/CustomPortal</context-root>

         </web>

      </module>

    <security-role id="SecurityRole_2405939848572">

        <description>An administrator</description>

        <role-name>twuser</role-name>

    </security-role>
</application>
```

Update the following attributes so that they do not conflict with the original Process Portal EAR file:

- The `application id` attribute uniquely identifies for the custom EAR file.
- The `display-name` attribute appears in the WebSphere Application Server console by default and is used to visually identify the app.
- The `description` attribute is for documentation purposes. (Optional)
- The `module id` attribute uniquely identifies the Responsive Portal WAR file inside the EAR file.
- The `context root` attribute defines the URL for Process Portal.
- The `security-role id` attribute uniquely identifies the security role for the EAR file.

Update the ibm-application-bnd.xmi file

Update the following file as shown in Example 5-5:

`IBM_BPM_ResponsivePortal_SingleCluster.ear\META-INF\ibm-application-bnd.xmi`

Example 5-5 The ibm-application-bnd.xmi file extract

```
  <authorizationTable xmi:id="AuthorizationTable_1457365628655">

    <authorizations xmi:id="RoleAssignment_1457365628655">

      <specialSubjects
xmi:type="com.ibm.ejs.models.base.bindings.applicationbnd:AllAuthenticatedUsers"
xmi:id="AllAuthenticatedUsers_1457365628655" name="AllAuthenticatedUsers"/>

      <role href="META-INF/application.xml#SecurityRole_2405939848572"/>

    </authorizations>

  </authorizationTable>

  <application href="META-INF/application.xml#CustomPortal_ID"/>
```

Update the following attributes so that they match the changes made to the `application.xml` file:

- The `role href` attribute must use the `security-role` value as specified in the `application.xml` file.
- The `application href` attribute must use the `application id` value as specified in `application.xml` file.

After you make all the modifications to the EAR file, complete the actions in the following sections.

Package and rename the Custom Process Portal EAR file

Package the WAR and EAR files using your chosen tools and rename the Custom Process Portal EAR file so that it is unique:

`IBM_BPM_CustomPortal_SingleCluster.ear`

Install the Custom Process Portal EAR file

Use the WebSphere Application Server Admin console to install the custom Custom Process Portal EAR file, as shown in Figure 5-39 on page 298. Follow these steps:

1. Select **Applications** → **New Application**.
2. Select **New Enterprise Application**.
3. Select **Choose File** and locate the `IBM_BPM_CustomPortal_SingleCluster.ear` file on the file system.
4. Click **Next** on this panel and on all subsequent panels to accept the default settings.
5. After the installation completes, remember to save to the master configuration.

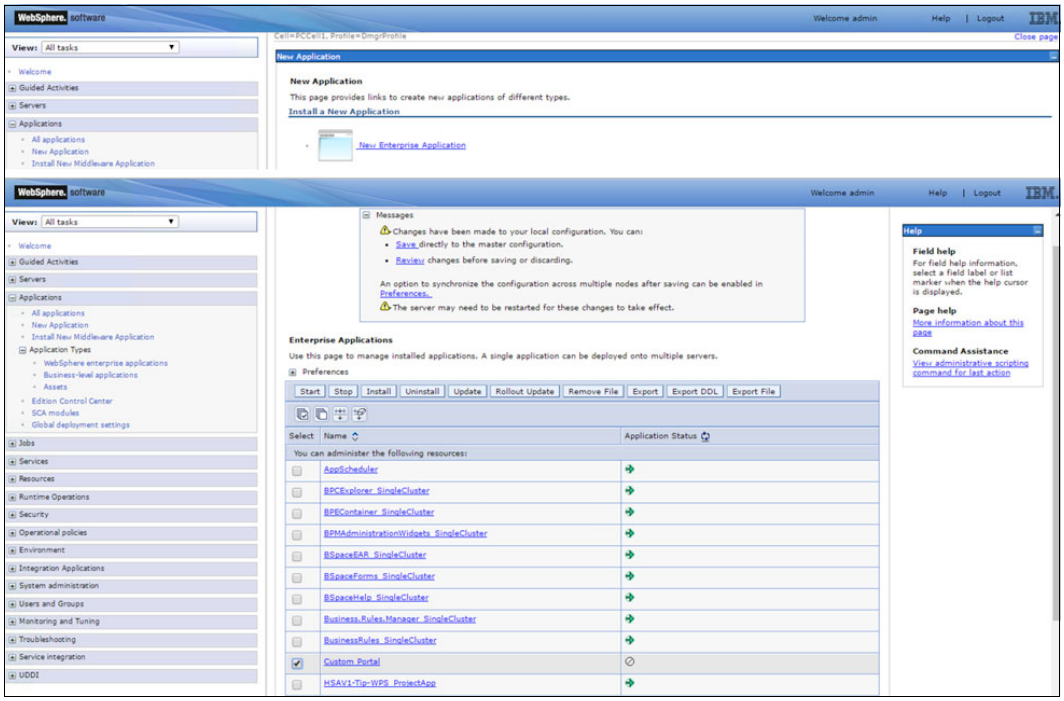


Figure 5-39 Installing the Custom Process Portal EAR file

6. Start the Custom Process Portal EAR file by using the WebSphere Application Server, as shown in Figure 5-40. Select **Custom Portal**, and click **Start**.

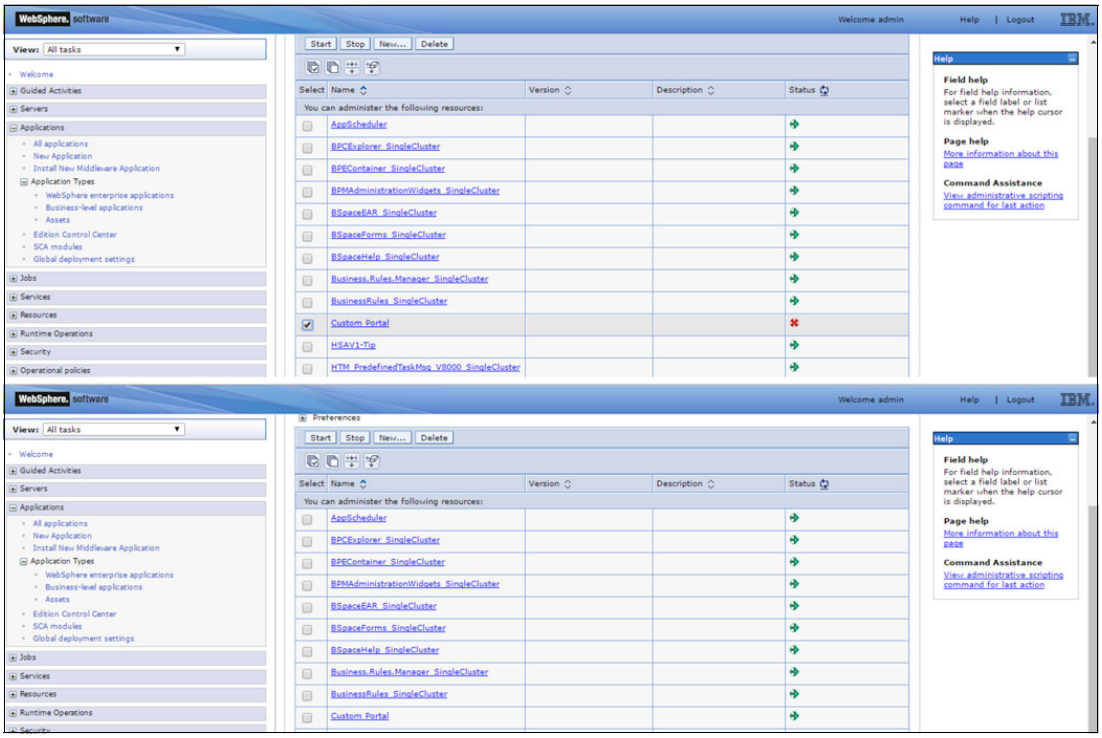


Figure 5-40 Starting the Custom Process Portal EAR file

Launch your Custom Process Portal

After you create a Custom Process Portal app and toolkit and the Custom Process Portal EAR file is running, you can launch your Custom Process Portal from a browser using the following URL (substituting the host name and port number accordingly):

`https://<host>:<port>/CustomPortal`

> **Tip:** The host name and port numbers for the Custom Process Portal are identical to those used for the standard Process Portal.

Now that you have created a Custom Process Portal, you can customize Process Portal.

5.2.8 Customize Process Portal

This section details how to make customizations to IBM Process Portal 8.5.7. Before making any customizations, create a clone of Process Portal, as detailed in 5.2.7, “Clone Process Portal” on page 286, to avoid invalidating your BPM product support.

Make customizations using the coach-view configuration options

The Custom Process Portal app includes both the Portal and the Work coach views. The Portal coach view is the top-level coach view and contains all the coach-based content within the Custom Process Portal. The Work coach view provides the definition for the work dashboard in the Custom Process Portal. The coach views included in the Custom Portal Process Toolkit are documented in IBM Knowledge Center (because these are copies of those in the Responsive Portal Components toolkit):

`http://ibm.biz/BPMResponsivePortalComponentsToolkit`

All the coach views in the Custom Process Portal app and the Custom Process Portal Toolkit have configuration options that you can use in your customizations.

Add a link to the Heritage Portal

This section illustrates how to make a customization to the Custom Process Portal app. This example adds a link to the Heritage Process Portal, so that you can access it directly from the Custom Process Portal. To add the link, you use the configuration option on the Portal coach view within the Custom Process Portal App.

To add a link to the Heritage Portal:

1. Open the Custom Process Portal app in Process Designer.
2. Open the `RESPONSIVE_PORTAL` client side coach view, as illustrated in Figure 5-41 on page 300. Click the Portal coach view in the Coach Editor, select the **Show links** configuration option, and add the following link to the Links configuration option:
   - Name: `Heritage Process Portal`
   - Value: `https://<host>:<port>/HeritagePortal`

   > **Tip:** The host name and port numbers for the Heritage Process Portal are identical to those used for the standard Process Portal.

3. Save your changes.

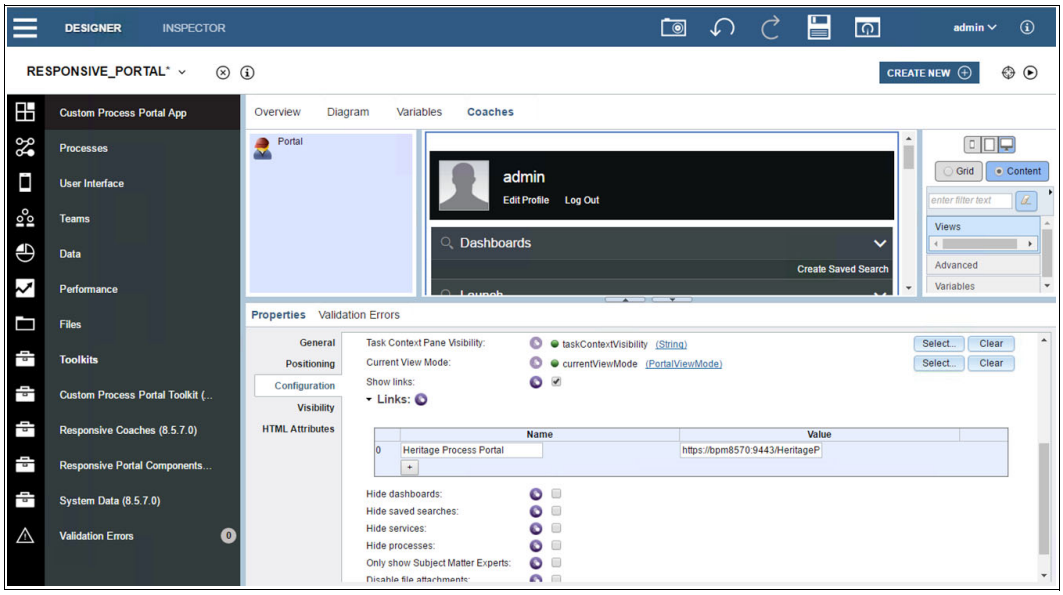


Figure 5-41 Configuration options within the Portal coach view

4. Launch the Custom Process Portal in your web browser. You see the Links section within the context menu that provide access to the Heritage Process Portal, as illustrated in Figure 5-42.

Figure 5-42 Links within the context menu providing access to the Heritage Process Portal

5. Click the **Heritage Process Portal** link to launch the Heritage Process Portal in a new browser window, as illustrated in Figure 5-43 on page 301.

> **Note:** You do not need to provide credentials for the Heritage Process Portal, because you are already signed in to the Custom Process Portal.

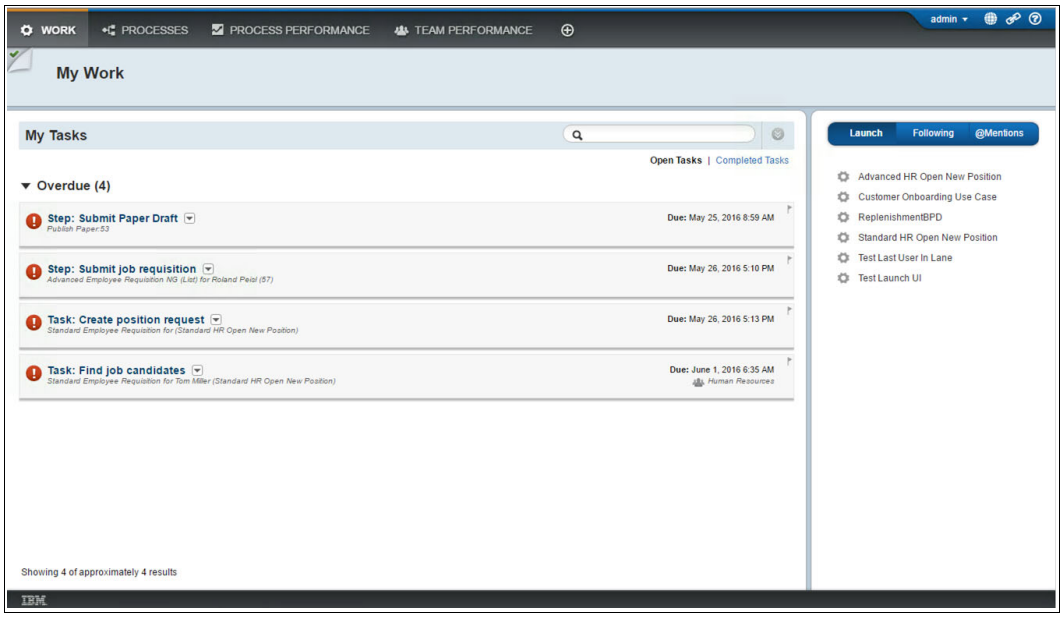


Figure 5-43 Heritage Process Portal

Customize by updating the coach-view implementation

This section describes how to customize the Custom Process Portal Toolkit. The example updates the User Profile coach view to open a dialog box to indicate that the profile displays.

To customize the Custom Process Portal Toolkit:

1. Open the Custom Process Portal Toolkit in Process Designer.
2. Open the User Profile coach view, go to the Behavior tab, and select **load** as illustrated in Figure 5-44.
3. Insert the following line of JavaScript into the load event handler, so that the user is notified when the User Profile coach view is loading:

```
alert("this is the User Profile loading");
```

4. Save the changes and take a new snapshot of the Custom Process Portal Toolkit.

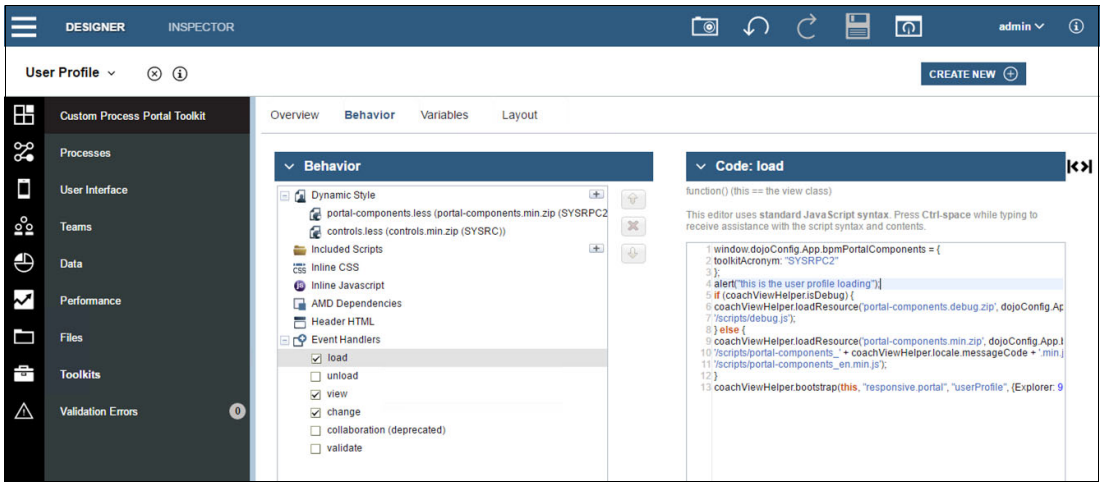


Figure 5-44 Customize the User Profile coach view within the Custom Process Portal Toolkit

5. Open the Custom Process Portal in your browser to test the customization. The dialog box opens as illustrated in Figure 5-45.

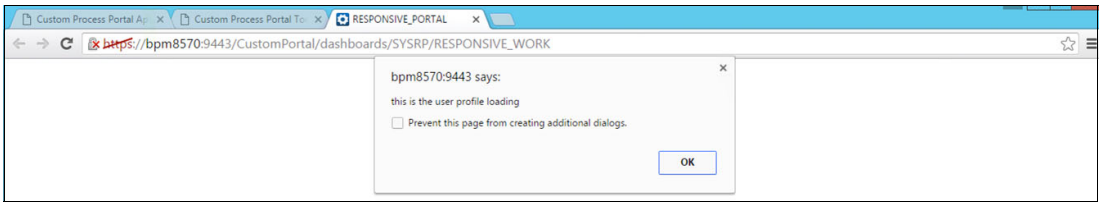


Figure 5-45 Testing the User Profile customization

Modify the JavaScript in the Custom Process Portal Toolkit

The JavaScript that is contained within the managed files in the Process Portal Toolkit is minimized,[3] which makes it difficult to customize. However, you can modify this default behavior by completing these steps:

1. Set up the build environment.
2. Make customizations to toolkit managed files.
3. Update the custom toolkit with the updated managed files.

Note: All custom toolkit references within the process app and toolkit must reference the Custom Process Portal Toolkit to ensure that the custom managed files are used.

Set up the build environment

Create a directory on your local file system to do the portal customization work. This directory can be named anything, but it is assumed to be named `custom-build` for the remainder of this book. Create the following directories:

- `custom-build\min`
- `custom-build\lib\ant-contrib`

Open the Custom Process Portal Toolkit in Process Designer. Click the **Files** link in the left menu bar, and double-click the `portal-components.debug.zip` file, as illustrated in Figure 5-47 on page 303. Click the folder icon to download the file. Decompress the file to your `custom-build` directory. Follow the same procedure to download the `portal-components.min.zip` file and extract it to your `custom-build/min` directory. The directory structure now looks like that shown in Figure 5-46.

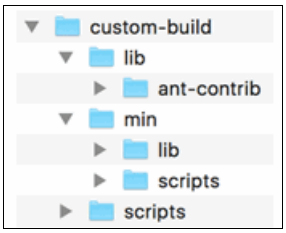


Figure 5-46 Directory structure

Tip: Save the `portal-components.debug.zip` and the `portal-components.min.zip` files to a safe location, because you replace them with modified versions. Having backups of these files is recommended.

[3] *Minification* is the process of removing all unnecessary characters from source code without changing its functionality. For more information, refer to: https://en.wikipedia.org/wiki/Minification_(programming)

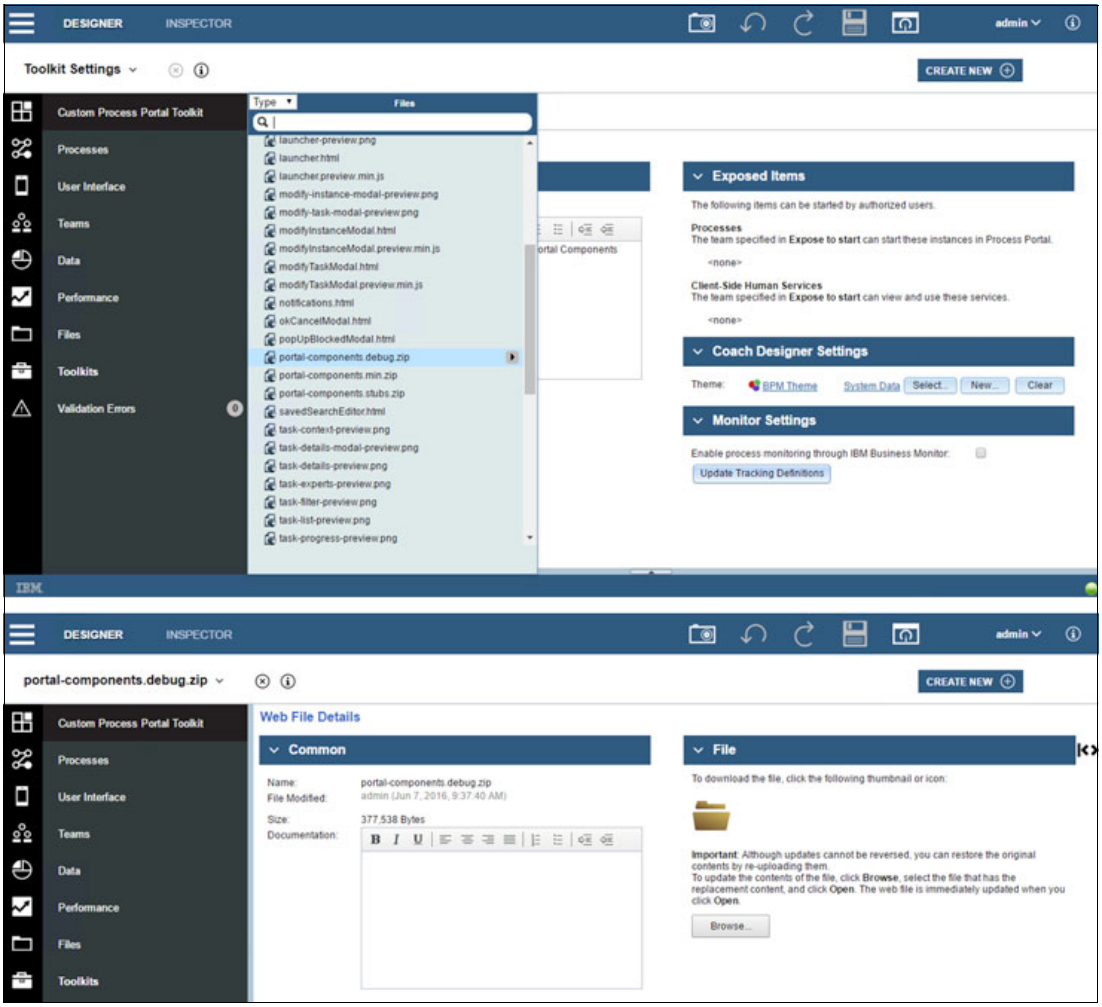


Figure 5-47 Saving Custom Process Portal Toolkit Managed Files

Download the `ant-contrib-1.0b3.jar` file from a trusted site and extract it into the following directory:

`cutom-build/lib/ant-contrib`

Download the `yuicompressor-2.4.8.jar` file from a trusted site and move it (without extracting it) into the following directory:

`custom-build/lib`

Make customizations to the toolkit managed files

This section describes how to customize the toolkit managed files using a simple example.

Make the following update to the `custom-build/scripts/dashboardContext.controller.js` file and save the file:

```
$scope.expireNewMentions = function() {
   var delay, now = new Date().getTime();
   console.log("this is now ", now);
```

Now build the `portal-components.min.zip` file. This file provides JavaScript for Portal at run time. The `dashboardContext.controller.js` file is minimized and built into the compressed file along with other files.

Open a command window and navigate to the `custom-build` directory. If you are doing this work on the same system that BPM is on, you can make use of the `ws_ant` script. Otherwise, you need to download both Java and Apache Ant to build the `portal-components.min.zip` file.

The `custom-build` directory contains the `create-min-zip.xml` build file.

If you have a BPM installation available, run:

```
<path-to-BPM-install-root>\bin\ws_ant -f create-min-zip.xml
```

If you need to use Apache Ant and Java, run:

```
ant -f create-min-zip.xml
```

If the build was successful, you will find the `portal-components.min.zip` file in the `custom-build/dist` directory.

Update the custom toolkit with the updated managed files

Now, you need to upload the new version of the `portal-components.min.zip` file to the Custom Process Portal Toolkit:

1. Open the Custom Process Portal Toolkit in Process Designer.
2. Select the `portal-components.min.zip` file from the Files link.
3. To replace the current version: click **Browse** and locate the `portal-components.min.zip` file that you just created in the `custom-build/dist` directory.
4. Click **Save** to upload the file to Process Designer.

Customize the Login Page

The Process Portal Login Page is the only aspect of the Process Portal 8.5.7 UI that is not implemented using the Coach Framework. However, you can still customize the Login Page by updating the Java implementation within the Custom Process Portal EAR file.

This section assumes that you have already created a Custom Process Portal EAR file, by cloning the reference implementation, as detailed in 5.2.7, “Clone Process Portal” on page 286.

The `login.jsp` file and supported files are located within the Custom Process Portal EAR file:

```
IBM_BPM_ResponsivePortal_SingleCluster.ear\responsivePortal.war\
```

The `login.jsp` file is responsible for displaying the login page and for authenticating the user. An experienced Java UI developer can modify the `login.jsp` file and any associated files (such as the CSS files) within the EAR file to update the styling and layout, add a company logo, and incorporate any additional behavior.

Build a custom process portal

You can use the Custom Process Portal (detailed in 5.2.7, “Clone Process Portal” on page 286) as a baseline to create your own custom portal either by reusing the human services and coach views that have been cloned from the IBM implementation, by creating new human services and coach views of their own, or a combination of both.

Replace the custom portal landing page

The `serviceName` parameter in the `web.xml` file is used to specify the landing page that is displayed after you are authenticated when logging in to Process Portal.

The web.xml file is located within the Custom Process Portal EAR file:

IBM_BPM_ResponsivePortal_SingleCluster.ear\responsivePortal.war\WEB-INF\web.xml

To replace the custom portal landing pages, for example, update the web.xml file as follows to use your own CUSTOM _PORTAL human service for the landing page:

```
<init-param>
   <param-name>serviceName</param-name>
   <param-value>CUSTOM_PORTAL</param-value>
</init-param>
```

Tip: Within your Custom Process Portal App, you can create a duplicate of the RESPONSIVE_PORTAL human service as a baseline for your CUSTOM_PORTAL human service. Remove the Portal coach view from the coach, and replace it with your own content.

5.2.9 SPARK Portal Builder toolkit

In some scenarios, customizing Process Portal might not be sufficient: you might need a custom designed portal that is built completely from scratch that still takes advantage of the Coach Framework. The SPARK Portal Builder toolkit provides configurable, ready-to-use coach views for creating custom portals using the Coach Framework and the SPARK programming model.

In these types of scenarios, keep in mind the following points of interaction for users:

- Display of a list of tasks, instances, or services.
- Display the UI for the selected item in the list.

The SPARK Portal Builder toolkit includes a set of coach views for each of these types of interaction points, with different variants, that are designed specifically for tasks, instances, and services, as shown in Figure 5-48 on page 306.

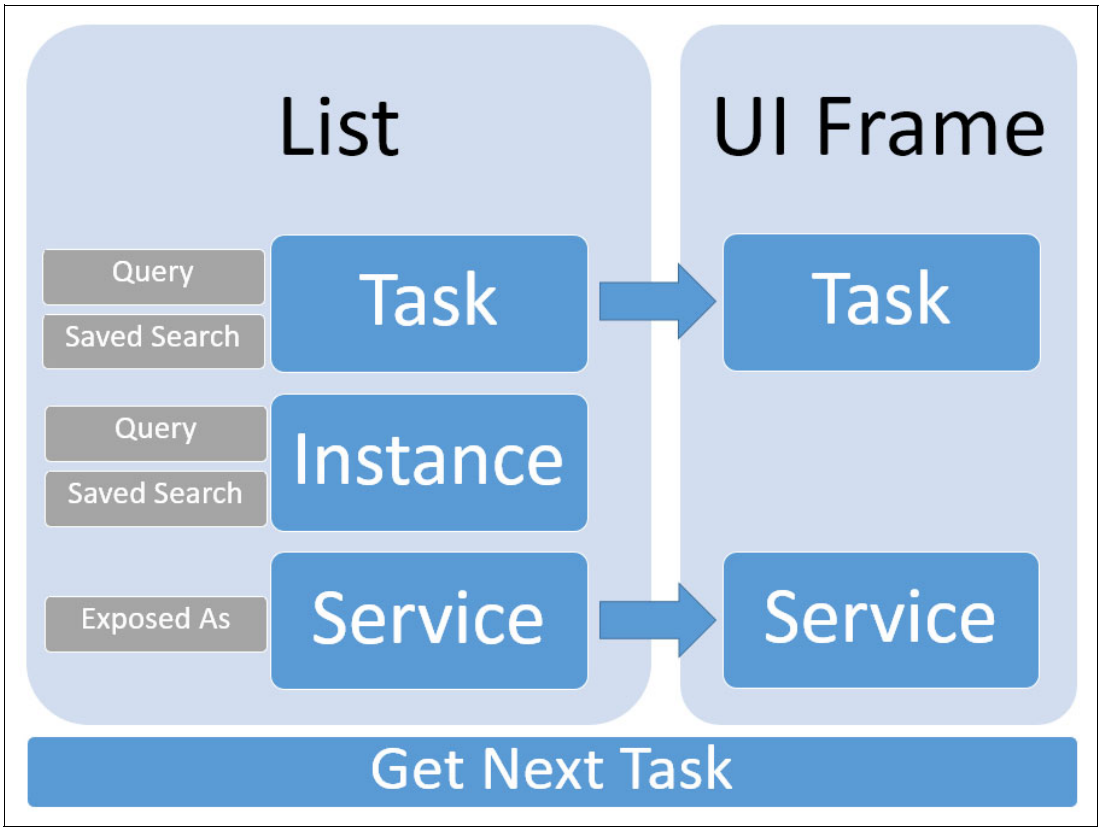


Figure 5-48 SPARK Portal Builder components

You can find a full description of the SPARK Portal Builder toolkit at:

http://ibm.biz/BPMSparkPortalBuilder

SPARK Portal Builder lists

Table 5-2 shows the list controls in the SPARK Portal Builder toolkit.

Table 5-2 Spark Portal Builder list controls

Control name	Description
Task List	Filters and sorts a list of tasks through configuration of search conditions
Task Search List	Filters and sorts a list of tasks by specifying an existing Saved Search

When you configure a list on a coach, the results display in a table to select a specific result, as shown in Figure 5-49.

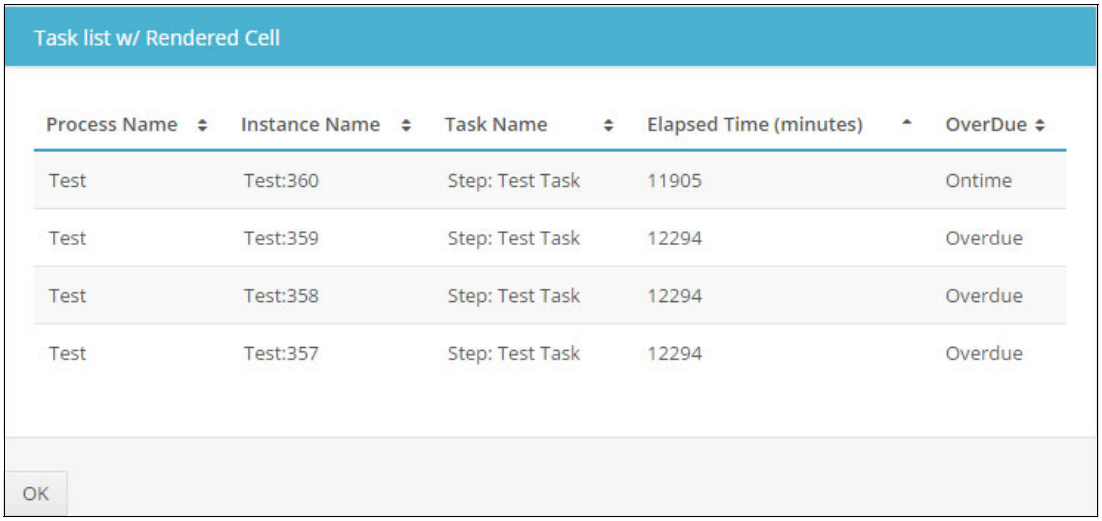


Figure 5-49 Task list

Because the controls in the toolkit are based on the same framework as the SPARK UI toolkit described in Chapter 4, “SPARK UI Toolkit” on page 153, developers can specify the action that occurs when an item is clicked through the Item Clicked event configuration option.

SPARK Portal Builder UI frames

In the case of a Service or Task based list, the selection of an item normally requires the item be displayed on the same coach. The SPARK Portal Builder UI Frame allows you to complete process tasks or run exposed human service implementations within a coach, as depicted in Figure 5-50 on page 308. The content in the UI frame comes from the human service implemented for the activity in the process.

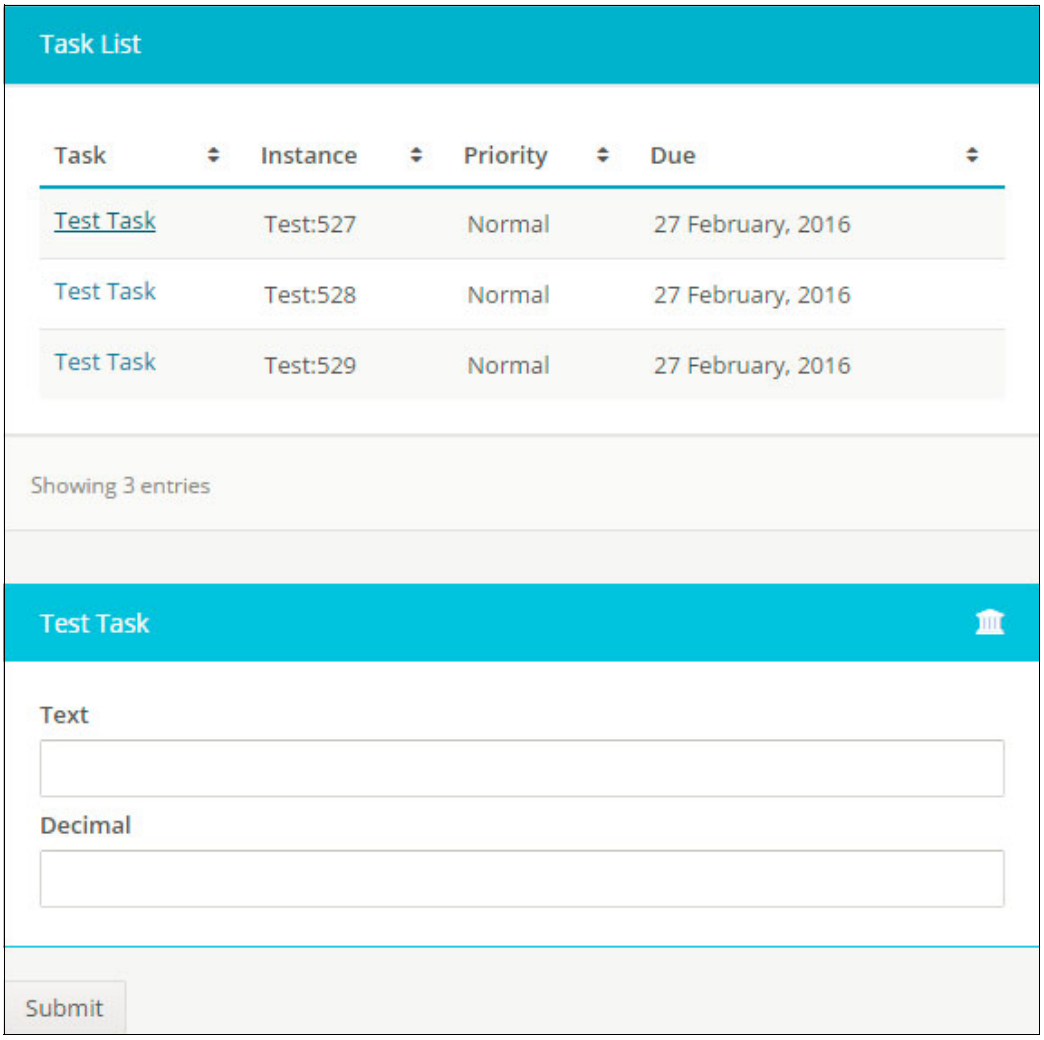


Figure 5-50 Example task list with a UI frame

Get Next task

There are some use cases that require work tasks to be *pushed* to users one at a time, instead of being *pulled* by the user from a set of assigned tasks. In these cases, you need a programmatic way to identify the set of tasks assigned to a user (or a user's team), select the task that needs to be pushed to the user and render the selected task on the page.

You can use the SPARK Portal Builder Get Next task control to identify and select a task for the user currently logged in. The logic used to select the task can be customized through an Ajax service.

When you combine the capability of the Get Next task control with a Task UI frame control, you can achieve the push model for work assignments as shown in Figure 5-51 on page 309, an example implementation using a SPARK Button, Get Next task, and Task UI coach view to achieve a work assignment push pattern. The event handler configuration of the Get Next Task control is also shown in Figure 5-51 on page 309.

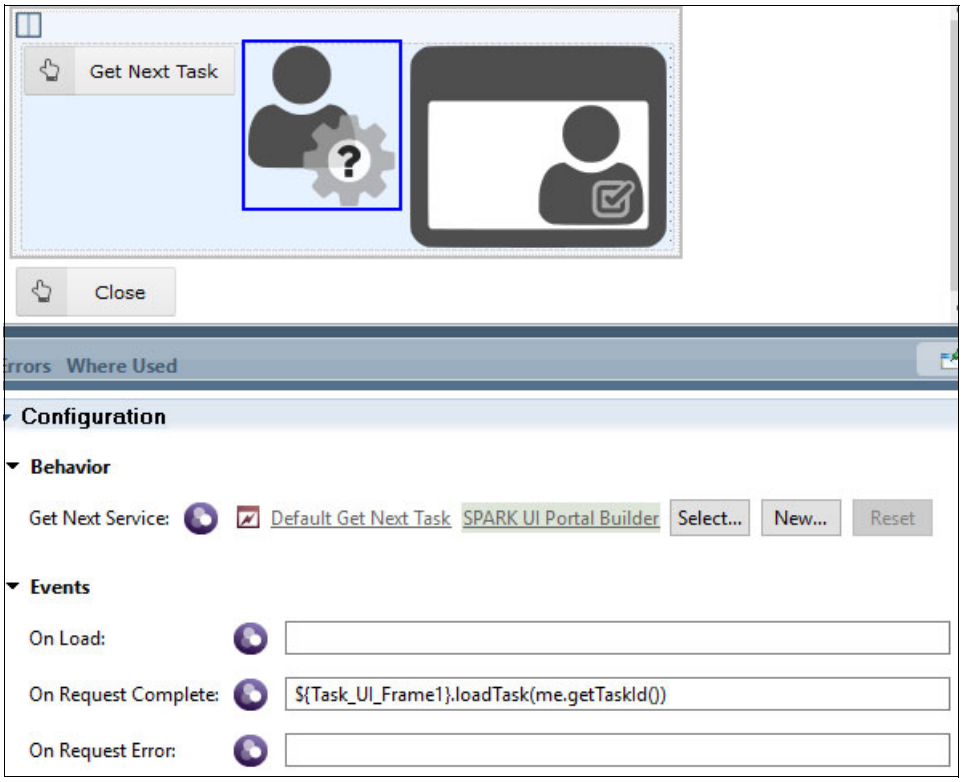


Figure 5-51 Example using a SPARK Button, Get Next task, and Task UI coach view

To invoke the selection service configured for the Get Next task coach view shown in Figure 5-52, the *requestNextTask* method must be invoked when the button is clicked, as show in Figure 5-52. The event handler configuration of a button that is invoking the *requestNextTask* method of a Get Next task control.

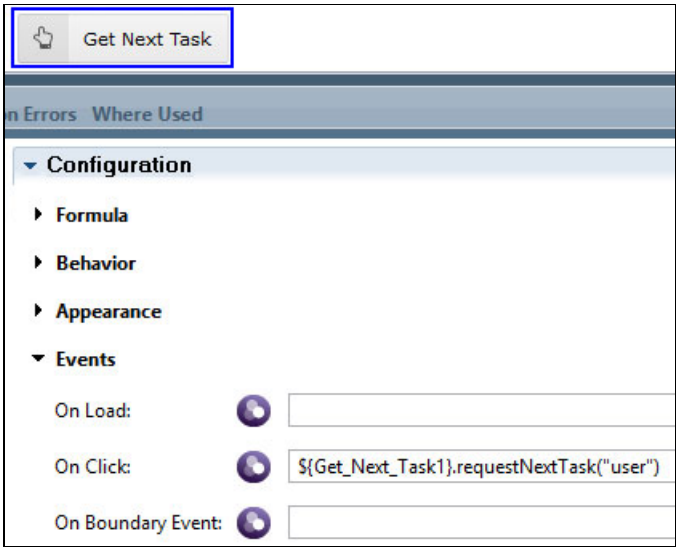


Figure 5-52 Button that invokes the requestNextTask method of a Get Next task control

Customizing the style of your portal

All of the coach views of the SPARK Portal Builder are based upon the same framework as the SPARK UI toolkit. You can style SPARK Portal Builder coach views in the same way you

customize SPARK UIs—through configuration options for each control. You can further customize the style by using the SPARK Style control to specify CSS overrides or by embedding your own CSS within a wrapping coach view or page.

5.3 Securely accessing IBM Process Portal from a mobile device

Deploy IBM Process Sever securely behind the corporate firewall to prevent malicious attacks on mission critical processes. Process Sever is responsible for process orchestration, integrates back-end systems, and delivers coach-based UIs to your web browser. When Process Portal, tasks, and dashboards are accessed from a desktop computer, you can rely on the standard corporate security as the desktop computer that is deployed safely behind the corporate firewall. However, when accessed from a mobile device or a desktop computer outside the office, additional security is required.

For an on-premise deployment of IBM BPM, use a secure virtual private network (VPN) connection. Some organizations also use a secure internet browser[4] for added security.

When using BPM on Cloud, no additional security considerations are required when exposing Process Portal to the Internet.

If the use of a VPN or BPM on Cloud are not acceptable, an alternative portal implementation might be required. For more information, refer to Chapter 6, “Combining the Coach Framework with other approaches” on page 313.

5.4 Process Federation Server

IBM Process Portal 8.5.7 can be used with either a single Process Server or in a federated topology using Process Federation Server. Process Federation Server enables the IT Administrator to manage the BPM infrastructure without significantly impacting the business user experience.

Note: Users might experience differences with Process Portal behavior when moving to a federated topology.

Process Federation Server is used to:

- Isolate processes
- Manage departmental deployments
- Massively scale the IBM BPM infrastructure
- Manage IBM BPM migrations
- Combine different task implementations such as, Business Process Execution Language (BPEL) and BPMN

Process Portal is deployed on a Process Server 8.5.7 instance whether connected to a Process Server or Process Federation Server. However, Process Federation Server can be

[4] See the IBM BPM System Requirements support site for a list of supported browsers:
http://ibm.biz/BPMAdvSysReqs
http://ibm.biz/BPMStdSysReqs
http://ibm.biz/BPMExpSysReqs

connected to multiple Process Servers (or deployment environments) using different versions of IBM BPM back to BPM 8.0.1.3, as illustrated in Figure 5-53.

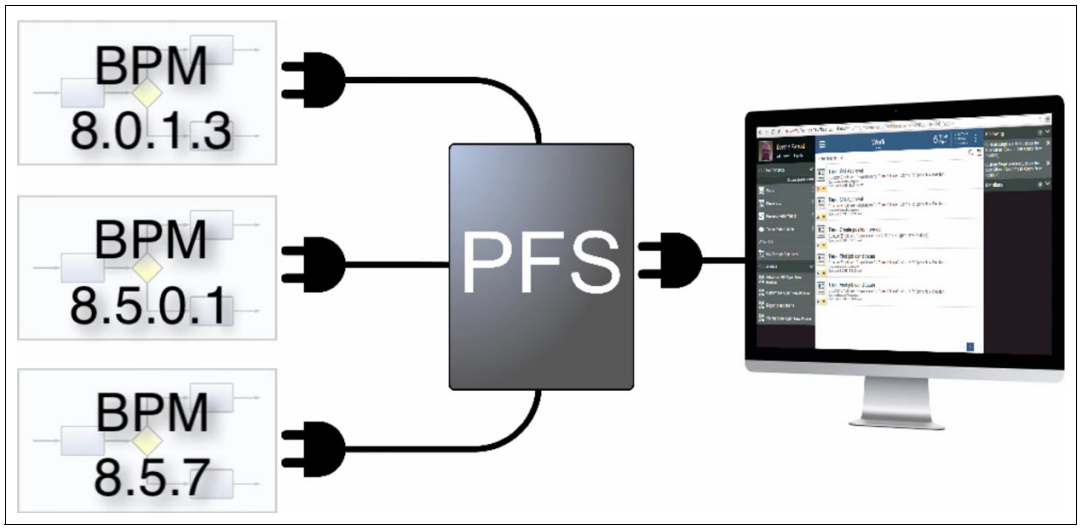


Figure 5-53 Process Federation Server connecting IBM Process Portal to multiple process servers

Process Federation Server federates the task list, the process launch list, the dashboard launch list, and saved searches from each of the connected Process Servers to provide access to a unified view through REST APIs.

Note: Team-based Saved Search, Follows, Mentions, and Performance Dashboards are not supported by the PFS in IBM BPM 8.5.7 at the time of writing this book.

Process Federation Server and Process Server APIs used by IBM Process Portal 8.5.7 are identical, so Process Portal is simply configured to connect to Process Federation Server when used in a federated topology.

Customers that have decided to develop their own task management user interface instead of using Process Portal can also leverage Process Federation Server capabilities using Process Federation Server REST APIs directly.

5.4.1 Process Federation Server Deployment

Process Federation Server is included within the IBM BPM 8.5.7 license. It is installed using the WebSphere Liberty profile and can either be deployed on its own physical host or virtual machine (VM), or share the same host/VM as one of the IBM Process Server 8.5.7 instances.

Similarly, IBM Process Portal 8.5.7 needs to be hosted by an IBM Process Server 8.5.7 instance. Thus, Process Portal can be deployed either on a dedicated Process Server or can reuse a Process Server that is being used for process execution.

5.4.2 More on Process Federation Server

For further information about Process Federation Server, refer to:

- Getting Started with Process Federation Server in IBM BPM 8.5.7 (video)

  http://ibm.biz/IBM_PFS_857_Video

- Process Federation Server in IBM Knowledge Center

  http://ibm.biz/IBM_PFS_857
- Interconnect 2016 BTB-2890 presentation, *Understanding the IBM BPM Process Federation Server*

  http://ibm.biz/IBM_PFS_InterConnect2016_BTB-2890

5.5 Conclusion

IBM Process Portal is an integral part of the IBM BPM UI. You can use it to perform work, to interact with BPM human tasks, and to gain visibility on process and team performance. This chapter provided information about Process Portal features from the business user perspective. It also explained how to personalize, extend, or customize Process Portal by using the Coach Framework. Finally, it discuss how Process Portal plays an important role in federating Process Servers as the business UI on the Process Federation Server.

The next chapter explores how to use alternative technologies and techniques to create BPM UIs either that are combined with coaches or that are completely independent.

6

Combining the Coach Framework with other approaches

IBM Business Process Manager (BPM) coaches and coach-based human services are not the only approach available to create user interfaces (UIs) to interact with BPM processes. An enterprise can have numerous other reasons to use alternative approaches to design and deliver BPM UIs, and each approach comes with consequences on the application's design as well as its security and performance.

This chapter defines design patterns to deliver BPM UIs, along with architectural and security considerations deriving from mixing the Coach Framework with alternative approaches. It includes the following topics:

- Decoupling the BPM UI: "Headless" BPM
- BPM UI design patterns
- Exposed pattern
- Embedded pattern
- Integrated pattern
- Decoupled pattern
- Independent pattern
- Hybrid pattern
- Combining and extending the BPM UI design patterns
- Conclusion

6.1 Decoupling the BPM UI: “Headless” BPM

This section describes the role of BPM in a service-oriented architecture (SOA) and the architectural implications of decoupling the BPM UI: generally referred to as *headless* BPM. This section is largely based on an IBM developerWorks® article, *The placement of BPM runtime components in an SOA,* which is available at:

http://ibm.biz/BPM_SOA

SOA refers at least in part to the exposure of re-usable interfaces from internal systems of record and operational systems. Although many enterprises are now using Representational State Transfer (REST) application programming interfaces (API) to expose those internal services rather than the SOAP-based web services of early SOA, many of the core principles remain the same.

For simplicity, this chapter refers to SOA and services; however, the same principles equally apply where an internal API layer is used. You can find a more detailed discussion about the differences between SOA and API initiatives in the developerWorks article, *Integration architecture: Comparing web APIs with service-oriented architecture and enterprise application integratio*n, which is available at:

http://ibm.biz/APIandSOA

6.1.1 How are business processes represented in the SOA reference architecture?

A commonly recognized reference architecture for SOA is The Open Group[1] layered model shown in Figure 6-1.

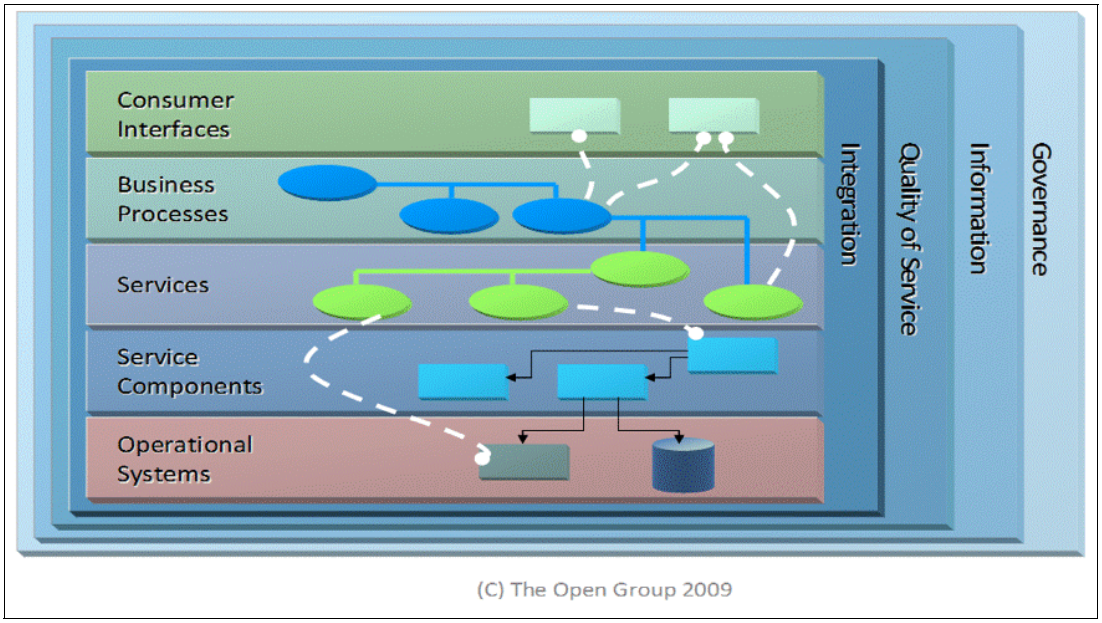


Figure 6-1 The Open Group's SOA Reference Architecture

In this layered model, *business processes* are shown in a dedicated layer to articulate their primary purpose, which is to orchestrate services that are exposed in the layer below the business processes.

[1] The Open Group SOA Reference Architecture, http://ibm.biz/SOA_Ref_Arch

The layered architecture is a logical model, not a map of the physical deployment of components. A component's position within the layers depends on the role it is playing, but some (indeed many) components perform more than one role. This approach means that they might in reality sit in multiple layers at the same time.

An example is given in Figure 6-2, for the operational systems along the base of the diagram in Figure 6-2. These operational systems are placed in the bottom layer because they are, from a SOA point of view, "providers" of function and data. Requests are made to them to retrieve or change data they hold, or to perform calculations that only they know about. However, in a modern complex environment, many of those operational systems are not completely autonomous, they also require data from other systems. In a governed SOA, requests between systems are done via calls to re-usable interfaces made available in the services layer.

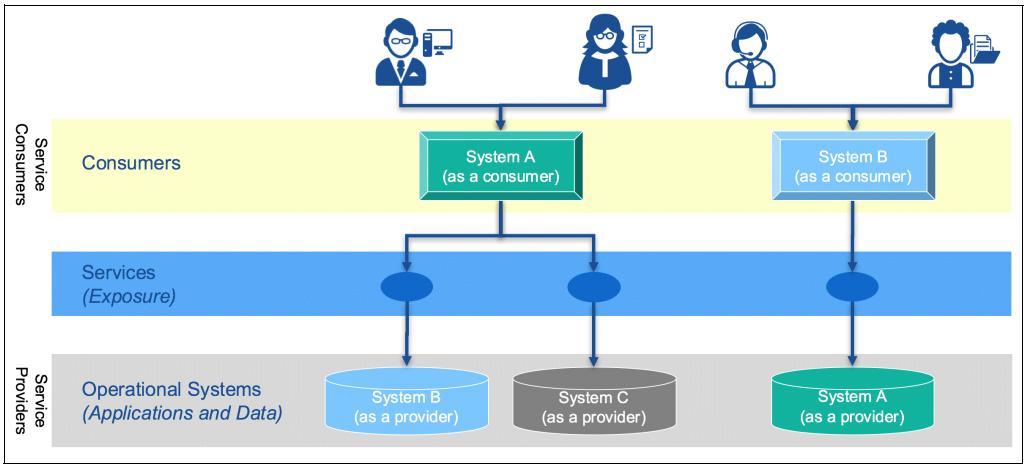


Figure 6-2 Systems acting as both providers and consumers simultaneously

When making these requests to other systems, suddenly the operational system is, in fact, acting in a different role. It becomes a *consumer*, and when performing that role, it is in the top layer of the diagram, as shown in Figure 6-2. Thus, an operational system can actually appear in two different layers of the architecture, depending on its role in the interaction.

When the system of interest is BPM, the BPM component can be broken up into its constituent parts:

- *Process run time*: A logical representation of the runtime component that holds the running processes, performing the orchestration specified in their process definitions.
- *Process*: A given process being managed by the run time.
- *APIs*: Interfaces that provide access to information about the running processes such as, pulling back a list of tasks within those processes that are currently waiting to be processed, and enabling actions to be performed on the processes, such as claiming a task, updating its data, and completing it.
- *BPM UI*: A BPM specific UI that enables users to view the process data, such as viewing task lists, and perform actions, such as completing tasks.

Figure 6-3 shows how the BPM process run time is an example of a component that can be both a consumer and a provider of services.

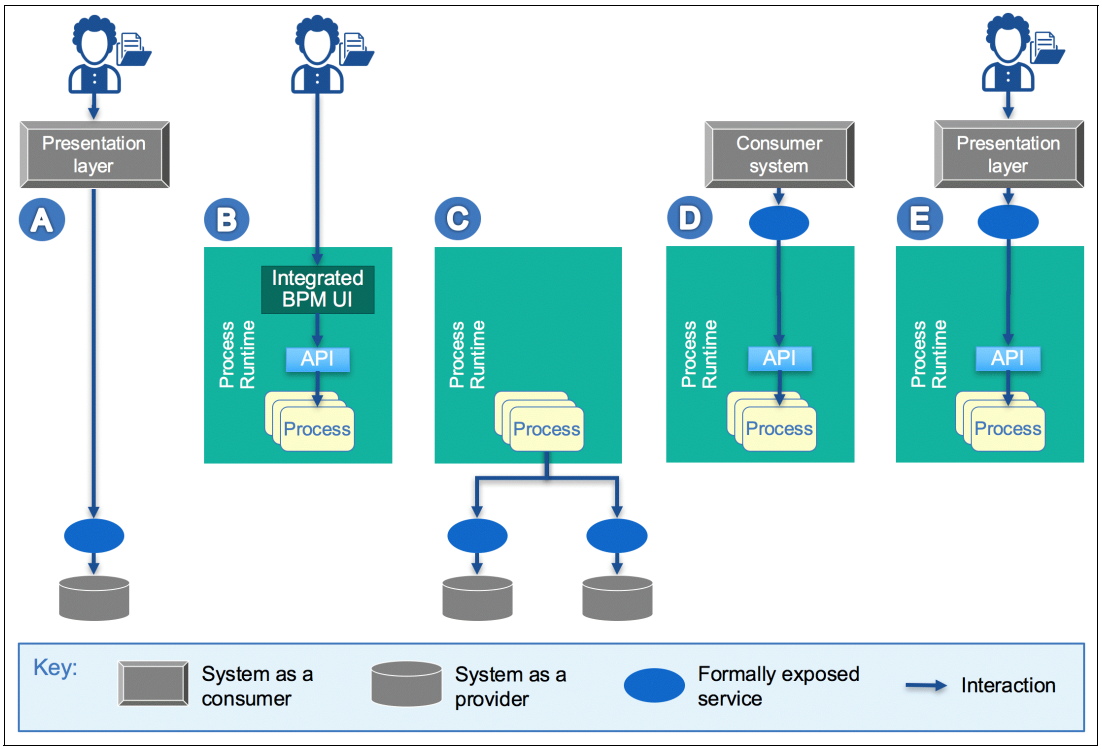


Figure 6-3 Logical constituents of a BPM component

Table 6-1 explains the scenarios shown in Figure 6-3.

Table 6-1 Roles of the BPM process run time

	Description	Role of the BPM process run time
Scenario A	**Typical consumer provider relationship:** The UI calls a formally exposed service, without involving the BPM process run time.	Neither consumer nor provider
Scenario B	**A self contained workflow:** No *services* are involved. Users work directly with internally provided BPM UIs.	Neither consumer nor provider
Scenario C	**"Headless" orchestration:** The process run time acts as a *consumer*, orchestrating services.	Consumer of services
Scenario D	**Straight-through process services:** The process run time acts as a *provider* of process and task related services, not requiring any user interaction.	Provider of services
Scenario E	**Process services:** The process run time acts as a *provider* of process and task related services, consumed by external UIs.	Provider of services

Note: The *headless* orchestration described here refers to BPM processes that do not require interaction with users. Alternatively, *headless* BPM is generically used to refer to BPM processes in which user interactions are required, but the BPM UI has been decoupled from those processes.The two terms are therefore different.

In any reasonably sized solution, BPM can play most, if not all, of these roles at the same time.

6.1.2 Business process orchestration and BPM UI

A business process is most easily recognized in a SOA by its role as an orchestrator of services, as in *scenario C* in Table 6-1 on page 316. A BPM process engine can take on some of the tasks in the process that otherwise need to be performed by people, and automate them by calling services in the service layer, improving speed, efficiency, and consistency of the process and reducing the number of human interaction points. In this role, BPM is clearly a *consumer* of services, as shown in the business process layer in Figure 6-4.

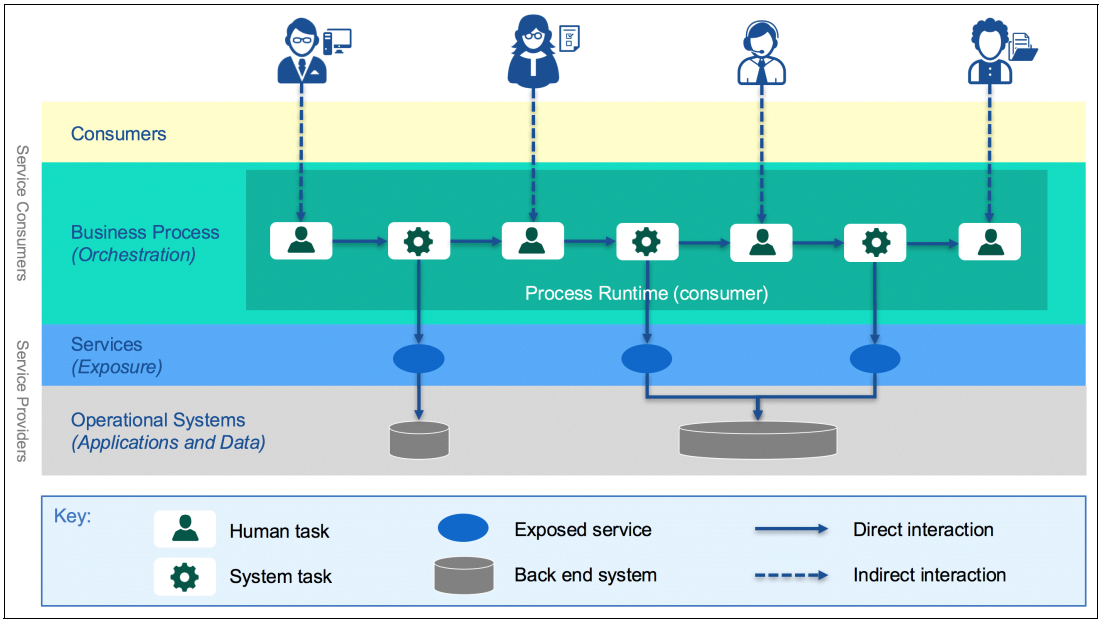


Figure 6-4 BPM as a consumer of Services

However, as with the operational systems in Figure 6-2 on page 315, this is not the only role that the business process component plays within the architecture.

The dotted lines in Figure 6-4 represent the fact that there is an over-simplification in the diagram that needs to be understood in more detail. Do the users really interact directly with the business process?

As a process progresses from step to step, the process engine assigns the next task in line to a specific group of users. Users need to be able to access a list of these tasks and work on them. This UI is often known as a *task portal*. The *task portal* enables users to view the current list of tasks assigned to someone via the groups they belong to. It also enables them to claim a task, and to work on it using screens that provide just the data needed by the user (data retrieved by the process run time from different operational systems) and asking for only the responses required by them.

It is this ability to provide a user with a context specific piece of work that is a key strength of BPM. Users do not need to understand how to use the underlying systems. Users need far less training to understand how and when to do tasks within a process. This approach also means that adjustments to the process can be more readily accepted because minimal re-training is required.

IBM BPM provides a ready to use component within the BPM system, which is represented in Figure 6-5 as the *Integrated BPM UI*, which includes functionality for both the *task portal* and the context-specific screens (that is, Coaches and coach-based human services):

- Task portal: The IBM Process Portal is described in Figure 6-5. It includes a simple, business-friendly inbox for effective task management and task completion with zero development effort.
- Context-specific screens: The coaches described in Chapter 2, “Creating user interfaces with coaches” on page 9. Coaches make it easy to adapt UIs when a business process changes, because UIs created with the Coach Framework are an integral part of the process development lifecycle. This method ensures that users are always presented with screens that are in sync with the correct process logic.

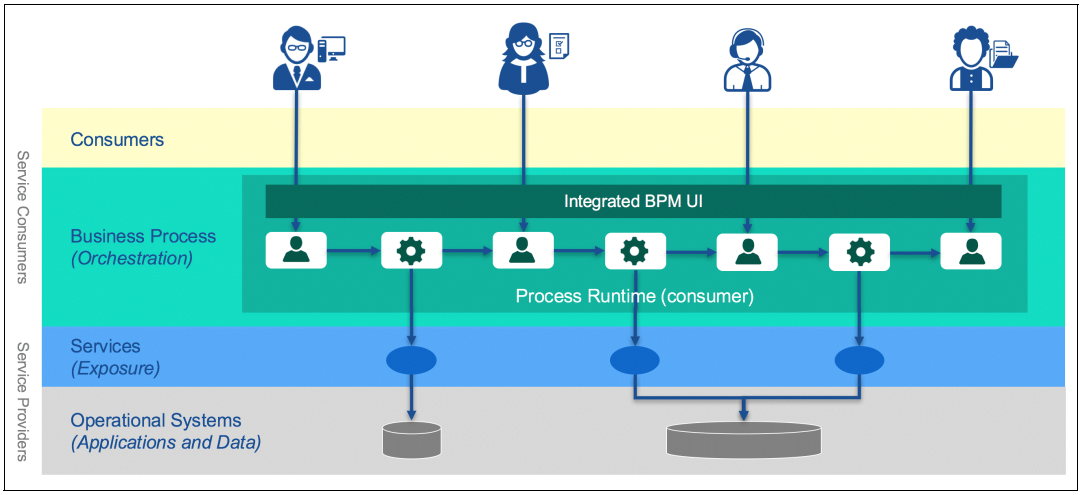


Figure 6-5 Positioning of the integrated BPM UI

When using IBM BPM, the BPM UI is fundamentally part of the same run time component that orchestrates the tasks and services, so it is able talk directly to the process engine to query what process and tasks are available, get task and process data, and complete tasks.

For many situations, this internal BPM UI provides an excellent approach for getting processes up and running quickly. Additionally, during the development lifecycle, it provides a simple way to *playback* the current understanding of the process to business sponsors. This ability to rapidly prototype a process using the internal portal and iteratively improve it via close communication with the business users is another key benefit of BPM.

6.1.3 Accessing tasks externally

There are use cases for which the internal BPM UI might not suffice such as:

- In recent times, the pervasive use of mobile devices has made the need of exposing tasks externally more pressing. Internal and external consumers expect to be able to access business processes either through direct mobile access or through mobile applications (*mobile apps)* often over the public Internet.

- Organizations might dictate a common *look and feel* across every function and application considered customer-facing. Those services that can be consumed or seen directly by their customers and not limited to BPM.
- Organizations might have a desire to embed task lists within other systems, such that the user of those systems does not even realize they are working with tasks in the BPM engine.
- In large enterprises, there are often important architectural standards in place that relate to each of the layers in the SOA reference architecture. The enterprise architecture can enforce, for example, that UIs must be built in a specific way, perhaps using a specific technology (such as node.js, AngularJS) or platform (such as IBM Bluemix®[2]).

In addition to the use cases described, a specific architectural standard that plays a paramount role in the BPM UI choice is security. When wanting to expose BPM UIs outside of the enterprise to the public Internet, integrated BPM UIs might pose several security risks that are intrinsically bound to the nature of their integrated implementation, such as:

- Denial of service attacks for Internet facing UIs might impact the entire internal processing done in BPM.
- BPM APIs are usually intended to expose the complete set of functionality provided by the Process Engine. As such, they are not safe to be exposed externally without a *filter*.
- Application and presentation tiers are not fully decoupled, causing additional risks, such as no full control over HTTP requests and an uncontrollable attack surface.[3]
- There is an increased vulnerability to cross-site scripting (XSS)[4] or to cross-site request forgery (CSRF)[5] attacks.
- Issues arise with coarse authorization granularity. Any process participant can potentially use APIs to introspect information around processes, even if not directly exposed to them, for example:
  - Information around duration of other tasks or the overall process
  - Query process attributes to which they should not have access

Purely from a security perspective, it is usually not recommended to expose integrated BPM UIs directly over the Internet. This recommendation, with particular emphasis on the word *directly*, holds also for IBM BPM. For more information, refer to the following resource:

`http://ibm.biz/InternetFacingBPMUI`

[2] For more information about IBM BlueMix, see: `http://www.ibm.com/Bluemix`
[3] For more information about attack surface, see: `http://ibm.biz/AttackSurface`
[4] For more information about cross-site scripting, see: `http://ibm.biz/OWASP_XSS`
[5] For more information about cross-site request forgery, see: `http://ibm.biz/OWASP_CSRF`

It is often unreasonable, in the use cases mentioned previously, to expect the internal IBM BPM UI to be a fit for all purposes and to comply with all standards. It is common for organizations to need to build an independent UI to surface tasks and task lists to external users, as depicted in Figure 6-6.

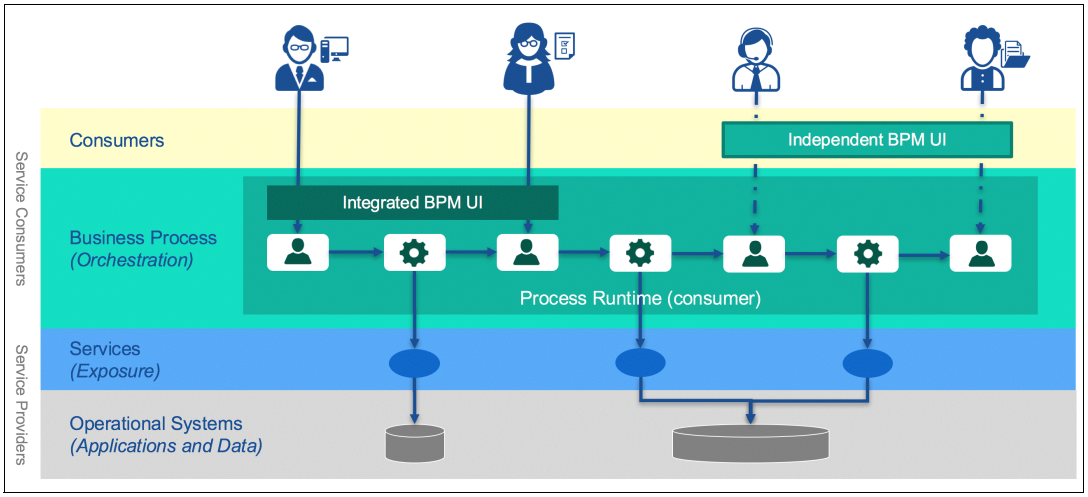


Figure 6-6 Introducing an independent BPM UI

The dotted line in Figure 6-6 is an over-simplification of the interaction between external UI and process run time.

Most BPM systems, including IBM BPM, cater to the common need to create independent BPM UIs by providing a set of APIs that independent UI applications can call to retrieve and update process and task information. The APIs are typically provided over common protocols such as, Hypertext Transfer Protocol (HTTP), SOAP, or REST, and also over messaging transports such as, Java Message Service (JMS).

From a logical point of view, these APIs on the BPM engine should be seen as *task services* and *process services*. In a common scenario in which those services are highly re-usable capabilities across the organization, they should therefore be treated as first class citizens of the SOA, and exposed formally via the service layer.

From a logical viewpoint, the process engine is therefore an orchestrator of services, and thereby a *consumer*. It is also a *provider* of services as a component in the operational systems layer as shown in Figure 6-7.

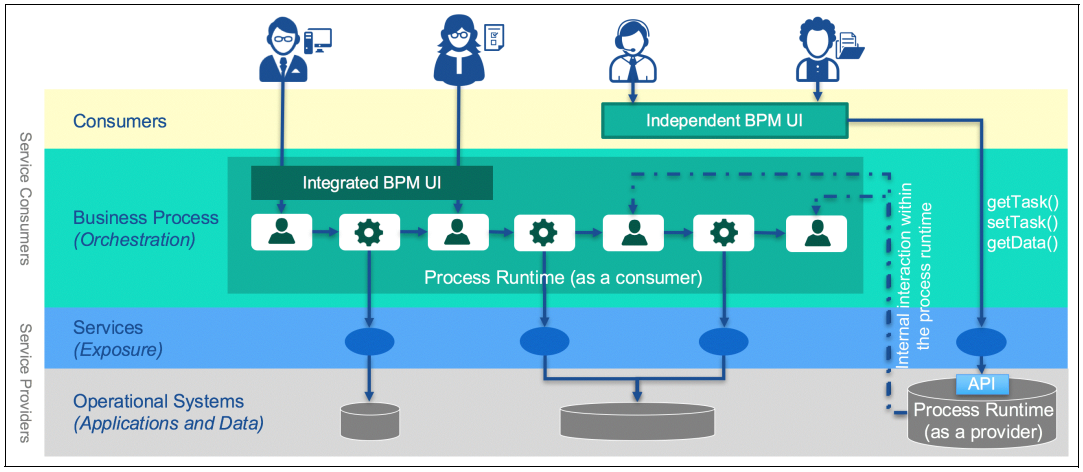


Figure 6-7 The process run time as a provider of services

The business process engine component provides a set of services for viewing and manipulating tasks and processes and these services are highly re-usable. So they are formally exposed through the service layer.

6.1.4 Exposing a process via a business service

There is one further type of interaction with the business process to consider. The services exposed for re-use described thus far have been essentially generic APIs, provided by the process engine for directly administering task and processes. They are flexible APIs enabling you to perform almost any action required to a task or process. However, they have a number of disadvantages:

- To use them effectively, a designer or developer must intimately understand the concepts of tasks and processes and know how to navigate them.
- The granularity of these APIs is often quite low level, and it might take multiple requests to achieve an apparently simple objective such as, completing a task.
- The business data retrieved from a task or required to update it, is not known to this generic API. Therefore, it is not possible for a consumer of the service to inspect the schema of the API in order to understand how to populate it or retrieve data from it. Separate documentation or schemas have to be prepared to explain the specific use of the API for each task in each process.

There are circumstances where the enterprise needs to enable consumers to perform actions on a process in a more business oriented way, without having to understand the intimate details of the process engine. In these circumstances, the user is typically unaware of the existence of a business process and its tasks, and simply needs to interact with an application that implements the required business functionality.

You can accomplish this task by providing a business specific service, which is then mapped to the appropriate lower level requests on the generic API.

The following common usage scenarios provide a process that can be exposed as a *business service*, as depicted in Figure 6-8 on page 323:

- Isolated business service

  Many business processes begin with an initial data capture phase, where a significant amount of data is collected, often from the actual customer via an online web site or a mobile app. Examples include applications for a quotation for an insurance product, applying for a bank loan, or populating an online shopping cart before submitting it to be processed.

  The UI from an internal BPM UI is typically not the appropriate mechanism for this data capture, as discussed in 6.1.3, “Accessing tasks externally” on page 318. Often, you might not even know that you are taking part in a process. So a dedicated independent UI, or even a device application, is likely used to collect and ultimately submit this data. The data submission services often are highly re-usable, and creators of these UIs or the applications that consume them have no need to understand the intricacies of the process engine. They also benefit from an interface schema that clearly defines the business data required so they know what they need to populate for a valid request.

- Correlated business service

  Midway through a process, there is a need to pause waiting for an event to happen. Typical examples are waiting for the receipt of further details from a customer, or the arrival of a manual quotation from a supplier.

  As in the previous use case, these events are likely to be captured by separate applications or UIs external to the process engine. The writers of those applications might only know that they must publish an event but not who will be listening for it. It is likely that they do not even know that they are delivering it to a process at all. They certainly do not want to understand which task in which process they should be passing the event to. In this situation, a business service is exposed by the service layer to be called when the event happens. A service component then translates that into an event for the BPM component. The subtlety here is that the event must correlate with the correct running process instance. IBM BPM has a mature correlation capability for incoming events exactly for this reason.

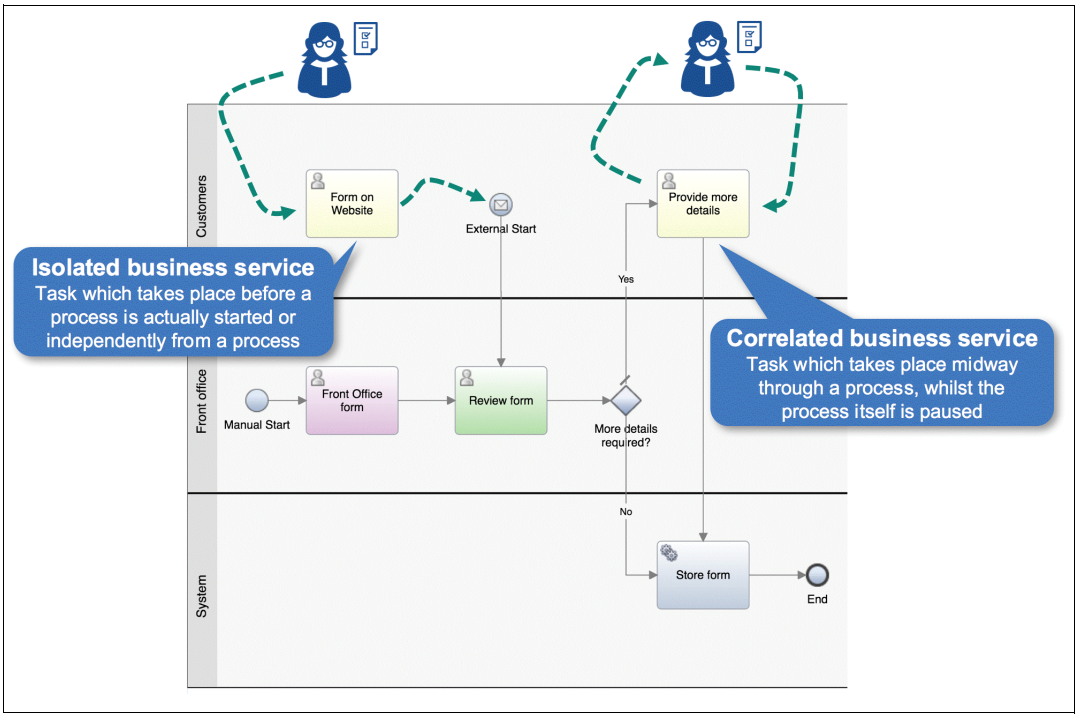


Figure 6-8 Processes and tasks exposed as business services

In both cases, a business specific service is required that specifies the valid business data model. This data model is then transformed into the relevant requests to the lower level generic API on the process engine.

This approach has advantages for the consumer of the service in terms of simplicity. The consumer is completely and functionally decoupled from the way in which the service is implemented, and is not even aware that the service is provided by a process engine. On the other hand, this results in extra work defining the business service for every process that needs to be triggered or for every event that meets with a process rather than using a generic API. In essence, the more re-usable the business service is, the more beneficial it is to use this technique.

6.1.5 IBM BPM REST APIs

As of version 7.5, IBM BPM provides a set of APIs that are implemented using RESTful services. A set of business process definition (BPD), Process and Human Service related REST resources are provided for accessing business process, human task and business category data. These REST APIs enable developers to build, for example, external BPM UIs such as, task implementations and task portals.

The IBM BPM REST APIs are usually *chained* to perform complex tasks. The result of a REST API call is used as an input parameter of a subsequent call, and potentially other ones, in a *chain* of API calls. This chain of API calls results in the required composite action.

It is worth noting that chaining REST API calls is a stateful action, meaning that state needs to be maintained across the entire chain. For this state to happen, session affinity and security context between the consumer application and IBM BPM must be kept during the entire sequence of API requests. This situation is not uncommon when implementing external UI applications calling *stateful* REST APIs. In addition, a number of patterns address this

topic (for example JSESSIONID cookies and LTPA tokens), although the topic is not covered in depth in this book.

You can find the full list of public IBM BPM REST APIs in IBM Knowledge Center:

http://ibm.biz/BPM_REST_API

In addition, you can find a brief of overview about how to use the APIs in the developerWorks article, *Using the REST APIs in IBM Business Process Manager V7.5*, which available at:

http://ibm.biz/Use_BPM_REST_API

IBM BPM REST API tester

IBM BPM also provides an interface for testing the exposed REST APIs, called REST API tester, or REST API test tool: a web application that can be used to prototype the following IBM BPM REST resources and their associated parameters:

- BPM resources for processes, BPD and tasks
- Federated BPM resources
- Business Process Execution Language (BPEL) related resources

The tool is installed on each Process Center and Process Server node as part of the IBM BPM installation and can be found at the following web address:

https://<host_name>:<port_number>/bpmrest-ui

Where *host_name* is the network name for the host of the process server, and *port_number* is the port number used by the REST API test tool. The port number depends on your system configuration. Default port numbers are *9080* (HTTP) and *9443* (HTTPS).

For example, to test the REST APIs available on port 9443 on server1, the URL to be used would be:

```
https://server1:9443/bpmrest-ui
```

To use the REST API test tool, it is sufficient to:

1. Select an API.
2. Optionally enter any associated parameters for the selected API.
3. Click **Execute Call** to start the test.

This procedure is described in Figure 6-9 on page 325 and Figure 6-10 on page 325.

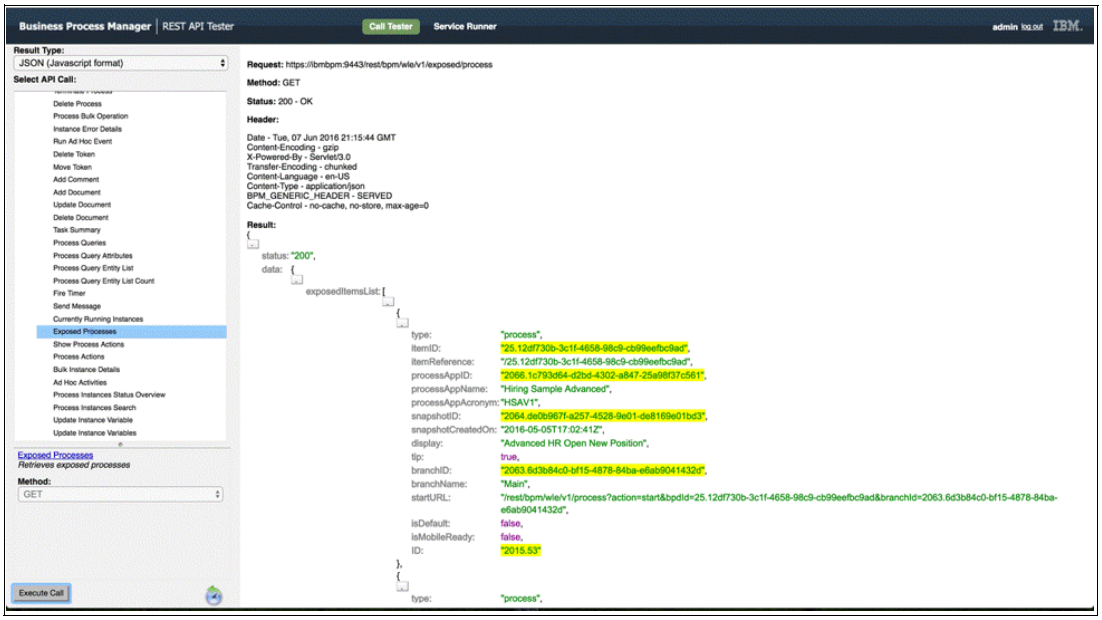


Figure 6-9 Use the REST API tester to retrieve all exposed processes

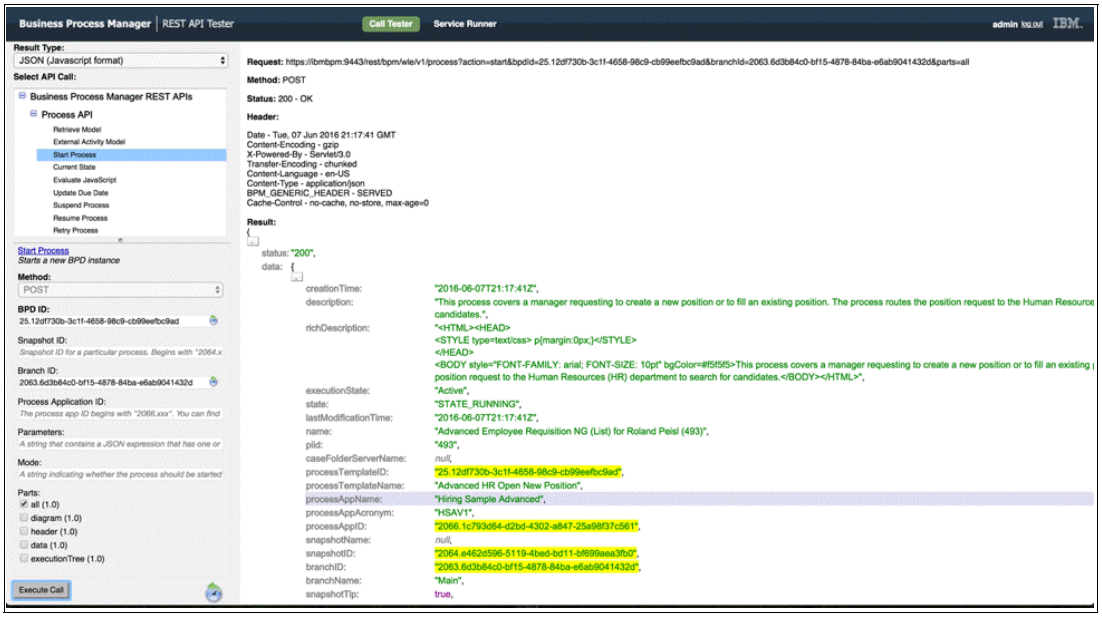


Figure 6-10 Use the REST API tester to launch an exposed process

Common REST API

In BPM 8.5.7, a simplified, top-down set of REST APIs is introduced in technical preview. This set of APIs, referred to as a *Common BPM API,* is designed to be secure, stateless, mobile-friendly, and cloud-ready. The architecture of the APIs follows the Swagger 2.0 specification,[6] the industry standard for REST API metadata, and dramatically simplifies the development of external BPM UI applications. The complexity of the Service Exposure layer is described in 6.1.6, “Pros and Cons of a formal service exposure layer for BPM APIs” on page 326.

[6] Swagger 2.0 specification found at http://swagger.io/

An example of a Swagger document for a BPM REST API is given in Figure 6-11.

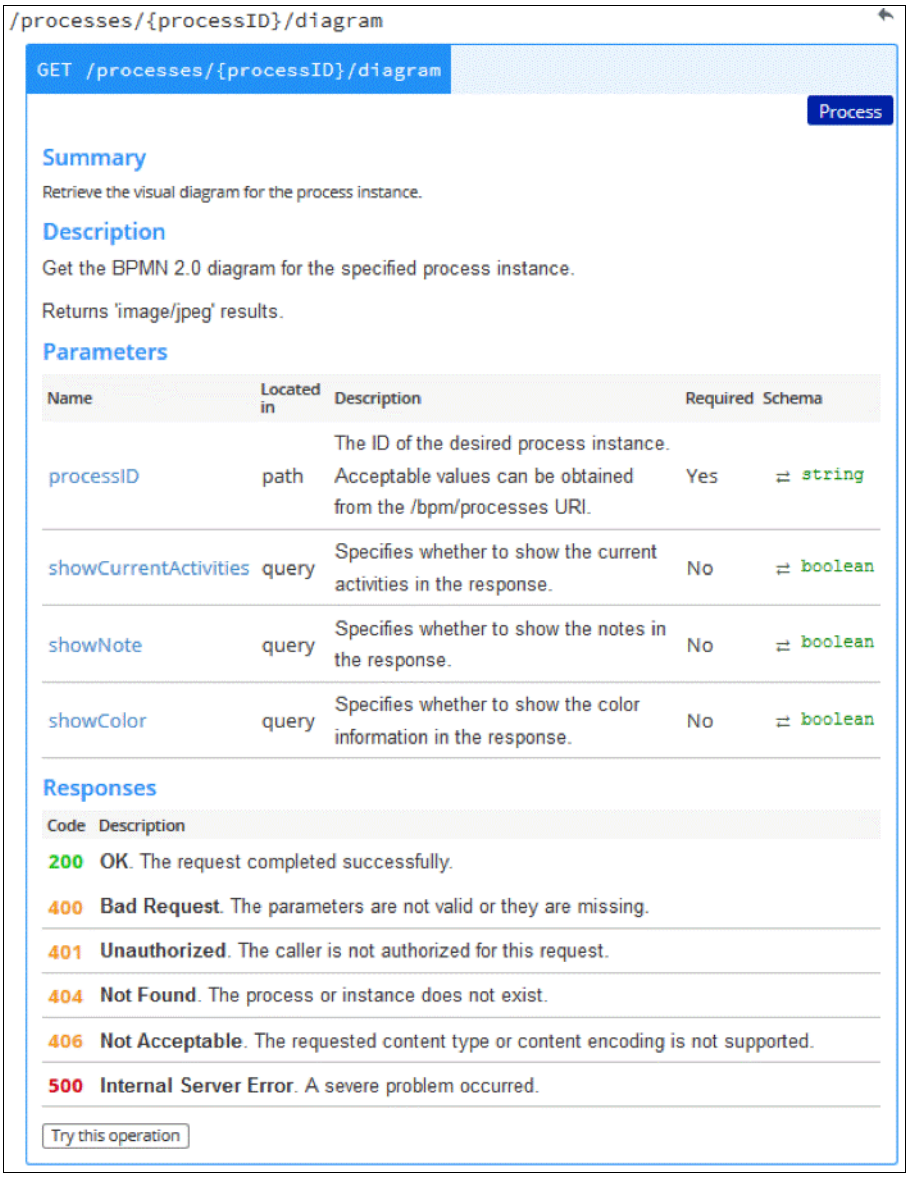


Figure 6-11 Swagger document for the process diagram REST API: Technical Preview only

The common BPM REST API technical preview is available at:

`http://ibm.biz/BPM_Common_API`

6.1.6 Pros and Cons of a formal service exposure layer for BPM APIs

The service exposure layer in Figure 6-7 on page 321 performs the function of exposing any truly reusable capability via the service layer, and all requests go through it for consistency. The APIs on the BPM engine should really be seen as *task* and *process* services. These services are highly reusable capabilities throughout the organization and, therefore, need to be treated as first class citizens of the SOA and exposed formally via the service layer.

There are several benefits in exposing services formally, when referring to BPM APIs, and some disadvantages, mainly centered around an increased complexity of the architecture.

This is a topic in its own right, and it is at the heart of the value of an SOA. With the increasing popularity of REST APIs, a high number or organizations recognize the need for tools to implement service exposure layers capable of being used internally and externally to the enterprise, such as IBM API Connect™ for more information about API Connect go to these web address:

- https://www.ibm.com/marketplace/cloud/api-management/us/
- https://developer.ibm.com/apiconnect/

In brief, a formal service exposure enables the following benefits:

- Decoupling

  Consumers of the service do not need to know about the implementation of the service, and they do not need to be aware of where the service is physically located. This approach enables the service to be changed easily, for example for reasons of alterations in infrastructure or for something more dynamic, such as improved performance at different times of the day. Consumers also do not need to know what programming language the service is written in or what operating system it is implemented on. Ideally, consumers should not even need to know the network protocols or data formats used by the implementation. The service exposure layer can perform all of this decoupling and more, minimizing the effect of changes in the implementation on the consumers of the service.

- Simplicity of reuse

  All services should use common mechanisms for description and usage to other services in the enterprise such that those who design and create new applications that consume services have less of a learning curve and are more inspired to innovate. This approach can include standardizing the way that services are discovered at design time, what protocols are used to expose them, and how enterprise headers are defined and propagated. It will certainly be important to limit the number of data formats used and it might be appropriate to use a common data model. A common service exposure layer ensures these standard enabling reuse can be more easily adhered to.

- Standardized management

  At run time, it is possible for the service layer to be monitored and controlled using a single set of tools for such capabilities as gathering monitoring statistics, checking service level agreements, throttling traffic, and scanning for malicious usage. These things are critical if the services are to be considered predictable and trustworthy. Only if all services are exposed in a standardized way, can this level of management be provided at a reasonable cost.

- Standardized governance

  From a design point of view, services should follow the same rules to ensure that they are simple to re-use, conform to the enterprise's security standards, use the same access control mechanisms, utilize protocols in the same way as other services, look after its data encryption needs in the same way, and handle versioning of services via an agreed approach. Generally, only if services are all exposed via standardized services layer, all these items can be controlled, and services can play their role within the architecture and contribute to a consistent, secure, and maintainable architecture.

  When the BPM component's capabilities are formally exposed via the services layer, they become more straightforward for future consumers to use, easier to manage, and more reliable. On the contrary, if they are not formally exposed via the service layer, any new consumers wanting to use them will have to become familiar with their specific security model, data format, and interaction patterns. They might need to suffer performance degradation or outages caused by other consumer's usage of the APIs.

A formal service exposure, however, can also infer added complexity, associated with two main areas which are as follows:

- Security model translation

  It is likely that the security mechanisms enforced in the service layer are different from those of the BPM engine. Indeed, this might well be one of the reasons the service layer is present, that is to provide a consistent security model throughout the enterprise. However, although most security model translations can be overcome within the service layer, they are often technically challenging first time around, and this is considered a significant spike[7] that needs to be tackled early to reduce risk to a project. In a formal service exposure layer, this functionality can be encapsulated in reusable services independent from the BPM process run time.

- Data model decoupling

  Should the data model provided by the underlying IBM BPM APIs be accepted as is? If the product API does not conform with the enterprise's protocol and data format governance standards, should the IBM BPM APIs data exposed model directly, or is the introduction of an independent service layer data model required? This issue is complex, but some of its obvious ramifications are that if IBM BPM APIs data model is used, each change to IBM BPM API and potentially even upgrades to new versions, might impact independent BPM UIs. However, if a new data model is introduced, then it will have to be designed, implemented, tested, and kept up-do-date with new product capabilities. This is a non-trivial task and introduces its own risks. IBM BPM has a well-thought-out API, especially in its latest form (described in 6.1.5, "IBM BPM REST APIs" on page 323).

  This API is a publicly-specified interface, and IBM is required to manage issues, such as versioning and backward compatibility when the interface changes. It is often better not to translate the data model unless absolutely necessary and to focus the decoupling on more non-functional concerns.

6.1.7 Design points relating to an independent BPM UI

There are certain implications of having an external BPM UI created and maintained in a separate technology to that of the process engine and separated by a formal service layer. This section of the book lists the main ones.

To put some of the points listed in this section in context, a brief recap of basic Business Process Management principles is provided as follows:

- A *task* is a single atomic unit of work that is started with the express expectation that it will be completed by the participant who claims it.
  - For a *system task*, the work is completed by some combination of systems.
  - For a *human task*, the work is completed by the individual participant that claims it.
- In a *human task* a participant is expected to complete the task in a single unit of work. This means without the participant needing to return to a task list or a portal and without the need to wait for some other participant or service to complete.
- In a *human task*, the unit of work can include calls to services, as long as there is an expectation that the service will return in a reasonable amount of time.

[7] For more details see SAFe SCALED AGILE, at web address http://scaledagileframework.com/spikes

- *Screen flow* is not the same as *process flow*:
  - In a *process flow*, the responsibility for the continuation of the work is passed to another participant (which can be a system, another team or even another member of the same team).
  - In a *screen flow*, all pages are accessed sequentially in the same unit of work.
- A Business Process Modeling and Notation (BPMN) process needs to reflect a business process flow and not a screen flow.
- When choosing to decouple the UI from the process engine, BPM developers need to be concerned only with the business process design. The UI design becomes a concern of the UI designers.

How is agility retained?

Fully separating the UI from the process probably means that the UI will then be written by a completely separate team and possibly in a different technology. Unless there is good collaboration between the UI and process teams, this method will likely slow down the pace at which iterations of the process can be designed, tested, and deployed. However, the greatest rate of change of the process and UI is at the early stages of process discovery. So, it is wise to continue to use the internal BPM UI for playbacks at this early stage and for as long as possible until the *wireframe* of the UIs and the data model for the process have stabilized.

Note that the wireframes created during this early phase cannot be directly reused when you move to the external UI (apart from few exceptions, as described in 6.6.2, “Use IBM MobileFirst to implement the Decoupled pattern” on page 373). Subtle styling such as color, format, and precise positioning should be done when developing the final external UI. This separation of wireframe design from detailed presentational design of the UI is commonplace in many development methodologies and helps to focus efforts on the requirements that matter in the early phases, rather than getting distracted by unnecessary details.

This progressive capture of detail is illustrated in a basic form in Figure 6-12.

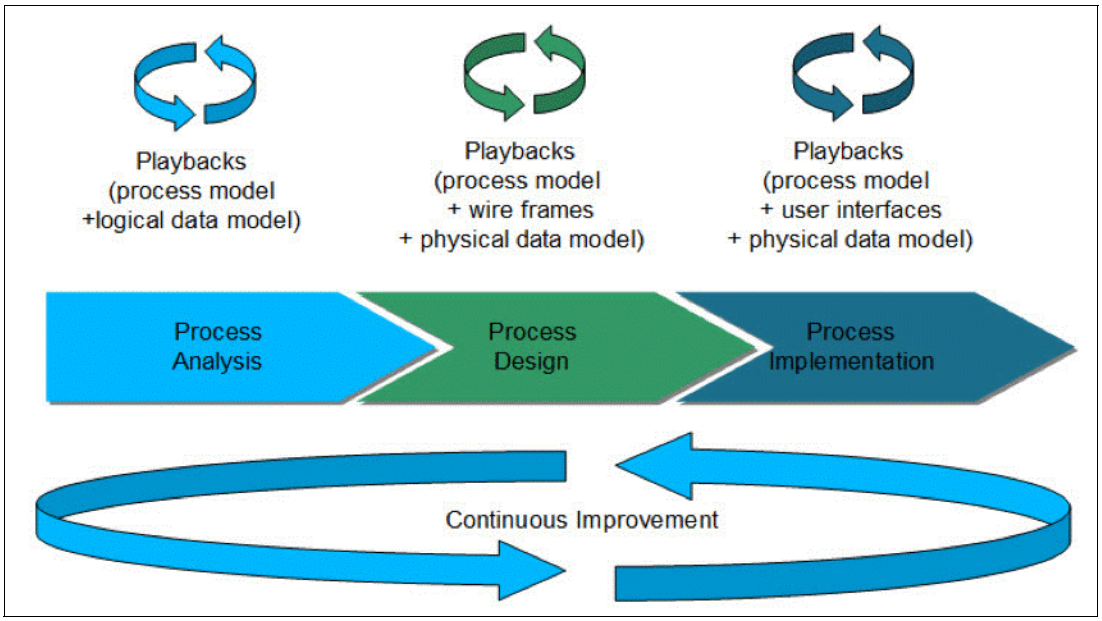


Figure 6-12 What to capture at each phase in relation to BPM

If at all possible, the benefits of the internal BPM UI should be retained through the process design phase, where significant refactoring is likely to happen. Ideally, the core interplay

between UI and process should be clarified and agreed upon in this phase. In the implementation phase, most of the effort is focused on independent concerns of the precision in the UI, and the detail of the integration with operational systems. There is always some leakage from changes affecting all layers, but retaining agility during the design phase by using the internal BPM UI provides significant advantages in helping the stabilization of the subsequent phases.

Page navigation

The UI required for a single user to complete a single task often in itself comprises of several separate pages. The user must navigate through these in a certain order, often dependent on the data they enter. This movement between pages while performing a task is called *page navigation*. It is most certainly not a business process, although it can, and often is, represented using a similar notation. It is inevitably a common mistake when modelling processes to confuse the page navigation and business processes, and there are many modelling best practices to help avoid this common mistake. Page navigation accidentally modelled within a process is relatively easy to spot, as you will see multiple tasks line in the same swim lane. Generally, subsequent tasks in a process diagram need to change lanes, passing control to a different team. If the work remains in the same team and swim lane, question why it is not all part of the same task.

As mentioned at the beginning of this section, screen flow and process flow are not the same thing, and a single task should encapsulate an entire unit of work, even if that requires displaying more than one page.

The two types of modelling look similar. Many BPM process design tools can capture page navigation alongside process diagrams and often allow for page design as well, as shown in Figure 6-13.

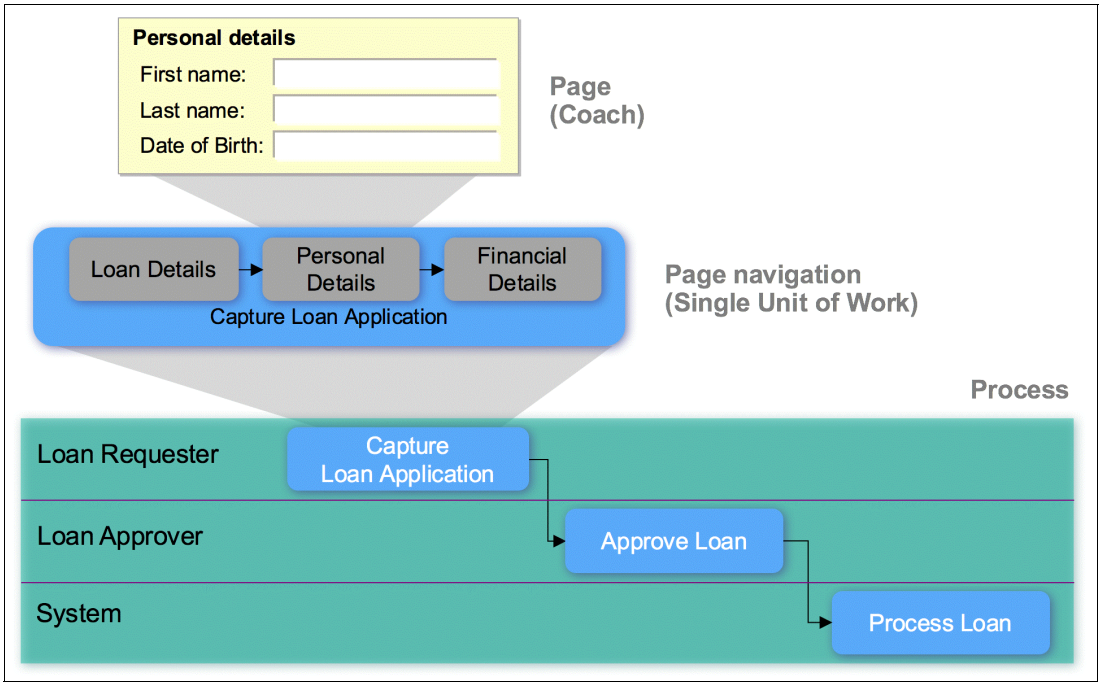


Figure 6-13 Process, page navigation and page design

When separating the graphical UI from the process, the pages belong in the UI technology and the process remains with BPM. It is tempting to conclude that because the notation is

similar, the page navigation can be part of the process definition. However, it is fundamental to understand how tightly coupled pages are with their associated navigation.

The following points illustrate how close the page design and page navigation really are:

- Navigation logic

  Where the page navigates to is likely to be highly dependent on the data entered on previous page.

- Data dependencies

  Data on one page can affect data on other pages in ways, such as field visibility and filtering lists.

- Performance

  Responsive UIs are critical to the front line staff, and performance is key. Pre-fetching data for the next page is done if the UI knows enough about where the user is likely to go next. Pages themselves might need to be pre-loaded.

- Data models

  It is likely that the pages will hold data in a data model that represents the data itself in a way that is simple to render on the pages, with data grouped and represented in the same way as it is across the pages. The pages are likely to make use of reusable UI components that hold the data in a specific form. The page navigation needs to work directly with this data model if it is to avoid continuously translating between two models. It is often the case that this data model is different than the model used by the underlying process.

Although process logic is completely separate from page navigation:

- The process is unaware of how a task is achieved, how many pages are present, how they are navigated, or how long page navigation takes. It only needs the data required to achieve the task and the data sent back by the task when it is complete.
- Process design tools such as IBM BPM deliberately simplify UI design so that it can be performed alongside process design, improving the value of the playbacks. However, process designers do not typically have extensive training in UI technologies or in UI design. UI design for front line users, such as those in customer facing scenarios, is typically a separate and highly specialized skill.

If an external UI is to be created to work the tasks, it needs to contain both the pages and the page navigation, as shown in Figure 6-14.

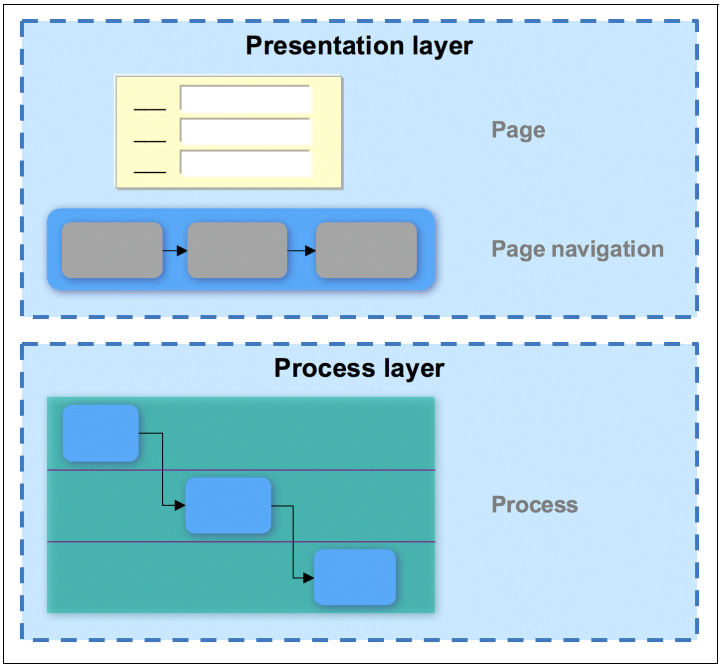


Figure 6-14 Separation of concerns for an independent UI in BPM

This separation of concerns ensures that the appropriately skilled teams can do what they know best and that the only thing that needs be agreed between them is the input and output data for the task in hand.

IBM BPM enforces this logically correct separation of concerns: *human services* (both client-side and server-side) are not part of a BPMN model as such, but in IBM BPM they provide a UI framework for implementing BPMN *human tasks*.

Human services are therefore designed separately from processes, and are considered *human tasks* within the processes themselves, which invoke them when necessary.

When decoupling BPM UI from a BPM processes, the correct decoupling should happen logically between the Process Layer and the Presentation Layer, as described in Figure 6-14 on page 332. The IBM BPM REST APIs expose functionality around an entire task for this specific reason. As an example, when a task is designed in IBM BPM as a *human service*, functionality around page navigation within the human service itself is not exposed via the REST API in a supported fashion. This implementation choice is the correct one, since when implementing an external UI application, that application should not use what was already defined within the human service itself but replace it in its entirety.

6.1.8 Pros and cons of an external BPM UI

As a general rule, the larger and more complex the enterprise, the more formal separation of concerns provided by the SOA reference architecture will need to be followed if the overall landscape is to remain manageable in the future.

When choosing an external implementation for the BPM UI, there are three main categories of advantages and drawbacks that an enterprise is expected to encounter:

- Reusable patterns
- Common look and feel
- Reusable business services

This list is not exhaustive but highlights the recommendations to maximize the benefits of external BPM UIs, while reducing the amount of additional effort required for their implementation.

Reusable patterns

Using external UI implementations for BPM human tasks simplifies the adoption of reusable UI patterns across the enterprise and provides a set of frameworks that make UI design easier and more consistent. These design patterns are described in 6.2, “BPM UI design patterns” on page 335.

Additionally, this choice allows the enterprise to be independent from the UI technology used for the integrated BPM UI. The UI layer is in the hands of the enterprise and the process capability provided becomes unaware of the technology chosen to implement external UIs.

As a consequence, UI technology can be changed or updated without impacting the process layer and without need of refactoring the process implementations.

However, this entails significant upfront work, in particular for the first project. The two major areas of work will consist of:

- Creating a service exposure layer for the BPM REST APIs

  Creating a service exposure layer for the BPM REST APIs as described in 6.1.6, “Pros and Cons of a formal service exposure layer for BPM APIs” on page 326. Even if the complexity of the service exposure layer is kept to a minimum, this introduces an additional layer in the architecture, which needs to be maintained and governed separately. It is indeed worth noticing that, when an external UI implementation invokes a service, access to that service also needs to be implemented externally to BPM.

- Refactoring existing processes

  It is common for existing processes that are designed to use the internal BPM UI to need a level of refactoring when an external UI implementation is chosen. A typical example is the use of a high number of service calls within a *human service*, which in turn become the responsibility of the external UI implementation. Which service calls should pertain to a client-side UI implementation, and which should live within the server-side BPM process run time?

As a general recommendation, consider external UI implementations only when multiple, highly independent consumers are required.

The trade off between a set of reusable UI patterns and a significant increase of the end-to-end business processes complexity should also be clearly analyzed, especially in enterprises that already make use of BPM integrated UIs.

Common look and feel

External UI frameworks allow the creation of a common look and feel across the enterprise, even for applications not implemented in BPM. A typical driver for common look and feel is given by customer-facing UIs, generally external to the enterprise.

However, these UI frameworks result in another layer of the architecture to be maintained, requiring a different set of skills and a parallel development lifecycle for each business process using an external UI.

As a consequence, the agility attained by developing both BPM process and BPM UI within the same tool, as described in Figure 6-12 on page 329, is impacted.

Within the same process, it is possible to use a combination of integrated BPM UIs and external UI implementations. The latter should be used only when the trade off between a common *look and feel* and an additional design layer (with the consequential loss of agility) is acceptable.

Reusable business services

As a follow-on from the previous categories, when a service exposure layer is used, a set of reusable business services can be formally exposed inside and outside the enterprise, making BPM consumption easier.

However, as mentioned previously, introducing an additional layer in the BPM applications' architecture increases the complexity of the overall solution, resulting in this additional layer to have to be maintained and governed separately.

A common pitfall to avoid is to embed UI logic and process logic in the service exposure layer in order to keep the external UI implementation and the process model simple. However, good process design requires that logic to live in the presentation and process layers, respectively.

The main recommendation in this category is therefore to avoid creating service compositions, data model replacement and complex service logic in the service exposure layer, which should be kept as transparent as possible.

Requirements around traffic management and security should also dictate when a formal service exposure layer is required.

6.1.9 External UI integration: anti-patterns

In the previous section the main advantages and disadvantages of using external UI implementations have been described, alongside with general recommendations on how to approach the integration of these UIs.

This section describes the most common anti-patterns encountered when using an external BPM UI implementation: the common mistakes that are to be avoided.

It is not surprising that most of these anti-patterns derive by an incorrect application of the BPM design principles described in 6.1.7, "Design points relating to an independent BPM UI" on page 328. As such, these anti-patterns can also be found when using an integrated BPM UI, but are more commonly exposed when external UI implementations are chosen.

Screen flow equals Process flow

In this anti-pattern, each user interaction within a screen flow is modelled and implemented as an individual process task.

As this implementation is driven by the UI implementation choice, rather than by correct business analysis, this results in business processes to be incorrectly modelled and too fine-grained. In turn, this can dramatically increase the complexity of the process orchestration and potentially impact performance.

The recommendation is to ensure that process analysis happens early, when customer journeys are identified and before UI design proceeds to an advanced stage. Process analysis should be the sole driver to process modelling.

In a correctly modelled process, multiple screens/user interactions are usually involved in a single business task, if part of the same *single unit of work*.

Embed business Logic in the Presentation Layer

In this anti-pattern, aspects of the business logic (for example, process orchestration, business rules, and so forth) are implemented in either the service exposure or the presentation layers. This dilutes the focus on the UI design, breaks the separation of concerns between the three layers (presentation, service exposure, and process layers) and fragments the implementation of business logic.

It can also potentially lead to process applications that are ungovernable and unmaintainable.

The presentation and service exposure layers should be kept simple and maintain clear design goals, in terms of concerns and responsibilities per layer. It is fundamental to ensure close collaboration between the presentation and the process delivery teams, to avoid confusion on design responsibilities.

All services are System Tasks

In this anti-pattern, all enterprise service integration is implemented via BPM system services, rather than using a formal service bus or service exposure layer. Because this implementation is driven by the desire to reduce integration from the presentation layer, rather than by correct business analysis, this type of implementation results in business processes being incorrectly modelled and too fine-grained, which can dramatically increase the process orchestration and potentially impact performance.

Since poor process design is at the base of this anti-pattern, it is not surprising that its effects can be the same as those of the first anti-pattern described previously in this section (also caused by bad process design): Screen flow equals Process flow.

Using external UI implementations necessarily increases the complexity of some or all of the application's architecture layers. A typical example is the need of invoking services directly from the presentation layer (through the service exposure layer), rather than delegating this role to the process layer.

The recommendation is to ensure that clear criteria for deciding when services invocation is to be implemented in the presentation layer rather than the process layer are defined. The way of defining those criteria should rely on best practices around process analysis and modelling, and not on the UI implementation choice (independent BPM UI verses integrated BPM UI).

6.2 BPM UI design patterns

When using BPM, traditionally users access human tasks within a business process through the *integrated BPM UI* defined in Figure 6-5 on page 318, which includes both the *task portal* and the context-specific screens: Coaches and coach-based human services.

Section 6.1.3, “Accessing tasks externally” on page 318 describes use cases in which an *independent* BPM UI is usually adopted. Additionally, 6.1.4, “Exposing a process via a business service” on page 321 lists typical examples in which this independent BPM UI is

decoupled from the process engine, where users are not even aware that they are interacting with a process, and usually do not access the UI through a task portal.

Additionally, in a mature BPM landscape, independent BPM UIs can often be used together with the integrated BPM UI provided by IBM BPM.

As a consequence of the various use cases mentioned previously, it is possible to identify six distinct design patterns when creating a BPM UI, based on the following dimensions:

- The choice of the BPM human tasks implementation between:
  - Coach-based human services
  - External UI implementations
- The choice of the task portal implementation between:
  - No portal, for human tasks that are not claimed from a task list, where the user is unaware of the existence of a business process
  - An alternative portal, either an existing enterprise portal or a custom-built lightweight portal
  - IBM Process Portal, provided with IBM BPM

The resulting patterns are defined in Table 6-2 and Figure 6-15 on page 337.

Table 6-2 BPM UI common design patterns

TASK PORTAL / HUMAN TASK	Coach-based human services	External UI implementation (not using BPM coaches)
No portal	Exposed pattern	Decoupled pattern
Alternative portal	Embedded pattern	Independent pattern
IBM Process Portal	Integrated pattern	Hybrid pattern

It is important to notice that the patterns defined in Table 6-2 refer to the implementation of a single BPM human task. However, typically business processes contain more than one human task. A combination of these patterns is possible within the same business process and is described in 6.9, “Combining and extending the BPM UI design patterns” on page 386.

When a BPM human task is implemented using a coach-based human service, the patterns identified are:

- Exposed pattern

  Exposes coach-based human services without the need of a task portal.

- Embedded pattern

  Embedded pattern uses an alternative task portal which *embeds* coach-based human services.

- Integrated pattern

  Integrated pattern uses the IBM Process Portal to launch coach-based human services. This pattern leverages the ready to use integrated BPM UI provided by IBM BPM.

When a BPM human task is implemented using an external UI implementation, the patterns identified are:

- Decoupled pattern

  The decoupled pattern creates BPM human tasks using an alternative UI technology, and exposes them without the need of a task portal. In this pattern, the presentation layer is decoupled from IBM BPM.

- Independent pattern

  The independent pattern uses an alternative task portal implementation and builds human tasks using an alternative UI technology. In this pattern, the presentation layer is completely independent from IBM BPM.

- Hybrid pattern

  The hybrid pattern launches BPM human tasks built using an alternative UI technology directly from the IBM Process Portal. In this pattern, the presentation layer is a hybrid between IBM BPM and alternative technologies.

These three latter patterns use BPM as the process engine and rely on external UI implementations to various degrees of complexity. All implementations are reliant on using the exposure mechanism for human tasks based on the public RESTful APIs described in 6.1.5, “IBM BPM REST APIs” on page 323.

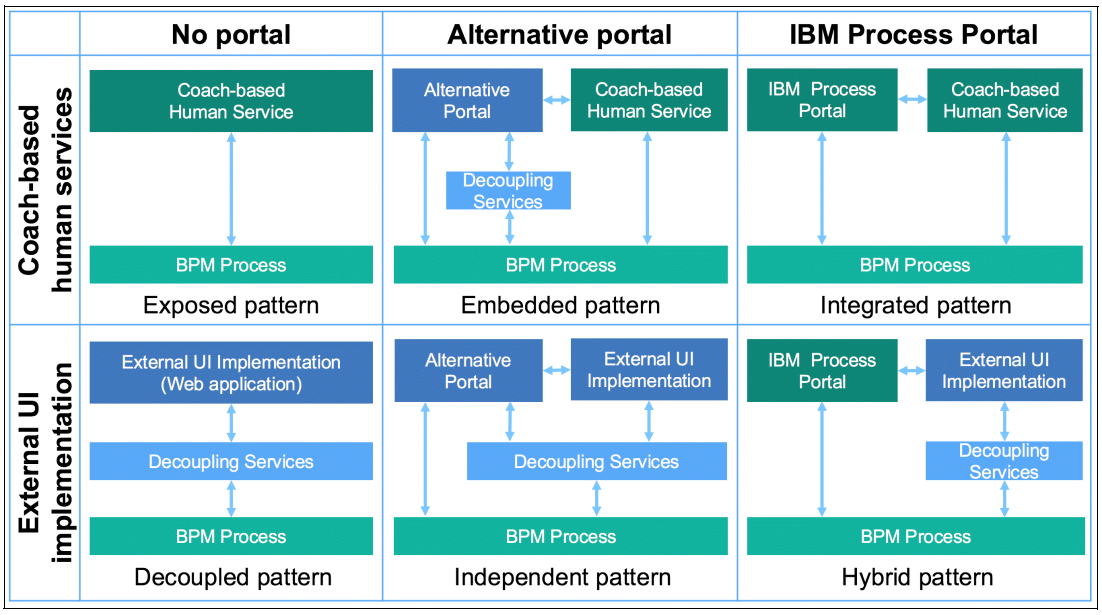


Figure 6-15 BPM UI common design patterns

The patterns defined in this section are derived from proofs of concept and implementations in the field that are carried out with various IBM BPM customers. The inception of this work was carried out in collaboration with the Royal Bank of Scotland,[8] who also presented their findings at the IBM InterConnect 2016 conference. You can find that presentation here:

`http://ibm.biz/RBS_BPM`

The remainder of this chapter discusses the six design patterns defined in more detail, providing implementation examples and recommendations. A compressed file that contains the samples used for this chapter is available. Refer to Appendix A., “Additional material” on page 395 for more information.

[8] Royal Bank of Scotland Group, `http://www.rbs.com`

6.3 Exposed pattern

The Exposed pattern, shown in Figure 6-16, is user experience oriented. In this pattern, a human task is exposed to the users as a business service, as defined in 6.1.4, “Exposing a process via a business service” on page 321. There is no need for a task portal, as users are unaware of the existence of BPM human tasks. They simply interact with an email or a URL.

In this pattern, the user experience is provided by an IBM BPM coach-based human service that is implementing a human task, which is in turn exposed, directly or indirectly, to the users.

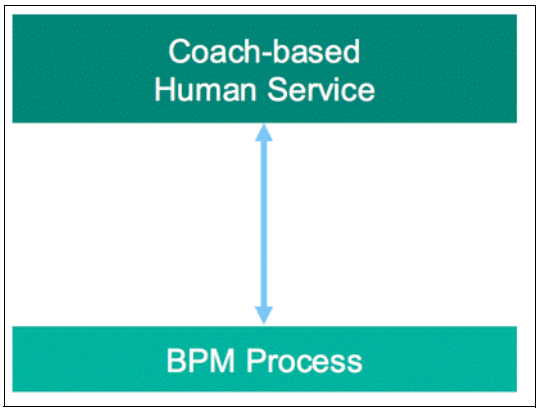


Figure 6-16 Exposed pattern

This pattern is broken down in two subpatterns:

- Expose human services directly without any additional layer between IBM BPM and consumers of the human services.
- Expose human services indirectly through an additional layer, or *filter*, between IBM BPM and consumers of the human services.

Additionally, two *access* options can therefore be defined for this pattern:

- Inside the enterprise the user accesses IBM BPM from inside the firewall on the organization’s intranet or VPN.
- Outside the enterprise the user accesses IBM BPM from the public Internet (meaning outside the organization’s firewall).

Exposing human services directly is primarily intended for internal users. On the converse, indirect exposure can, within certain caveats, be used for users both internal and external to the enterprise.

As described in Chapter 1, “Delivering modern UI for IBM BPM using the Coach Framework and other approaches” on page 1, IBM BPM provides two supported constructs to build a human task: heritage human services and client-side human service.

Heritage human services tightly couple the human task implementation to the process engine, as the human service itself is essentially executed server-side, and only the UI pages (coaches) are rendered on the client side.

Additionally, in heritage human services, each boundary event can potentially go back to the server for validation and all variables passed in a boundary event are set as server-side variables, independently from their binding.

Alternatively, client-side human services are more—even if not completely—loosely coupled, depending on their design. They are executed mostly client-side.

Note: It is possible in client-side human services to define server-side calls. Depending on their design, therefore, the execution of these human services can happen on both client and server side.

A client-side human service is never completely decoupled from IBM BPM. If that is the specific requirement, use the Decoupled pattern (described in 6.6, “Decoupled pattern” on page 367).

Therefore, the task exposed as a business service is to be exposed outside of the enterprise to the public Internet, the following reccomendations apply:

- Do not expose any type of human service, either heritage human service and client-side human service, directly to the Internet.
- Do not expose, either directly or indirectly, heritage human service to the Internet.

As a consequence of these recommendations, when wanting to expose a BPM coach-based human service to the Internet, the recommendation is to use client-side human services indirectly exposed, using the additional architectural and security layers for indirect exposure provided by the SPARK External Participant Support (EPS) toolkit, which are described in 6.3.2, “Expose human services indirectly through the EPS toolkit” on page 347.

Note: This recommendation is valid only after the security implications of this choice are understood and considered acceptable by the enterprise. Security considerations of exposing human services, although indirectly, to the Internet, are described in the “Security considerations for the EPS toolkit” on page 356 of the Indirect exposure subpattern.

Table 6-3 provides a summary of the recommended approaches for the Exposed pattern and its subpatterns.

Table 6-3 Recommended usage of the Exposed pattern and its subpatterns for human services

	DIRECT Exposure subpattern (section 1.3.1)	**INDIRECT Exposure subpattern (section 1.3.2)**
Inside the Enterprise	Client-side human service or heritage human service	Client-side human service or heritage human service (through EPS)
Outside the Enterprise	Not recommended	Client-side human service only (through EPS)

In both subpatterns described in Table 6-3, human tasks implemented as human services and exposed as business services typically follow the two usage scenarios described in 6.1.4, “Exposing a process via a business service” on page 321:

- Exposed human services in isolation

  A user accesses an exposed human service which is not part of an overall process. The human service might, for example, be used to initiate a process or access a report. This scenario is described as *isolated business service* in 6.1.4, “Exposing a process via a business service” on page 321.

- Exposed process instance tasks

  A user completes an exposed process instance task. As such, this task is part of a running process. This scenario is described as a *correlated business service* in 6.1.4, “Exposing a process via a business service” on page 321.

This adds another dimension to what is described in Table 6-3, based on the specific *usage* scenario. A consolidated view of the resulting implementation options is given in Table 6-4, along with links to the corresponding implementation description, provided in the next parts of this section.

Table 6-4 Recommended use of the Exposed pattern and its subpatterns based on usage scenarios

	Direct Exposure subpattern (6.3.1, “Expose human services directly”)		**Indirect Exposure subpattern (6.3.2, “Expose human services indirectly through the EPS toolkit”)**	
	Isolated business services	Correlated business services	Isolated business services	Correlated business services
INSIDE the Enterprise	Directly expose human services in isolation	Directly expose process instance tasks	Indirectly expose human services in isolation	Indirectly expose process instance tasks
OUTSIDE the Enterprise	Not recommended		Indirectly expose human services in isolation	Indirectly expose process instance tasks

6.3.1 Expose human services directly

This section describes the *Direct Exposure* subpattern listed in Table 6-4.

A human task implemented using a client-side human service can be exposed directly to BPM internal users, provided that those users are authenticated against the Process Server where the human service is running.

An example is provided in Figure 6-17.

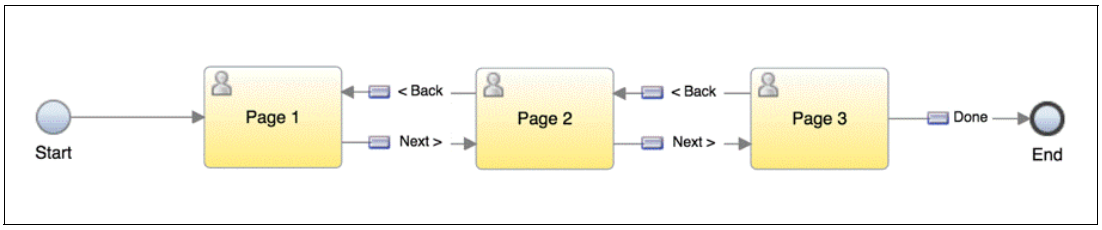


Figure 6-17 Client-side human service with three coaches

To use a client-side human service, the Overview tab of the human service implementation provides four options, as showed in Figure 6-18.

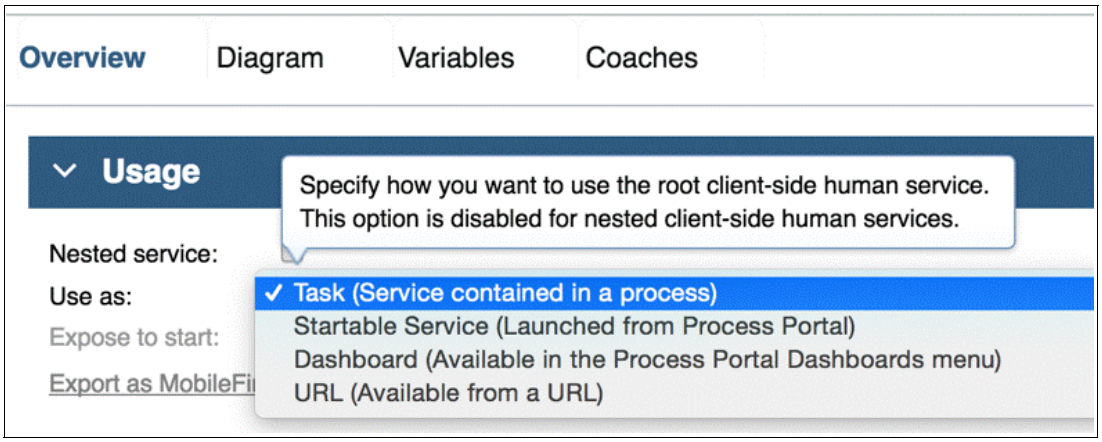


Figure 6-18 Use a client-side human service

The available options neatly relate to the usage scenarios previously described:

- The Startable Service, Dashboard, and URL options are intended for tasks that might be independent from processes. Therefore, they can be used in an Exposed human service in an isolation scenario.
- The Task option is intended for tasks contained in processes and is used in an Exposed process instance task scenario.

You can find more information about client-side human service exposure in IBM Knowledge Center:

`http://ibm.biz/BPMExposedHS`

Albeit not detailed in this book, the subpattern described in this section can also be implemented using heritage human services. You can more information about heritage human services exposure in IBM Knowledge Center:

`http://ibm.biz/BPMExposedHHS`

Directly expose human services in isolation

This section describes the *isolated business service* usage scenario for the *direct exposure* subpattern described in Table 6-4 on page 317. The Startable Service, Dashboard, and URL options in Figure 6-18 expose a client-side human service as an isolated business service.

Taking the URL option as an example, when this usage is selected, the human service is exposed as URL only, and Process Designer generates the URL to invoke the client-side human service directly, without the need of a *task list*, as shown in Figure 6-19.

Figure 6-19 Expose a client-side human service as a URL

This URL might not work across all environments and is primarily meant for testing purposes. For production purposes, the client-side human service URL can be retrieved through a REST API call, as described in IBM Knowledge Center:

`http://ibm.biz/REST_Exposed_Items`

Similarly, URLs can be retrieved for client-side human services exposed as *Startable Services or Dashboards*.

The Usage tab in Figure 6-19 on page 341 also allows process developers to choose the user group to which this client-side human service is made accessible.

It is important to note that exposing a client-side human service as a URL does not, however, directly implement an interaction between said client-side human service and a BPM process. If the intended behavior is for the client-side human service to execute in isolation, and then trigger a BPM process, this can be attained through the use of Undercover Cover Agents (UCA). The use of UCAs pertains process design in IBM BPM, and is not strictly part of the topic treated in this book. You can find information in IBM Knowledge Center:

`http://ibm.biz/BPM_UCA`

Directly expose process instance tasks

This section describes the *correlated business service* usage scenario for the *Direct Exposure* subpattern described in Table 6-4 on page 340. Commonly, human tasks exposed as business services are correlated with a BPM process: their usage falls in the *Correlated business service* usage scenario.

Figure 6-20 shows a process with two human tasks:

- The first one not exposed, in grey
- The second one exposed, in blue

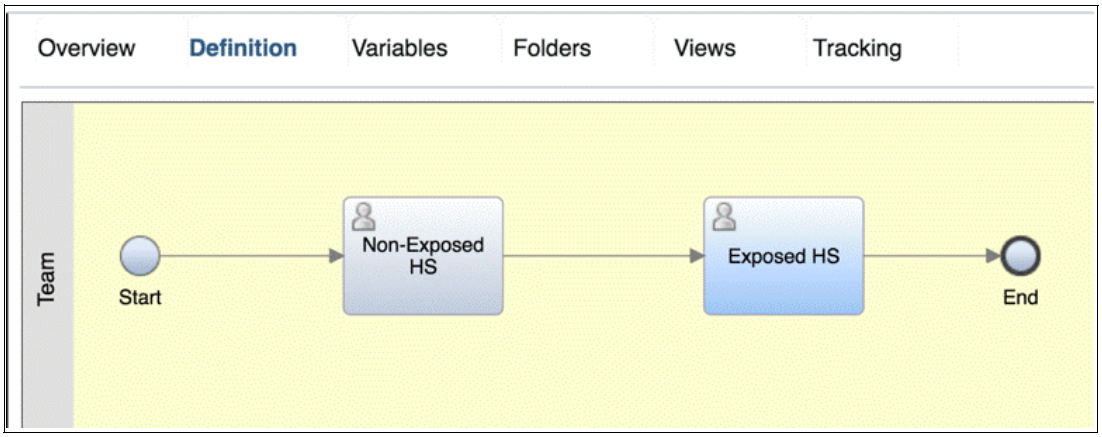


Figure 6-20 Sample process with one exposed human service

Within this example, when the process execution reaches the blue task, the task instance gets assigned a unique task ID.

When the selected Usage of the client-side human service defined for the blue human task (as shown in Figure 6-18 on page 341) is Task, IBM BPM provides a unique URL for each process task instance at run time, using its task ID.

Note: The URL defined in Figure 6-19 on page 341 invokes a separate, isolated implementation of the task. To invoke a task instance within a specific process instance, additional information to identify the task instance itself is needed.

The unique URL to invoke the task instance can be retrieved programmatically using the IBM BPM REST API, as described in IBM Knowledge Center:

http://ibm.biz/BPM_Get_Task

It is however trivial to compose the unique URL for the process instance task, using the template provided in Example 6-1.

Example 6-1 URL template for programmatic retrieval of a process instance task

```
https://<host_name>:<port_number>/<contextRootPrefix>/process.lsw?zWorkflowState=1
&zResetContext=true&zTaskId=<TASK_ID>
```

Where:

- *contextRootPrefix* depends on your system configuration. The default context root is teamworks
- *host_name* is the network name for the host of the process server
- *port_number* depends on your system configuration. Default port numbers are 9080 (HTTP) and 9443 (HTTPS)
- *TASK_ID* is the unique identifier of the task instance

Note: The process.lsw API is the one in use at the time this book is written. Future IBM BPM implementations might make use of a different API. To programmatically retrieve a unique URL invoking a task instance use the following web address:

http://ibm.biz/BPM_Get_Task

For example, to invoke task 1123 on server1, with the default configuration, use the following URL:

```
https://server1:9443/teamworks/process.lsw?zWorkflowState=1&zResetContext=true&zTa
skId=1123
```

The URL can be then served to a user, which invokes it to complete the task instance.

Directly expose process instance tasks via email

A common extension of the implementation described for an exposed process instance task is to send an email notification to a user or user group with a link to the task instance, so that they can execute the process instance task directly. The receivers of the email are internal participants to the process: IBM BPM users.

IBM BPM provides a mechanism to send notifications via email to users in a team to which a task is assigned. Instructions to configure this feature are detailed in IBM Knowledge Center:

http://ibm.biz/EmailTask

Note: An SMTP server is needed in order for IBM BPM to send emails.

Each IBM BPM user has a profile that can be associated with an email address and with an option to receive a email notifications when a task is assigned to the team they are part of.

This profile can be configured by BPM administrators or directly by the users within IBM Process Portal, as depicted in Figure 6-21.

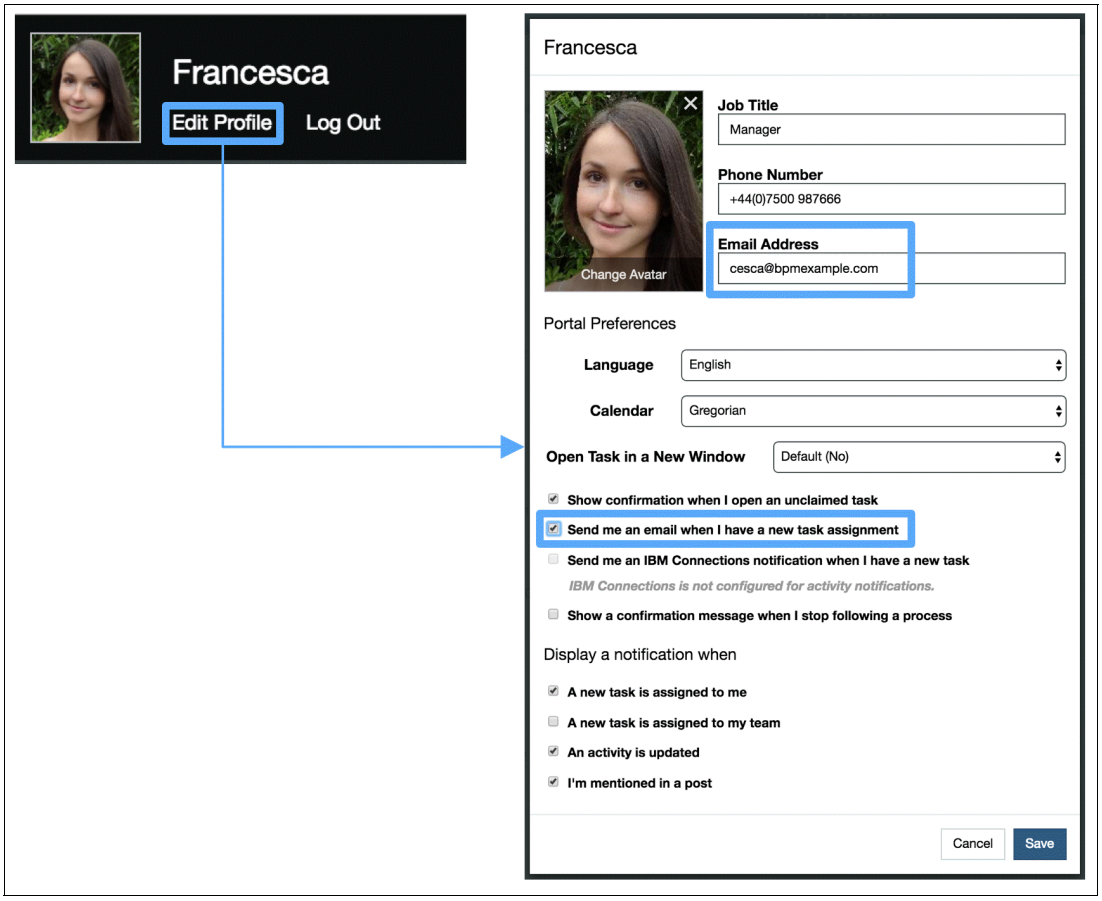
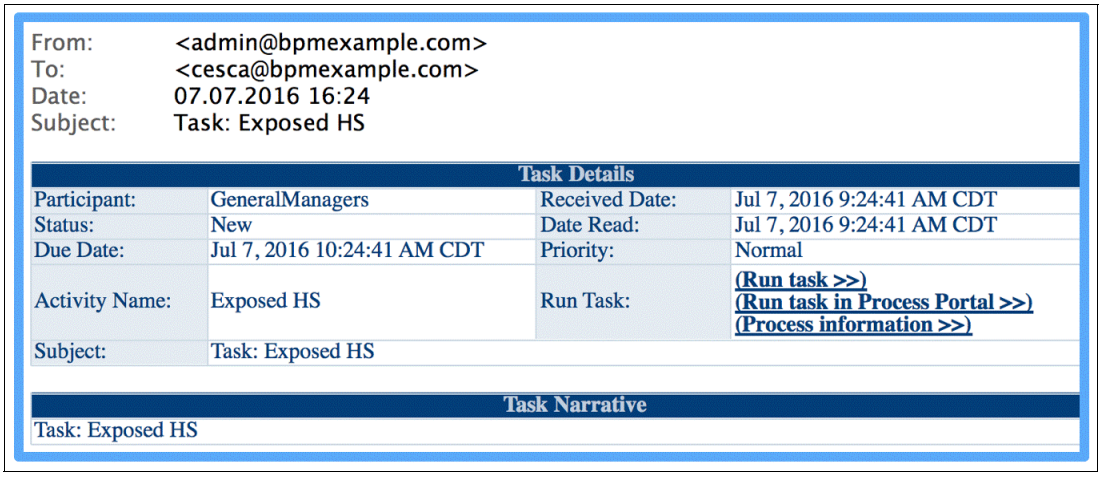


Figure 6-21 Customize BPM user profile to add email address and task assignment notifications

If this feature is enabled, when a task is assigned to a user who has opted for email notifications, BPM interacts with the SMTP server to send an email with a link to the specific process instance task, as shown in Figure 6-22.


From: <admin@bpmexample.com>
To: <cesca@bpmexample.com>
Date: 07.07.2016 16:24
Subject: Task: Exposed HS

Task Details			
Participant:	GeneralManagers	Received Date:	Jul 7, 2016 9:24:41 AM CDT
Status:	New	Date Read:	Jul 7, 2016 9:24:41 AM CDT
Due Date:	Jul 7, 2016 10:24:41 AM CDT	Priority:	Normal
Activity Name:	Exposed HS	Run Task:	(Run task >>) (Run task in Process Portal >>) (Process information >>)
Subject:	Task: Exposed HS		

Task Narrative
Task: Exposed HS

Figure 6-22 Directly exposed human service via email, as shipped implementation

The email in Figure 6-22 contains a link to the process instance task, which is a URL automatically generated following the template described in Example 6-1 on page 343. The internal BPM user can click the link, and complete the process instance task directly from the browser.

Both the generated URLs and the email template are customizable. To do so, developers can edit the `100Custom.xml` file and add to it a customized version of the email template that is contained in the `99Local.xml` file. You can find more information about these files and their usage in IBM Knowledge Center:

`http://ibm.biz/100CustomXML`

If the level of customization available for the ready to use email notifications is not sufficient, or if the chosen notification mechanism is different from emails (for example SMS or social media notifications), a different implementation of this solution can be modelled explicitly in BPMN, as depicted in Figure 6-23.

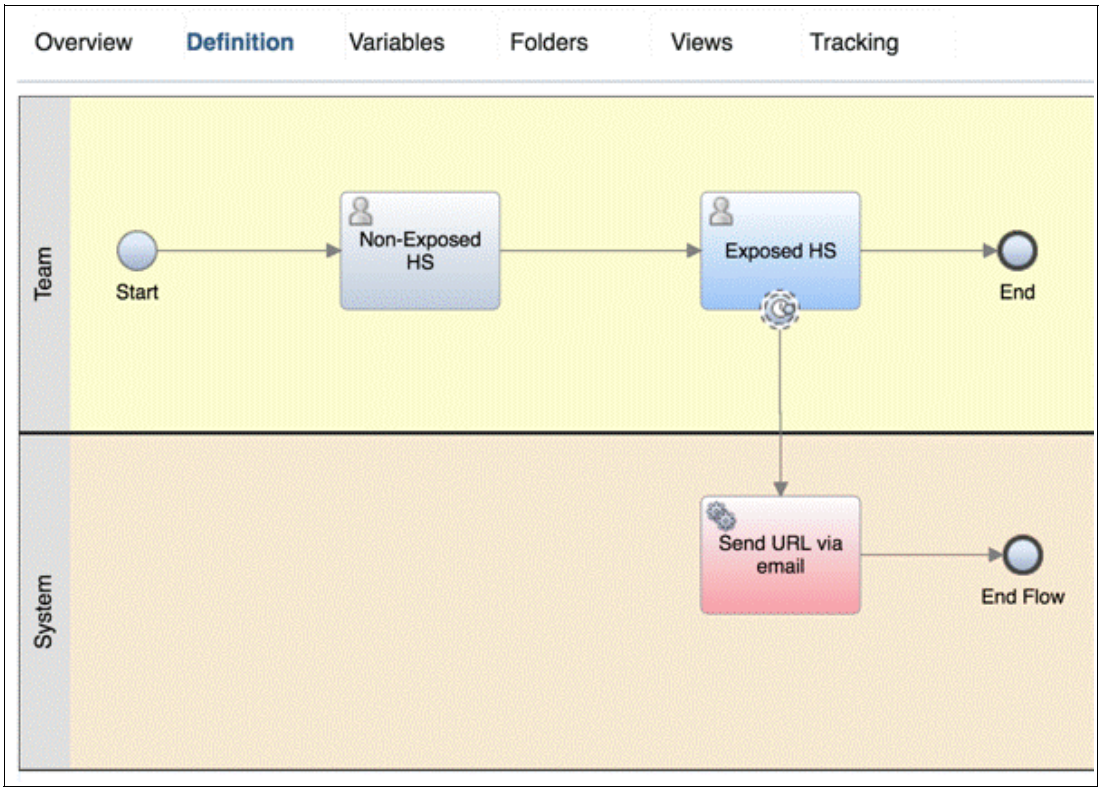


Figure 6-23 Directly exposed human service via email: Explicitly modelled implementation

In this example, when the process execution reaches the blue task, a non-interrupting timer event is triggered after 0 seconds, capturing the task ID and assigning it to a local variable, as shown in Figure 6-24.

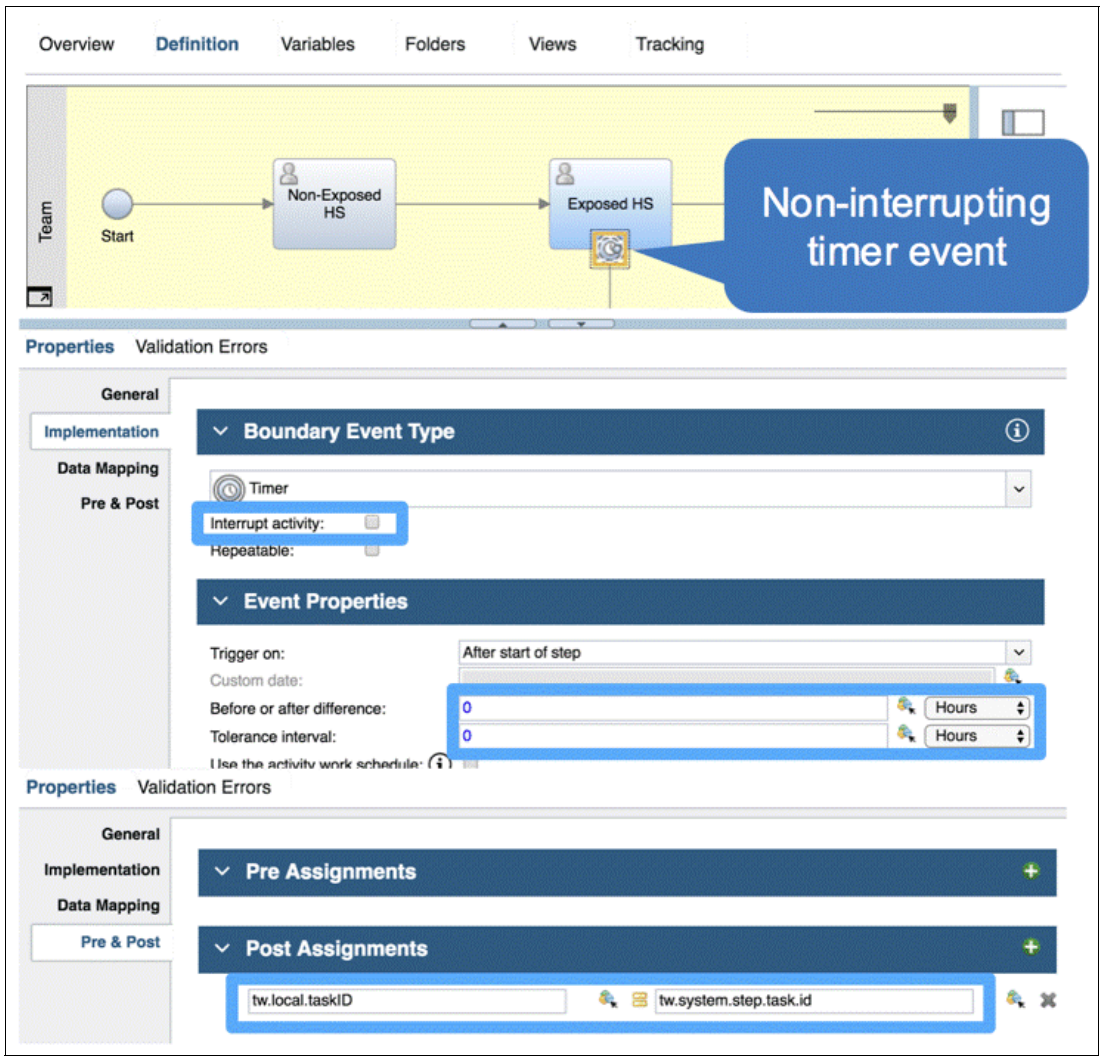


Figure 6-24 Timer event configuration

The value contained in the `tw.local.taskID` local variable is then passed to the red system task in Figure 6-23 on page 345, which programmatically creates a unique URL for the task instance (using the template described in Example 6-1 on page 343) and sends an email or a different type of notification to the relevant user with a link to the task instance, as summarized in Figure 6-25.

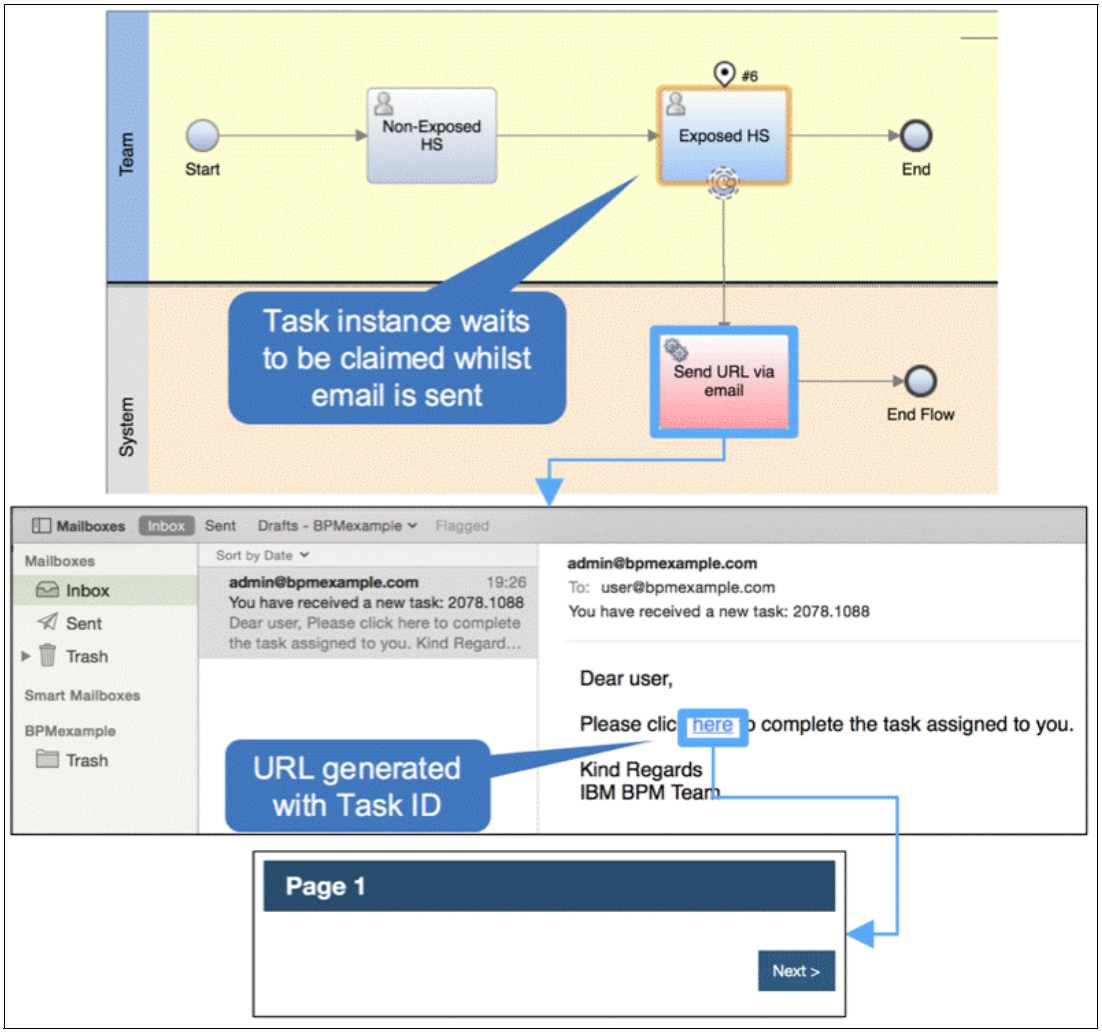


Figure 6-25 Directly exposed human service via email, explicitly modelled implementation and execution

The full implementation of this example is available in the `SG24-8355-ch6-patterns.twx` file, refer to Appendix A, “Additional material” on page 395 of this book.

6.3.2 Expose human services indirectly through the EPS toolkit

This section describes the Indirect Exposure subpattern in Table 6-4 on page 340 and the use the SPARK External Participant Support (EPS) toolkit[9] for the implementation of this subpattern.

[9] This toolkit is an IBM BPM toolkit that was created by Salient Process, an IBM Business Partner specializing in IBM Smarter Process consulting services and innovation. You can find more information about the toolkit here: `http://ibm.biz/BPMSparkEPSToolkit`

In fact, the SPARK EPS toolkit allows coach-based human services to be *indirectly* exposed to external participants, by adding additional architectural and security layers between IBM BPM and consumers of the human services. An external participant or external user in this context is defined as a generic user, including both IBM BPM users and, most notably, non-BPM users (individuals that are not defined in the IBM BPM user registry with an ID and a password).

The EPS toolkit offers an implementation for the indirect exposure of a *human task* as a business service, as described in 6.1.4, "Exposing a process via a business service" on page 321. Both usage scenarios described at the beginning of 6.3, "Exposed pattern" on page 338, Isolated business services and Correlated business services are supported by the EPS toolkit.

For either of these usage scenarios, the two access options supported are as follows:

- Inside the enterprise

  The external participant (whose department and security directory might not be associated with IBM BPM) accesses IBM BPM from inside the firewall on the organization's intranet or VPN

- Outside the enterprise

  The external participant (who can be an organization's customer or business partner) accesses IBM BPM from the Internet (meaning outside the organization's firewall), through a public-facing web server.

The mechanics of the EPS toolkit in those usage scenarios are described in the next section, independently from the chosen access option (further detailed in "Access options" on page 353).

EPS toolkit architectural overview

The core functionality of the EPS toolkit (and its supporting modules) includes:

- Managing the access, authorization, and lifecycle of external requests
- Creating and sending requests to external users
- Provisioning temporary credentials that external users can run as when they access IBM BPM UIs externally
- Presenting coach-based human services to external users.

The primary architectural elements in the EPS toolkit are as follows:

- EPS toolkit

  A toolkit containing services that generate *tokens* used to authenticate external participants and to serve coach-based human services to them.

- EPS App

  A web application which handles requests containing the generated *tokens*, and maps them to BPM requests. The EPS App is an Enterprise Archive (EAR) file deployed to WebSphere Application Server.

- EPS Trust Association Interceptor (TAI)

  An EPS module using the mechanism by which WebSphere Application Server enables an external component to authenticate users and to assert their identity. Without a TAI, external requests cannot run under pre-provisioned user IDs.

When using the EPS toolkit, access to IBM BPM for external requests is achieved through the four-step sequence described in Figure 6-26.

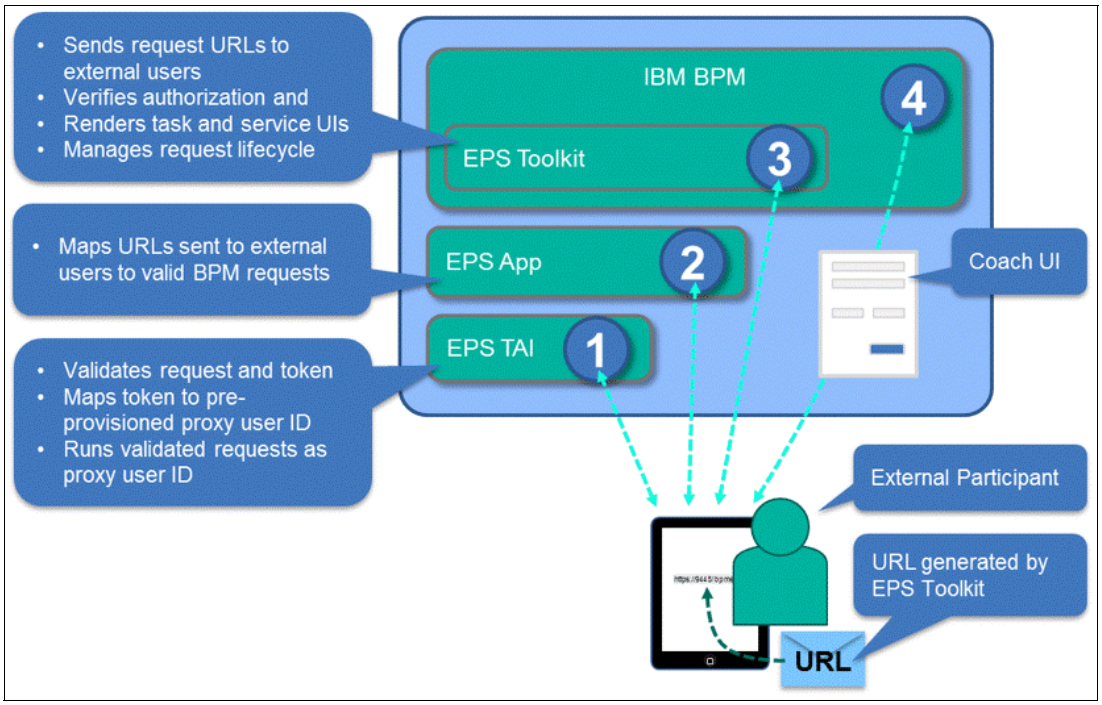


Figure 6-26 High-level access sequence for externally-served IBM BPM UIs

When step 4 is reached, the EPS toolkit lets IBM BPM render the coach UI and does not intervene again until the human service is completed by the external participant, at which point the URL sent to the external user becomes invalid.

Note: The diagram in Figure 6-26 is architecturally abstract on purpose. The EPS toolkit and its associated modules can add layers of security for external requests made from outside an organization's firewall. More detailed architectural diagrams are provided in “Access options” on page 353.

There are multiple components needed to install and configure for the EPS toolkit. The instructions for installation of these components can be found on the Salient Process support site.[10]

Indirectly expose human services in isolation

This section describes the Isolated business service usage scenario for the Indirect Exposure subpattern described in Table 6-4 on page 340. In this scenario, a BPM human service needs to be used by an external participant in isolation: meaning not as part of a BPM process. The *S*tartable Service, Dashboard and URL options in Figure 6-18 on page 341 expose a client-side human service as an isolated business service.

[10] Salient Process support site for installation details, http://ibm.biz/installEPS

A special General System Service (named “Assign And Notify External Service Participant”) in the EPS toolkit can be used to programmatically generate a perishable[11] URL and send an email to an external user. The URL generation and human service access scenario is described in Figure 6-27.

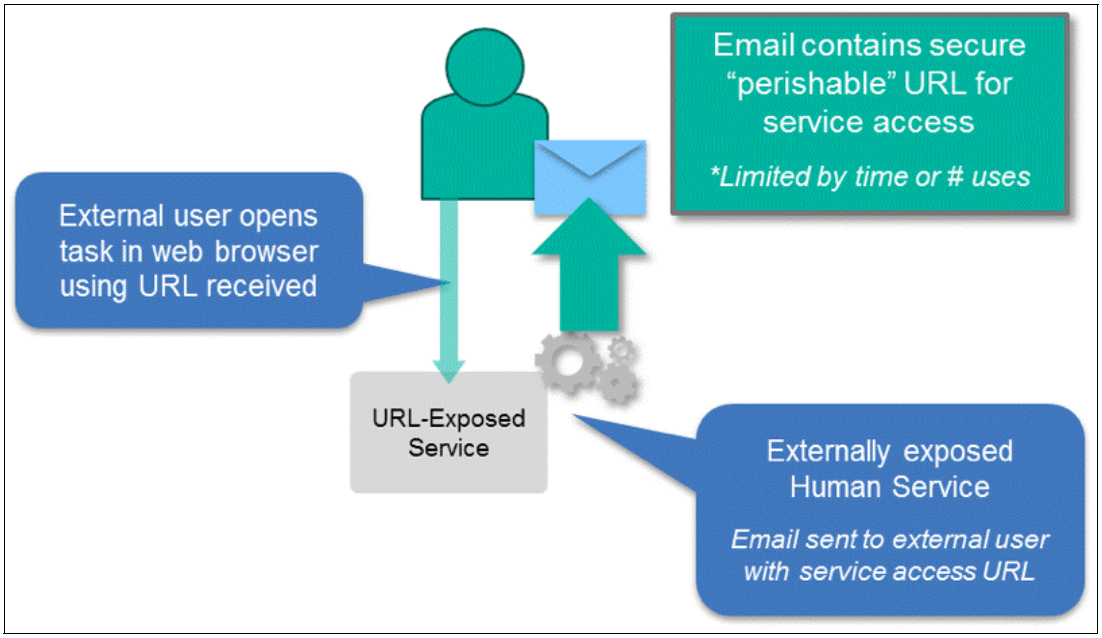


Figure 6-27 Exposed human service in isolation using the EPS toolkit

The diagram in Figure 6-28 and its explanation provide details about the steps in the EPS toolkit to support the Exposed human service in isolation usage scenario.

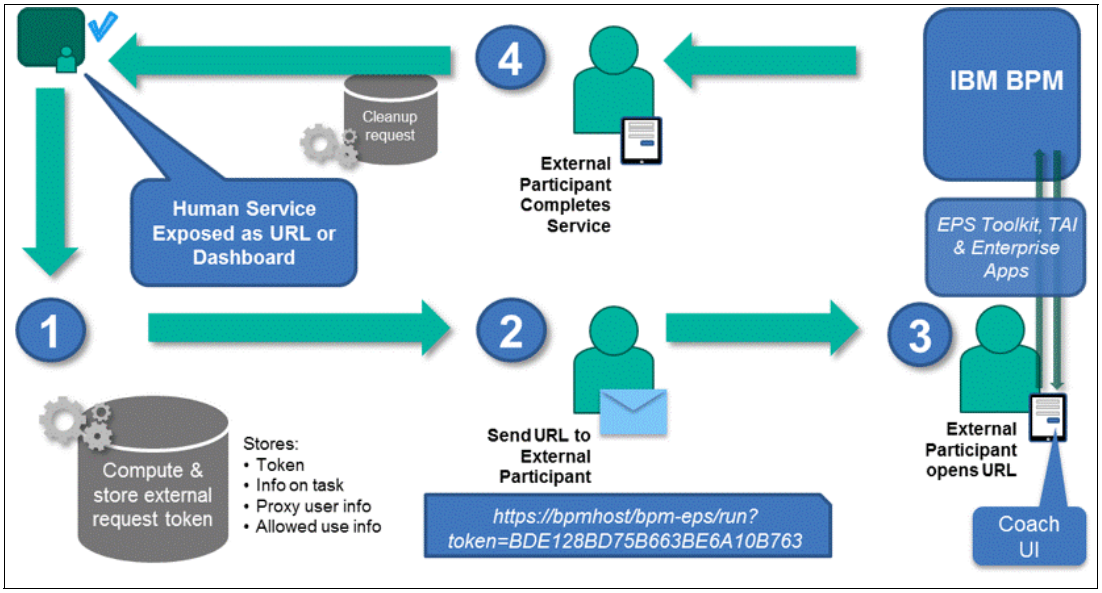


Figure 6-28 Exposed human service in isolation using the EPS toolkit: Completion sequence

[11] A URL that is only available for a limited amount of time or number accesses, after which it expires.

The sequence in Figure 6-28 consists of the following detailed steps (each step is numbered after the corresponding step numbers in the diagram):

1. An EPS request token is generated and stored in the External Participant table in one of the IBM BPM internal databases (this table is created by running a service inside the EPS Toolkit), along with some key information that is used by the EPS web application:
   - Service information: Service name, process app name, and potentially service invocation context, are stored to ensure the correct service is picked up when the external user clicks the URL.
   - Proxy user information: Depending on the configuration options set during the registration of the EPS task, this can be a unique user ID that is created ahead of time, or a new user ID can be generated dynamically to allow for more security, by preventing the ID from being used again. The external participant will authenticate against IBM BPM using the proxy user.
   - Valid use information: This includes the request date, the amount of time the token will be valid for, the maximum number of allowed requests, and the count of the requests to access the service.

   Note: If the token expires before the user accesses the URL, the human service is not e accessible without generating and sending out a new request.

   Note: The token for the EPS invocation is randomly-generated and contains no encrypted credentials or related information.

2. The generated token is then used as a part of the URL that is created and sent to the external user.
3. When the external user accesses the URL, the token is checked for validity twice before human service access is allowed:
   - First the EPS TAI authenticates the external user against IBM BPM as the proxy user mapped to the token.
   - Then the EPS App maps the request to a valid BPM request.
4. The external user can, at this point, interact with the coach-based human service in the browser. When the external user completes the task, the token is invalidated, and the URL can no longer be used. Alternatively, if the expiration time or the maximum number of allowed accesses is reached, the token is also invalidated.

Indirectly expose process instance tasks

This section describes the Correlated business service usage scenario for the Indirect Exposure subpattern described in Table 6-4 on page 340. In this scenario, a running BPM process needs to involve an external participant. The Task option in Figure 6-18 on page 341 exposes a client-side human service as a Correlated business service.

In Figure 6-29, when the process execution reaches the point in which control needs to be handed to an external user, a team filter service (or a general system service[12]) is fired, which registers a request for the external participant action. The URL generated for the external request can be sent to an external user via email.

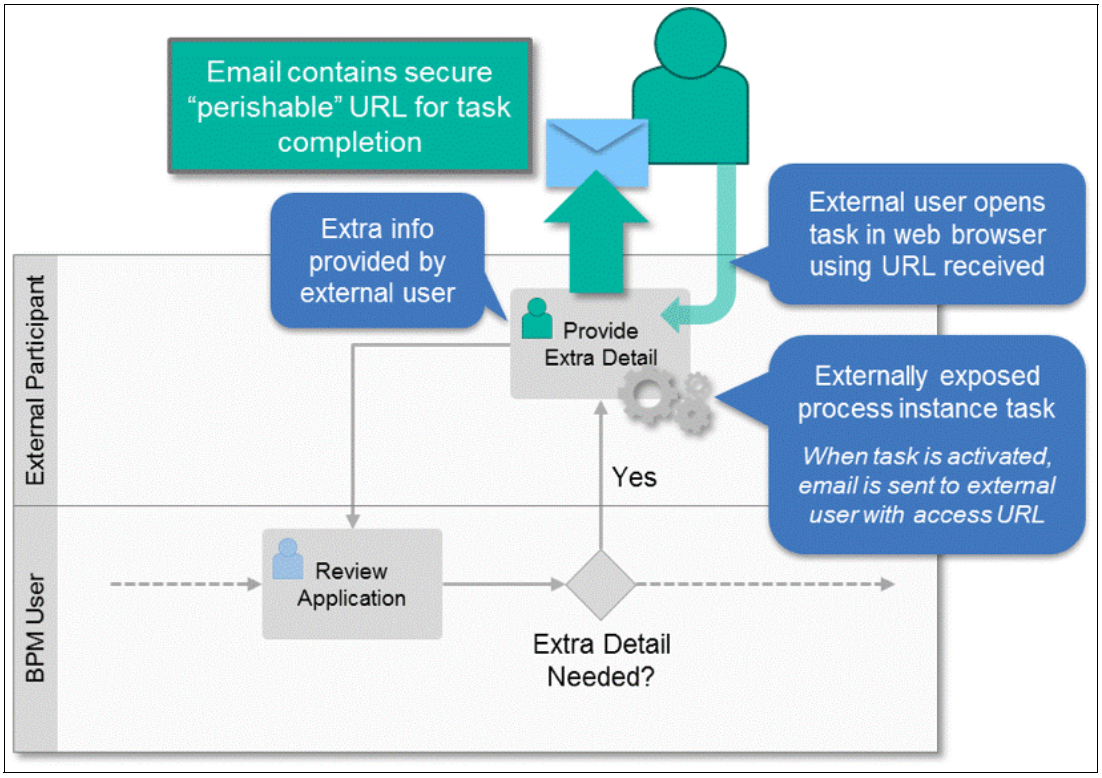


Figure 6-29 Exposed process instance task using the EPS toolkit

[12] To choose between a team filter service and a general system service refer to http://ibm.biz/exposeEPS

Figure 6-30 describes the steps in the EPS toolkit to support the Exposed process instance task usage scenario. Those steps are similar to the ones described in Figure 6-28 on page 350. The only difference is that no service information is stored in the IBM BPM database tables, as in this scenario the task ID is sufficient to identify and correlate the exposed human service to the correct process instance.

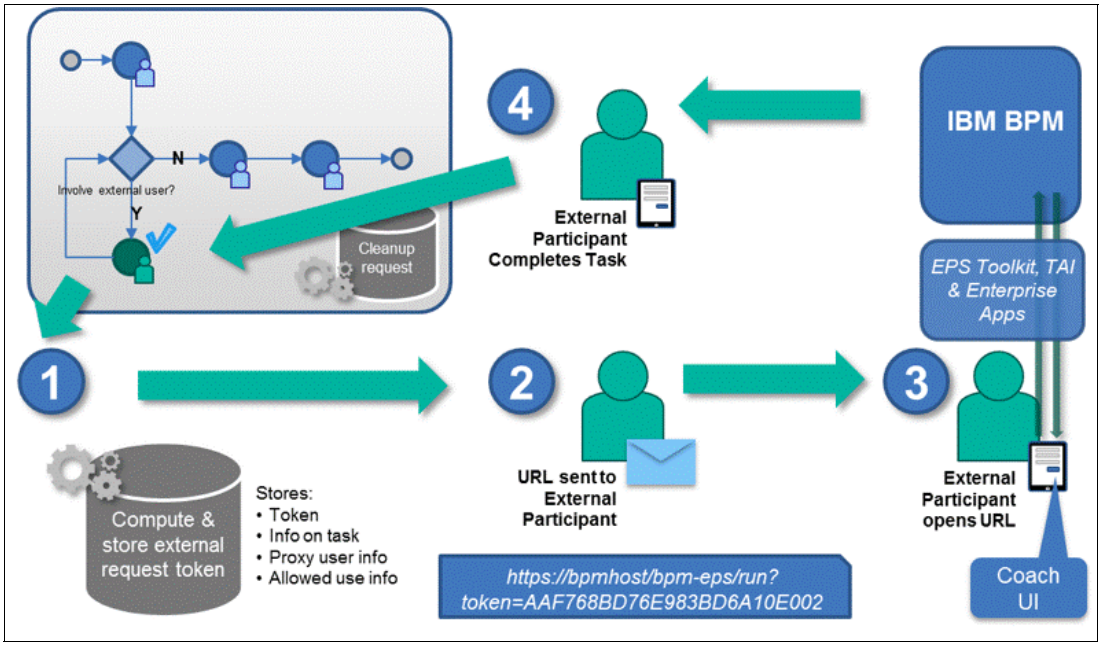


Figure 6-30 Exposed process instance task using the EPS toolkit: completion sequence

Access options

Both Exposed human service in isolation and Exposed process instance task usage scenarios behave similarly on the surface, whether accessed by external participants inside or outside the enterprise (meaning in front or behind the firewall). The two access options are further explained in the following sections, along with their security and architectural differences.

Inside the enterprise

The EPS toolkit allows BPM and non-BPM users within an organization (who are behind the firewall) to interact with IBM BPM processes and services indirectly. The toolkit handles the authentication of the user through the use of generated tokens which are perishable[13] based on configurable settings within the toolkit. The architectural diagram in Figure 6-31 on page 354 shows the various layers involved in processing external participant requests after a user accesses the URL sent by the EPS toolkit.

[13] A token which is used in a URL that is only available for a limited amount of time or accesses, after which it *expires*.

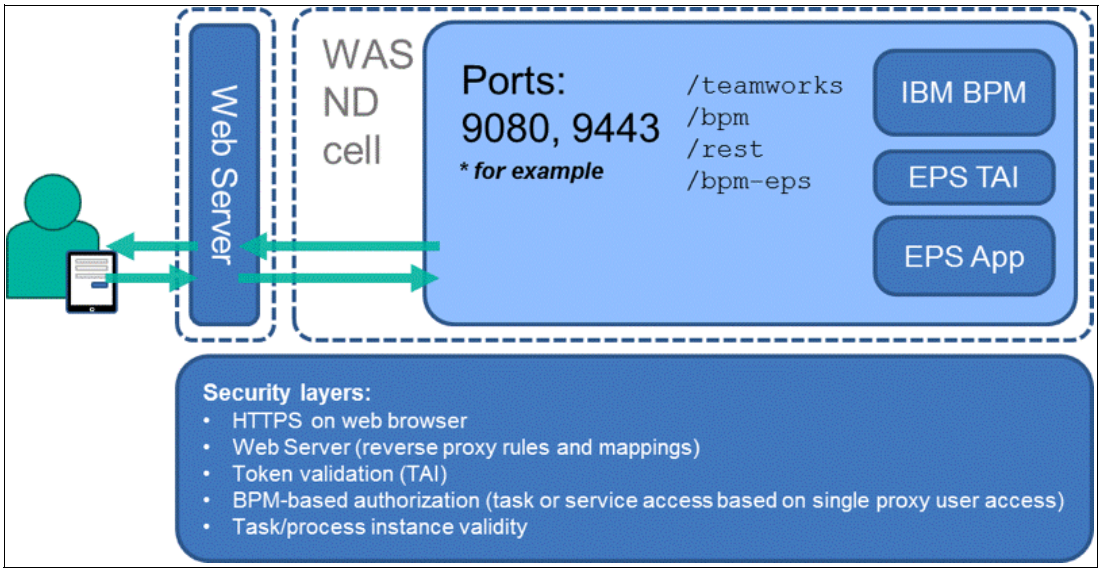


Figure 6-31 Inside the enterprise BPM UI access by external participant

From inside the enterprise, after the EPS TAI establishes a session for the external request (using the previously-provisioned proxy user ID for the external interaction), the original request URL is rerouted to the direct access URLs to start tasks, access services, make REST API calls, complete tasks, and so forth.

Note: The “inside the enterprise” option can potentially give an external participant as much control over IBM BPM as a regularly defined BPM user. BPM authorization is enforced, but any BPM interaction that does not require specific authorization can be performed by the external user.

Outside the enterprise

In some cases, processes might require participation of users that are external to the enterprise and outside the firewall. This poses several security challenges, including the need to allow users to authenticate against IBM BPM without exposing the system to attacks from malicious parties trying to use their authenticated session, as discussed in 6.1.3, “Accessing tasks externally” on page 318.

The mechanisms described in “Inside the enterprise” on page 353 still applies to this option when the EPS toolkit is used. The main difference is in the additional layers of security that external requests must go through before reaching the BPM server. These layers are depicted in Figure 6-32.

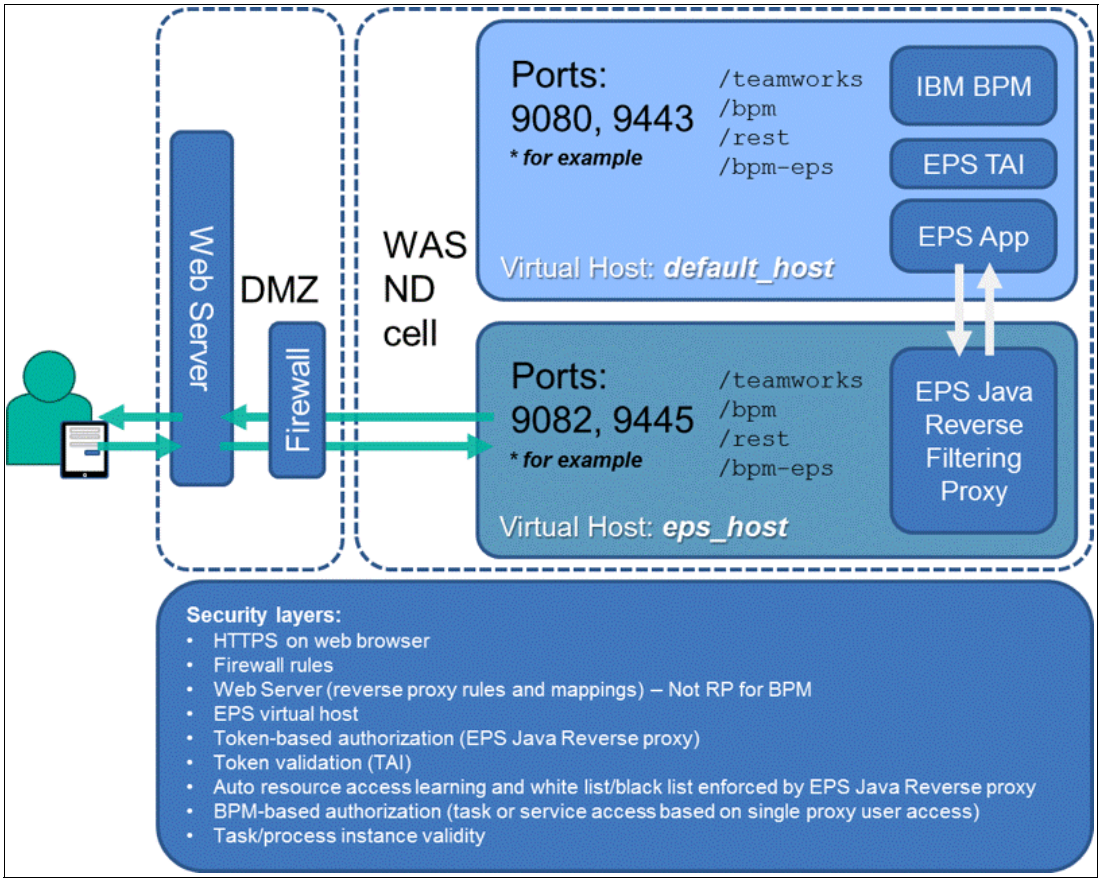


Figure 6-32 Outside the enterprise BPM UI access by external participant

This scenario introduces an additional Java reverse proxy module, running on dedicated ports, that maps to a separate virtual host in the WebSphere Application Server environment running IBM BPM. With a firewall that only allows access to those dedicated ports and a public-facing web server located in front of the firewall, the EPS reverse proxy becomes the only access medium to IBM BPM from outside the enterprise's firewall. This allows the proxy to enforce constrains based on black lists and white lists, to narrow the interactions of external users with IBM BPM.

The EPS proxy module also provides a *learning mode*, which intelligently captures the interactions expected for a particular task or human service. When a process instance task or human service in isolation is indirectly exposed through the EPS toolkit, it is associated with an *interaction*. An interaction is an EPS concept that allows the gathering and grouping of all server requests related to the exposed task or service. In learning mode, the EPS Proxy records all requests for an interaction (for example, initial load of a task for a particular process, access of AJAX services to populate select control items, and so on) and adds variability when appropriate to tolerate different *IDs* for tasks, process instances and other parts of the interaction at run time.

The learning mode effectively creates an adaptive white list that is enforced at run time for all external interaction tokens associated with a particular task or human service. This allows

organizations to restrict what can be done with IBM BPM by external users to a narrow, situation specific, and expected set of requests captured and learned by the EPS proxy.

Security considerations for the EPS toolkit

The EPS Toolkit, especially if using the mechanisms described in the “Outside the enterprise” option, protects the IBM BPM environment through several additional layers of security (as described in Figure 6-32 on page 355). This section lists those layers from the outside-in and provides details about their respective role.

- Web browser

  Uses HTTPS to encrypt the data transfer between the external user and the web server for the duration of the session.

- Web server

  Can be configured to prevent Denial of Service (DoS) attacks.

  Can enforce Origin Header checks to mitigate CSRF attempts.

- Firewall rules

  Can be configured to constrain access to the WebSphere Application Server environment running IBM BPM to the two ports used by the EPS reverse proxy module.

- EPS Virtual Host (WebSphere Application Server)

  Only exposes the EPS Reverse Proxy module on two configurable ports (for HTTP and HTTPS access).

- EPS Reverse Proxy
  - Provides standard configurable black lists and white lists to constrain allowed operations against the IBM BPM environment.
  - Provides adaptive while lists that are generated after capturing an external interaction in learning mode. This constrains allowed actions (meaning HTTP requests) to only the types of operations captured during the training phase.

- EPS Trust Association Interceptor (TAI)

  Only allows valid tokens (including non-expired tokens) to attempt to establish a session on behalf of an external participant.

- EPS Toolkit

  Only allows valid (available) tasks to be started by external participants.

- IBM BPM

  Only allows tasks and services to be opened by the correct proxy user ID specified for the external interaction.

Note: The configuration of the various security aspects (based on the options and security constraining capabilities discussed in this section) needed for a secure environment is up to system, network, WebSphere Application Server, BPM administrators and solution designers and developers.

Despite the additional security layers provided by the EPS toolkit, it is fundamental to make important security considerations *before* exposing IBM BPM human services over the Internet (“Outside the enterprise” access option), although indirectly. The most relevant of these considerations are:

- An *unexpected delegation* can occur, meaning that any external user with a valid EPS request URL contains a valid non-expired token mapping to an available exposed human

service can open the request and complete the task. Thus, an external user can potentially forward the URL to anyone to open and complete a task or service on their behalf. The EPS toolkit makes no attempt to use third-party authentication to verify that the person working with the human service is the original recipient of the external request URL.

- White-listing by the EPS proxy mitigates, but does not fully eliminate, the risks of CSRF. If a particular external interaction makes use of specific IBM BPM REST APIs (for example to query or alter the state of the IBM BPM environment), a malicious site might be able to take advantage of those APIs calls.
- Coach-based UIs containing non-BPM links (URLs from external sites) broaden the possibility of CSRF uses.
- If a single user is used for all EPS requests, no full auditing will be available for those requests, as they will be recorded as executed by the same user. On the converse, if distinct users are used, this choice can fill the BPM user registry with temporary, yet undeletable, user accounts.

Note: These considerations are valid at the time in which this book is being written. However, security and REST API hardening for IBM BPM human services are in continuous evolvement.

There might be several reasons for which the security provided by the EPS toolkit is not sufficient when wanting to expose BPM human tasks outside of the enterprise. In these circumstances, the Decoupled pattern described in 6.6, “Decoupled pattern” on page 367 should be used instead.

6.3.3 Summary of the Exposed pattern implementation options

The *Exposed pattern* is broken down into two subpatterns: *Direct* and *Indirect Exposure* of human tasks as business services. For both exposure subpatterns, usage scenarios for the business services might require a different implementation style in IBM BPM. Finally, access options for those business services also play a key role in the adopted implementation.

The available implementation options introduced in Table 6-4 on page 340 and discussed in this section, are summarized in Table 6-5, along with the relevant recommendations.

Table 6-5 The Exposed pattern and subpatterns: Options and recommendations, based on usage scenarios

	Direct Exposure subpattern (6.3.1, “Expose human services directly”)		**Indirect Exposure subpattern (6.3.2, “Expose human services indirectly through the EPS toolkit”)**	
	Isolated business services	**Correlated business services**	**Isolated business services**	
INSIDE the Enterprise	Directly expose human services in isolation: heritage human services and client-side human services as Dashboards, Startable Services and URLs	Directly expose process instance tasks: heritage human services and client-side human services as Tasks	Indirectly expose human services in isolation: heritage human services and client-side human services through the EPS toolkit	Indirectly expose process instance tasks: heritage human services and client-side human services through the EPS toolkit

	Direct Exposure subpattern (6.3.1, “Expose human services directly”)	Indirect Exposure subpattern (6.3.2, “Expose human services indirectly through the EPS toolkit”)	
OUTSIDE the Enterprise	Not recommended	Indirectly expose human services in isolation: client-side human services *only* through the EPS toolkit	Indirectly expose process instance tasks: client-side human services *only* through the EPS toolkit

6.4 Embedded pattern

The Embedded pattern in Figure 6-33 is task oriented. In this pattern an alternative task portal is used to consume human tasks implemented as IBM BPM coach-based human services. The alternative task portal is either an existing enterprise portal or a custom-built lightweight portal embedding the coach-based human services.

This pattern is intended for scenarios requiring basic task management with integration to human tasks implemented as coach-based human services, or to expose those task to an existing enterprise portal.

A custom-built lightweight portal usually accesses BPM via a set of reusable decoupling services, acting as service exposure layer, wrapping the BPM RESTful APIs. An enterprise portal typically accesses the BPM layer directly.

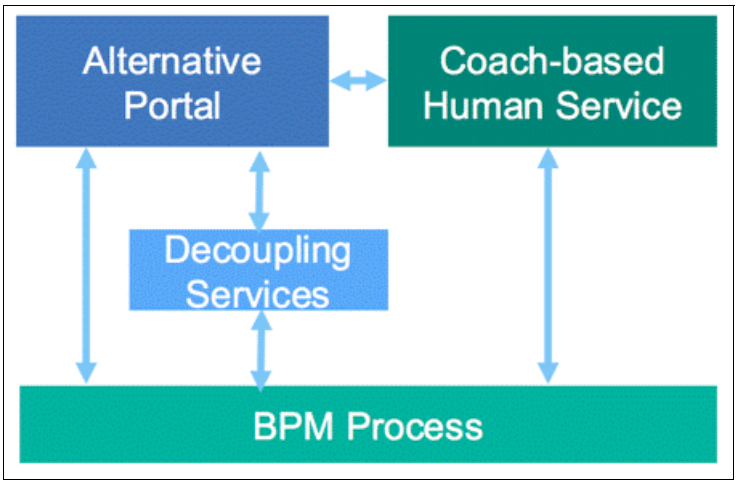


Figure 6-33 Embedded pattern

This pattern is primarily intended for internal staff users working across multiple processes and tasks.

Examples of alternative portals are:

- A lightweight portal built as a custom web application, leveraging the BPM REST APIs
- A mobile portal application, built leveraging the BPM REST APIs
- A custom portal created using the SPARK Portal builder toolkit, as described in 5.2.9, “SPARK Portal Builder toolkit” on page 305

- An enterprise portal, such as WebSphere Portal or Microsoft SharePoint

Note: Using a custom IBM Process Portal, as detailed in 5.1.7, “Performance dashboards” on page 269, falls into the Integrated pattern described in 6.5, “Integrated pattern” on page 366, as the customized portal process application leverages the same functionality of the standard IBM Process Portal.

The *Embedded pattern* is, from an implementation perspective, a combination of coach-based human services for human tasks, implemented according to the Exposed pattern, and an alternative portal to implement a task list.

The main option to embed coach-based human services into alternative task portals is to use an *inline frame* (iframe). An iframe is an HTML document embedding the human service within the task portal. This approach is valid independently from the task portal implementation choice.

Additionally, IBM BPM provides specific mechanisms to embed coach-based human services in the following alternative portal implementations:

- IBM BPM iOS mobile app
- WebSphere Portal

6.4.1 Embed a human service as an iframe in an alternative task portal

When using an alternative task portal, the functionality to interact with tasks is delivered by the portal itself: basic operations such as, get next task, claim task, and complete task are part of the core capabilities of a task portal.

Enterprise portals such as, WebSphere Portal and Microsoft SharePoint offer these capabilities (or integration with these capabilities) ready to use.

If the alternative task portal is implemented as a custom lightweight portal or as a mobile application, the IBM BPM REST APIs expose the standard functionality expected from a BPM task portal, and developers can select their technology of choice for its implementation.

The main mechanism to embed a coach-based human service in an alternative task portal is to expose the human service as an HTML document, and to reuse that document within the portal. The direct use of BPM human services as an HTML document has been discussed within Exposed pattern (see 6.3, “Exposed pattern” on page 338).

In fact, embedding a human service in an iframe corresponds to expose the human service through a URL, and embedding the HTML served by that URL in the alternative task portal. All additional functionality around task manipulation should be absolved by the task portal implementation.

In essence, the *Embedded pattern* using an iframe uses the same principles described in the Exposed pattern to expose coach-based human services, but serves them to the users for manipulation through an alternative task portal implementation.

An example of a technique to implement the Embedded pattern is described in the article *BPM YouPortal,* at web address:

`http://ibm.biz/BPMYouPortal`

6.4.2 Embed a human service in the IBM BPM iOS mobile app

An iOS application for iPad and iPhone is provided as a sample implementation of the Embedded pattern for mobile apps. This sample application implements a lightweight portal mobile app that follows the principles described in 6.4.1, “Embed a human service as an iframe in an alternative task portal” on page 359. It exposes coach-based human services as iframes and builds a task portal using the IBM BPM REST API.

You can download IBM BPM for use with iPhone and iPad from the Apple App Store or from iTunes:

http://ibm.biz/BPM_App

Provided the runtime Process Server is exposed to the mobile device, either over the intranet or via VPN, the app is pointed at the Process Server host, as shown in Figure 6-34.

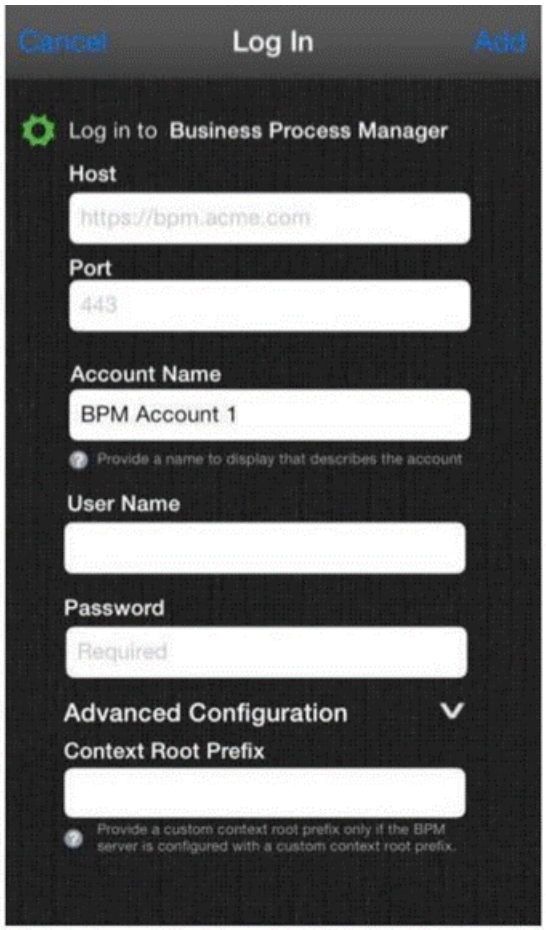


Figure 6-34 Configuration of the IBM BPM iOS app

Where:

- Host is the URL of the Process Server
- Port depends on your system configuration. Default port numbers are *9080* (HTTP) and *9443* (HTTPS)
- Account Name is *name* to identify the Process Server instance within the app
- User Name is the user name that will be used to login to the Process Server

- Password is the password of that user
- Advanced Configuration, Context Root Prefix is the context root prefix for the deployment environment you want to connect to

Configuration parameters: When using the sample iOS app with IBM BPM on Cloud, use the following configuration parameters:

- Development Environment context root prefix: `bpm/dev`
- Test Environment context root prefix: `bpm/test`
- Run Environment context root prefix: `bpm/run`

When configured, the app allows you to launch new processes (Figure 6-35), to view the task list, claim tasks and work with them, add attachments, leave comments, and so on (Figure 6-36 on page 362).

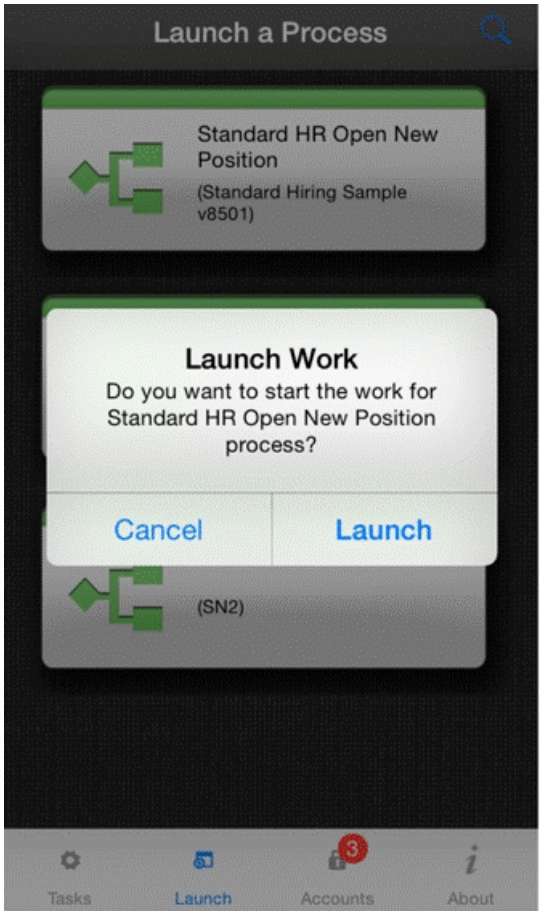


Figure 6-35 Launch a new process from the IBM BPM iOS app

Figure 6-36 Work with the IBM BPM iOS app

6.4.3 Embed a human service in IBM WebSphere Portal

The IBM enterprise portal, WebSphere Portal, provides a framework to make the implementation of the Embedded pattern out of the box. This framework provides two main mechanisms to embed BPM coach-based human services in WebSphere Portal, as shown in Figure 6-37 on page 363:

- Generating JSR 286 portlets (only available for heritage human services exposed as *Dashboards*)
- Embedding human services into the Unified Task List portlet

Generate at JSR 286 portlet

Commonly used to standardize portal and portlets interactions, Java Specification Request (JSR) 286 is a Java Portlet Specification that defines a contract between a portlet container (a portal) and portlets. It provides a programming model for Java portlet developers and includes event support for interactions on portal pages. You can find more information about the JSR 286 specification at:

`http://ibm.biz/UnderstandJSR286`

WebSphere Portal can consume portlets adhering to JSR 286 specification.

IBM BPM heritage human services exposed as Dashboards can be used to generate JSR 286 portlets, which can be embedded into WebSphere Portal (Figure 6-37).

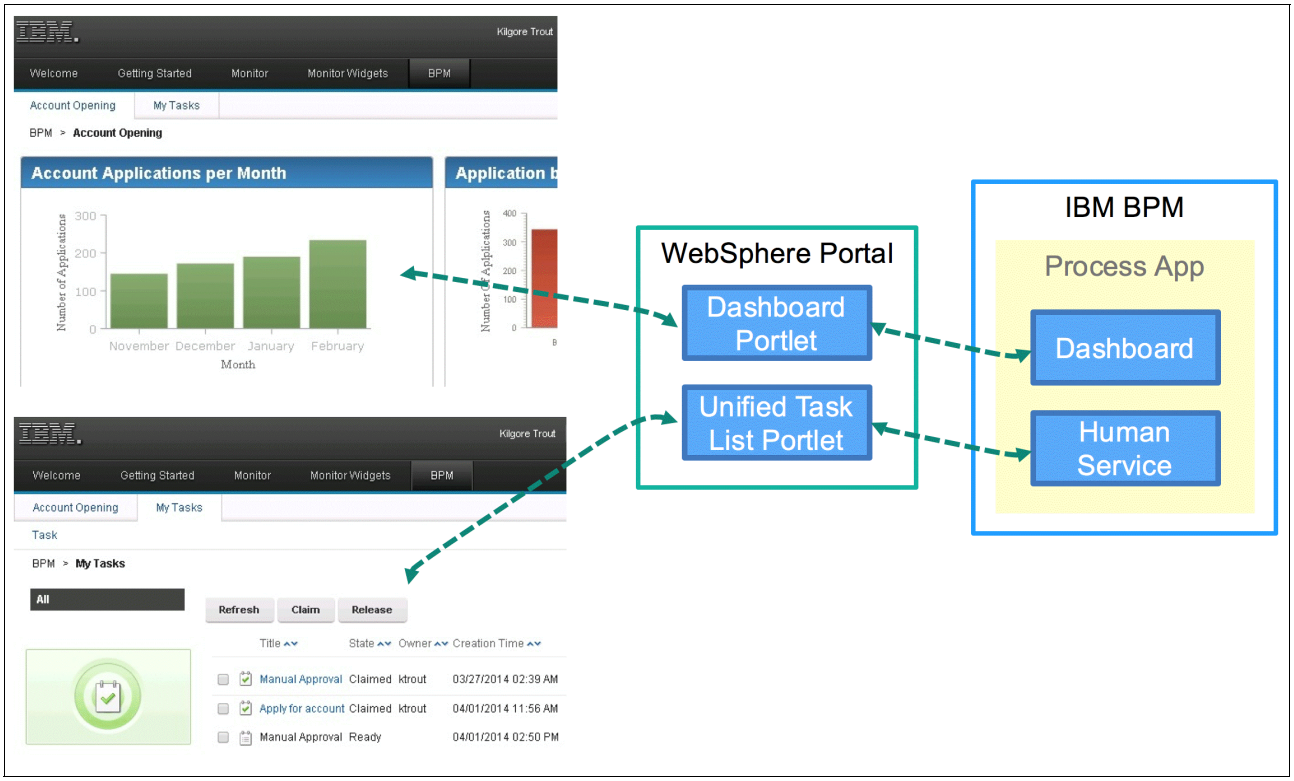


Figure 6-37 Embedded pattern in IBM WebSphere Portal

Note: BPM human services exposed as Dashboards are intended to be used as Isolated business services, as described in 6.1.4, “Exposing a process via a business service” on page 321, and not as part of an overarching process.

This feature is not available for client-side human services.

Using this mechanism, Process Designer generates a JSR 286 portlet, embedding the heritage human service exposed as a Dashboard, and produces a web archive file (WAR file) that can be installed in WebSphere Portal, as shown in Figure 6-38.

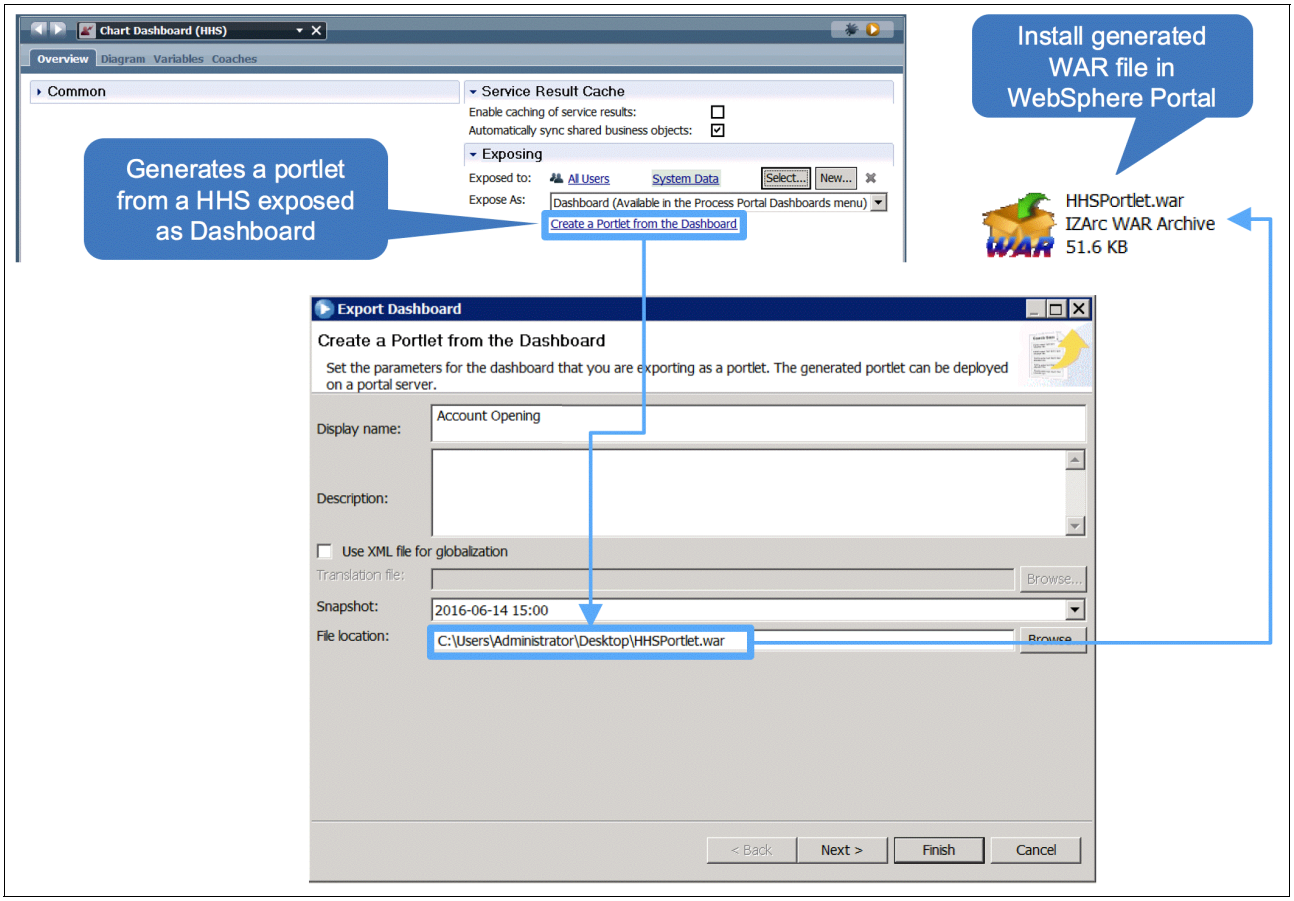


Figure 6-38 Generating a JSR 286 portlet for heritage human services exposed as Dashboard

You can either deploy the WAR file on the BPM Process Server and access it through the Web Services for Remote Portlets (WSRP) 2.0 protocol or deploy it directly to the WebSphere portal server.

A full set of instructions to use the generated JSR 286 portlets is available in IBM Knowledge Center:

`http://ibm.biz/BPMGeneratePortlet`

An example, available in the `Custom_Report_Dashboard.twx` and `SG24-8355-ch6-HHSPortlet.war` files is shown in Appendix A, “Additional material” on page 395.

Figure 6-39 on page 365 shows the heritage human service dashboard embedded in JSR 286 portlets in WebSphere Portal.

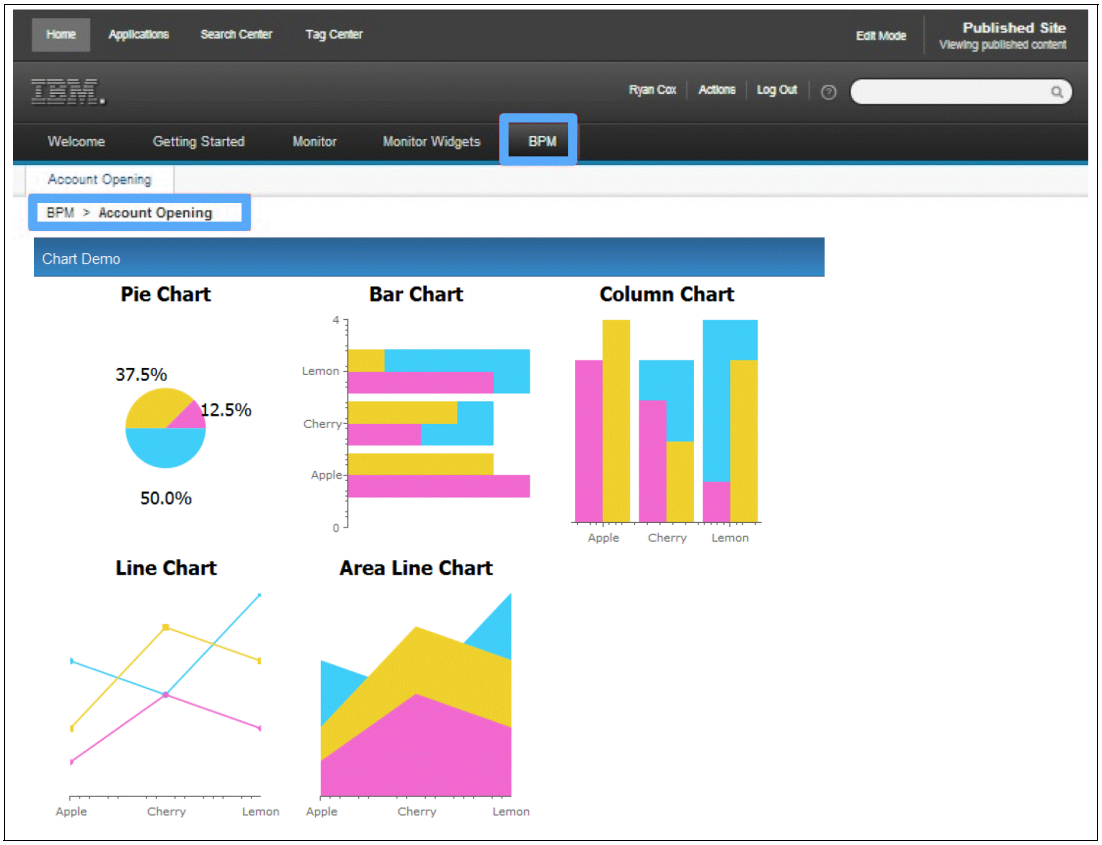


Figure 6-39 Heritage human service dashboard embedded in JSR 286 portlets in WebSphere Portal

Use the Unified Task List portlet

The Unified Task List (UTL) is a portlet which aggregates tasks and activities from multiple systems into a single UI, implementing the task portal functionality of the Embedded pattern as a portlet running in WebSphere Portal. Users can complete human tasks using task completion portlets available within the Unified Task List. Those portlets directly embed the coach-based human service selected in the UTL, as shown in Figure 6-40.

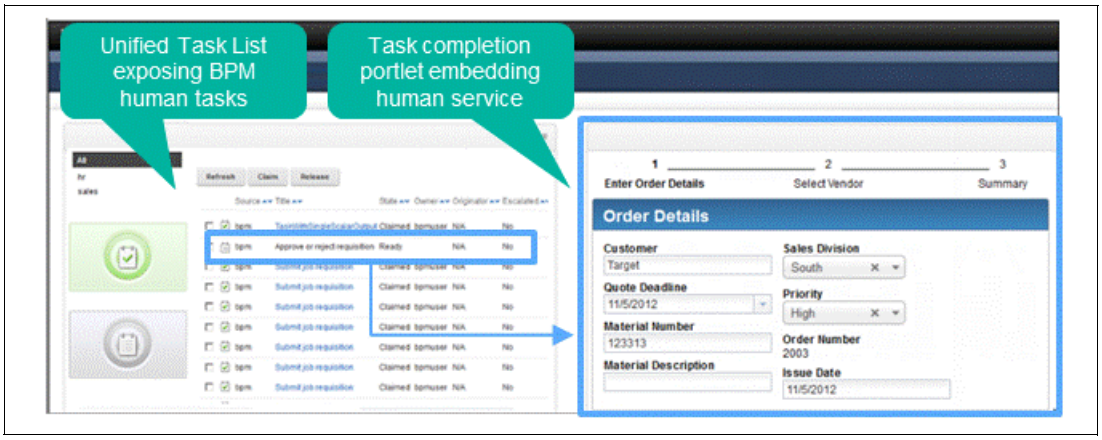


Figure 6-40 Using the UTL to implement the Embedded pattern in WebSphere Portal

The task completion portlet embeds the coach-based human service using an iframe. Therefore, the UTL mechanism is essentially a ready to use implementation of the principles

as described in 6.4.1, “Embed a human service as an iframe in an alternative task portal” on page 359, applied to IBM WebSphere Portal.

You can download the Unified Task List from the IBM WebSphere Portal Unified Task List Portlet web page:

http://ibm.biz/PortalUTL

You can download detailed instructions about how to configure the UTL to work with IBM BPM at:

http://ibm.biz/UTLusage

6.5 Integrated pattern

The Integrated pattern in Figure 6-41 is task oriented. In this pattern, IBM Process Portal is used for task management and all other interactions with the BPM system. Human tasks executed within the portal are implemented as IBM BPM coach-based human services.

This pattern provides users with the powerful functionality of IBM Process Portal.

In essence, this pattern implements the ready to use integrated UI provided by IBM BPM and depicted in Figure 6-5 on page 318, which has been described in Chapter 2, “Creating user interfaces with coaches” on page 9 and Chapter 5, “IBM Process Portal” on page 261 of this book. This functionality is delivered by IBM Process Portal and coach-based human services. Both client-side and heritage human services are used in this pattern.

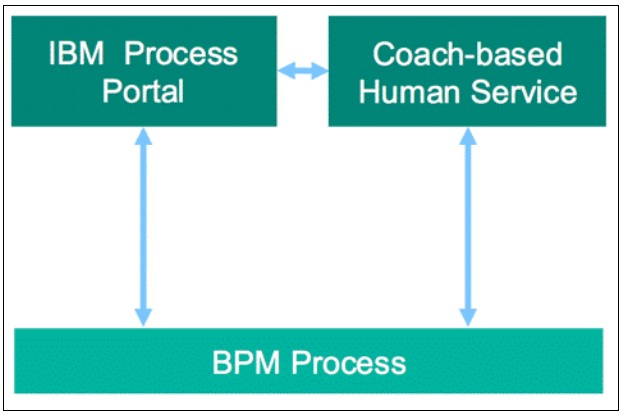


Figure 6-41 Integrated pattern

This pattern is primarily intended for internal users who require the full functional breadth of IBM Process Portal.

Note: Using a custom IBM Process Portal, as detailed in 5.2.7, “Clone Process Portal” on page 286, also falls into this pattern, as the customized portal process applications leverages the same functionality of the standard IBM Process Portal.

Note: A variant of the Integrated pattern, in which coaches in human services embed an iframe, is discussed in 6.9.2, “Embed iframes in coaches” on page 388.

6.6 Decoupled pattern

The Decoupled pattern in Figure 6-42 is user-experience oriented. In this pattern a human task is exposed to the users as a business service, as defined in 6.1.4, “Exposing a process via a business service” on page 321. There is no need for a task portal, as users are unaware of the existence of BPM human tasks, they simply interact with an application.

In this pattern, the user experience is provided by an external UI implementation decoupled from IBM BPM (for example, a web application using a different UI technology) for a human task, which integrates with BPM at specific points to update an underlying, loosely-coupled process model. The application controls all user interactions and screen flows, providing a rich user experience that is driven primarily by user requirements and not by the underlying process model structure.

BPM coach-based human services are not used in this pattern.

The external UI implementation usually accesses BPM via a set of reusable decoupling services, acting as service exposure layer, wrapping and securing the BPM RESTful APIs.

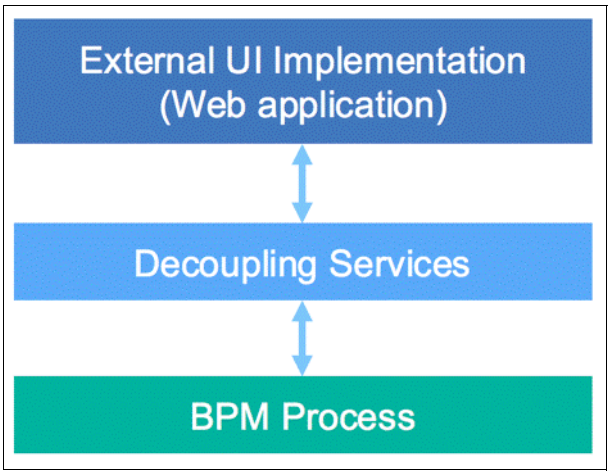


Figure 6-42 Decoupled pattern

This pattern is primarily intended for users external to the enterprise, typically accessing a web site or using a mobile app over the public Internet.

The Decoupled pattern is implemented by creating an external UI implementation for a BPM human task, in any chosen technology, as discussed in 6.6.1, “Use an external UI implementation within a BPM process” on page 367. In 6.6.2, “Use IBM MobileFirst to implement the Decoupled pattern” on page 373, a set of sample implementations is provided, using IBM MobileFirst™ Platform Foundation, the IBM platform to build, secure, and deploy mobile applications.

6.6.1 Use an external UI implementation within a BPM process

This section illustrates how IBM BPM can be used to easily integrate with external UI implementations, implementing the Decoupled pattern.

In the example in Figure 6-43 on page 368, a business process defined in BPM is made up by two human tasks and a system task. The human tasks are to be implemented using an external UI, each of them representing, respectively, an independent implementation of the Decoupled pattern.

The two human tasks depicted in this section are not part of a single unit of work, and have been modelled as two separate BPM tasks. They follow the Correlated business service usage scenario defined in 6.1.4, “Exposing a process via a business service” on page 321.

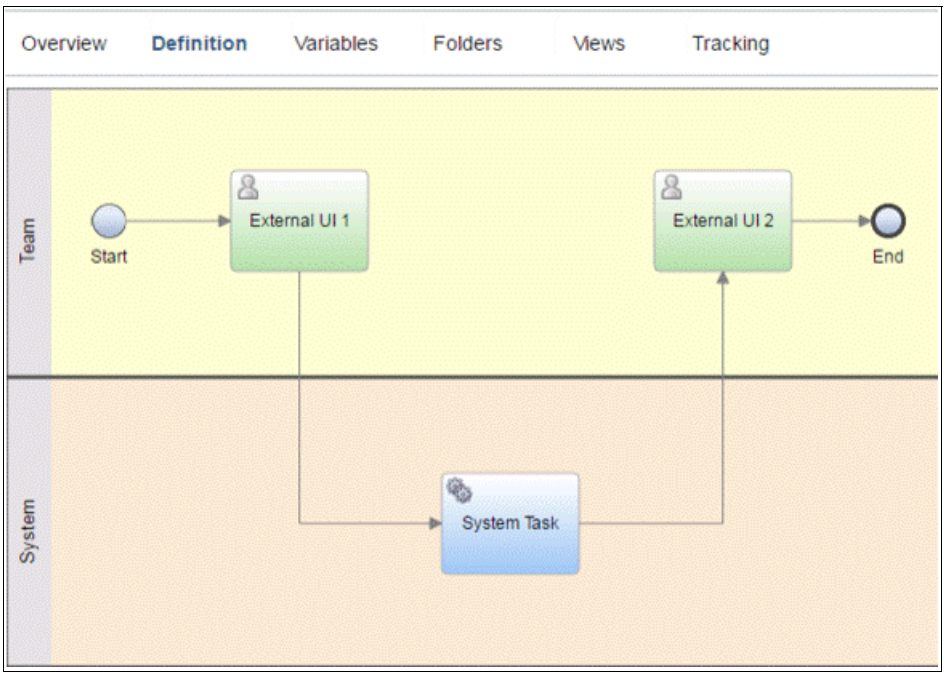


Figure 6-43 Sample process with two external UI implementations

For this process, a number of private variables has been defined, as described in Figure 6-44.

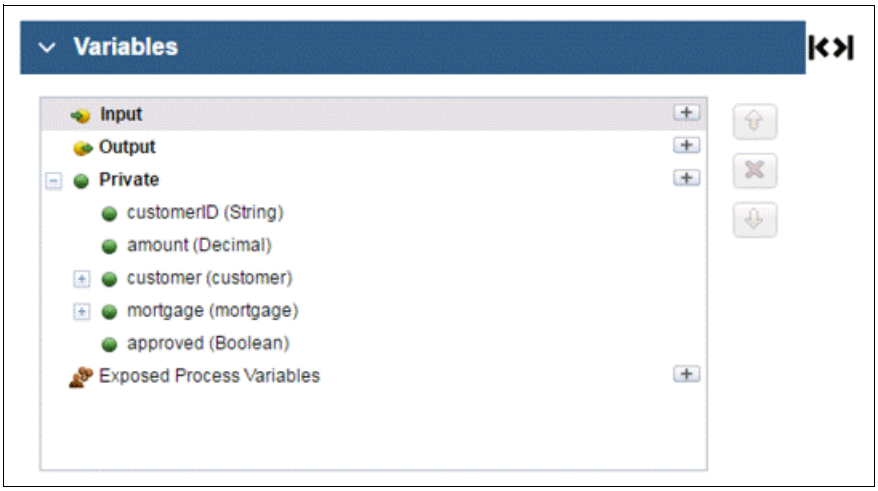


Figure 6-44 Sample process with its private variables

To allow the two human tasks in the process to be implemented in a separate presentation layer, they have to be defined in BPM as External Implementations, as shown in Figure 6-45 and Figure 6-46.

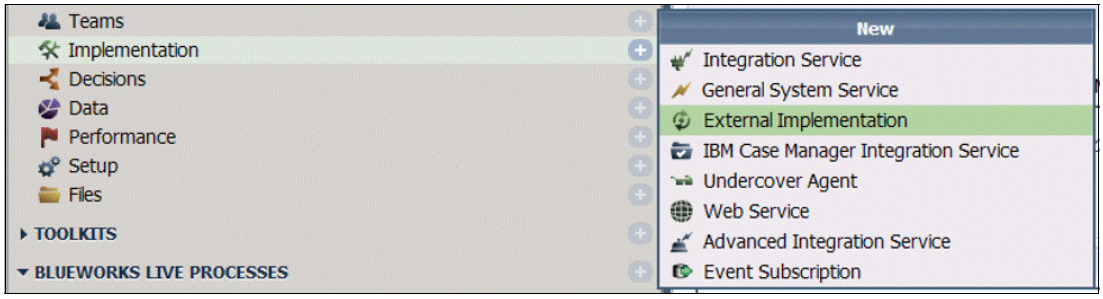


Figure 6-45 External implementation

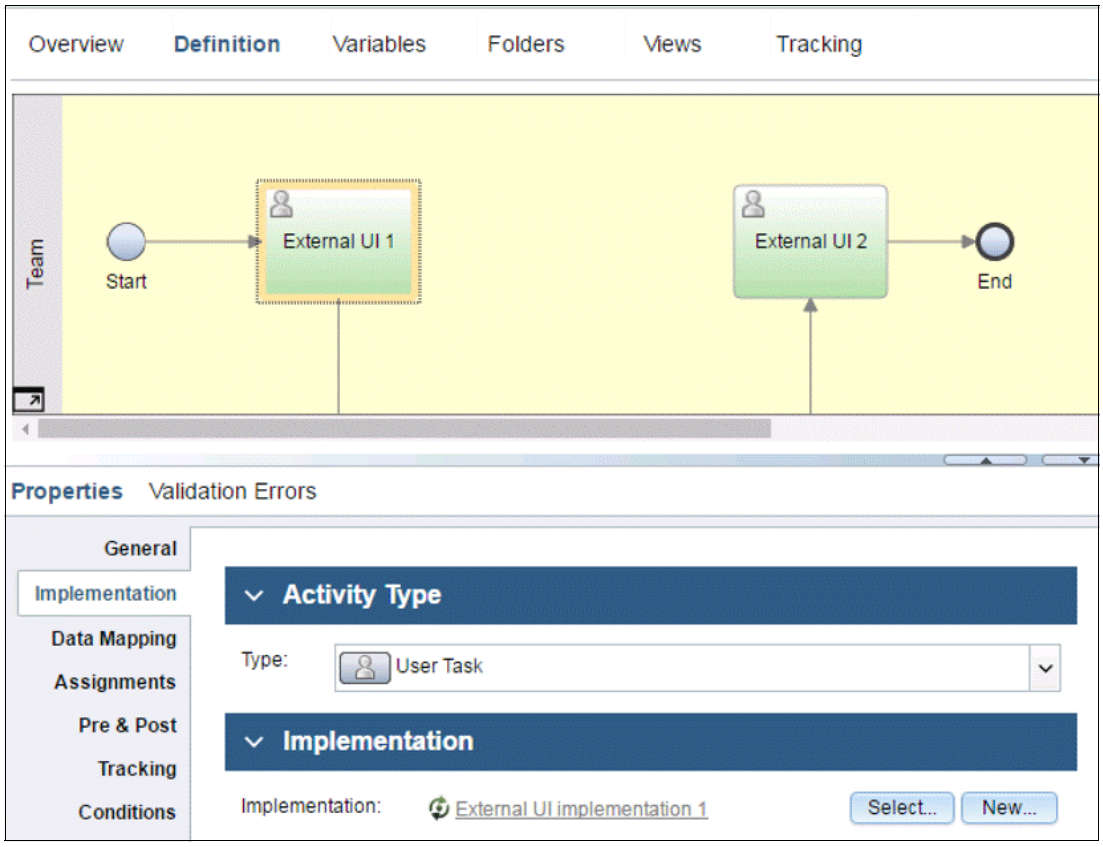


Figure 6-46 Implementing a human task as an external implementation

An External Implementation is a type of implementation service in IBM BPM that allows the process developer to define the following configuration options for an externally implemented human task, as depicted in Figure 6-47 on page 370:

- The *input* and *output* parameters: The parameters passed to and received from the external UI implementation.
- The exposed URL[14] of the external UI implementation, or a URL template using internal BPM variables for replacement at run time.[15]
- Ajax authorization services.
- Custom properties.

Note: Specifying a URL is not a required step to implement the decoupled pattern, as the external UI implementation will claim the task and interact with it independently from a URL being specified in the external implementation configuration. In turn, it is a required parameter to implement the Hybrid pattern described in 6.8, “Hybrid pattern” on page 382.

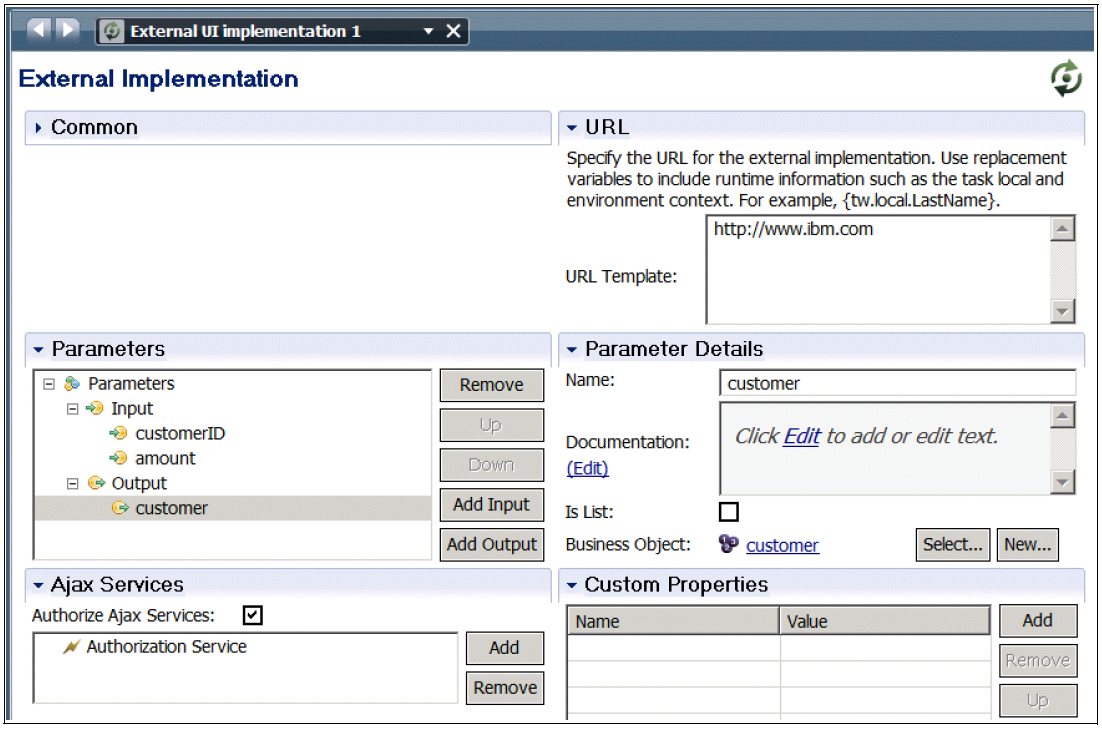


Figure 6-47 External Implementation configuration options

After defining the input and output parameters for the external implementation, they can be mapped to the process variables, as described in Figure 6-48.

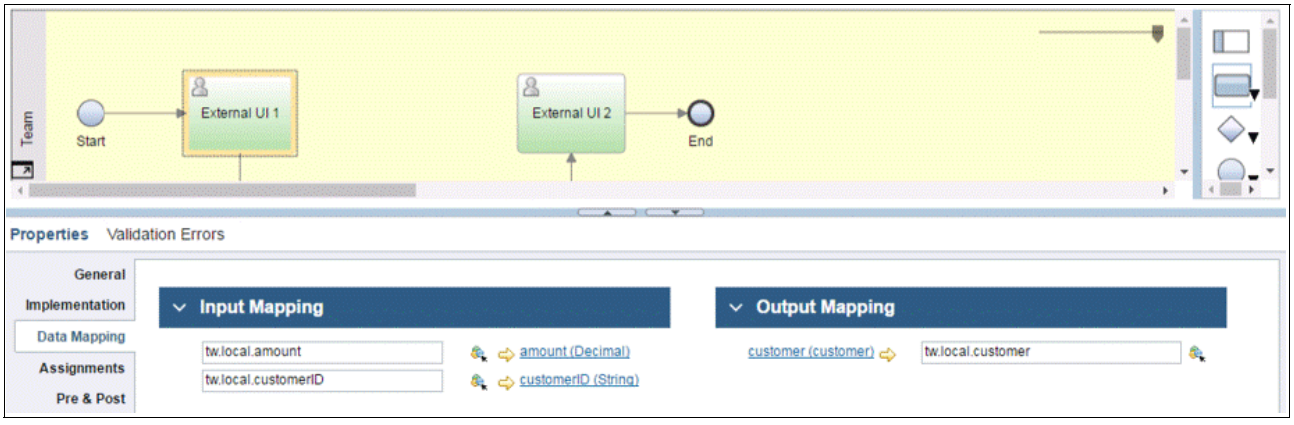


Figure 6-48 Parameter mapping for external activity

14 Both fully qualified URLs and relative URLs are supported. As an example, the following are both valid URLs for external implementations: `http://www.ibm.com/ExternalImplementationSample/ExternalImp.jsp` (redirects to the same server where IBM BPM is running)

15 IBM variables can be passed in the URL as in this example: `/ExternalImplementationSample/ExternalImp.jsp?taskId=<#=tw.system.task_id#>?amount=<#tw.local.amount#>`

The same procedure can be applied for the second task, to link it to a separate external implementation. Finally, the system task would typically be a call to a back-end system or to a service exposed through an enterprise service bus, and its exposure layer.

In the example, when a step implemented as an external implementation is reached during the execution of a business process, the execution stops and IBM BPM waits for an external system to complete the activity. To move the business process forward, the system or application implementing the external UI programmatically connects to IBM BPM and claims and completes the activity. To do that, it uses the IBM BPM REST APIs described in 6.1.5, “IBM BPM REST APIs” on page 323, and specifically the task APIs.

A sample implementation of this example is available in the `SG24-8355-ch6-patterns.twx` file attached to the Appendix A, “Additional material” on page 395 of this book.

You can find a full description on the use of external implementations in IBM BPM Knowledge Center:

`http://ibm.biz/BPMExternalActivities`

When designing external UI implementations for IBM BPM, it is important to consider two fundamental aspects that go beyond UI design:

- What is the correct way to invoke services?
- How can the external UI implementations be triggered?

These two aspects are described in the remainder of this section.

Invoke services from an external UI implementation

It is fundamental to note that the IBM BPM REST API does not allow invocation of services other than human or Ajax, as a service bus or a service exposure layer would do. As an example, *Advanced Integration Services* and *General System Services* are not exposed via the IBM BPM REST API. This is not a limitation of the product, but a correct architectural choice: if BPM system services are to be reused, they should be formally exposed through a service bus or a service exposure layer, and not simply through the BPM layer’s REST API. The benefits of a formal service exposure layer have been described in 6.1.6, “Pros and Cons of a formal service exposure layer for BPM APIs” on page 326.

As a consequence, when external UI implementations need to invoke system services that are not simple Ajax calls, they should do so via a service exposure layer.

Furthermore, this architectural choice makes it more difficult to fall in a common anti-pattern encountered when using external BPM UI implementations: *All services are System Tasks*, described in 6.1.9, “External UI integration: anti-patterns” on page 334.

Trigger external UI implementations

In the example in Figure 6-43 on page 368, because no task portal is used in the Decoupled pattern, when BPM execution reaches the second human task, no direct notification is sent to the users (unless a notification mechanism is explicitly modelled in the process). For the correlated business service’s external UI implementation (External UI 2) to be triggered, it needs to know that the process has reached that stage of the execution.

The recommended approaches to trigger an external UI implementation for a Correlated business service are:

- Create a mechanism to notify users that a task is available, so that they can invoke the external UI implementation manually.

- Programmatically push a notification to the external UI implementation, rather than relying on it to *wait* or *poll* until the human task is available to be claimed.

To notify the user, for example, via email, the same mechanism described in section “Directly expose process instance tasks via email” for the Exposed pattern can be used for the Decoupled pattern. An email notification with a link to the external UI implementation is triggered when the process instance task is available. When notified, the user can then manually invoke the external UI implementation.

Note: For example, the external UI implementation can be exposed as a JSP page. In this case, you need to edit the URL template for the email that is automatically generated by IBM BPM (contained in the `99Local.xml` and `100Custom.xml` files) to point to the URL for the JSP page.

You can find more information about this topic in IBM Knowledge Center:

`http://ibm.biz/EmailTask`

If a human task is implemented as an External Implementation and a URL for the implementation has been specified in the configuration options in Figure 6-47 on page 370, the BPM REST API can be used to retrieve that URL programmatically. You can find more information in IBM Knowledge Center:

`http://ibm.biz/BPM_Get_Task`

The Task Instance Client Settings API needs to be invoked using the “External” option, as shown in Example 6-2.

Example 6-2 Programmatically retrieve URL associated the External Implementation of a task instance

```
https://<host_name>:<port_number>/rest/bpm/wle/v1/task/<TASK_ID>/clientSettings/External
```

Where the variables are as follows:

- *host_name* is the network name for the host of the process server
- *port_number* depends on your system configuration. Default port numbers are 9080 (HTTP) and 9443 (HTTPS)
- *TASK_ID* is the unique identifier of the task instance

Example 6-3 shows an example of retrieving the External Implementation URL for task 1124 on server1, with default configuration, the REST API call.

Example 6-3 Retrieving the External Implementation URL

```
https://server1:9443/rest/bpm/wle/v1/task/1124/clientSettings/External
```

This API call returns the URL specified in the configuration options in Figure 6-47 on page 370, if present.

Note: If a human task is implemented as an External Implementation, the process.lsw API described in Example 6-1 on page 343 will not retrieve its external UI implementation, as the `process.lsw` API is only supported for human tasks implemented as coach-based human services.

To programmatically trigger an *external UI implementation* before interacting with the user, the IBM BPM's Dynamic Event Framework (DEF) can be used. The DEF gathers and publishes events when a process instance reaches the desired state. External UI implementations then consume those events directly, without needing manual intervention from the users.

You can find more information about the DEF in IBM Knowledge Center:

http://ibm.biz/BPM_DEF

To create and send outbound events, the Dynamic Event Framework uses a specific send-events function. For instructions about how to install and use this function, refer to IBM Knowledge Center:

http://ibm.biz/BPMSendEvents

6.6.2 Use IBM MobileFirst to implement the Decoupled pattern

IBM BPM provides tight integration with IBM MobileFirst for the development of external UI implementations as mobile apps, through the use of IBM BPM MobileFirst *adapters*. These adapters can be used in an IBM MobileFirst mobile app to translate IBM BPM functions. The IBM BPM MobileFirst adapters encapsulate authentication and other IBM BPM REST APIs so that developers can simply use adapter functions to start IBM BPM processes and manipulate tasks. In essence, when implementing the *Decoupled pattern*, the MobileFirst adapters performs the role of the Decoupling Services described in 6.1.6, “Pros and Cons of a formal service exposure layer for BPM APIs” on page 326.

There are two IBM BPM MobileFirst adapters provided by IBM BPM:

- A generic IBM BPM MobileFirst adapter, which is required
- An optional IBM BPM app-specific adapter

IBM MobileFirst also allows implementation of offline task completion using the IBM Mobile First JSONStore capability.

The IBM BPM app-specific adapter allows you to perform activities related to exposed processes, tasks, or services related to a process app. This adapter serves as a façade to the generic adapter and introspects details, such as *process ID* and *snapshot ID*, which can be otherwise onerous and error-prone for the developer to capture.

You can find more information about MobileFirst and the MobileFirst adapters in IBM Knowledge Center:

http://ibm.biz/BPMMobileFirst

Figure 6-49 shows a sample architectural view of the components of an IBM MobileFirst app, which includes IBM BPM MobileFirst adapters, JSONStore, and IBM BPM.

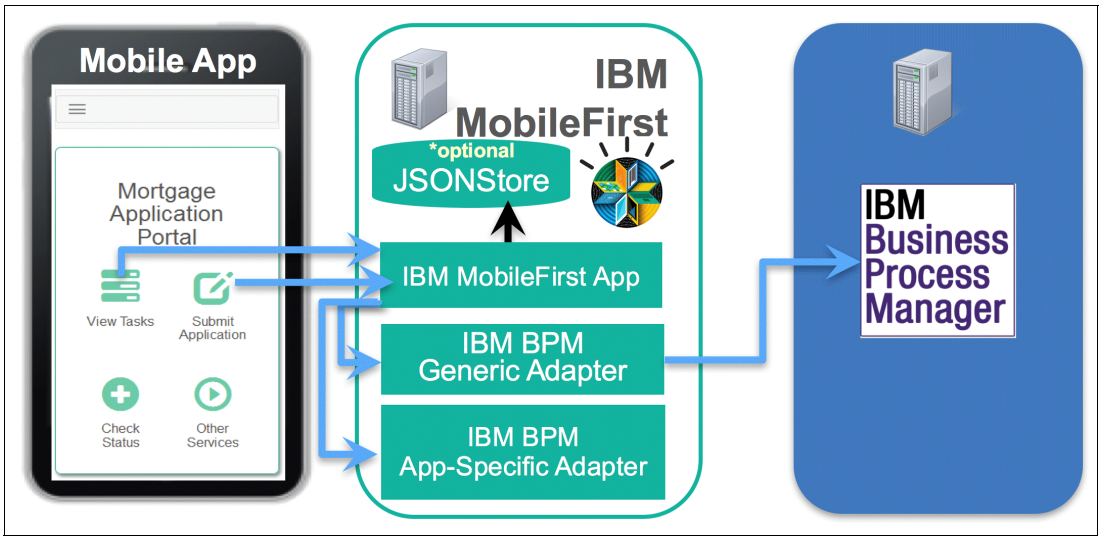


Figure 6-49 BM MobileFirst mobile app solution components

On-line mobile app example

This section describes a sample implementation of an online mobile app developed in MobileFirst triggering a business process in IBM BPM. The example used is one for a mortgage application, and is available in the `SG24-8355-ch6-online_app.twx` and `SG24-8355-ch6-MortgageOnlineApp.zip` files. You can find more information about these files in Appendix A, “Additional material” on page 395.

Note: The protocol that is used to communicate with the IBM BPM server is HTTP. You can change the `bpm.server.protocol` and `bpm.server.port` properties in the `worklight.properties` file to change the protocol to HTTPS.

To use the HTTPS protocol when the SSL certificate of the IBM BPM server is a self-signed certificate, use the WebSphere Administrative Console or the Java Keytool Keystore command to extract the certificate from the server. Then, import it into the Java Runtime certificate store on which the MobileFirst server is running. The Java Runtime certificate store file name is `cacerts` and is located in the `<JRE Root>\lib\security` file.

The IBM BPM MobileFirst adapters can be either retrieved through a service discovery mechanism in IBM MobileFirst, which is the preferred method or exported directly from IBM BPM Process Center, as shown in Figure 6-50.

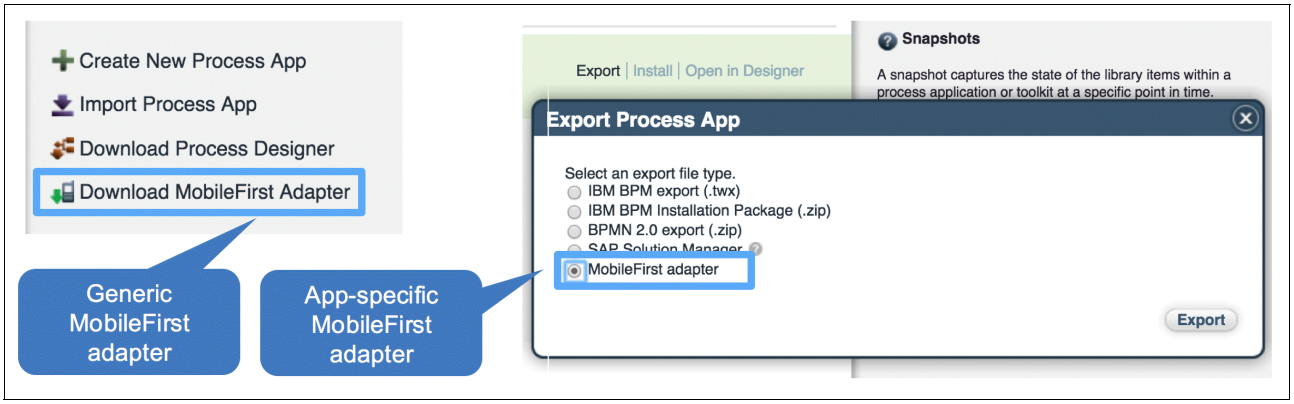


Figure 6-50 Export the MobileFirst adapters from Process Center

The sample mobile app is built in MobileFirst using both adapters. From the app, you can start a process in IBM BPM by submitting a mortgage application, as shown in Figure 6-51.

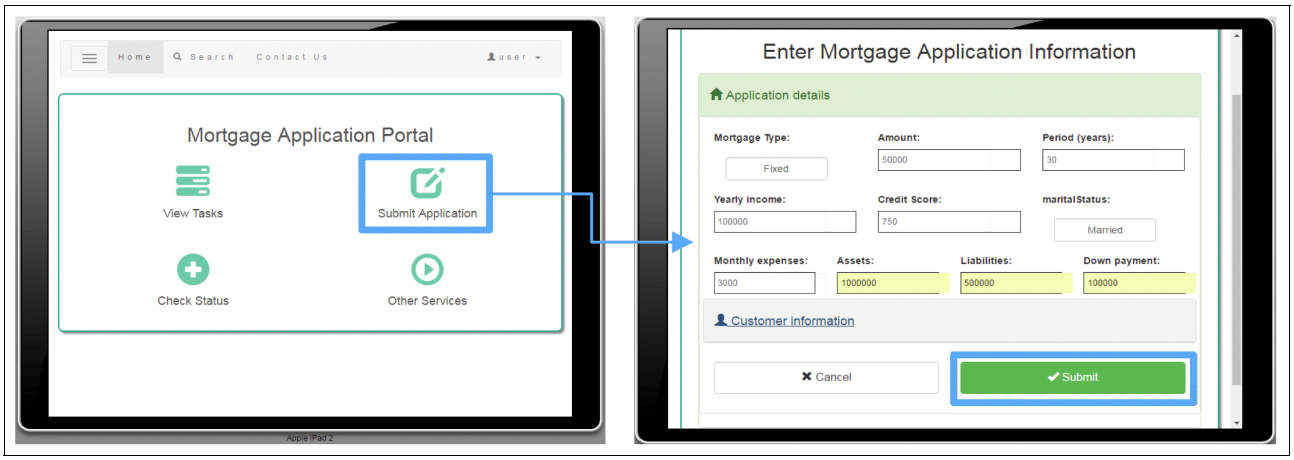


Figure 6-51 Sample MobileFirst mobile app starting a BPM process

When you click the **Submit** button in the app:

- IBM BPM is invoked through the generic MobileFirst adapter.
- The BPM mortgage process is started with the data captured in the app, through the app-specific MobileFirst adapter.

The mortgage process is then orchestrated by IBM BPM, and its human tasks (implemented with coach-based human services) can be accessed through IBM Process Portal, as shown in Figure 6-52.

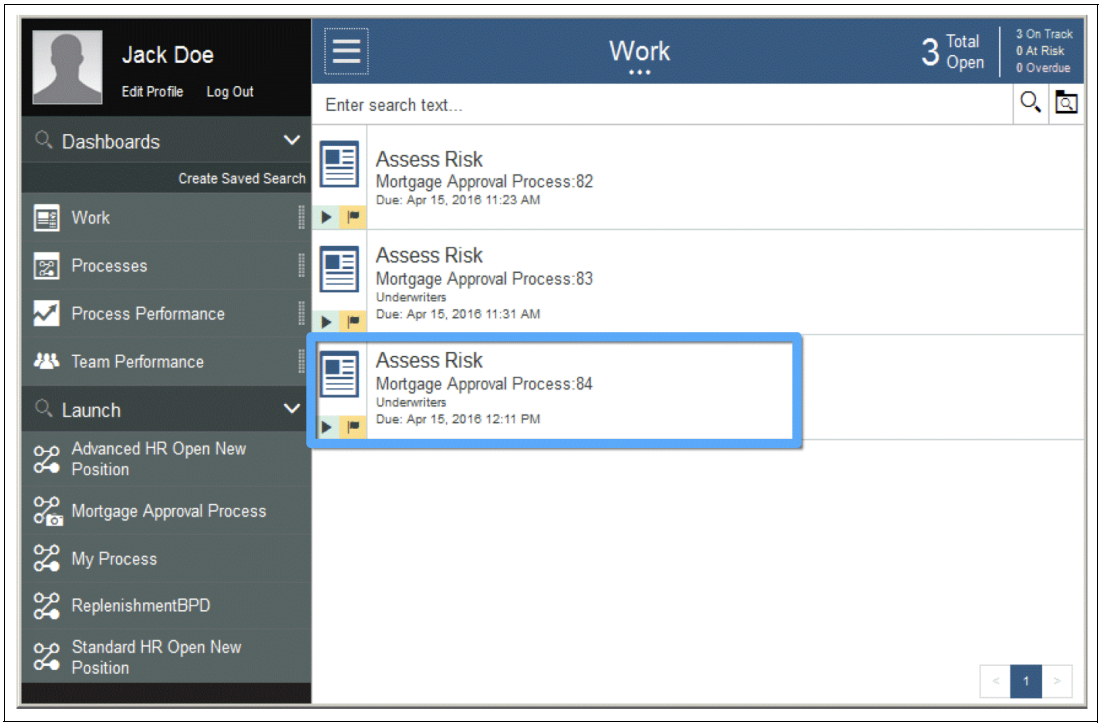


Figure 6-52 BPM process started from the mobile app, accessed via IBM Process Portal

Off-line mobile app example

As organizations deploy more mobile applications to empower their field workers, a common usage pattern that needs to be addressed is offline work.

This section describes a sample implementation of an offline mobile app developed in MobileFirst that triggers a business process in IBM BPM. The example is a service request executed by a field technician and is available in the `SG24-8355-ch6-offline_app.twx` and `SG24-8355-ch6-FieldService.zip` files. You can find more information about these files in Appendix A, “Additional material” on page 395.

Note: The protocol that is used to communicate with the IBM BPM server is HTTP. You can change the `bpm.server.protocol` and `bpm.server.port` properties in the `worklight.properties` file to change the protocol to HTTPS.

To implement the offline capabilities, the mobile app in this example leverages the JSONStore available in MobileFirst. JSONStore is a lightweight, document-oriented storage system that provides persistent storage of JSON documents and is an optional feature of IBM MobileFirst. You can find more information about JSONStore at:

`http://ibm.biz/MFJSONSTore`

This scenario uses the following steps:

1. A customer submits a field service request using the FS_CustomerApp, which triggers the Field Service BPM process shown in Figure 6-53.

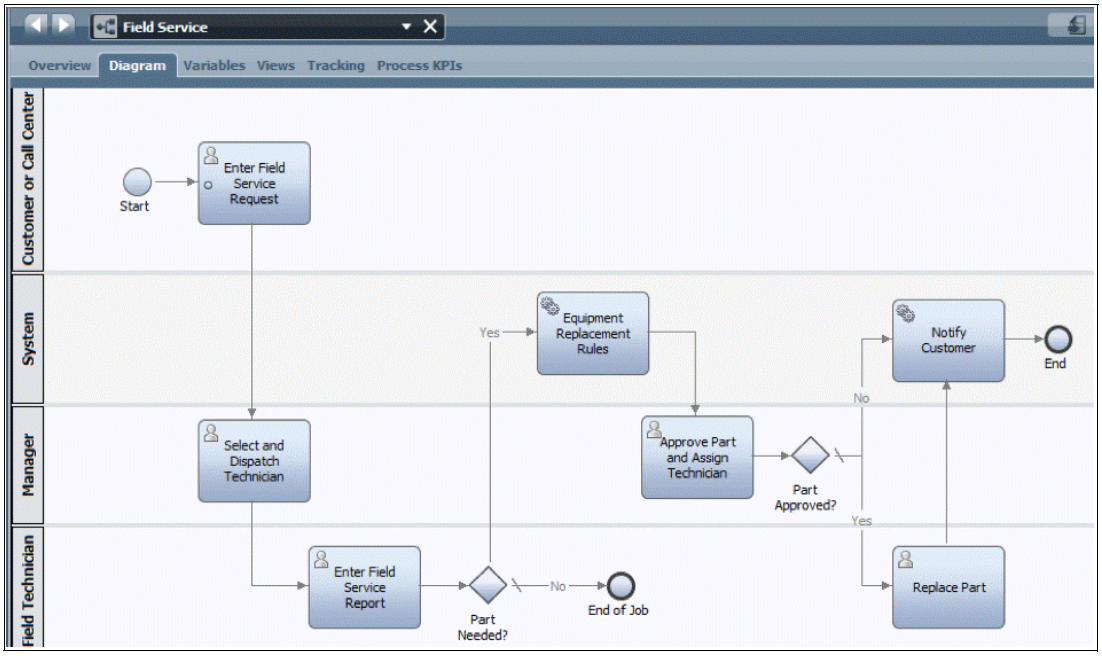


Figure 6-53 Field Service process in IBM BPM

2. A manager selects and dispatches a technician.
3. The technician submits a service report using FieldServiceApp. Because the technician is offline (Figure 6-54), the report is stored in the JSONStore for subsequent completion in IBM BPM (Figure 6-55 on page 378).

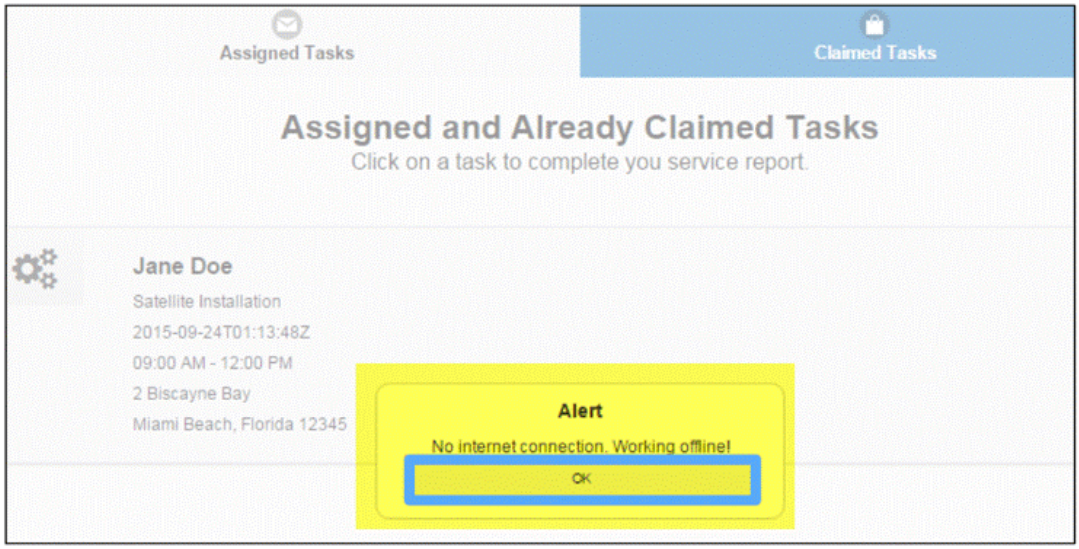


Figure 6-54 Technician accessing mobile app when offline

Figure 6-55 Service report is stored in JSONStore

4. When the technician gets back online, the app prompts the technician regarding whether to sync the reports. The technician chooses to sync, and the report is submitted successfully. The completed task is shown in IBM BPM (Figure 6-56).

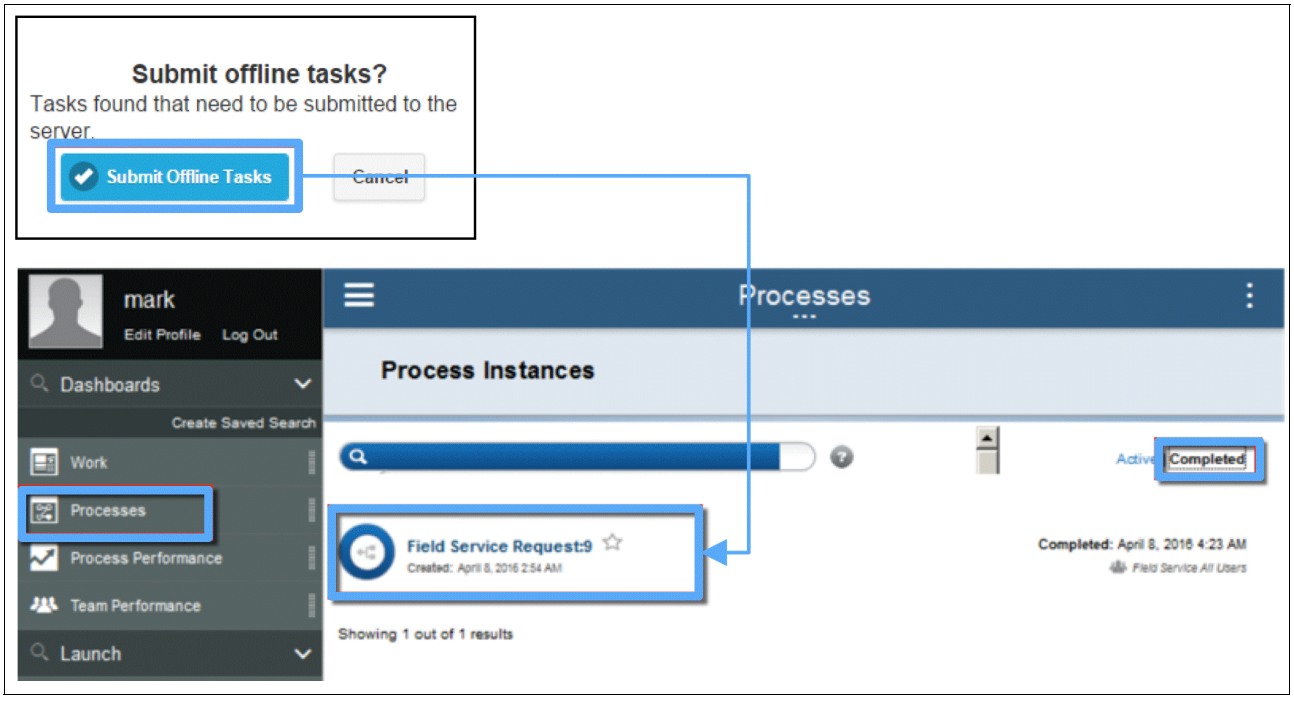


Figure 6-56 Back online, the mobile app submits the report to IBM BPM

Generate a hybrid MobileFirst App from a human service

The two samples provided in this section thus far assume that the external UI implementations for the human task designed following the Decoupled pattern are created from scratch in the MobileFirst platform.

However, IBM BPM 8.5.7 provides, in technology demonstration, the ability to export a client-side human service directly as an IBM BPM MobileFirst project. This export mechanism dramatically accelerates the creation of MobileFirst mobile apps, because the generated MobileFirst project includes both a generic IBM BPM MobileFirst adapter and UI code that is used to render the coaches and coach views in the client-side human service. In addition, it integrates them together in a basic mobile app that is near-ready to be run. However, the preferred approach is to use the export mechanism as the *base* of a MobileFirst project and to customize it to meet the specific app's needs.

The export mechanism is only available for those types of human tasks that in 6.1.4, “Exposing a process via a business service” on page 321have been defined as Isolated business services: human tasks that are not part of an overall process (for example, a human task to initiate a process or access a report).

Only client-side human services that are intended for use on multiple devices can be exported, when they are exposed (as shown in Figure 6-18 on page 341) as either a startable service, a dashboard, or a URL.

Note: These are the typical exposure mechanisms to be expected when a human service is exposed to the users in the isolated business service usage scenario described in 6.1.4, “Exposing a process via a business service” on page 321. Thus, they are appropriate for the Exposed and Decoupled patterns.

This feature is not supported for heritage human services.

The example in Figure 6-57 is a client-side human service with one coach and a system service used to start a process. This client-side human service is intended to be used in isolation.

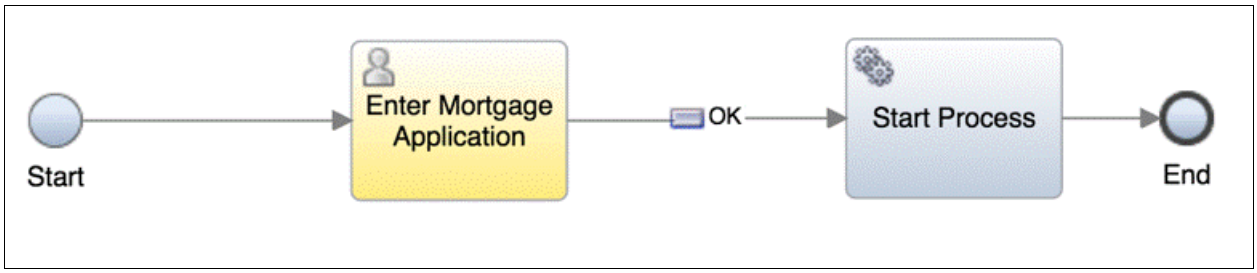


Figure 6-57 Sample client-side human service for MobileFirst export

If the client-side human service is intended for use on multiple devices and is exposed as, for example, a URL, you can export it as a MobileFirst project, as shown in Figure 6-58.

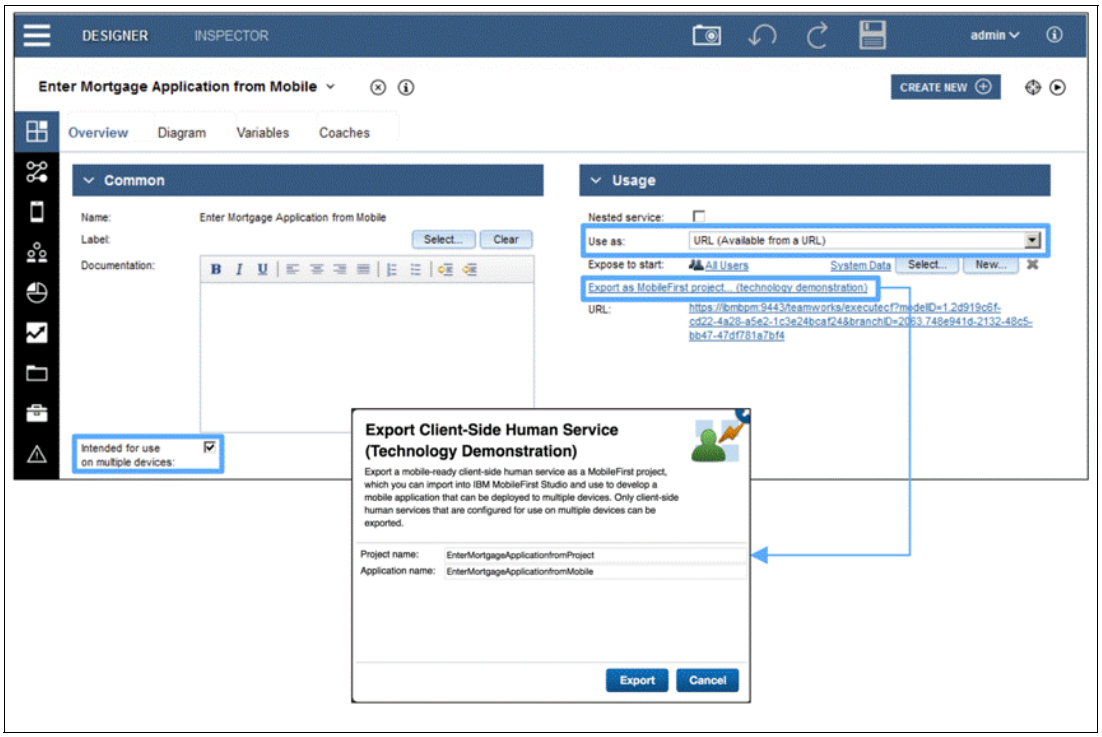


Figure 6-58 Export sample client-side human service as MobileFirst project

The exported project is an IBM MobileFirst hybrid project (meaning that it can support multiple environments, such as iOS, android and Windows), which can be imported for further authoring using the MobileFirst studio.

The project uses the IBM BPM MobileFirst adapter previously described and contains the codebase of the client-side human service, which, if correctly designed, can now run in isolation from the Process Server.

Using MobileFirst studio, the mobile app developer can:

- Edit security and authentication configuration.
- Change the look and feel of the mobile app.
- Add execution environments, such as iPhone and Android.

Each of these tasks, however can start from the framework supplied in IBM BPM, rather than starting from scratch.

The example described in this section is available in the `SG24-8355-ch6-hybrid_app.twx` and `SG24-8355-ch6-MortgageHybridApp.zip` files. For more information about these files, refer to Appendix A, “Additional material” on page 395.

6.7 Independent pattern

The Independent pattern shown in Figure 6-59 is task oriented. In this pattern an alternative task portal is used to allow users to interact with tasks in a presentation layer completely independent from BPM. The alternative task portal is either an existing enterprise portal or a custom-built lightweight portal, surfacing the human tasks implemented as an external UI for example a web application using a different UI technology.

This pattern is intended for scenarios requiring basic task management with integration to tasks implemented using an external UI or to expose those task to an existing enterprise portal.

BPM coach-based human services are not used in this pattern.

Both a custom-built lightweight portal and the external UI implementation usually access BPM via a set of reusable decoupling services, acting as service exposure layer, wrapping the BPM RESTful APIs. An enterprise portal typically accesses the BPM layer directly.

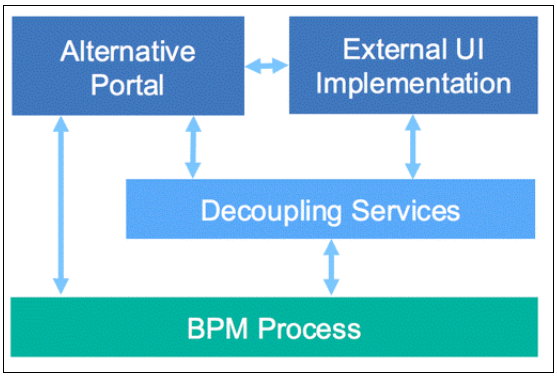


Figure 6-59 Independent pattern

This pattern is primarily intended for internal staff users working across multiple processes.

Examples of alternative portals are:

- A lightweight portal built as a custom web application, leveraging the BPM REST APIs
- A mobile portal application, built using the BPM REST APIs
- A custom portal created using the SPARK Portal builder toolkit, as described in 5.2.9, “SPARK Portal Builder toolkit” on page 305
- An enterprise portal, such as WebSphere Portal or Microsoft SharePoint

Note: Using a custom IBM Process Portal, as detailed in 5.2.7, “Clone Process Portal” on page 286, falls into the Hybrid pattern described in “Hybrid pattern” on page 382, because the customized portal process application uses the same functionality of the standard IBM Process Portal.

The Independent pattern is, from an implementation perspective, a combination of the Decoupled pattern to implement the external UI for a human task and an alternative portal to implement a task list. As such, a BPM human task implemented using an external UI can be created, integrated and exposed following the guidance given in 6.6.1, “Use an external UI implementation within a BPM process” on page 367.

When using an alternative task portal, the functionality to interact with tasks is delivered by the portal itself: basic operations such as, get next task, claim task, and complete task are part of the core capabilities of a task portal.

Enterprise portals such as, WebSphere Portal and Microsoft SharePoint offer these capabilities (or integration with these capabilities) ready to use.

If the alternative task portal is implemented as a custom lightweight portal or as a mobile application, the IBM BPM REST APIs expose the standard functionality expected from a BPM task portal, and developers can select their technology of choice for its implementation.

Additionally, IBM BPM provides a specific mechanism to use external BPM UI implementations in WebSphere Portal, using the IBM Script Portlet (a tool that enables a script developer to quickly develop portlets for WebSphere Portal without needing to know about Java, portlets or the JSR 286 portlet specification) as described in *Business Process Manager Sample for IBM Script Portlet in WebSphere Portal*, which is available at:

`http://ibm.biz/DXScriptPortlet`

6.8 Hybrid pattern

The Hybrid pattern shown in Figure 6-60 is task oriented. In this pattern, IBM Process Portal is used for task management and all other interactions with the BPM system. Human tasks executed within the portal are implemented as an external UI, for example a web application using a different UI technology.

This pattern provides users the powerful functionality of IBM Process Portal. BPM coach-based human services are not used in this pattern. The external UI implementation usually accesses BPM via a set of reusable decoupling services, acting as service exposure layer, wrapping the BPM RESTful APIs.

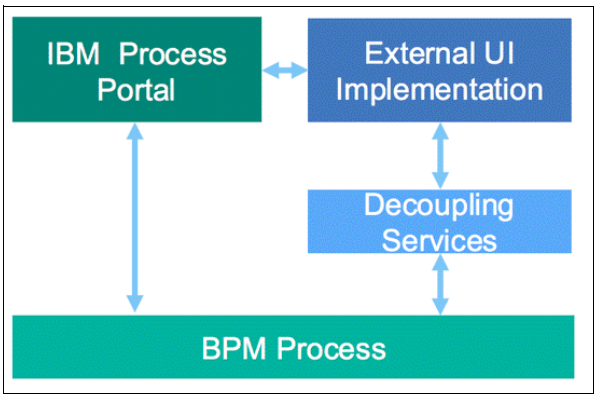


Figure 6-60 Hybrid pattern

This pattern is primarily intended for internal users who require the full functional breadth of IBM Process Portal but need to launch tasks implemented as an external UI.

6.8.1 Launch external activity from IBM Process Portal

IBM BPM 8.5.7 provides a ready to use implementation of the Hybrid pattern. The Hybrid pattern is, in essence, a combination of the Decoupled pattern to implement the external UI for a human task and the IBM Process Portal to implement a task list. As such, a BPM human

task implemented using an external UI can be created, integrated and exposed following the guidance given in 6.6.1, “Use an external UI implementation within a BPM process” on page 367.

The sample process used in 6.6.1, “Use an external UI implementation within a BPM process” on page 367, and depicted in Figure 6-43 on page 368, can be used to describe how to implement the hybrid pattern in IBM BPM.

In IBM BPM 8.5.7, you can now link an external implementation directly to a URL, as shown in Figure 6-61.

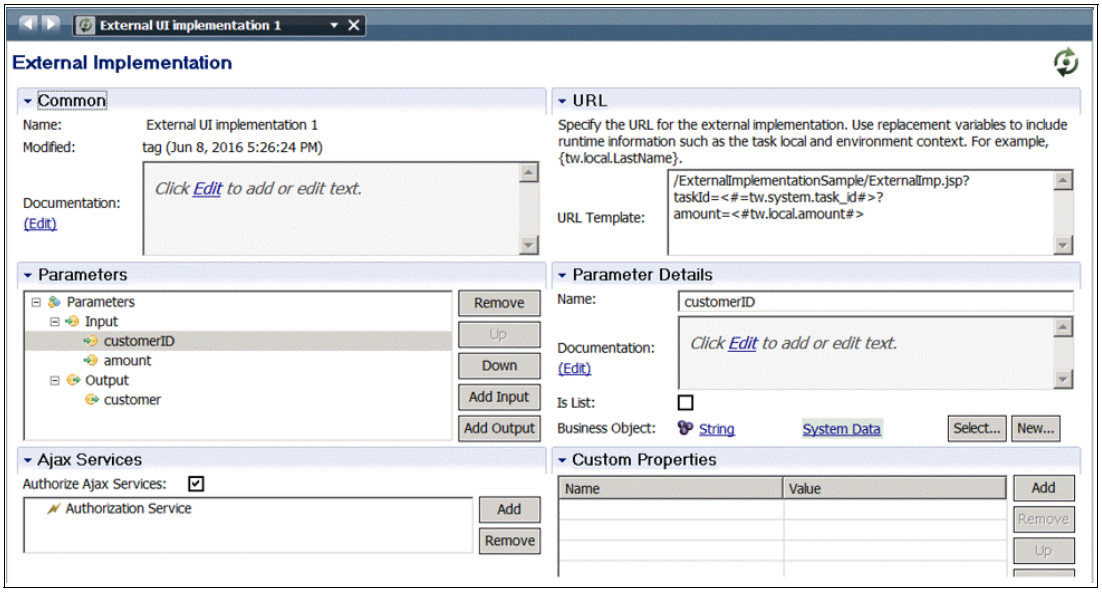


Figure 6-61 External Implementation configuration options, with URL

In this example, the external UI implementation for the first human task of the process is a JavaServer Pages (JSP) page pointing at an enterprise archive (EAR) file deployed to the same WebSphere Application Server instance running the BPM Process Server, as summarized in Figure 6-62.

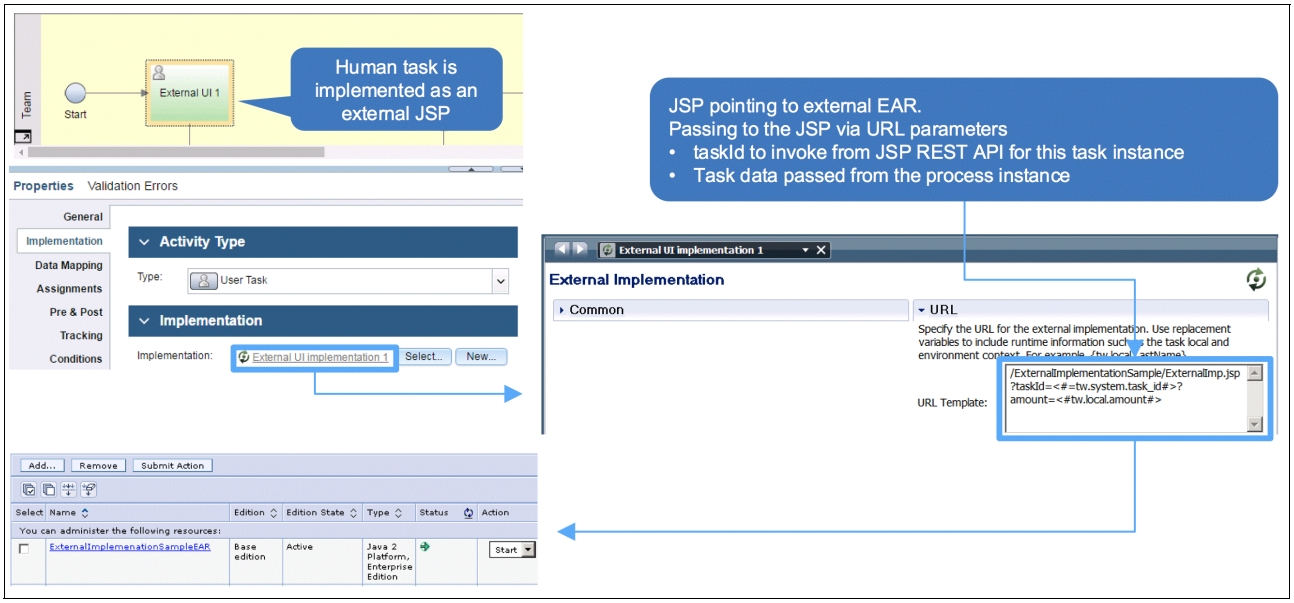


Figure 6-62 External UI 1 implemented as an external JSP running on the same server: BPMN

Note: External implementations can also be used to redirect the human task to other systems, in an assisted swivel chair work style. For example, the external implementations can point at bespoke web applications or existing UIs for different systems, such as SAP. (For a swivel chair integration with SAP implemented as is without configuration, refer to 6.9.2, “Embed iframes in coaches” on page 388.)

Not pertaining to this specific example, this implementation is also possible if the task is a BPEL task rather than a BPMN task, as shown in Figure 6-63.

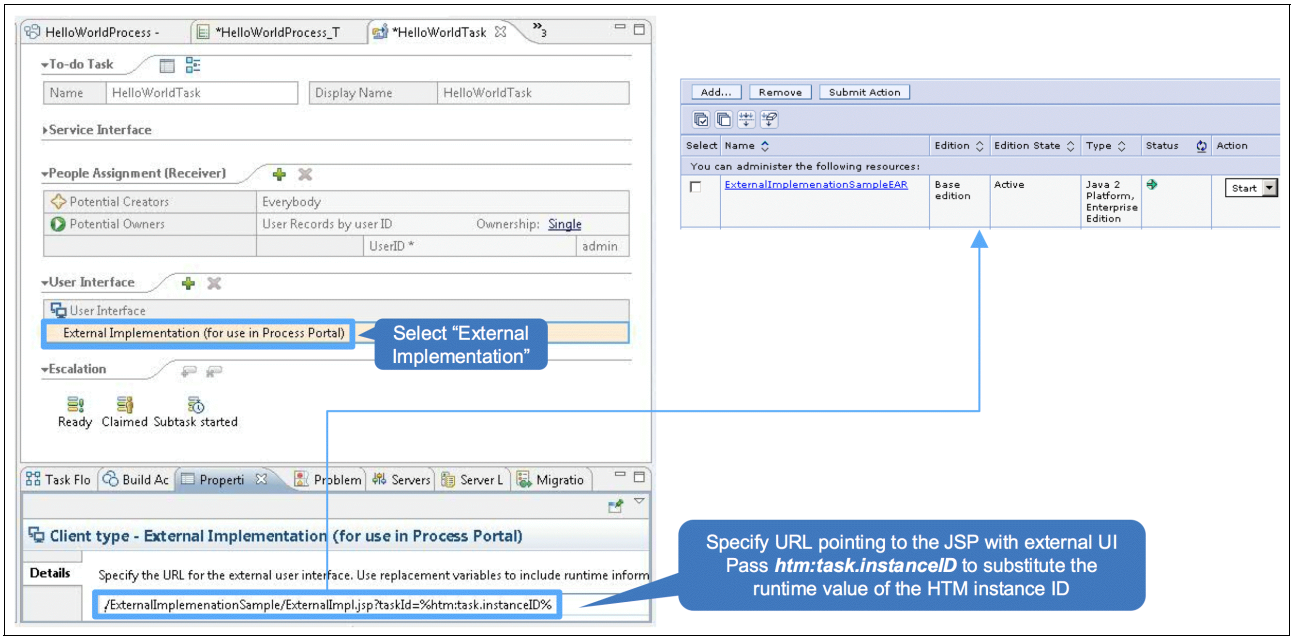


Figure 6-63 External UI 1 implemented as an external JSP running on the same server: BPEL

When the step implemented as an external implementation is reached during the execution of a business process, a corresponding task appears in the IBM Process Portal. That task is assigned to the user group that is defined in the process swim lane.

When a user selects the task from the IBM Process Portal, the URL specified in the external implementation configuration is then used to directly launch the external UI implementation from the portal, as depicted in Figure 6-64.

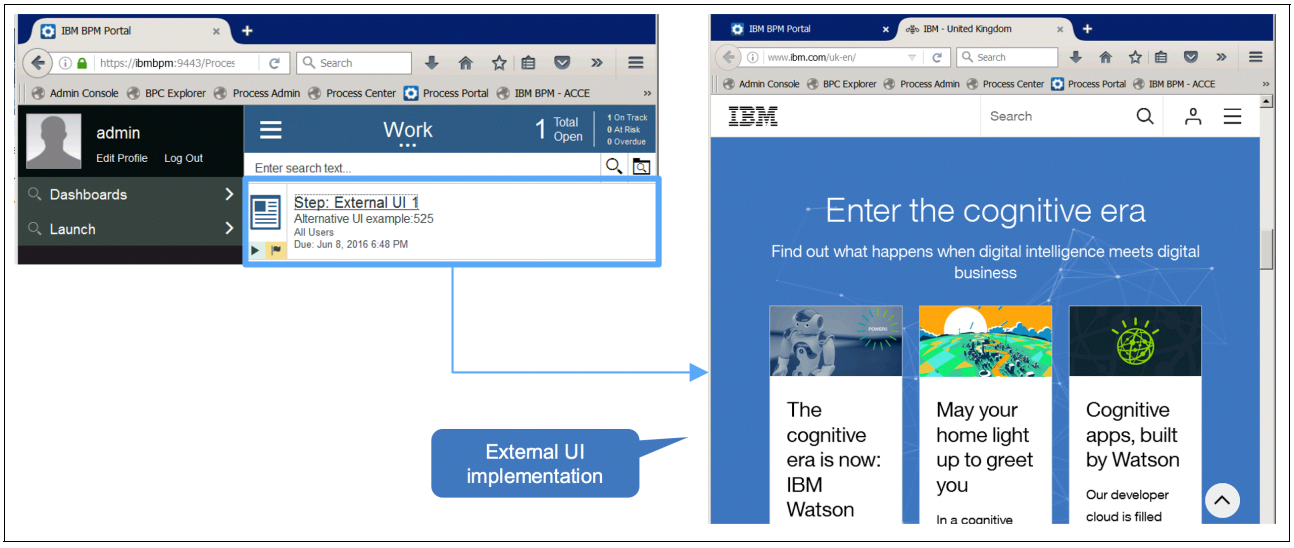


Figure 6-64 Launch an external activity from IBM Process Portal

Note: It is still the responsibility of the external UI implementation—in this case contained in the EAR file—to claim the task, get and set the task data, and finish the task. All these actions are exposed through the public REST API.

To attain a Hybrid pattern implementation, it is sufficient to use the IBM Process Portal to execute human tasks that are implemented as an external UI.

6.9 Combining and extending the BPM UI design patterns

The six patterns defined in 6.2, “BPM UI design patterns” on page 335 refer to the implementation of a single BPM human task. Typically, however, business processes contain more than one human task. A combination of the patterns in Table 6-2 on page 336 is possible within the same business process.

Potentially a single BPM business process can implement many patterns at once, depending on the choice made for each human task implementation, and the required usage scenario for business services, as shown in Figure 6-65.

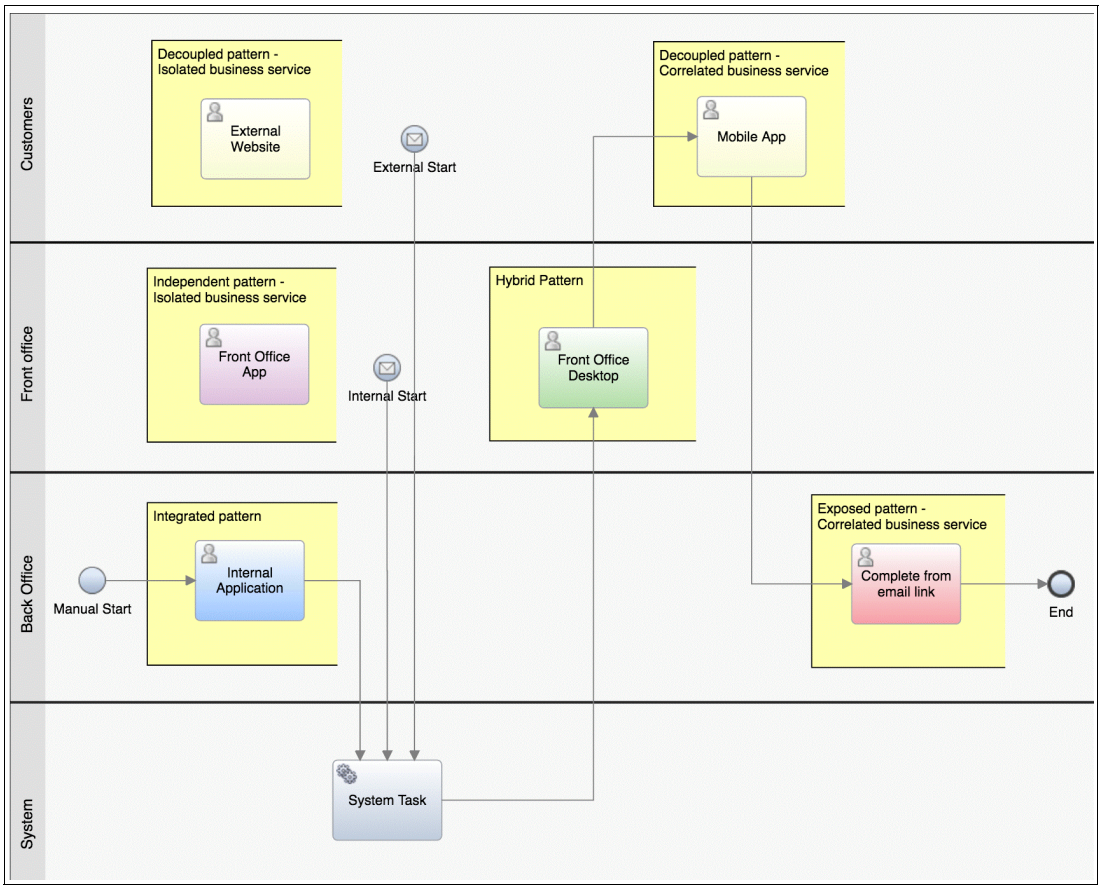


Figure 6-65 BPM process implementing many patterns at once

In the example in Figure 6-65 on page 386, the following roles are affected:

- Back office staff uses coach-based human services, accessed either through IBM Process Portal (Integrated pattern) or directly (Exposed pattern).
- Front office staff uses external UI implementations for human tasks, accessed either through a custom lightweight portal (Independent pattern) or through IBM Process Portal (Hybrid pattern).
- External customers interact with the process as different types of business services (as per the usage scenarios defined in 6.1.4, “Exposing a process via a business service” on page 321) to initiate a process (isolated business service) or complete a task (correlated business service), using external UI implementations (Decoupled pattern).

The scenario in Figure 6-65 on page 386 is just one example and is in no way meant to be prescriptive. In fact, it is rare for the same process to implement more than two or three of the design patterns described in this chapter at the same time. However, this example is useful to describe how the usage scenario, access mechanism, and presentation layer choices specific to each human task can drive its implementation.

It is also important to note that process modelling feeds into the implementation choice, but it is not influenced by it. Process analysis should be the sole driver to process modelling, which then influences the implementation choice, and not vice-versa.

6.9.1 Take advantage of IBM Process Portal

One of the benefits of using IBM Process Portal is its ability to deliver a ready to use Hybrid pattern, which is described in “Hybrid pattern” on page 382. This pattern is particularly valuable when working with a BPM business process that uses a mixture of coach-based human services and external UI implementations, because the IBM Process Portal can serve both types of human task implementations seamlessly to the user, as shown in Figure 6-66.

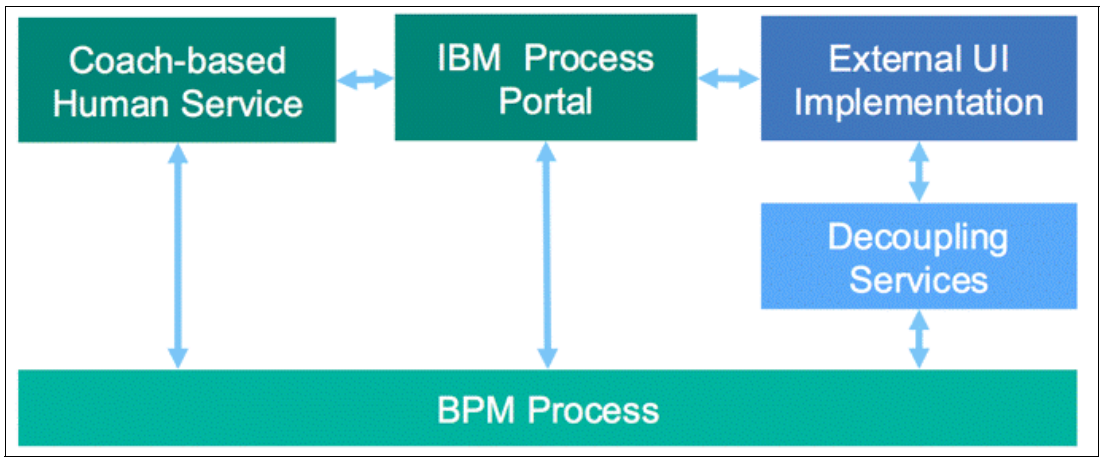


Figure 6-66 Combining the Hybrid and the Integrated patterns

In essence, serving both types of tasks means combining the Hybrid pattern and the Integrated pattern, using IBM Process Portal as the unique task portal implementation for all BPM UI types.

This function is even more important if this particular design is adopted for multiple BPM processes and, as an extension, for multiple BPM Process Servers. In fact, the Process Federation Server, described in 5.4, “Process Federation Server” on page 310 allows you to

federate tasks that are generated from multiple IBM BPM sources and to serve them in a single IBM Process Portal, independently from their implementation style.

6.9.2 Embed iframes in coaches

The patterns described in Table 6-2 on page 336 are defined taking into account the following dimensions:

- The choice of the BPM human tasks implementation
- The choice of the task portal implementation

No particular distinction has been made so far in regards to what is contained within the BPM human tasks. However, in the same way in which coach-based human services can be embedded in a separate presentation layer as an iframe, a different UI provider can be embedded in human tasks. When that happens, the design still falls back in one of the six patterns that are described in this chapter. However, it is important to consider the architectural impact of embedding iframes into human tasks, such as:

- Security
- Cross-site scripting
- Cross-site request forgery
- Passing of context and variables across the iframe
- Authentication and single-sign on
- And others

These considerations are independent from the implementation choice for human tasks: using coach-based human services or external UI implementations.

IBM BPM provides an implementation of this subpattern when integrating to SAP, in what is called SAP guided workflow. The SAP guided workflow implementation is, in essence, a variant of the Integrated pattern described in “Integrated pattern” on page 366, which uses coach-based heritage human services and iframes that embed the SAP web graphical user interface (GUI). The SAP guided workflow is described in the remainder of this section.

The SAP Guided Workflow Toolkit

The SAP Guided Workflow Toolkit that ships with IBM BPM provides the capability to wrap SAP transactions UIs (native SAP web GUI screens) and display them in coach-based human services. This capability enables the creation of processes which guide SAP users through the correct sequence of SAP transactions.

Additional process steps complementing SAP functionality can also be added to the process model, such as approvals, escalations, exceptions and exception handling, value management, and non-SAP automated activities. Moreover, SAP users can take advantage of BPM process monitoring and process improvement capabilities, such as gaining real-time insight into business performance issues and identifying process improvement opportunities.

The toolkit contains default implementations of various types of the SAP web GUI, such as Fiori or SAP WebDynpro, as shown in Figure 6-67.

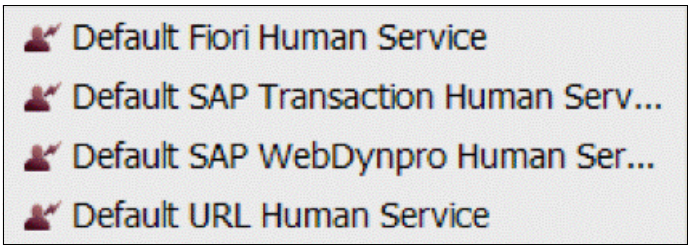


Figure 6-67 SAP web GUI default human services

The most common type of SAP GUI rendering within the toolkit is the SAP Transaction human service, a heritage human service, which displays SAP Transaction Codes.

Note: In IBM BPM 8.5.7, this feature is not available using client-side human services.

The recommended approach to use the SAP Transaction human service is to:

1. Copy the desired SAP human service to a Process App or Toolkit.
2. Customize it if needed.
3. Use it as implementation for human services in a BPM process.

The main reason to copy and customize the default SAP Transaction human service is to change the look and feel of the coach that embeds the SAP Transaction. For example, as shown in Figure 6-68, designers can add the corporate header and footer (Global Communications) and the transaction name in a Create Sales Order coach view section.

Note: The SAP VA01 transaction window is displayed inside an iframe using a special coach view that is supplied in the Default SAP Transaction human service.

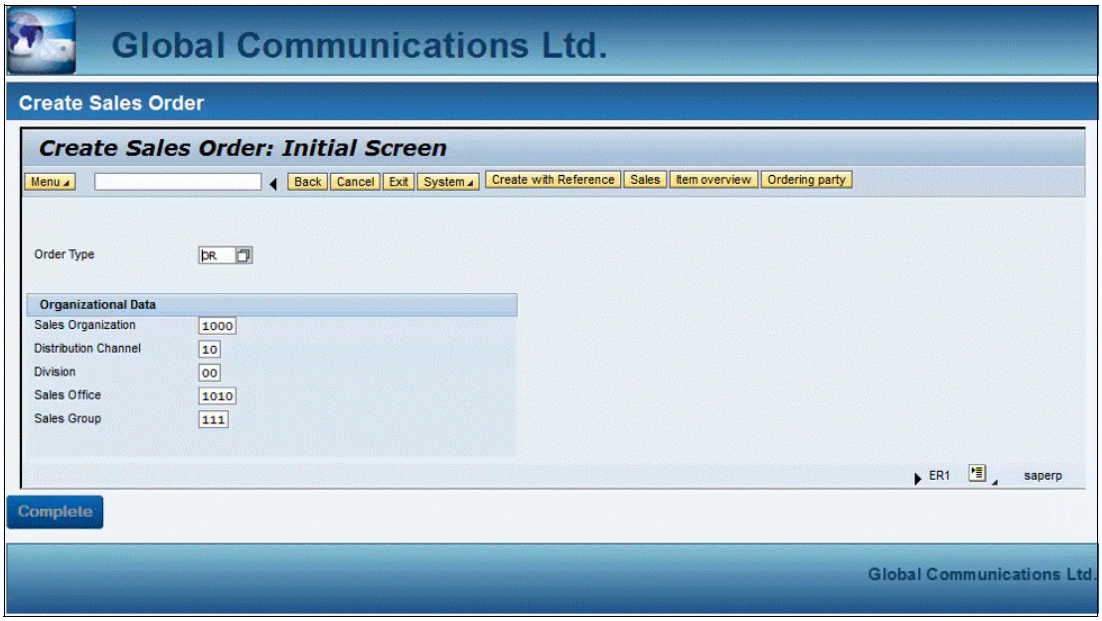


Figure 6-68 A coach in a custom SAP Transaction human service

Other possible types of customization include:

- Adding other buttons, for example a Help button
- Adding other coach views, for example data to be copied into the SAP Transaction

After you customize the SAP Transaction human service, you can add it as the implementation of a BPM human task, as shown in Figure 6-69.

Figure 6-69 Human service implementation set to a Custom SAP Transaction human service

The data required to launch the SAP Transaction Human Service needs to be mapped to the human service, as shown in Figure 6-70. In particular, the following required parameters must be provided:

- defaultServerInfo
- defaultTransaction

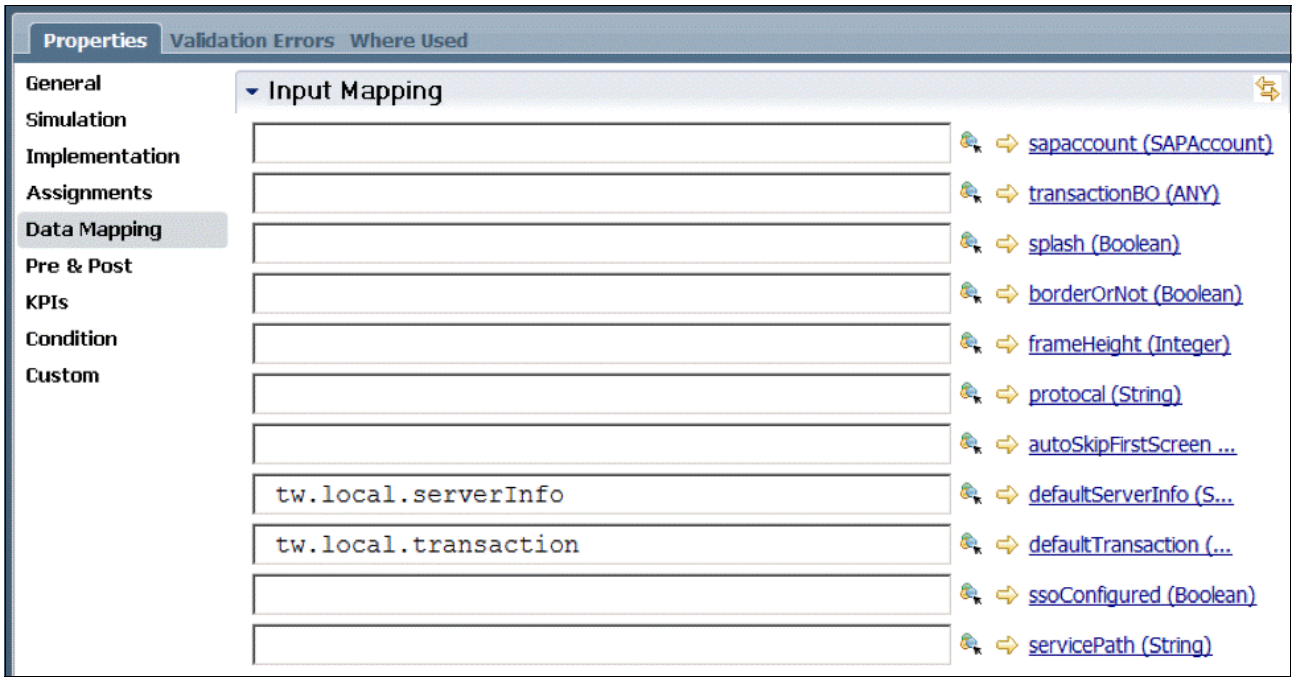


Figure 6-70 Required parameters for an SAP Transaction human service

The defaultServerInfo variable identifies the target SAP server on which the activity will run. You should use an *Environment* variable to change the server configuration after a snapshot is deployed. In Figure 6-71 on page 391, however, a private variable that contains the SAP server information is used. This example illustrates an alternative approach to defining the server location. Instead of using Environment variables, you can use a fully qualified host name for the SAP Server.

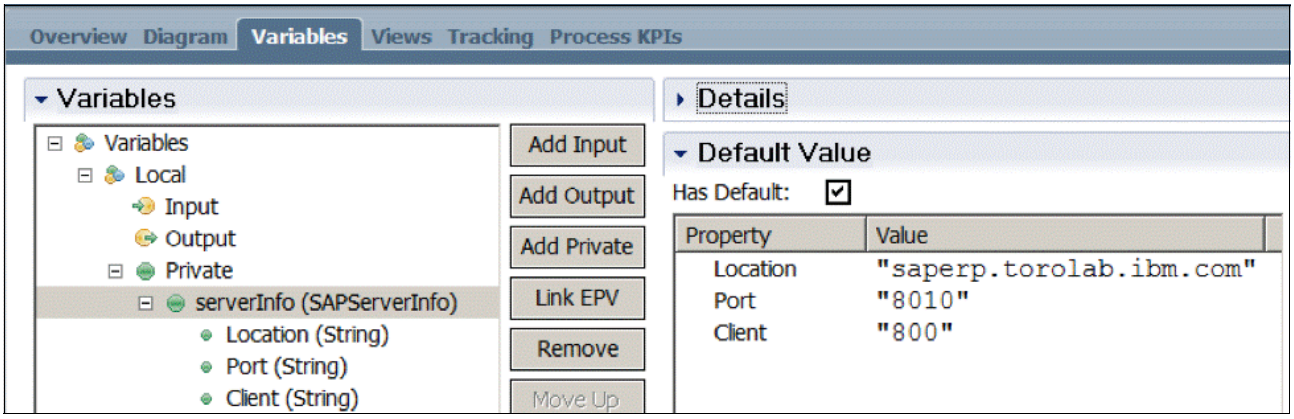


Figure 6-71 SAP server information

The defaultTransaction variable specifies the transaction that the human task will embed and run. Typically, only a single variable of this type is used in a business process and the transaction.object value is dynamically changed prior to calling the human service. This value identifies the actual transaction in SAP. Figure 6-72 shows an example.

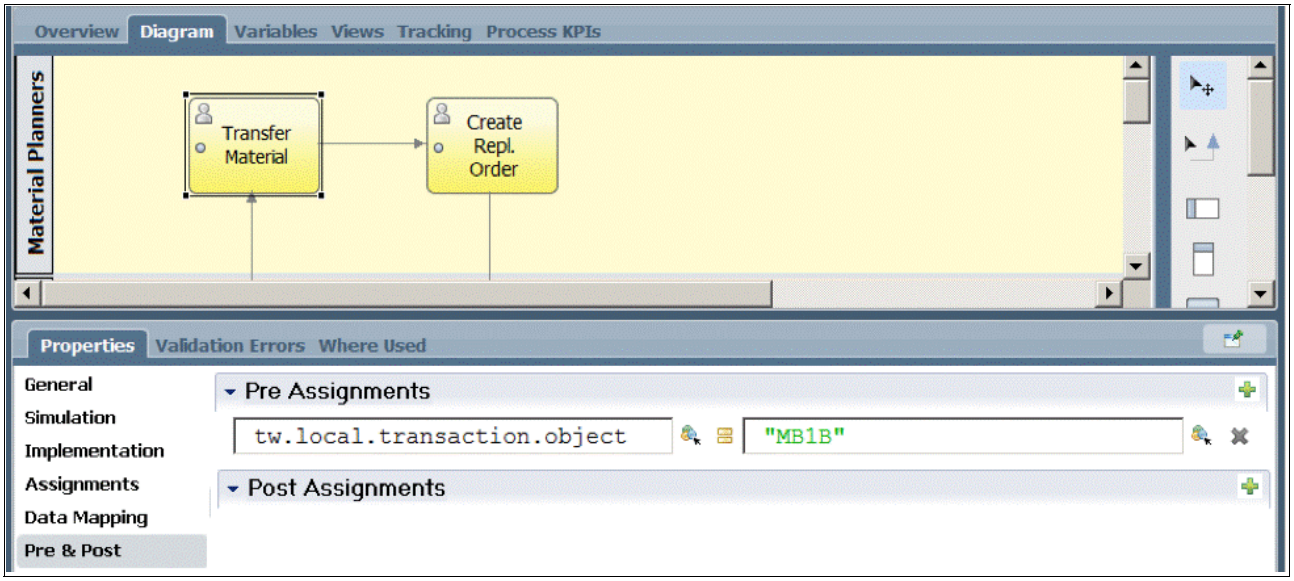


Figure 6-72 Setting SAP transaction value

The additional mapping options in Figure 6-70 on page 390 allow for the configuration of advanced features in the SAP guided workflow capabilities, such as:

- The ability to use a business object defined in the BPM process to initialize the SAP transaction panel
- The ability to disable the Complete button on the SAP Transaction coach, if an SAP transaction is not completed
- The ability to copy the result of an SAP transaction back to a business object defined in the process

Figure 6-73 shows how the Order field in the SAP panel is initialized by passing the value of `14447` from the output of an SAP transaction completed in a previous human task and kept in a business object associated with the BPM process. In the example, the Complete button is displayed as enabled because the SAP transaction was completed, as indicated by the `Delivery 80017179 has been saved` message that is displayed at the bottom of the iframe.

Finally, when the Complete button is clicked, the 80017179 delivery number is stored in a business object used by the BPM process, so that it can used as an input to the next SAP transaction.

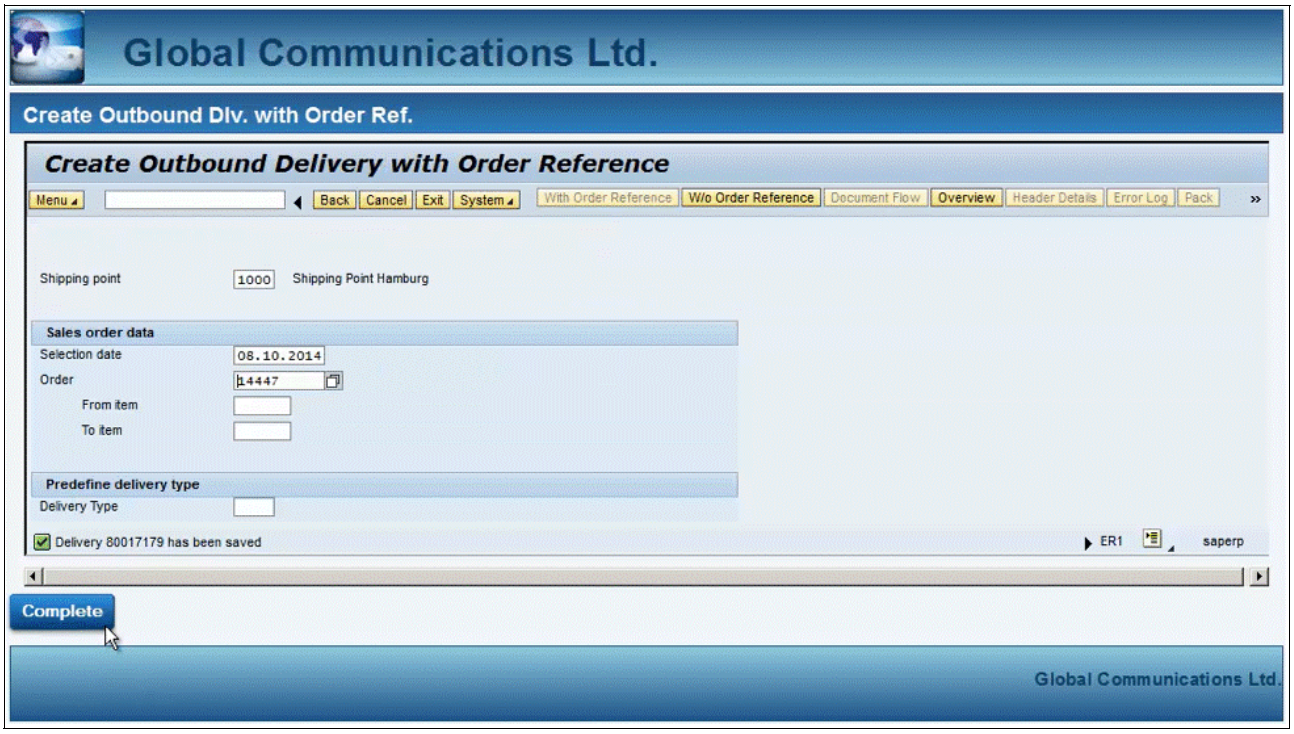


Figure 6-73 Advanced features in SAP guided workflow

You can find more information about the SAP guided workflow advanced features in IBM Knowledge Center:

`http://ibm.biz/BPM_SAP`

6.10 Conclusion

This chapter included information about *headless* BPM and the use of an external UI together with the IBM BPM process engine. It provided information that analyzed the architectural and security implications of headless BPM, along with common advantages and drawbacks of this choice.

This chapter also discussed the use of external UI implementations together with the integrated UI provided by the IBM BPM Coach Framework. Furthermore, it provided information about the role of a formal service exposure layer for the BPM APIs and the requirement of exposing IBM BPM to the Internet.

Based on these considerations, this chapter introduced six design patterns for the creation of BPM UIs, which included using the integrated UI provided by IBM BPM, an external UI

implementation, and various combinations of these two options. In addition, for each design pattern, this chapter provided implementation examples, along with an explanation of how IBM BPM technology enables the design pattern.

Finally, the six design patterns and implementation examples allowed the discussion to include the exposure of coach-based human services over the Internet, the integration of coaches into other solutions, the integration of external UIs into BPM, and the use IBM Mobile First and IBM WebSphere Portal with IBM BPM.

A

Additional material

This book, SG24-8355, refers to additional material that you can download from the Internet as described in the following sections.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG248355`

Alternatively, you can go to the IBM Redbooks website at:

ibm.com`/redbooks`

Select the Additional materials and open the directory that corresponds with the IBM Redbooks form number, SG248355.

Using the Web material

Some of the `.zip` files are large. So the additional materials include separate files for each chapter. You only need to download the files that you are interested in.

File name	*Description*
SG24-8355-ch2.zip	Compressed code samples for chapter 2
SG24-8355-ch3.zip	Compressed code samples for chapter 3
SG24-8355-ch5.zip	Compressed code samples for chapter 5
SG24-8355-ch6.zip	Compressed code samples for chapter 6

The sections that follow provide more details about these compressed files.

SG24-8355-ch2.zip

SG24-8355-ch2.zip file contains the SG248355-ch2.twx file for the sample process used in Chapter 2, “Creating user interfaces with coaches” on page 9. Import this sample process application using the Process Center console.

SG24-8355-ch3.zip

The SG24-8355-ch3.zip file contains the following samples used in Chapter 3, “Building controls using coach views” on page 73:

- The SG24-8355-ch3-start.twx file, sample application
- The SG24-8355-ch3-completed.twx file, completed reference

Import these sample process applications using the Process Center console.

SG24-8355-ch5.zip

The SG24-8355-ch5.zip file contains the following samples used in Chapter 5, “IBM Process Portal” on page 261:

- The Custom_Process_Portal_App.twx file, baseline custom process portal app
- The Custom_Process_Portal_Toolkit.twx file, baseline custom process portal toolkit
- The IBM_BPM_CustomPortal_SingleCluster.ear file, baseline custom process portal EAR file
- The Custom_Report_Dashboard.twx file, sample custom report dashboard

Use the baseline custom process portal app, toolkit, and EAR file to make customizations to IBM Process Portal 8.5.7 or as a baseline to create your own custom process portal, as detailed in Chapter 5, “IBM Process Portal” on page 261.

Import the sample process apps and toolkits using the Process Center console.

Deploy the custom process portal EAR using WebSphere Admin console, as detailed in Chapter 5, “IBM Process Portal” on page 261.

SG24-8355-ch6.zip

The SG24-8355-ch6.zip file contains the following sample files used in Chapter 6, “Combining the Coach Framework with other approaches” on page 313:

- The SG24-8355-ch6-patterns.twx file, design patterns examples
- The SG24-8355-ch6-HHSPortlet.war file, generated portlet
- The SG24-8355-ch6-online_app.twx file, process for online app
- The SG24-8355-ch6-MortgageOnlineApp.zip file, Mobile First online app
- The SG24-8355-ch6-offline_app.twx file, process for offline app
- The SG24-8355-ch6-FieldService.zip file, Mobile First offline app
- The SG24-8355-ch6-hybrid_app.twx file, process for hybrid app
- The SG24-8355-ch6-MortgageHybridApp.zip file, Mobile First hybrid app

Details of these sample files are described in Chapter 6, “Combining the Coach Framework with other approaches” on page 313.

Downloading and extracting the Web material

Create a subdirectory (folder) on your workstation, and extract the contents of the Web material `.zip` files into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

Online resources

These websites are also relevant as further information sources:

- Binding data and configuration options

  http://ibm.biz/BPMCoachViewBindingAndConfigOptions
- BPM community article: *How to re-develop the IBM Process Portal*

  http://ibm.biz/BPMYouPortal
- BPM Performance Dashboards

  http://ibm.biz/BPM85_Process_Performance_Video
  http://ibm.biz/BPM85_Team_Performance_Video
- Calling Ajax services from coach views

  http://ibm.biz/BPMCallingAjaxServicesFromCoachViews
- Chart Control

  http://ibm.biz/BPMChartControl
- The change event handler

  http://ibm.biz/BPMCoachViewChangeEventHandler
- Coach API

  http://ibm.biz/BPMCoachAPI
- Configuring the design-time appearance of coach views

  http://ibm.biz/BPMCoachViewsDesignTimeAppearance
- Content box

  http://ibm.biz/BPMCoachViewsContentBox
- The context object

  http://ibm.biz/BPMCoachViewContextObject
- Controls in the Content Management toolkit

  http://ibm.biz/BPMContentManagementToolkit
- Controls in the Responsive Coaches toolkit

  http://ibm.biz/BPMResponsiveCoachesToolkit
- Control (Coach View) Visibility Properties

  http://ibm.biz/BPMCoachViewVisibilityProperties
- Custom HTML

  http://ibm.biz/BPMCoachViewCustomHTML

- developerWorks resources
  - *The placement of BPM runtime components in an SOA*

    http://ibm.biz/BPM_SOA
  - *Comparing web APIs with SOA and EAI*

    http://ibm.biz/APIandSOA
  - *Using the REST API in IBM BPM*

    http://ibm.biz/Use_BPM_REST_API
- Digital Experience community article: *BPM sample for IBM Script Portlet in WebSphere Portal*

  http://ibm.biz/DXScriptPortlet
- Dojo reference guide

  https://dojotoolkit.org/reference-guide/1.10/dojo/dom.html
  https://dojotoolkit.org/reference-guide/1.10/dojo/dom-construct.html
  https://dojotoolkit.org/reference-guide/1.10/dojo/dom-attr.html
  https://dojotoolkit.org/reference-guide/1.10/dojo/dom-class.html
  https://dojotoolkit.org/reference-guide/1.10/dojo/dom-prop.html
  https://dojotoolkit.org/reference-guide/1.10/dojo/dom-style.html
  https://dojotoolkit.org/reference-guide/1.10/dojo/request/xhr.html
- Dojo tutorials: Dojo DOM Functions

  http://dojotoolkit.org/documentation/tutorials/1.10/dom_functions/index.html
- Download Interconnect 2016 BTB-2890 presentation: Understanding the IBM BPM Process Federation Server:

  http://ibm.biz/IBM_PFS_InterConnect2016_BTB-2890
- Enabling JavaScript debugging for coaches

  http://ibm.biz/BPMEnableJSDebugging
- Event handlers for coach views

  http://ibm.biz/BPMCoachViewEventHandlers
- Event handlers for coach view design-time preview

  http://ibm.biz/BPMCoachViewsDesignTimePreviewEventHandlers
- Example: Creating a jQuery button control

  http://ibm.biz/BPMCoachViewExampleJQueryButton
- Example: creating a custom AngularJS progress bar coach view

  http://ibm.biz/BPMCoachViewExampleAngularJSProgressBar
- Framework managed versus view managed content for coaches

  http://ibm.biz/BPMCoachViewsFrameworkVsViewManaged
- Generating a unique ID for a coach view

  http://ibm.biz/BPMGenerateUniqueCoachViewID
- IBM BPM System Requirements

  http://ibm.biz/BPMAdvSysReqs
  http://ibm.biz/BPMStdSysReqs
  http://ibm.biz/BPMExpSysReqs

- IBM Icon Font

  http://ibm.biz/IBMIconFont
- IBM Knowledge Center and the BPM Developer Center
  - BPM Developer Center

    http://ibm.biz/BPMDevCenterUIVideos
    http://ibm.biz/BPMDevCenterUXResources
    http://ibm.biz/BPMDevCenterMobileResources
  - Building Coaches

    http://ibm.biz/BPMBuildingCoaches
  - Creating user interfaces

    http://ibm.biz/BPMCreatingUI
  - Modeling client-side human services

    http://ibm.biz/BPMModelingCSHS
  - Process Federation Server

    http://ibm.biz/IBM_PFS_857
  - IBM Process Portal

    http://ibm.biz/IBM_Process_Portal_857_Video
    http://ibm.biz/IBM_Process_Portal_857
- IBM Technote: *Internet facing BPM UIs*

  http://ibm.biz/InternetFacingBPMUI
- Improving coach view performance

  http://ibm.biz/BPMCoachViewPerformance
- InterConnect presentation: *RBS BPM to Digital Platform integration*

  http://ibm.biz/RBS_BPM
- Java.net: *Understanding the Java Portlet Specification 2.0* (JSR 286)

  http://ibm.biz/UnderstandJSR286
- JavaScript API for client-side human service development

  http://ibm.biz/BPMCSHSJSAPI
- Localizing process applications

  http://ibm.biz/BPMLocalization
- Less: Getting Started

  http://lesscss.org/
- Minification

  https://en.wikipedia.org/wiki/Minification_(programming)
- Mobile First community article: *JSONStore*

  http://ibm.biz/MFJSONSTore
- Open Web Application Security Project resources
  - Attack Surface

    http://ibm.biz/AttackSurface

  - Cross-site scripting

    http://ibm.biz/OWASP_XSS

  - Cross-site request forgery

    http://ibm.biz/OWASP_CSRF

- Process Federation Server

  http://ibm.biz/IBM_PFS_857_Video
  http://ibm.biz/IBM_PFS_857
  http://ibm.biz/IBM_PFS_InterConnect2016_BTB-2890

- Process Portal Configuration

  http://ibm.biz/BPMProcessPortalConfigMashups
  http://ibm.biz/BPMProcessPortalConfig100CustomXML

- Process Portal Toolkits

  http://ibm.biz/BPMResponsivePortalComponentsToolkit
  http://ibm.biz/BPMDashboardToolkit
  http://ibm.biz/BPMSparkPortalBuilder

- Regular Expression

  https://en.wikipedia.org/wiki/Regular_expression

- REST interface for BPD-related resources: Service Resources, POST (start) Method

  http://ibm.biz/BPMRESTAPIStartService

- Salient Process: External Participant Toolkit

  http://ibm.biz/BPMSparkEPSToolkit

- Salient Process Enhanced support for SPARK Toolkit:

  http://ibm.biz/BPMSparkToolkitsAnnouncement

- Salient Process Spark UI Toolkit Controls

  http://ibm.biz/BPMSparkUIToolkitControls

- SOA reference architecture

  http://ibm.biz/APIandSOA

- The Swagger specification

  http://swagger.io

- Tips for debugging coach view lifecycle method inside client-side human services

  http://ibm.biz/BPMDebugCoachViewLifecycleInCSHS

- Tips for debugging coach views in Process Portal

  http://ibm.biz/BPMDebugCoachViewsInProcessPortal

- Using AngularJS helper provided in the Responsive Coaches toolkit

  http://ibm.biz/BPMCoachViewsAngularJSHelper

- The unload event handler

  http://ibm.biz/BPMCoachViewUnloadEventHandler

- Validating coaches in client-side human services

  http://ibm.biz/BPMCoachValidation

- View the Getting Started with Process Federation Server in IBM BPM 8.5.7 video:

  http://ibm.biz/IBM_PFS_857_Video

- Validate event handler

  http://ibm.biz/BPMCoachViewValidateEventHandler

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

Deliver Modern UI for IBM BPM with the Coach Framework and Other Approaches

SG24-8355-00

ISBN 0738442011

IBM

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8355-00

ISBN 0738442011

Printed in U.S.A.