IBM OpenPages GRC
Version 8.0.0

OpenPages Business Process Author's Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 59.

Product Information

This document applies to IBM OpenPages GRC Version 8.0.0 and may also apply to subsequent releases.

Contents

Introduction

You can use IBM® Business Process Manager to develop and implement automated business process solutions that meet the needs and requirements of IBM OpenPages® GRC Platform.

Audience

The *IBM OpenPages Business Process Author's Guide* is intended for business process authors. These users develop workflow solutions and understand how IBM OpenPages GRC Platform and IBM Business Process Manager are integrated.

Please read the following important information regarding IBM OpenPages GRC documentation

IBM maintains one set of documentation serving both cloud and on-premise IBM OpenPages GRC deployments. The IBM OpenPages documentation describes certain features and functions which may not be available in OpenPages GRC on Cloud. For example, OpenPages GRC on Cloud does not include integration with IBM Business Process Manager and certain administrative functions.

If you have any questions about the functionality available in the product version that you are using, please contact IBM OpenPages Support via the IBM Support Community.

Finding information

To find product documentation on the web, including all translated documentation, access IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter).

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products. IBM OpenPages GRC Platform documentation has accessibility features. PDF documents are supplemental and include no added accessibility features.

Forward-looking statements

This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Chapter 1. Overview

Through the integration of IBM OpenPages GRC Platform and IBM Business Process Manager, you can access an enhanced level of GRC process automation. IBM Business Process Manager is an industry-leading process automation system that is both scalable and highly configurable.

You can develop workflow solutions that align with your requirements. You can also configure custom coach pages that show object information in a form that uniquely meets the needs of the user task. When a business process calls a coach page, the user must input information to continue the business process. Additionally, you can use a set of integration toolkits, which align with IBM OpenPages GRC Platform APIs and leverage existing data and configuration.

Users can launch and work on GRC processes by working with the embedded IBM BPM Process Portal on the Home page. When users click the **Process Portal** tab, the system opens the native BPM Process Portal.

You need to have a solid understanding of IBM Business Process Manager before you begin the integration with IBM OpenPages GRC Platform.

To find information about IBM Business Process Manager, access http://www.ibm.com/support/knowledgecenter/SSFPJS_8.5.7/com.ibm.wbpm.main.doc/kc-homepage-bpm.html (http://www.ibm.com/support/knowledgecenter/SSFPJS_8.5.7/com.ibm.wbpm.main.doc/kc-homepage-bpm.html).

Chapter 2. OpenPages Platform Toolkit

The OpenPages Platform Toolkit is used to integrate parts of the IBM OpenPages GRC Platform and the IBM Business Process Manager.

Resources in the OpenPages Platform Toolkit

The OpenPages Platform Toolkit contains resources that are used to define business processes and build interaction between business processes and OpenPages. Using a dependency, you can link a process application to the OpenPages Platform Toolkit.

The OpenPages Platform Toolkit is a standard toolkit that is updated by OpenPages releases and fix packs. Do not edit the toolkit artifacts directly.

The resources available in the OpenPages Platform Toolkit are illustrated in the following graphic.

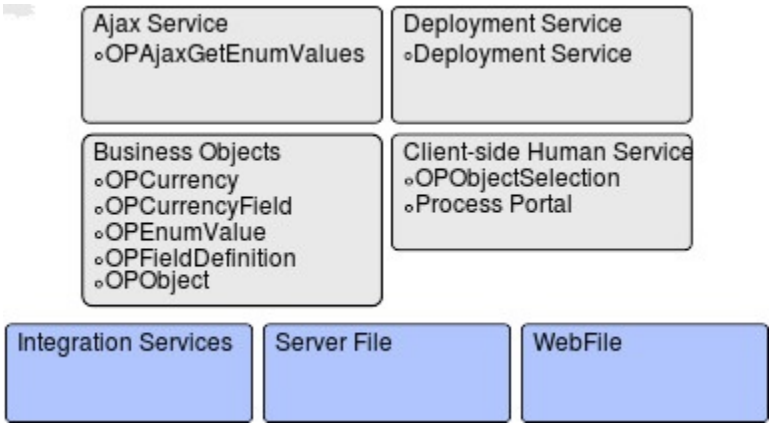


Figure 1. OpenPages Platform Toolkit

Ajax service

Ajax services interact with OpenPages from within coach views. An Ajax service in a coach view can make a REST API call to OpenPages. For example, you can use the Ajax service when you want a user to populate a drop-down control with values from OpenPages. For more information, see http://www.ibm.com/support/knowledgecenter/SSFPJS_8.5.0/com.ibm.wbpm.wle.editor.doc/develop/topics/tajaxservice.html (http://www.ibm.com/support/knowledgecenter/SSFPJS_8.5.0/com.ibm.wbpm.wle.editor.doc/develop/topics/tajaxservice.html).

Deployment service

The Deployment service adds calls and scripts that complete specific functions. It does this when a process application is deployed on a server in another environment.

Business objects

The integration services use the following business objects as input and output: OPCurrency, OPCurrencyField, OPEnumValue, OPFieldDefinition, and OPObject. A business object is a custom type that can be used as a variable for business data. This data must be available to the process and its services. Variables capture the business data that is used by activities in a business process definition, or by steps in services such as integration or human services.

Client-side human services

A human service is a component that creates a task for a user to work on. Human services provide the inclusion of coach components, which are the descriptions of screen data to be presented to users and data to be retrieved from users. Client-side human services include OPObjectSelection, which is used when users select an OpenPages object to work with in a business process. For an example of how to use OPObjectSelection, see “Using OPObjectSelection client-side human service ” on page 17. The **Process Portal** tab on the OpenPages Home page is also a client-side human service. Users access it to launch business processes, claim tasks, and work on tasks.

Integration services

Integration services interact with OpenPages data. The services are designed and built to support OpenPages. For more information, see Chapter 4, “OpenPages integration services,” on page 7.

Resource bundle groups

Resource bundle groups include the following:

- OPPlatformTexts is used by OPObjectSelection for UI-elements that are associated with OpenPages objects. Do not use OPPlatformTexts in your business processes.
- OPSystemFieldLabels is used by OPObjectSelection for field labels that are associated with OpenPages objects. Do not use OPSystemFieldLabels in your business processes.

Server file and WebFile

The server file and WebFile are automatically generated. The server file contains Java™ code that is deployed in the toolkit. WebFile contains image icons and stylesheets that are used when you design coach pages.

Chapter 3. OpenPages Solutions Toolkit

The OpenPages Solutions Toolkit is used to integrate parts of the IBM OpenPages GRC Platform and the IBM Business Process Manager.

Resources in the OpenPages Solutions Toolkit

The OpenPages Solutions Toolkit contains resources that give you access to business objects based on your unique object schema. You can access all business objects, their fields, and the relationships between fields. You can also access field labels, locales, and application text in multiple languages. The resources are used to build coach screens and workflow activities that give you access to the object data in IBM OpenPages GRC Platform. Using a dependency, you can link a process application to the OpenPages Solutions Toolkit.

The OpenPages Solutions Toolkit is generated based on your object schema. Therefore, it is unique for each OpenPages customer. You must regenerate it if you change the data model. Similar to the OpenPages Platform Toolkit, the OpenPages Solution Toolkit can be updated by OpenPages releases and fix packs. Do not edit the toolkit artifacts directly.

The resources available in the OpenPages Solutions Toolkit are illustrated in the following graphic.

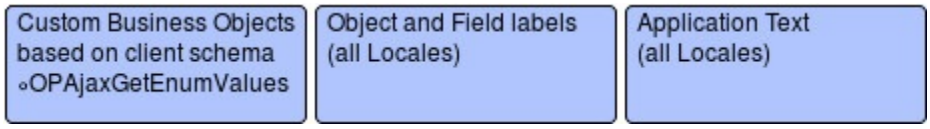


Figure 2. OpenPages Solutions Toolkit

Custom business objects based on client schema

OpenPages business objects that are specific to each customer begin with OP. For example, the OpenPages *SOXRisk* object type is named *OPSOXRisk* in IBM BPM. Variables capture the business data that is used by activities in a business process definition, or by steps in services such as integration or human services.

Object and field labels

Resource bundle groups give you access to copies of object texts and application texts in OpenPages. You can access resources in the process definitions to control what language to display in coach views for object and field labels.

- Use OPObjectLabels to access localized labels for object types. Both singular and plural labels are available.
- Use OPFieldLabels to access localized labels for object fields.

OpenPages data is specific to each customer.

Application text

Application text is specific to each customer. You can access this resource in process definitions to control what language to display in coach views for application text.

Use the OPAppText* resources to access localized application text by category. By default, the toolkit contains OPAppTextLabels, OPAppTextTitles, OPAppTextValidationMessages, and OPAppTextCustom. You can optionally copy more categories. Use the `-appTextCategories` option with the `op-bpm-tool.jar` command line tool.

For an example of how to use these resources, see “Using localization resources” on page 35.

Chapter 4. OpenPages integration services

The OpenPages Platform Toolkit contains integration services that you can use in business processes to access data in OpenPages.

Input and output parameters are required unless noted otherwise.

OPAssociateChildren

OPAssociateChildren is used to associate children with a specified object.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – object ID or full path to the parent object. String.

`children` – list of object IDs or full paths to associate. String list.

Output parameters

none

OPAssociateParents

OPAssociateParents is used to associate parents with a specified object.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – an object ID or full path to the child object. String.

`parents` – a list of object IDs or full paths to associate. String list.

Output parameters

none

OPCopyObjects

OPCopyObjects is used to copy one or more objects to a specific target parent.

You can copy an unlimited number of objects with OPCopyObject. However, if you copy a large number of objects, the operation can fail with an HTTP(S) read timeout. You can avoid this problem if you use a script to break the request into multiple REST API calls.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`targetParentId` – an object ID or full path to the target parent object. String.

`sourceObjectIds` – a list of source object IDs or full paths. String list.

`includesChildren` – a flag used to include children of source objects or copy source objects. Boolean.

`conflictBehavior` – a parameter that determines conflict behavior. Optional. String. Can be one of the following values:

- CREATECOPYOF – creates a copy and adds a prefix to the name of the copy in the target location. The prefix is `Copy of` for the first instance of a conflict, or `Copy (n) of` for the nth instance.
- OVERWRITE – overwrites the GRC object at the destination with the GRC object from the source.

- USEEXISTING – keeps the GRC object at the destination and associates it with the source objects that are being moved.
- ERROR – prompts an OpenPagesException if a conflict exists.

`childrenTypesToCopy` – object type IDs or names to copy. Optional. String list.

Output parameters

`copiedObjects` – a list of copied objects.

OPCreateObject

OPCreateObject is used to create an object based on specified field values.

For an example of how to use OPCreateObject, see "Using OPObjectSelection client-side human service " on page 17.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`inObject` – the object to create. The type of the object must be one of business objects from the OpenPages Solution Toolkit. Any.

Output parameters

`outObject` – a created object with an ID. The type of the object is the same as the `inObject`.

OPDeleteObject

OPDeleteObject is used to delete an object that is specified by the `objectId`.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – an object ID or full path to the object to be deleted. String.

Output parameters

none

OPDissociateChildren

OPDissociateChildren is used to dissociate children from a specified object.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – an object ID or full path to the parent object. String.

`children` – a list of object IDs or full paths to dissociate. String list.

Output parameters

none

OPDissociateParents

OPDissociateParents is used to dissociate parents from a specified object.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – an object ID or full path to the child object. String.

`parents` – a list of object IDs or full paths to dissociate. String list.

Output parameters

none

OPExecuteReportFragment

OPExecuteReportFragment is used to execute a given report fragment and return a value that is an HTML fragment.

Ensure that the HTML fragment does not contain links to icons or image files. They are not rendered in coach pages because the links are resolved within IBM Business Process Manager rather than Cognos® or IBM OpenPages GRC Platform.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – an object ID or full path to the object. String.

`field` – a field name in the `bundle-name:field-name` format. String.

Output parameters

`fragment` – an executed report fragment value (an HTML fragment).

OPGenericObjectQuery

OPGenericObjectQuery is used to issue a generic GRC object query to the IBM OpenPages GRC Platform REST API. This service is highly configurable and customizable. For more information about the query statement syntax, see the *IBM OpenPages GRC REST API Reference Guide*.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`queryStatement` – a Query SELECT statement using OpenPages API Query Syntax, for example, `SELECT [Resource ID], [Name] FROM [SOXIssue]`. String.

`isPrimary` – if joining multiple types in a query, set to `true` to honor primary associations between parents and children or set to `false` to return all associations. Boolean.

`isCaseInsensitive` – if comparing String values in a query, set to `true` to ignore differences in case. Default is `false`. Boolean.

`start` – starting index for results, `0` for the first result. Use to retrieve a given number of results rather than an entire list. For example, if `start` is 0 and `pageSize` is 500, the service returns up to the first 500 results (0-499) that are found by the query. If `start` is 500 and `pageSize` is still 500, the next 500 results are returned (500-999). Using `start` can improve performance for large data sets. Integer.

`pageSize` – the maximum number of query results that the request to REST API can return. Specify `pageSize` as `0` to return all query results that match the query filters. Integer.

`returnType` – if specified, the `returnType` is the business object type that will be returned by the query. If not specified, an OPObject business object is returned. If the query provided does not return columns for the business object, the behavior is unexpected. Optional. String.

Output parameters

`returnObjects` – a list of GRC objects that are returned for the query of the specified `returnType`, or OPObject if not specified. The queryStatement must select fields on the GRC object that match the fields in the specified `returnType`. Any fields in the `returnType` GRC object that are not part of your queryStatement's SELECT clause will be null.

OPGetBaseURL

OPGetBaseURL is used to retrieve the base URL for the OpenPages server, for example, `http://opserver:port/`. You can use this service, for example, to construct a URL to a Detail View page.

For an example of how to use OpGetBaseURL, see "Sending email notifications " on page 49.

Input parameters

None

Output parameters

`baseURL` – the base URL for the OpenPages server.

OPGetChildAssociations

OPGetChildAssociations is used to retrieve a list of child associations.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`ObjectId` – an object ID or full path to the parent object. String.

Output parameters

`children` – a list of OPObject types. It populates three fields of the OPObject: `id`, `path`, and `typeDefinitionId`. Other parameters of OPObject type are null.

OPGetEnumeratedValues

OPGetEnumeratedValues is used to retrieve a list of enumerated values (drop down values) for an enumerated field. When the dependency picklist is configured, the service caller passes the `controllingValue` argument to retrieve available enumerated values.

For an example of how to use OPGetEnumeratedValues, see "Adding enumerated fields with a dependent picklist to a coach page" on page 21.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`ObjectTypeName` – the name of the object type, for example, SOXBusEntity. String.

`fieldName` – the name of the field in the `bundle-name:field-name` format, example, `OPSS-BusEnt:Entity Type`. String.

`ControllingValue` – the value of the controlling field used when the dependency picklist is configured. Optional. String.

Output parameters

`outObject` – a list of enumerated values. The type is a list of values of OPEnumValue.

OPGetObject

OPGetObject is used to return business object information from a GRC object.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`ObjectId` – an object ID or full path to the object. String.

`computeReportFields` – a flag used to retrieve computed field values or leave them null. Boolean.

`Fields` – a list of field names to retrieve in the `bundle-name:field-name` format. Optional. String list.

Output parameters

`outObject` – a retrieved OpenPages object. The type is determined based on the object type to retrieve. It is one of the business objects defined in the OpenPages Solution Toolkit.

OPGetParentAssociations

OPGetParentAssociations is used to get a list of parent associations.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – an object ID or full path to the child object. String.

Output parameters

`parents` – a list of OPObject types. Three fields are populated in the OPObject: `id`, `path`, and `typeDefinitionId`. Other parameters of OPObject type are null.

OPHierarchicalAssigneeQuery

OPHierarchicalAssigneeQuery is used in special cases to make a hierarchical query for child objects under a specific parent instance. This method allows for a single field to be returned along with system fields for the child object type. The field is specified by the `assigneeField` parameter to issue a generic GRC object query to the IBM OpenPages GRC Platform REST API.

For an example of how to use OPHierarchicalAssigneeQuery, see "Defining basic hierarchical processes" on page 37.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`parentType` – the parent object type name to query. String.

`parentId` – the ID of the parent instance. String.

`childType` – the child object type name to query and return. String.

`assigneeField` – a field name that identifies a field on the child object type to be returned in the results. You can qualify the field using the `field_group:field` naming convention. String.

`isPrimary` – if joining multiple types in a query, set to `true` to honor primary associations between parents and children, or set to false to return all associations. Boolean.

`isCaseInsensitive` – if comparing string values in a query, set to `true` to ignore differences in case. The default is `false`. Boolean.

`filters` – used to apply additional filter conditions to the query. Must be specified in valid query API syntax: `[Object Type].[Field Group:Field Name]`. The filters must be for fields on either the parent type or child type for the query. Null is passed if no additional filters are needed. String.

`start` – starting index for results, 0 for the first result. Use to retrieve a given number of results rather than an entire list. For example, if `start` is 0 and `pageSize` is 500, the service returns up to the first 500 results (0-499) that are found by the query. If `start` is 500 and `pageSize` is still 500, the next 500 results are returned (500-999). Using `start` can improve performance for large data sets. Integer.

`pageSize` – the maximum number of query results that the request to REST API can return. Specify `pageSize` as `0` to return all query results that match the query filters. Integer.

`isDirect` – used to specify whether the query checks the entire hierarchy for the child type or immediate children. Boolean.

Output parameters

`returnObjects` – a list of GRC objects that are returned as children of the parent, including the field specified by the `assigneeField`.

OPHierarchicalQuery

OPHierarchicalQuery is used to make a hierarchical query for child objects under a specific parent instance.

For an example of how to use OPHierarchicalQuery, see "Retrieving a list of child objects" on page 48.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`parentType` – the parent object type name to query. String.

`parentId` – the ID of the parent instance. String.

`childType` – the child object type name to query and return. String.

`isPrimary` – if joining multiple types in a query, set to `true` to honor primary associations between parents and children, or set to false to return all associations. Boolean.

`isCaseInsensitive` – if comparing string values in a query, set to `true` to ignore differences in case. Default is `false`. Boolean.

`filters` – used to apply additional filter conditions to the query. Must be specified in valid query API syntax: `[Object Type].[Field Group:Field Name]`. The filters must be for fields on either the parent type or child type for the query. Null is passed if no additional filters are needed. String.

`fields` – if specified, is an explicit list of fields that will be returned by the query. Fields must be comma-delimited and specified in the `field_group:field` format. Must be specified in valid query API syntax and will be used as the SELECT clause of the query statement in place of the defaults. Specifying only fields from the child type is allowed. If not specified, only system fields from the child type are returned by default. Optional. String.

`fields` – if specified, is an explicit list of fields that will be returned by the query. Fields must be comma-delimited and either specified in the `[return_type].[field_group:field]` format or with one of the following special keywords: `Id` for only the resource ID or `System Fields` for all the system fields from the child type. Must be specified in valid query API syntax and will be used as the SELECT clause of the query statement in place of the defaults. Specifying only fields from the child type is allowed. If not specified, only the resource ID system field from the child type is returned by default. Optional. String.

`start` – starting index for results, 0 for the first result. Use to retrieve a given number of results rather than an entire list. For example, if `start` is 0 and `pageSize` is 500, the service returns up to the first 500 results (0-499) that are found by the query. If `start` is 500 and `pageSize` is still 500, the next 500 results are returned (500-999). Using `start` can improve performance for large data sets. Integer.

`pageSize` – the maximum number of query results that the request to REST API can return. Specify `pageSize` as `0` to return all query results that match the query filters. Integer.

`isDirect` – used to specify whether the query checks the entire hierarchy for the child type or immediate children. Boolean.

Output parameters

`returnObjects` – a list of GRC objects that are returned as children of the parent.

OPLaunchChildProcesses

OPLaunchChildProcesses is used to launch child processes in a hierarchical process.

For an example of how to use OPLaunchChildProcesses, see "Defining advanced hierarchical processes " on page 41.

Input parameters

`ProcessName` – name of the business process definition to launch. String.

`InputValues` – each entry in the list is passed as an input value to a new process that is launched. The size of the list determines how many processes are launched. The InputValues map matches the Input type variables of the process that is being launched. The key in the map is the Input variable name and the value in the map is the value passed to the variable. Map list.

`Delay` – time in milliseconds that the service pauses after launching a process. Use a longer delay to avoid performance issues. Integer.

`AddDependency` – adds a dependency from the launched child process to the parent process. Making the parent dependent on the launched child processes indicates to IBM Business Process Manager that there is a relationship between the two processes. The parent process cannot complete until all children processes are completed. Boolean.

Terminating the parent process also terminates all child processes.

Output parameters

`ChildPIDs` – list of the child process instances' IDs. String list.

OPLockObject

OPLockObject is used to lock an object that is specified by the `objectId`.

For an example of how to use OPLockObject, see "Locking and unlocking objects" on page 49.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`ObjectId` – an object ID or full path to the object. String.

Output parameters

none

OPMakeAddNewLink

OPMakeAddNewLink is used to make an HTML hyperlink tag to open the Add New wizard in OpenPages. You can add a specific new object type and optional parameters for parent, parent type, and view name. The link opens in a new window.

For an example of how to use OPMakeAddNewLink, see "Adding file attachments to a coach page" on page 28.

Input parameters

`newObjectType` – the name of the object type to add. String.

`parentId` – the resource ID for the default selected parent in the Add New wizard. Optional. String.

`parentType` – the object type of the default parent. Optional. String.

`viewName` – a specific view for the new object type. Optional. String.

`linkText` – the text to display for the link. String.

Output parameters

`addNewLink` – the HTML code used to create a link to open the Add New wizard for the object. String.

OPMakeDetailLink

OPMakeDetailLink is used to make an HTML hyperlink tag that opens an object's Detail View in OpenPages. You can use an optional parameter provide a profile view name.

For an example of how to use OPMakeDetailLink, see "Adding a link to an OpenPages Detail page on a coach page " on page 25.

Input parameters

`fileId` – the resource ID of the linked object in OpenPages. String.

`viewName` – a specific view for the object type. If not specified, the default is the detail view. Optional. String.

`rootOPAppContext` – the OpenPages root context path, for example, `openpages`

`linkText` – the text to display for the link. String.

Output parameters

`detailViewLink` – the HTML code used to open the Detail View for the object. String.

OPMakeDocumentLink

OPMakeDocumentLink is used to make an HTML hyperlink tag to download a file directly from OpenPages.

For more information about task-oriented hyperlinks, see https://www.ibm.com/support/knowledgecenter/SSFUEU_8.0.0/op_grc_admin/c_adm_task_oriented_hyperlinking.html.

For an example of how to use OPMakeDocumentLink, see "Downloading file attachments on coach pages" on page 30.

Input parameters

`fileId` – the resourceId of the document file. Must be an object type of SOXDocument. String.

`rootOPAppContext` – the OpenPages root context path, for example, `openpages`

`linkText` – the text to display for the link. String.

Output parameters

`documentLink` – the HTML code used to create a link to download a file. String.

OPMoveObjects

OPMoveObjects is used to move one or more objects to a specific target parent.

You can move an unlimited number of objects with OPMoveObject. However, if you move a large number of objects, the operation can fail with an HTTP(S) read timeout. You can avoid this problem if you use a script to break the request into multiple REST API calls.

Input parameters

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`targetParentId` – the object ID or full path to the target parent object. String.

`sourceObjectIds` – a list of source object IDs or full paths. String list.

`conflictBehavior` – a parameter that determines conflict behavior. Optional. String. Can be one of the following values:

- OVERWRITE – overwrites the GRC object at the destination with the GRCObject from the source.

- USEEXISTING – keeps the GRC object at the destination and associates it with the source objects that are being moved.
- ERROR – prompts an OpenPagesException if a conflict exists.

`childrenTypesToMove` – object type IDs or names to move. Optional. String list.

Output parameters
None

OPPerformRESTGet

OPPerformRESTGet is a generic integration service that you can use to access IBM OpenPages GRC Platform REST APIs. You can use it if no integration service meets your requirements but a REST API does.

Input parameters
`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`requestPath` – IBM OpenPages GRC Platform REST API path relative to `<OpenPages application URL>/grc/api`.

You can add parameters to the end of the URL. For example, if `requestPath` is the following:

```
/contents/12345/permissions/effective?user=orm
```

The effective access control is retrieved for the `orm` user on the object whose ID is 12345.

Output parameters
`response` – raw string, typically in REST API JSON format, from the REST API invocation. You can use IBM Business Process Manager scripting to extract data from the raw string. Example:

```
var jsonResult = JSON.parse(tw.local.response);
       log.info("Effective permission  for a user:" + jsonResult.securityPrincipal);
       log.info("\tcanRead:" + jsonResult.canRead);
       log.info("\tcanWrite:" + jsonResult.canWrite);
```

OPUnlockObject

OPUnlockObject is used to unlock an object that is specified by the `objectId`.

For an example of how to use OPUnlockObject, see “Locking and unlocking objects” on page 49.

Input parameters
`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`ObjectId` – the object ID or full path to the object. String.

Output parameters
none

OPUpdateObject

OPUpdateObject is used to update an object based on specified field values.

Input parameters
`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`inObject` – the object to update. The type of the object must be one of the business objects from the OpenPages Solution Toolkit. Any.

Output parameters
`outObject` – the updated object. The type of the object is the same as `inObject`.

Team Filter by Object Field

Team Filter by Object Field is a Team Filter Service that is used to assign tasks to a user whose name is in an object field, for example, Control Owner or Risk Owner. You can use it for user groups and for multi-select users and groups.

Input parameters

`originalTeam` – a parameter reserved for a Team Filter Service. Team.

`systemTask` – a flag used to run this operation as a system account or current user. Boolean.

`objectId` – the object ID or full path to the object. String.

`field` – field name in the `bundle-name:field-name` format. String.

`expandGroups` – flag to resolve group memberships recursively and to include only users in the filtered team. This can impact system performance if the group contains many users and subgroups. Because groups from OpenPages might not be available in IBM Business Process Manager, set to `true` in the LDAP configured environment. Boolean.

Output parameters

`filteredTeam` – a parameter that is reserved for a Team Filter Service. Team.

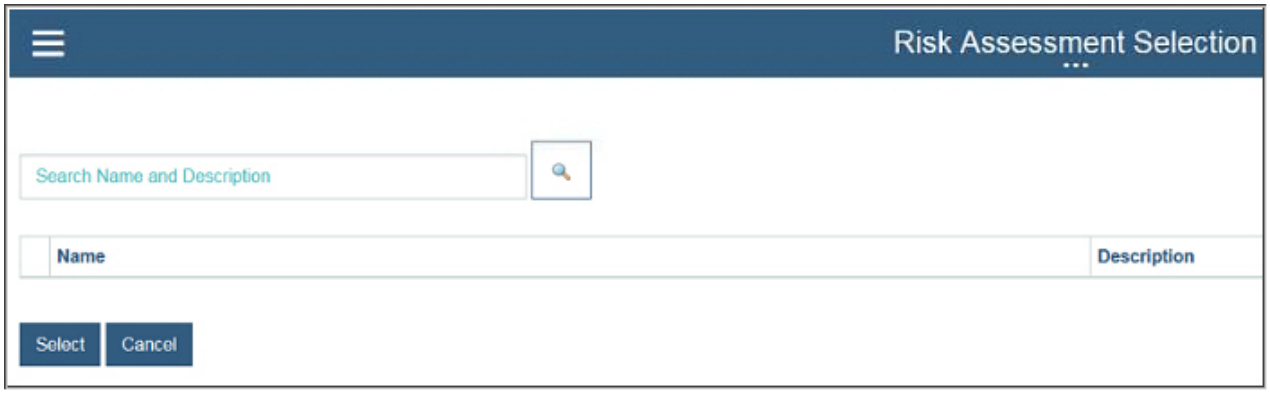
Chapter 5. Process authoring examples

Process authors can use the examples in this document to find solutions to common problems that occur when implementing IBM Business Process Manager with IBM OpenPages GRC Platform.

Using OPObjectSelection client-side human service

You can use the OPObjectSelection client-side human service to give users the ability to select an OpenPages object in a business process.

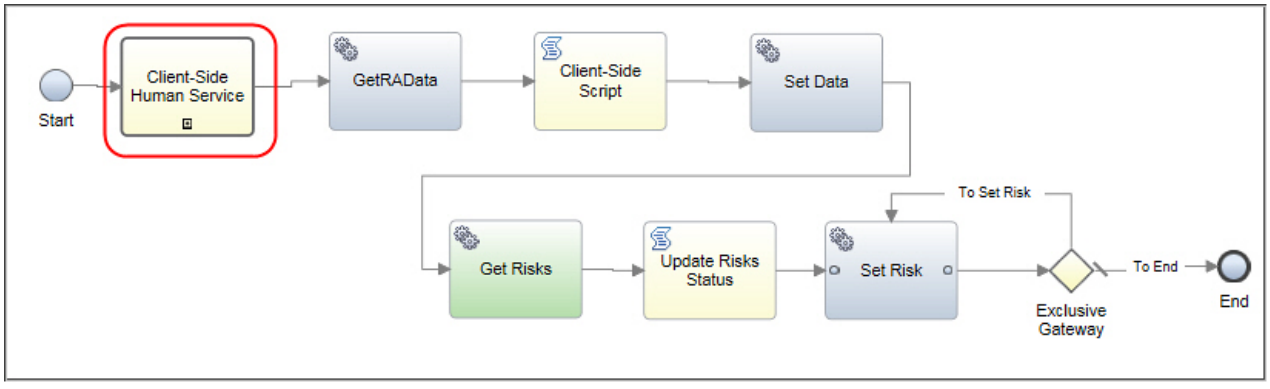
The OPObjectSelection client-side human service displays the following user interface to a user:



The title of the interface is the object type. Otherwise, the interface is the same regardless of what object type is used.

A user types a value and clicks the search icon. The search returns a list of objects where a match is found in the name or description. Users can then select an object or cancel. The object that is selected is passed back to the business process and the process continues.

In the following example, OPObjectSelection is inserted as the first step in a client-side human service:



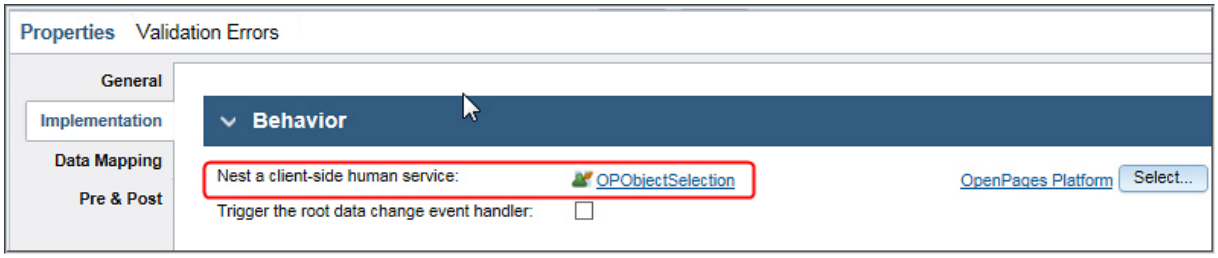
The object type that is used in OPObjectSelection is defined in a variable or an earlier task, depending on how the business process is defined.

In addition, you can use OPObjectSelection as a model to design your own object selection client-side human services.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.

3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Drag **Nested client-side human service** from the palette onto the canvas.
7. On the **Properties** pane, select the **Implementation** tab.
8. In **Nest a client-side human service**, select OPObjectSelection.



9. Select the **Data Mapping** tab, and define the input and output mapping.

10. You can now define the overall client-side human service.

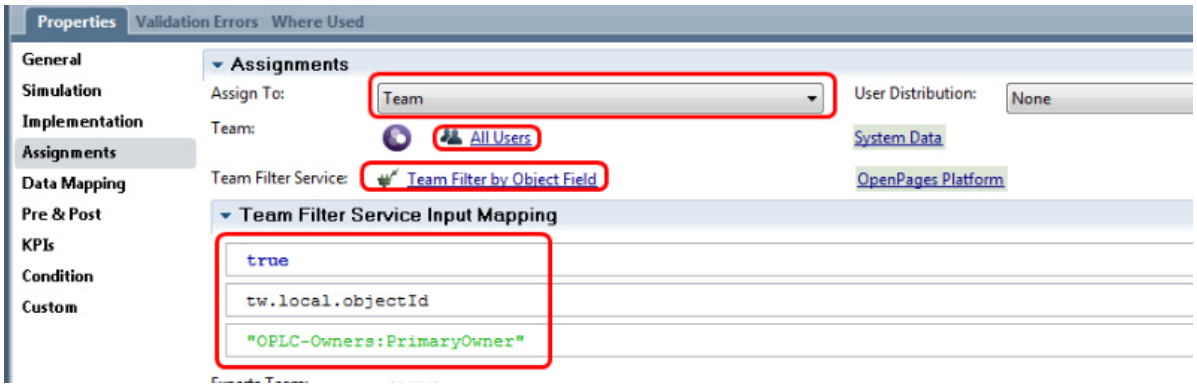
Assigning a process task based on a field value

You can assign a process task to a user based on the value of a specified field. In this example, you create a new activity in a process and assign that activity to an object's primary owner.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Create a process.
6. Drag an **Activity** from the palette onto the canvas.
7. On the **Properties** pane, select the **Assignments** tab.
   a) In **Assign to**, select Team.
   b) In **Team**, select All Users.
   c) In **Team Filter Service**, select Team Filter by Object Field.
   d) Specify **Team Filter Service Input Mapping** as:

- In **systemTask**, define whether the task can access OpenPages as system account or the current account.
- In **objectId**, enter the ID of the object. It can be a local variable.
- In **field**, enter a field definition string. It can be a local variable.



8. Save the process.

Adding rich text fields to a coach page

If you define coach pages that have rich text fields, different formatting behavior occurs in the OpenPages rich text editor (CKEditor) when compared to the IBM BPM rich text editor. In this example, you add a rich text field to a coach page.

For rich text fields that you define on a coach page with the BPM editor, not all formatting is preserved in the CKEditor. After a user enters text in a rich text field on a coach page, the field appears in read-only mode on a Detail page in OpenPages. The field displays with the formatting that the user specified on the coach page. However, if the user edits the object by using the CKEditor, the following formatting is removed:

- font size
- font type
- paragraph alignment (left, right, center)
- style attributes on indent tags that use `<blockquote>`

Similarly, for rich text fields that you define in OpenPages with the CKEditor, not all formatting is preserved in the BPM editor. After a user enters text in a rich text field in OpenPages, the field can be viewed on a coach page. Most of the formatting is preserved when the text is displayed on a coach page, with the following exceptions:

- emoticons
- paragraph alignment (justified)
- document bookmarks
- insert tables
- font colors
- styles for CKEditor-specific CSS classes

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.

a) Click **+** next to **Toolkits** and select OpenPages Platform.

b) Click **+** next to **Toolkits** and select OpenPages Solutions.

5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Click the **Coaches** tab and select a template.
7. Drag the **Text Area** view from the palette onto the canvas.
8. In the **Properties** pane, click the **Configuration** tab.
9. Set **Text area type** to Rich Text.
10. Save the coach page.

Adding single or multiple enumerated fields to a coach page

You can add single or multiple enumerated fields to a coach page. In this example, you create a new client-side human service and define a single enumerated field for a coach page.

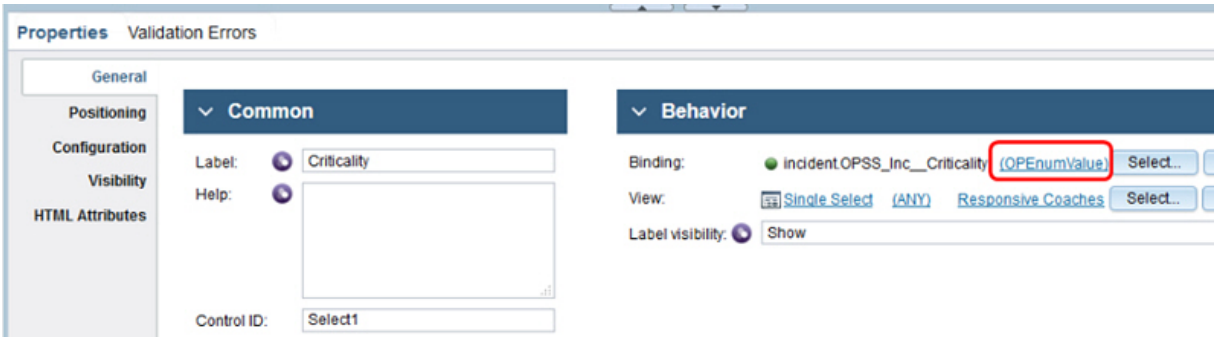
Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.

a) Click **+** next to **Toolkits** and select OpenPages Platform.

b) Click **+** next to **Toolkits** and select OpenPages Solutions.

5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Click the **Coaches** tab and select a template.
7. Drag the **Single Select** view or the **Multiple Select** view from the palette onto the canvas.
8. In the **Properties** pane, specify a label and bind the control to a variable. In this example, the variable is bound to an Incident object's `OPSS-Inc` field group, `Criticality` field. Ensure that the bound variable type is OPEnumValue.



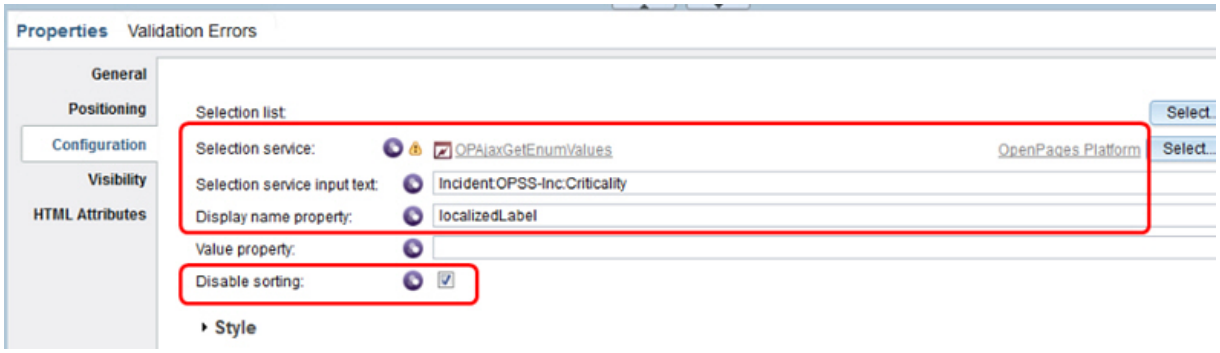
9. Click the **Configuration** tab and define the following fields:

a) In **Selection service**, select the OPAjaxGetEnumValues integration service from the OpenPages Platform Toolkit.

b) In **Selection service input text**, specify an object type, field group, and field in the following format: `<object type>:<field group>:<field>`.

c) In **Display name property**, select localizationLabel.

d) Select **Disable sorting** to maintain the order that is configured in OpenPages.



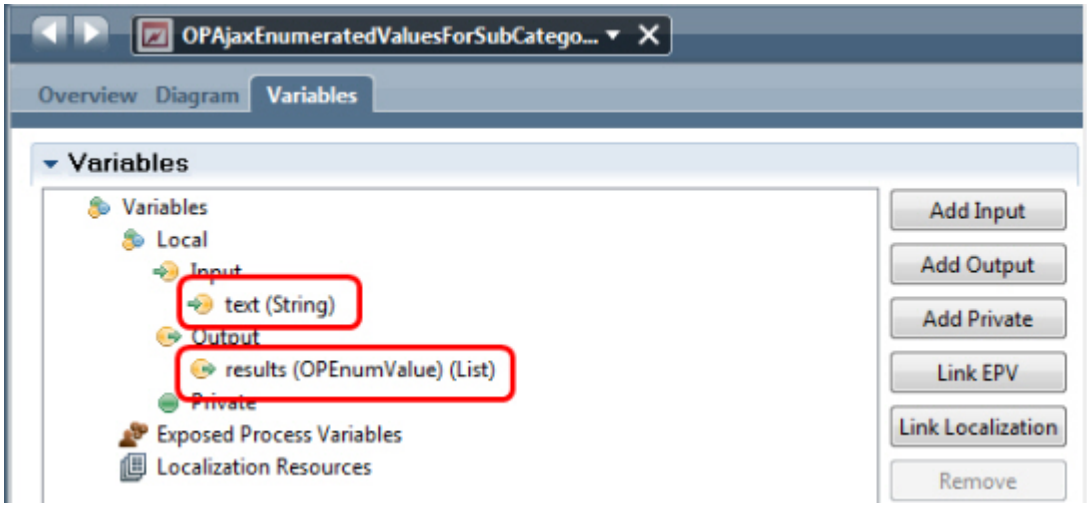
10. Save the coach page.

Adding enumerated fields with a dependent picklist to a coach page

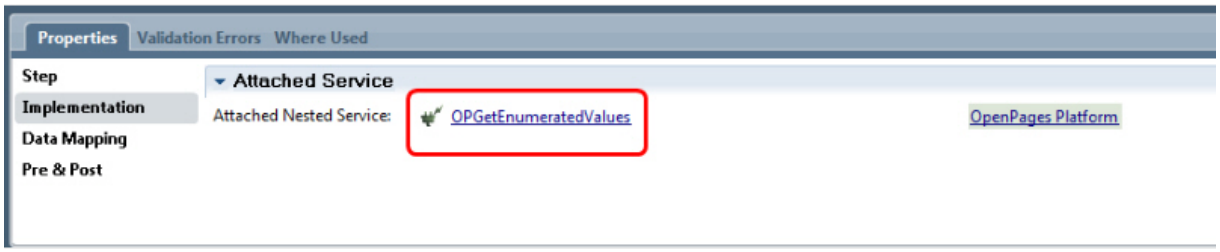
You can add enumerated fields with dependent picklists to a coach page. In this example, you create a new client-side human service and define an enumerated field with a dependent picklist for a coach page.

Procedure

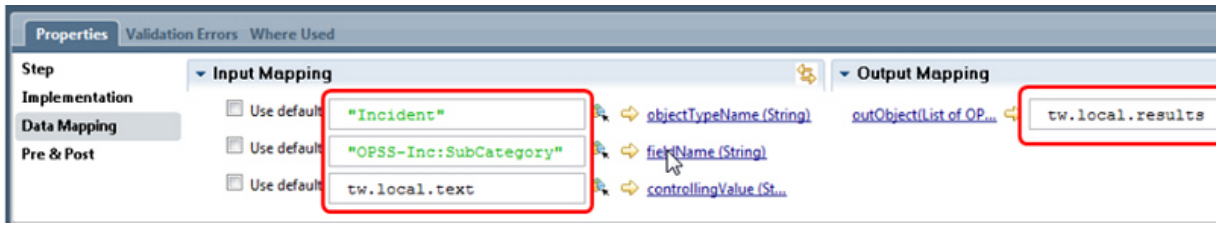
1. Open the Process Designer desktop editor.
2. Open the Process Designer.
3. Add dependencies to the OpenPages toolkits.
   a) Click **Toolkits** and select OpenPages Platform.
   b) Click **Toolkits** and select OpenPages Solutions.
4. Click **New** beside **User Interface** > **Ajax Service**, and enter a name for your service.
5. In the **Properties** pane, add the `text` input variable and the `results` output variable. The variable name and variable type must match.



6. Click the **Diagram** tab and drag a **Nested Service** from the palette onto the canvas.
7. In the **Properties** pane, click the **Implementation** tab, and select the OPGetEnumeratedValues service from the OpenPages Platform Toolkit.



8. Click the **Data Mapping** pane. Specify the following parameters:
   a) In **objectTypeName**, enter the object type of the enumerated field that is being picked.
   b) In **fieldName**, enter the object field group and field of the enumerated field that is being picked.
   c) In **controllingValue**, enter `tw.local.text`.
   d) In **outObject**, enter `tw.local.results`.



9. Save the Ajax service.
10. Open the Process Designer web editor.
11. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
12. Click the **Coaches** tab and select a template.
13. Drag the **Single Select** view from the palette onto the canvas.
14. In the **Properties** pane, specify a label and bind the control to an `OPEnumValue` variable.
15. Click the **Configuration** tab and define the following fields:
    a) In **Selection service**, select the service that you created.
    b) In **Selection service input text**, enter the name field of the controlling enumerated field.
    c) In **Display name property**, select localizedLabel.
    d) Select the **Disable sorting** checkbox to maintain the order that is configured on the OpenPages side.

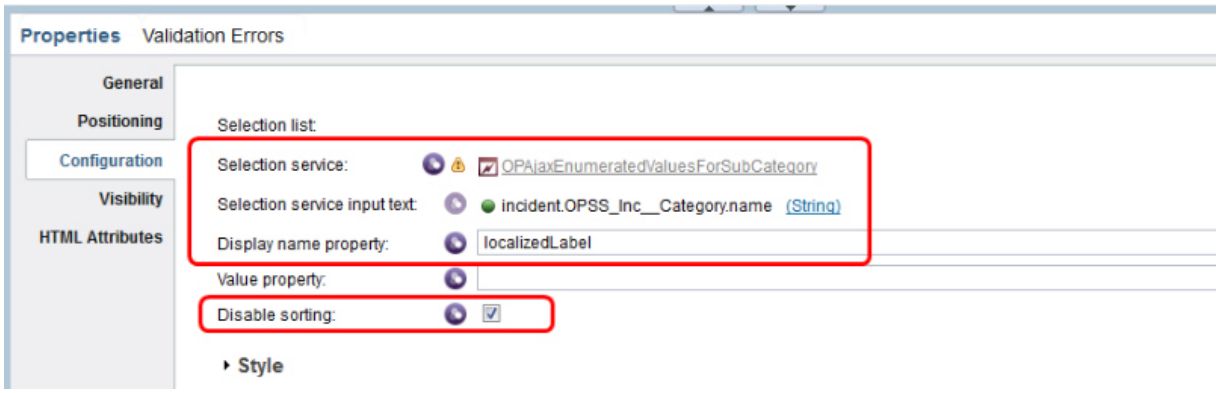


Figure 3. Defining the fields on the Configuration tab

16. Save the coach service.

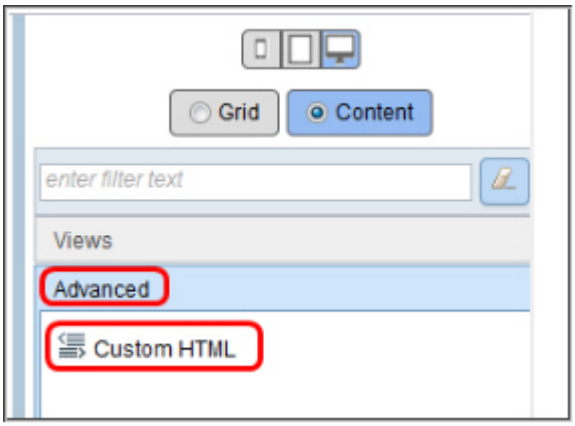
Adding computed fields to a coach page

You can add computed fields to a coach page. In this example, you create a new client-side human service and define a computed field for a coach page.

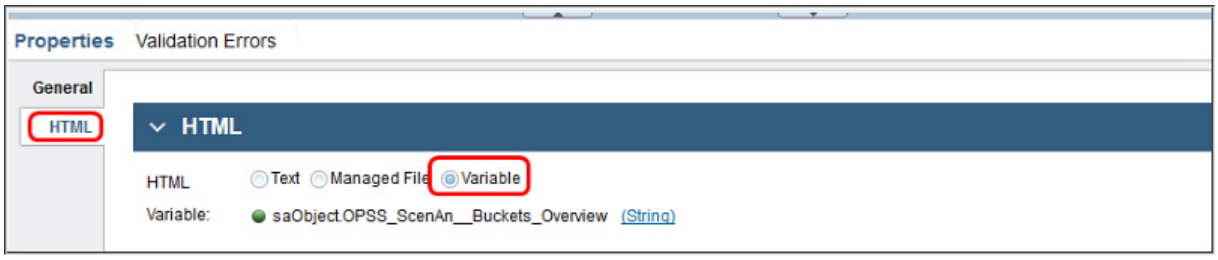
The business object type of computed fields is String. The value normally contains raw HTML tags.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Click the **Coaches** tab and select a template.
7. Click **Views** > **Advanced** and drag **Custom HTML** from the palette onto the canvas.



8. In the **Properties** pane, select the **HTML** tab.
9. Select **Variable**.
10. Specify a label and bind the control to a variable. In this example, the variable is bound to Scenario Analysis' OPSS-ScenAn field group, `Buckets Overview` field.



11. Save the coach page.

Adding Owner fields to a coach page

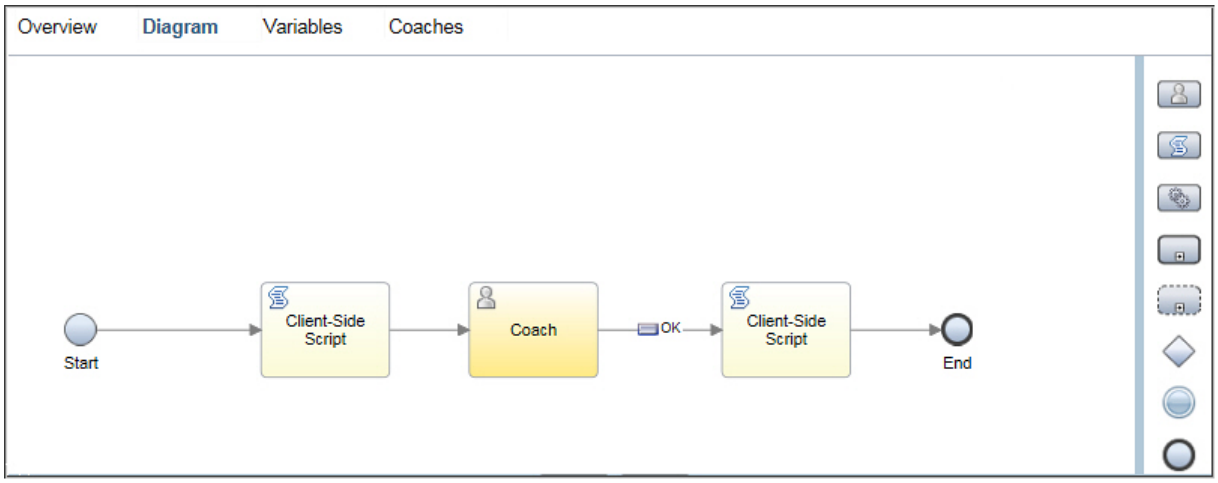
You can add Owner fields to a coach page. In this example you create a new client-side human service, define a coach page, and add an Owner field to it.

Owner fields that use the multiple user selector display type are displayed on a coach page as `$;<value>$;`. For example, the value `John Smith` for a Primary Owner is displayed on a coach page as `$;John Smith$;`. You must programmatically remove `$;`. Owner fields that use the single user selector display type do not display `$;`.

In this example, assume that a client-side human service has an Owner field value in a variable named `OwnerField`. It contains a value of `$;OpenPagesAdministrator$;itg$;fcm$;`. Before you display the value on a coach page, you must convert it to a more user readable format. After the value has been changed by a coach page, it must be converted back to the OpenPages format.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Click the **Variables** tab and add an `OutputOwnerField` private variable of type String.
7. Click the **Diagram** tab.
8. Drag a **Client-side script** from the palette onto the canvas. Place it after the Start. Connect the arrows link from the Start node to the new Client-side script node, and connect the Client-side script node to the Coach.
9. Drag a second **Client-side script** node from the palette onto the canvas. Place it after the Coach and connect the arrows. Give all the nodes descriptive names.



10. Click the first Client-Side Script node.
    a) In the **Properties** pane, click **Script**.
    b) Enter the script code. The following code converts the value in the `OwnerField` variable from the OpenPages format to the display format and returns it in `OutputOwnerField`.

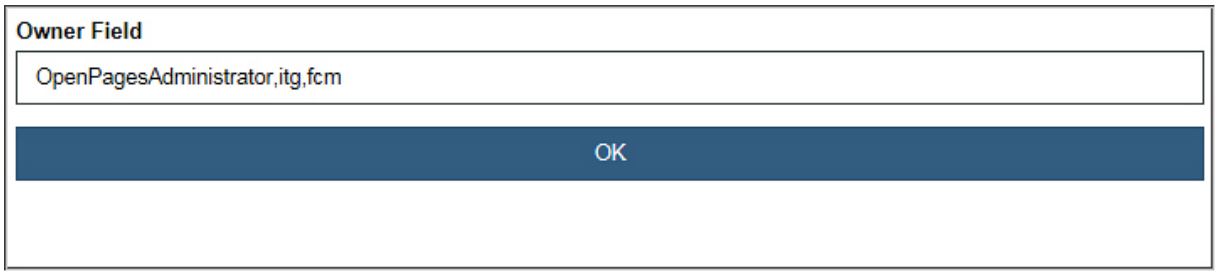
```
if( tw.local.OwnerField !== null){
//split based on the OP delimiter for multi-user selectors
    var names = tw.local.OwnerField.split("$;");

    //array has a trailing and leading empty string
 //due to the extra delimiters in OP value
```

```
    names.pop();
    names.shift();

    //use JS arrays.join() to concatenate the array of names
 //into comma-delimited list format
    tw.local.OutputOwnerField = names.join(",");
}
```

11. Click the second Client-Side Script node.
    a) In the **Properties** pane, click **Script**.
    b) Enter the script code. The following code reverses the value in the `OutputOwnerField` variable to the OpenPages format. It returns it in the `OutputOwner` variable, which you can then save back to the original OpenPages object.

```
if( tw.local.OutputOwnerField !== null){

    var names = tw.local.OutputOwnerField.split(",");



    //array needs trailing and leading empty string
    //due to the extra delimiters in OP value

    names.push("");

    names.unshift("");



    //use JS arrays.join() to concatenate the array of names
    //into OP multi-user list format

    tw.local.OwnerField = names.join("$;");

}
```

12. Click the **Coaches** tab and select a template.
    a) Drag a **View**, such as the **Text** view, from the palette onto the canvas.
    b) On the **Properties** pane, next to **Label** type `Owner Field`.
    c) Next to **Binding**, click **Select** and choose the `tw.local.OutputOwnerField` variable.
13. Save the coach page.
14. Run the client-side human service. The coach page displays a text field that contains user names in comma-delimited format.



15. If you click OK, it returns the value in the text field. The second script converts the value to the OpenPages format. Your process can then, for example, update the object's owner field with the new value.

    Note: The code in this example does not validate names that users enter. Users must enter multiple names as comma-delimited values.

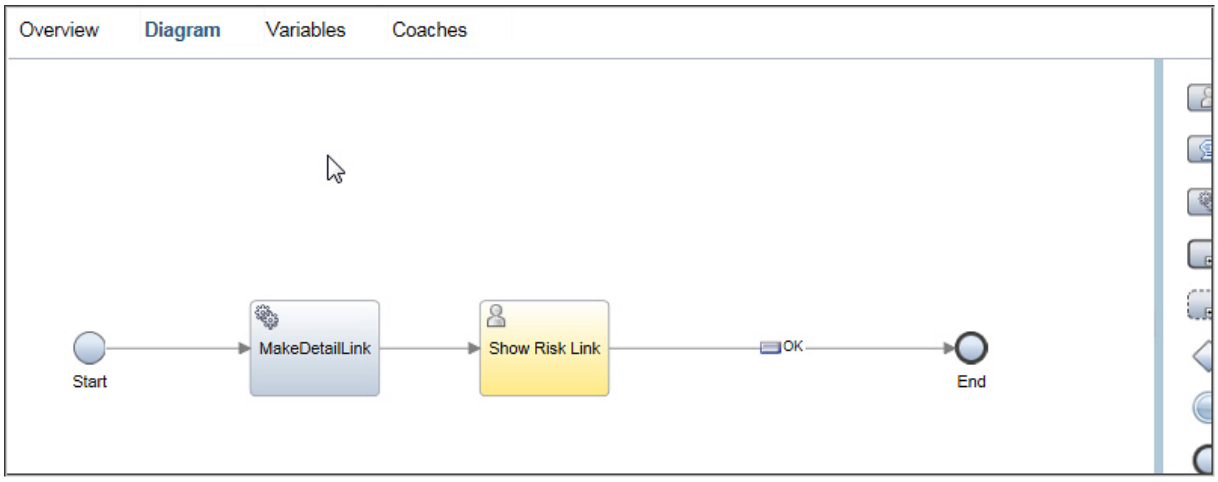
Adding a link to an OpenPages Detail page on a coach page

You can add a link to an OpenPages Detail page on a coach page. In this example, you create a new client-side human service and define a link on a coach page that opens an OpenPages Detail page. For

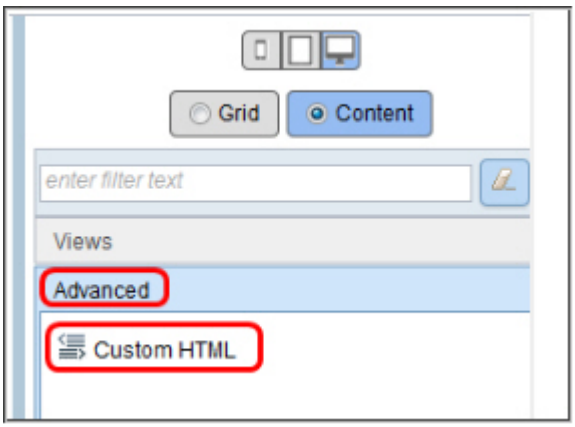
testing, you enter the value of an existing object's resource ID in OpenPages. In a real implementation, you specify the ID of the object you want to link to with an input variable on the client-side human service.

Procedure

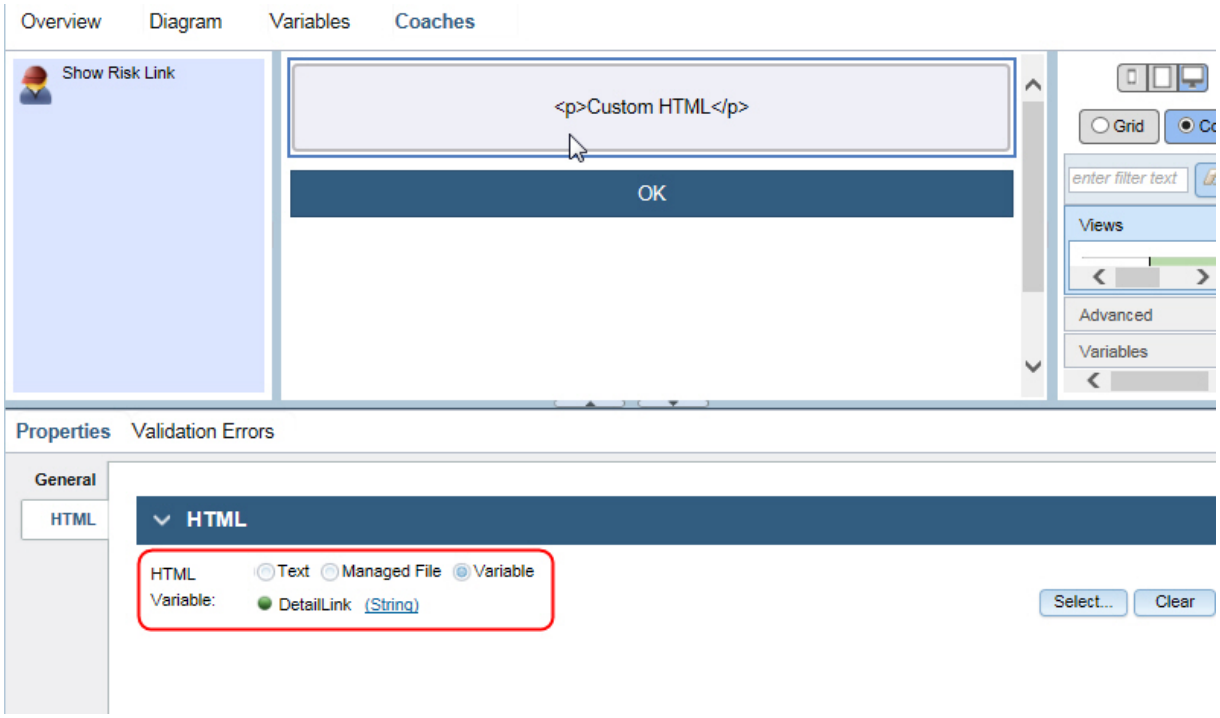
1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface, such as, `Risk Detail Link`.
6. Click the **Variables** tab, and add a `DetailLink` private variable of type String.
7. In **Input**, add a `RiskId` input variable of type String. Select **Has default**. For testing, enter the value of an existing Risk object's resource ID in OpenPages.
8. Click the **Diagram** tab.
9. Drag a **Service** from the palette onto the canvas. Place it after the Start. Connect the arrows link from the Start node to the new Service node, and connect the Service node to the Coach. Give the nodes descriptive names.



10. Click the service node, and select the **Implementation** pane.
11. In the **Behavior** section, select **Call a Service** and select the OPMakeDetailLink integration service from the OpenPages Platform Toolkit.
12. Select the **Data Mapping** pane and specify the following parameters:
    a) In **fileId**, select the `RiskId` input variable that is named `tw.local.RiskId`.
    b) In **linkText**, enter `See Details in OpenPages`.
    c) Optionally, in **viewName**, enter the name of a profile view for the object type. If left blank, the default view for the user's profile is used.
    d) In **Output Mapping**, enter `tw.local.DetailLink` for **documentLink**.
13. Save the changes to the service
14. Click the **Coaches** tab and select a template.
15. Click **Views** > **Advanced** and drag a **Custom HTML** from the palette onto the canvas.



16. In the **Properties** pane, select the **HTML** tab. Click **Variable** and select `DetailLink` private variable.



Alternatively, you can use an Output Text view rather than a Custom HTML view in the coach page. Select the Output Text, under **Binding** select the `DownloadLink|DetailLink` private variable. Click the **Properties** pane, and select the **Configuration** tab. Change the setting for **Disable HTML encoding** from cleared to selected.

17. Save the coach page.

Note: When a user, who is not logged in, clicks the link from the coach page, that user is prompted to log in to OpenPages.

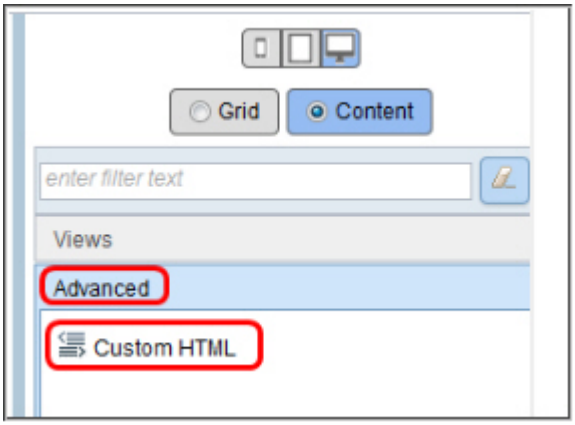
Adding a link to a Cognos report on a coach page

You can add a link to a Cognos report on a coach page. In this example, you create a new client-side human service and define a link on a coach page that opens a Cognos report.

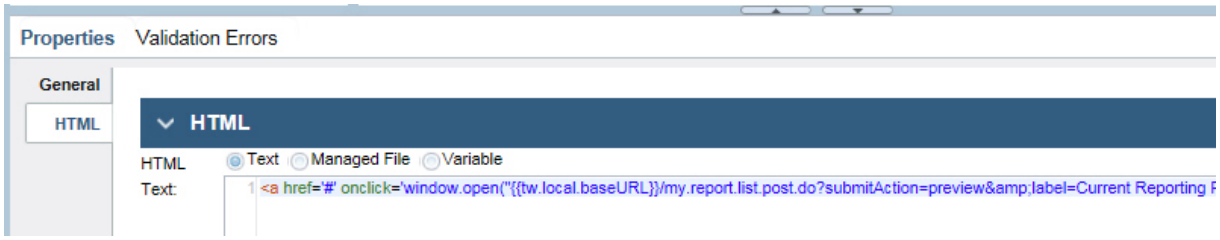
Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.

4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Click the **Variables** tab, and add a baseURL private variable of type String.
7. Click the **Coaches** tab and select a template.
8. Click **Views** > **Advanced** and drag **Custom HTML** from the palette onto the canvas.



9. In the **Properties** pane, select the **HTML** tab and add an HTML snippet:



The snippet can be:

```
<a href='#' onclick='window.open("{{tw.local.baseURL}}/my.report.list.post.do?
submitAction=preview&amp;label=Current Reporting Period&amp;reportId=899",
"Title","toolbars=no,status=yes,menubars=no")'>Audit Plan Report</a>
```

BPM substitutes the text in double brackets {{}} with the variable value.

10. You can substitute the report ID, 899 in this example, with the report you want to run. The report ID can be found in the URL when you run the report from OpenPages system. The baseURL variable is the OpenPages application URL, and it is redirected to the Cognos report.
11. Save the coach page.

Adding file attachments to a coach page

You can add a link to open a file attachment on a coach page. In this example, you create a new client-side human service and define a single link to open the Add New wizard to upload a new file to OpenPages. File attachments are stored on the OpenPages server.

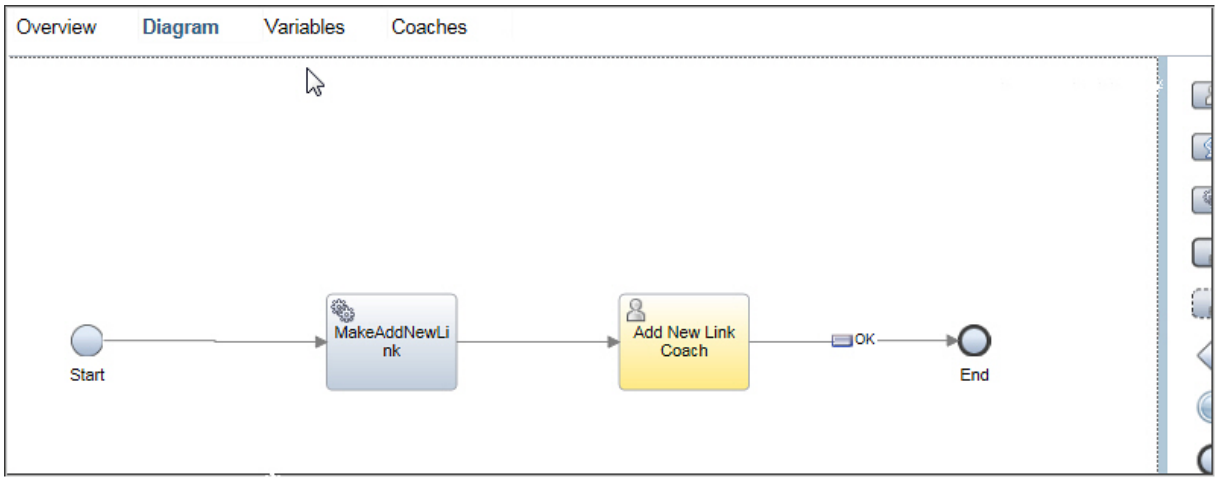
Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.

a) Click **+** next to **Toolkits** and select OpenPages Platform.

b) Click **+** next to **Toolkits** and select OpenPages Solutions.

5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface, such as, `Add New Document`.
6. Click the **Variables** tab, and add an `AddNewLink` private variable of type String.
7. Click the **Diagram** tab.
8. Drag a **Service** from the palette onto the canvas. Place it after the Start node. Connect the arrows link from the Start node to the new Service node, and connect the Service node to the Coach. Give the nodes descriptive names.



9. Click the service node, and select the **Implementation** pane.
10. In the **Behavior** section, select **Call a Service** and then select the OPMakeAddNewLink integration service from the OpenPages Platform Toolkit.
11. Select the **Data Mapping** pane, and specify the following parameters:

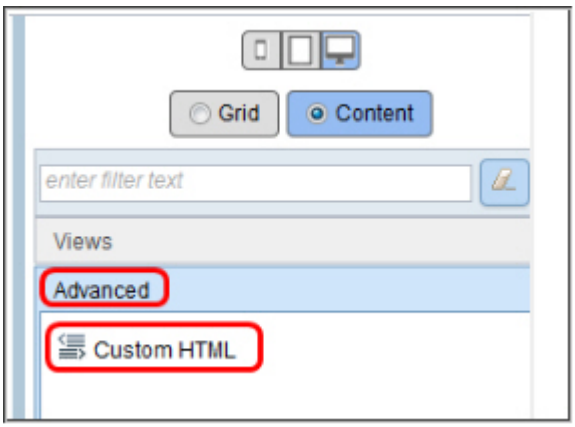
    a) In **newObjectType**, enter `SOXDocument`, which is the object type for file attachments.

    b) In **linkText**, enter `Add New Document`.

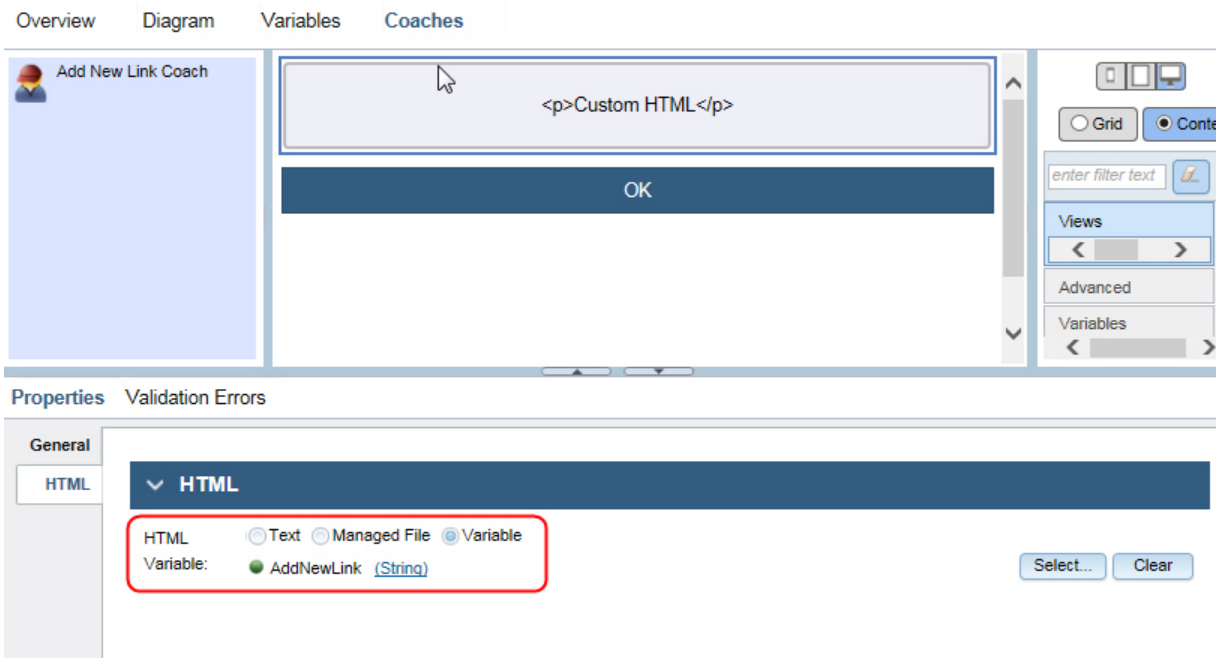
    c) Optionally, specify the behavior of the Add New wizard:

    - In **parentId**, enter the resource ID for the default parent that will be preselected as the primary parent in the Add New wizard.
    - In **parentType**, select the object type of the default parent, such as, SOXRisk.
    - In **viewName**, provide a creation view for the new object type.

    d) In **Output Mapping**, enter `tw.local.AddNewLink` for **addNewLink**.
12. Save the changes to the service.
13. Click the **Coaches** tab, and select a template.
14. Click **Views** > **Advanced**, and drag a **Custom HTML** from the palette onto the canvas.



15. In the **Properties** pane, select the **HTML** tab, click **Variable**, and select the `AddNewLink` private variable.



16. Save the coach page.

    Note: When a user, who is not logged in, clicks the link from the coach page, that user is prompted to log in to OpenPages. After the user saves a new File attachment (or other object type) with the Add New wizard, the wizard returns to the initial Add New page. The user can add another object of the same type, and can, therefore, easily add multiple entries.

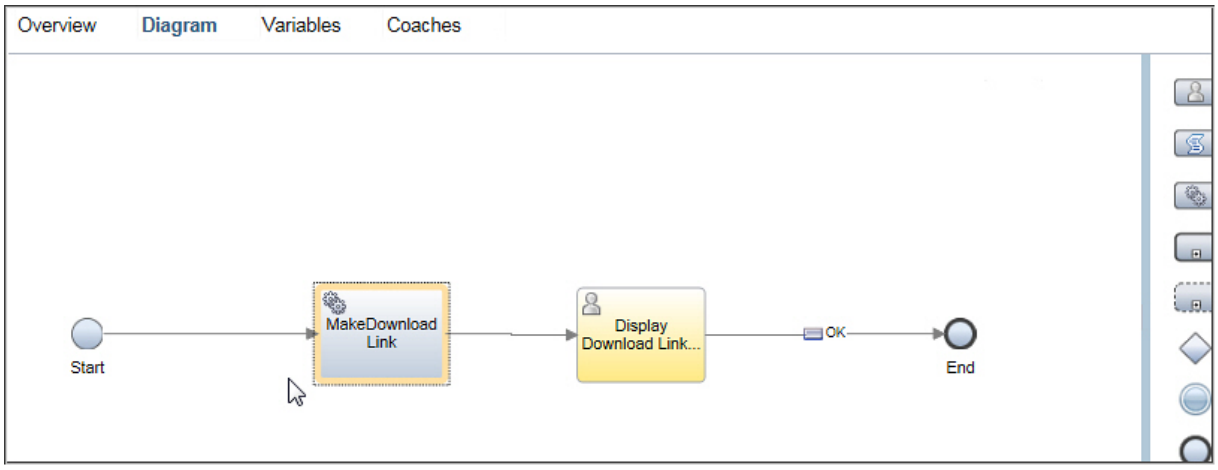
Downloading file attachments on coach pages

You can add a link to download a file attachment on a coach page. In this example, you create a new client-side human service and define a single link to download an existing file attachment from OpenPages. For testing, you enter the value of an existing file attachment's resource ID in OpenPages. In a real implementation, you specify the ID of the Document object to link to using an input variable on the client-side human service.

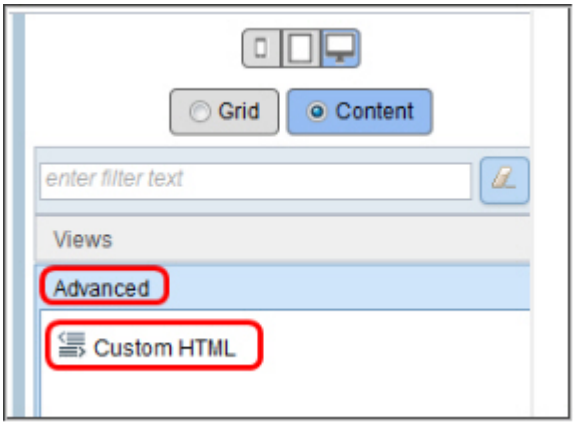
Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.

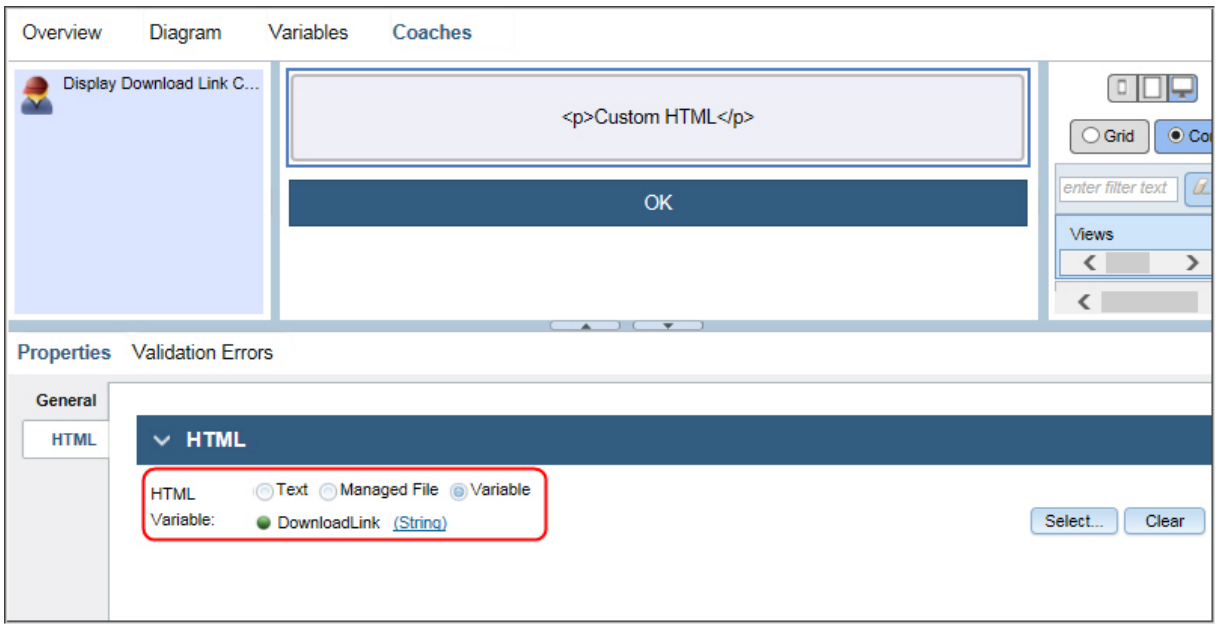
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface, for example, `Download Document`.
6. Click the **Variables** tab, and add a `DownloadLink` private variable of type String.
7. In **Input**, add a `DocId` input variable of type String. Select **Has default**. For testing purposes, enter the value of an existing File Attachment Document's resource ID in OpenPages.
8. Click the **Diagram** tab.
9. Drag a **Service** from the palette onto the canvas. Place it after the Start. Connect the arrows link from the Start node to the new Service node, and connect the Service node to the Coach. Give the nodes descriptive names.



10. Click the service node and select the **Implementation** pane.
11. In the **Behavior** section, select **Call a Service** and select the OPMakeDocumentLink integration service from the OpenPages Platform Toolkit.
12. Select the **Data Mapping** pane and specify the following parameters:
    a) In **fileId**, select the `DocId` input variable that is named `tw.local.DocId`.
    b) In **linkText**, enter `Download File`.
    c) In **Output Mapping**, enter `tw.local.DownloadLink` for **documentLink**.
13. Save the changes to the service
14. Click the **Coaches** tab and select a template.
15. Click **Views** > **Advanced**, and drag **Custom HTML** from the palette onto the canvas.



16. In the **Properties** pane, select the **HTML** tab, click **Variable**, and select the `DownloadLink` private variable.



Alternatively, you can use an Output Text view rather than a Custom HTML view in the coach page. Select the Output Text, under **Binding** select the `DownloadLink|DetailLink` private variable. Click the **Properties** pane, and select the **Configuration** tab. Change the setting for **Disable HTML encoding** from cleared to selected.

17. Save the coach page.

    Note: When a user, who is not logged in, clicks the link from the coach page, that user is prompted to log in to OpenPages. In this example, when a user clicks the link in the coach page, the user's browser saves the file. It does not navigate to the File attachment details page. To link to the details page, you must use the OPMakeDetailLink integration service. For more information, see "OPMakeDetailLink" on page 14.

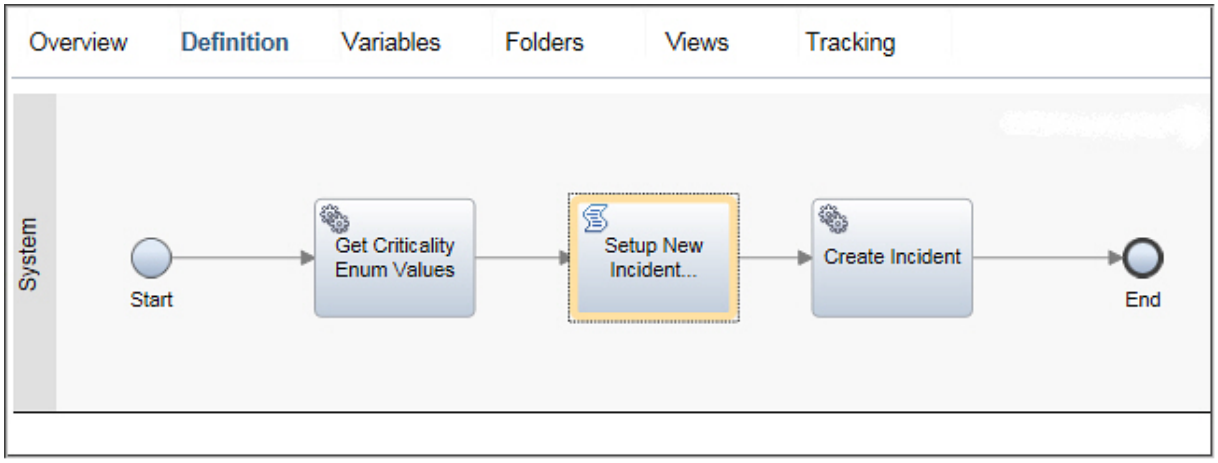
Creating OpenPages objects with an integration service

You can use an integration service to create OpenPages objects without user input. In this example, you define a business process that creates an Incident object and sets the Criticality field to an enumerated field value.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **Processes** > **Process**, and name it `Create Incident Process`.
6. Click the **Variables** tab, add two private variables:
   - `incident` of type OPIncident
   - `criticalityEnums` of type OPEnumValue. Select the **List** attribute.
7. Click the **Definition** tab.
8. Drag **Start** and **End** event nodes from the palette to the System lane. You can delete the Team swimlane. It is not used in this example.

9. Drag an **Activity** node from the palette onto the canvas.
   a) In the **Properties** pane, click the **General** tab and name it `Get Criticality Enum Values`.
   b) In the **Properties** pane, click the **Implementation** tab and change **Type** to System Task. Select the OPGetEnumeratedValues service from the OpenPages Platform Toolkit.
10. Drag an **Activity** node from the palette onto the canvas.
    a) In the **Properties** pane, click the **General** tab and name it `Setup New Incident Variable`.
    b) In the **Properties** pane, click the **Implementation** tab and change **Type** to Script.
11. Drag an **Activity** node from the palette onto the canvas.
    a) In the **Properties** pane, click the **General** tab and name it `Create Incident`.
    b) In the **Properties** pane, click the **Implementation** tab and change **Type** to System Task. Select the OPCreateObject service from the OpenPages Platform Toolkit.
12. Link the nodes from left to right in the order listed.



    You are now ready to configure the data mapping for the activities.
13. Click the `Get Criticality Enum Values` node and select the **Data Mapping** pane.

    In **Input Mapping** define the following fields:

    - In **systemTask**, enter `true`.
    - In **objectTypeName**, enter "`Incident`".
    - In **fieldName**, enter "`OPSS-Inc:Criticality`".

    In **Output Mapping** define the following field:

    - In **outObject**, select the `tw.local.criticalityEnums` variable.
14. Click the `Create Incident` node and select the **Data Mapping** pane. In **Input Mapping** define the following fields:

    In **Input Mapping** define the following fields:

    - In **systemTask**, enter `true`.
    - In **inObject**, select the `tw.local.incident` variable.

    In **Output Mapping** define the following field:

    - In **outObject**, select the `tw.local.incident` variable.

    You are now ready to enter the script.
15. Click the `Setup New Incident Variable` node. In the **Properties** pane, click the **Script** tab. Enter the following example code:

```
tw.local.incident = new tw.object.OPIncident();
tw.local.incident.base = new tw.object.OPObject();
tw.local.incident.base.name = "BPM Test Object creation " + new java.util.Date();
```

```
tw.local.incident.OPLC_Owners__PrimaryOwner = "OpenPagesAdministrator";
tw.local.incident.OPSS_Inc__Criticality = tw.local.criticalityEnums[0];
```

The script initializes the `tw.local.incident` variable with an empty OPIncident object. It then sets various fields as required by the use case. Using the properties of the `incident` variable and its nested properties, such as `tw.local.incident.base`, you can set any field on the Incident object. In this example, the following fields are set:

- The name is set to a dynamically created value.
- The PrimaryOwner is set to OpenPagesAdministrator.
- The Criticality enumerated field is set to one of the values from the `criticalityEnum` variable that was populated in the first activity.

If you know the name or ID of an enumerated value for a field, you can set it directly without using the OPGetEnumeratedValues service. For example, enter the following code to set the Critically field to High:

```
tw.local.incident.OPSS_Inc__Criticality = new tw.object.OPEnumValue();
tw.local.incident.OPSS_Inc__Criticality.name = "High";
```

16. Save the changes to the process.

Updating OpenPages objects with a client-side human service

You can use an integration service to update OpenPages objects in a client-side human service. Users access a coach page, update fields on a specific object, and the changes are then reflected in OpenPages. You use the OPUpdateObject OpenPages integration service. However, you cannot call OPUpdateObject directly from the client-side human service because it uses an input parameter of ANY business object type. To get around this, you can create an integration service that calls an object type that users can update. In this example, you define an integration service that can update an object type. Then, you call that service in a client-side human service that uses OPUpdateObject.

Procedure

1. Open the Process Designer desktop editor.
2. Open the Process Designer.

   First, create an integration service that calls an object type, which is OPIncident in this example.
3. Add dependencies to the OpenPages toolkits.
   a) Click **Toolkits** and select OpenPages Platform.
   b) Click **Toolkits** and select OpenPages Solutions.
4. Click **New** beside **Implementation** > **Integration Service**, and name it `OPUpdateIncident`.
5. Click the **Variables** tab, and add two variables:
   - in **inObject**, select `OPIncident` as input
   - in **outObject**, select `OPIncident` as output

   Define the `inObject` type as OPIncident rather than ANY so that you can call the variable from a client-human service.
6. Click the **Diagram** tab, and drag a **Nested Service** node from the palette onto the canvas.
   a) Name it `Update Incident`.
   b) In the **Properties** pane, click the **Implementation** tab. Select OPUpdateObject OpenPages integration service.
   c) Click the **Data Mapping** tab. In **Input Mapping**, enter `false` in **systemTask** and `tw.local.inObject` in **inObject**. In **Output Mapping**, enter `tw.local.outObject` in **outObject(ANY)**.
7. Connect the Start and End nodes before and after the `OPUpdateIncident` node.

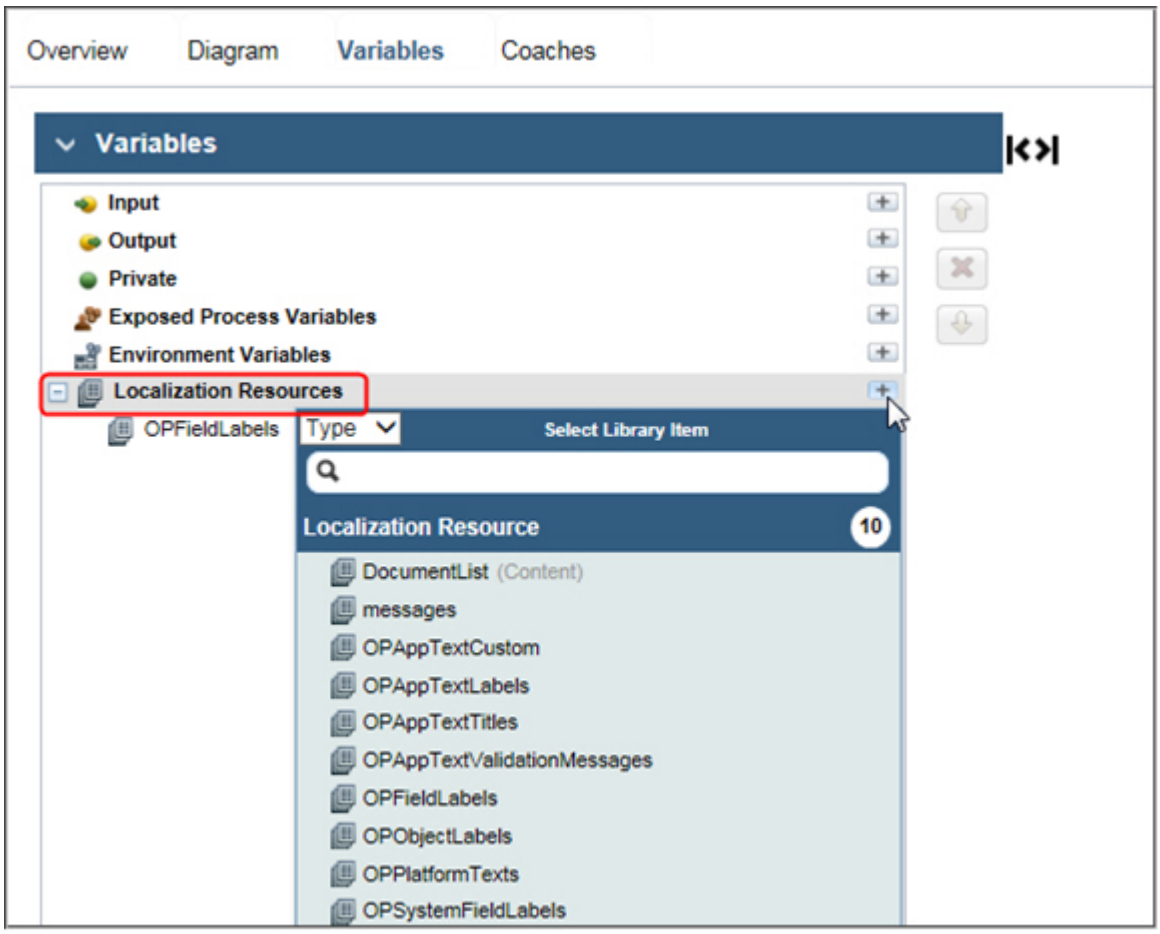
8. Save the `OPUpdateIncident` integration service. You are now ready to define the client-side human service.
9. Click the plus icon beside **User Interface** > **Client-side Human Service**, and give it a name, such as, `View and Edit Incident`.
10. Click the **Variables** tab, and add two variables:
    - `objectId` as an input variable of String type
    - `Incident` as a private variable of OPIncident type
11. Click **Diagram**.
12. Drag a **Service** node from the palette onto the canvas. This node retrieves an Incident object.
    a) In the **Properties** pane, click the **General** tab and name it `Retrieve Incident`.
    b) In the **Properties** pane, click the **Implementation** tab and select the OPGetObject service from the OpenPages Platform Toolkit.
    c) Click the **Data Mapping** tab. In **Input Mapping**, enter `false` in **systemTask** and `tw.local.objectId` in **objectId**. In **Output Mapping**, enter `tw.local.incident` in **outObject(ANY)**.
13. Edit the default coach node. Name it `View and Edit Incident`. Click the **Coach** tab and drag controls to the coach page so that users can view and edit an Incident object.
14. Drag a **Service** node from the palette onto the canvas. This node updates the Incident object with the edits the user entered.
    a) In the **Properties** pane, click the **General** tab and name it `Update Incident`.
    b) In the **Properties** pane, click the **Implementation** tab and select the OPUpdateIncident service that you created previously.
15. Connect the nodes from left to right in the order of the steps.
16. Save the client-side human service.

Using localization resources

You can display field labels and text on a coach page in localized languages. In this example, you create a new client-side human service and define a variable that allows fields to be displayed in localized languages.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
    a) Click **+** next to **Toolkits** and select OpenPages Platform.
    b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Click the **Variables** tab.
    a) Expand **Localization Resources**.
    b) Define variables for the elements that you want to be able to display in localized languages, for example, select `OPFieldLabels` or `OPPlatformTexts`. After you have defined the variables, you can begin designing the coach page.



7. Click the **Coaches** tab and select a template.
   a) Drag a **View**, such as, the **Single Select** view, from the palette to the canvas.
   b) On the **Properties** pane, click the icon next to **Label**.
   c) Click **Select** and choose a field.

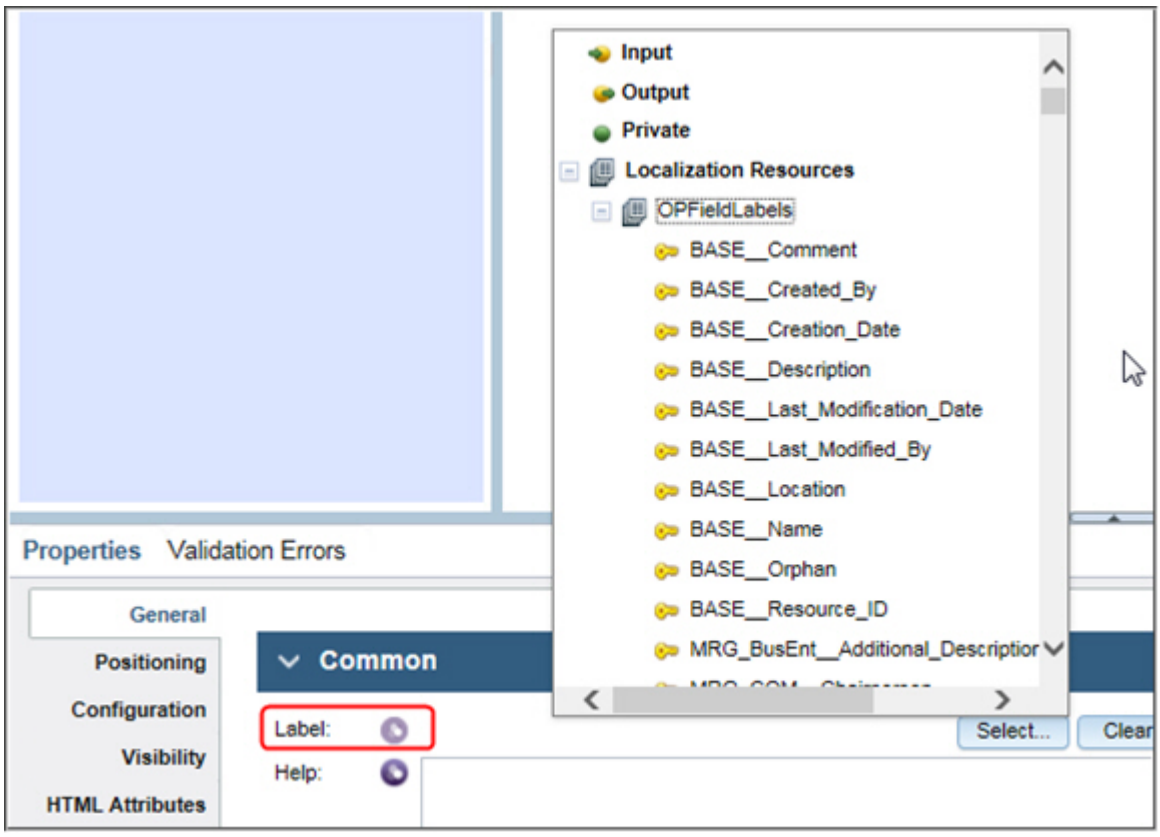


Figure 4. Defining a coach page

8. Bind the control to a variable and continue defining the view.
9. Save the coach page.

Defining basic hierarchical processes

In a hierarchical process, a parent object in one process can launch child processes for objects that are associated with the parent object.

In this example, Issue objects are retrieved from OpenPages. They can have one or more action items. After the action items are remediated, the issue can be closed.

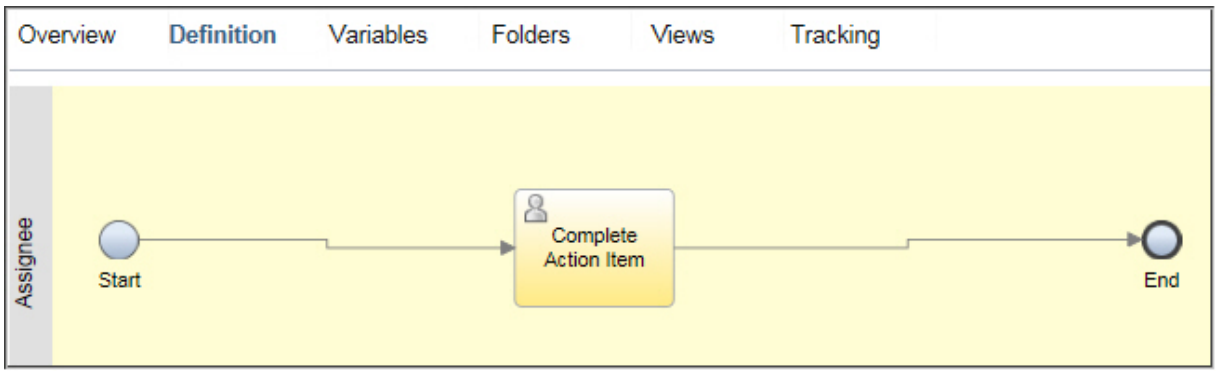
In this example, a small number (10 to 20) of child objects are launched. The parent process uses the OPHierarchyAssigneeQuery integration service and a multi-instance loop type. If a larger number of child objects are launched, you can build the processes to optimize server performance. For information, see “Defining advanced hierarchical processes ” on page 41.

Defining a child process

Define a child process that is named Complete Action Item. This process contains a user task so that it can be launched in parallel.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.



4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Create a process and name it `Complete Action Item`.
6. Click the **Variables** tab. Add an input variable and name it `actionitemId`.
7. Create a new swimlane or rename an existing one to Assignee.
8. Add a single Activity node to the Assignee swimlane and connect it to the Start and End nodes. Click the **Properties** pane, and define the following:
   a) Click **Implementation** and define a User Task to use the Default Human Service. You can later replace this with a different coach UI.
   b) Click **Data Mapping** and set up inputs for the coach UI definition. Pass the `tw.local.actionItemId` variable, and map it to an Input variable for the ID used by the coach to get the OpenPages object.
   c) Click **Assignments** and specify that the user who is named on the action item will be the assignee. Under **Assignments**, define the following fields:
      - In **Assign to**, select Team.
      - In **Team**, select All Users.
      - In **Team Filter Service**, select Team Filter by Object Field.

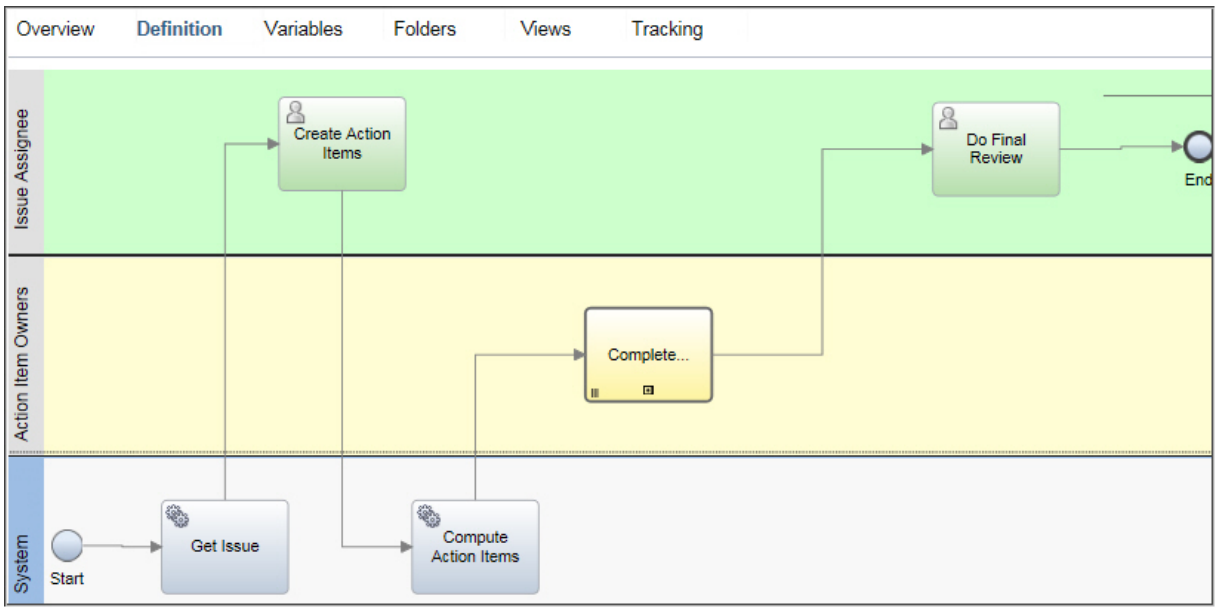
      Under **Input Mapping** define the following fields:
      - In **systemTask**, type `true`.
      - In **objectId**, type `tw.local.actionId`.
      - In **field**, type `OPSS-AI:Assignee`.
      - In **expandGroups**, type `false`.
9. Save the process.
10. Define a coach page. Ideally, Complete Action Item will be a custom human service (coach) that displays information from the Action Item, which you will retrieve using the OPGetObject integration service from the OpenPages Toolkit. This allows the Action Item assignee to get a Task that prompts them to perform work on the Action Item that is assigned to them for the Issue.

Defining a parent process

Define a parent process that is named Issue Remediation. It determines what action items exist. It uses the OPHierarchyAssigneeQuery integration service. You link to the Complete Action Item process and start a process for each child Action Item under the Issue. It uses looping to complete the processes.

Procedure

1. Create a process and name it `Issue Remediation`.



2. Click the **Variables** tab.
   a) Create a variable whose type is OPSOXIssue. Name it `issue`.
   b) Create a list variable whose type is OPSOXTask. Name it `actions`.
3. Click the **Definition** tab and create three swimlanes:
   - Issue Assignee
   - Action Item Owners
   - System
4. In the Issue Assignee swimlane, create a User Task activity node:
   a) Name it `Create Action Items`.
   b) Click **Assignments**.

      In **Assignments**, define the following fields:

      - In **Assign to**, select Team.
      - In **Team**, select All Users.
      - In **Team Filter Service**, select Team Filter by Object Field.

      In **Input Mapping**, define the following fields:

      - In **systemTask**, type `true`.
      - In **objectId**, type `tw.local.issue.base.id`.
      - In **field**, type `OPSS-Iss:Assignee`.
      - In **expandGroups**, type `false`.

5. In the Issue Assignee swimlane, create another User Task activity node:
   a) Name it `Do Final Review`.
   b) Click **Assignments**.

      In **Assignments**, define the following fields:

      - In **Assign to**, select Team.
      - In **Team**, select All Users.
      - In **Team Filter Service**, select Team Filter by Object Field.

      In **Input Mapping**, define the following fields:

- In **systemTask**, type `true`.
- In **objectId**, type `tw.local.issue.base.id`.
- In **field**, type `OPSS-Iss:Assignee`.
- In **expandGroups**, type `false`.

6. In the System swimlane, create a System Task activity node:
   a) Name it `Get Issue`.
   b) Click **Implementation**.
   c) In **Implementation**, select the OPGetObject integration service. Use this for testing. In a real implementation, you could launch a Process Coach UI that allows a user to select an Issue.
7. In the System swimlane, create another System Task activity node:
   a) Name it `Compute Action Items`.
   b) Click **Implementation**. In **Implementation**, select the OPHierarchicalAssigneeQuery integration service. This will query the children of the Action Items under Issue.
   c) Click **Data Mapping**. In **Input Mapping**, define the following fields:
      - In **systemTask**, type `true`.
      - In **parentType**, type `SOXIssue`.
      - In **parentID**, type `tw.local.issue.base.id`.
      - In **childType**, type `SOXTask`.
      - In **assigneeField**, type `OPSS-AI:Assignee`.
      - In **isPrimary**, type `true`.
      - In **isCaseInsensitive**, type `true`.
      - Leave **filters** blank.
      - Leave **pageSize** blank.
      - In **isDirect**, type `true`.
8. In the Action Item Owners swimlane, create a Linked Process node.
   a) Name it `Complete Action Item`.
   b) Click **General**. In **Behavior**, select Multi-instance loop in **Loop type**.
   c) In **Multi-instance loop**, define the following fields:
      - In **Start quantity**, type `tw.local.actions.listLength`.
      - In **Ordering**, select Run in parallel.

        If you select Run in parallel, each child process is launched immediately and simultaneously. This can lead to scalability issues since IBM BPM does not limit the child execution. If the child process is performing only System tasks, for example, creating new objects in OpenPages, you might want to instead select Ordering with Run Sequential to minimize performance load. If the child process has tasks that must be run in parallel and there is the potential for a large number of child processes (more than 10-20), see “Defining advanced hierarchical processes ” on page 41 for an alternative design.
      - In **Flow condition**, select Wait for all to finish (All).
   d) Click **Implementation**. In **Implementation**, select Linked Process in **Type**.
   e) Click **Data Mapping**. In **Input Mapping**, define the following field:
      - In **actionItemId**, type `tw.local.actions[tw.system.step.counter].base.id`.
9. Connect the lines.
10. Save the process.

Defining advanced hierarchical processes

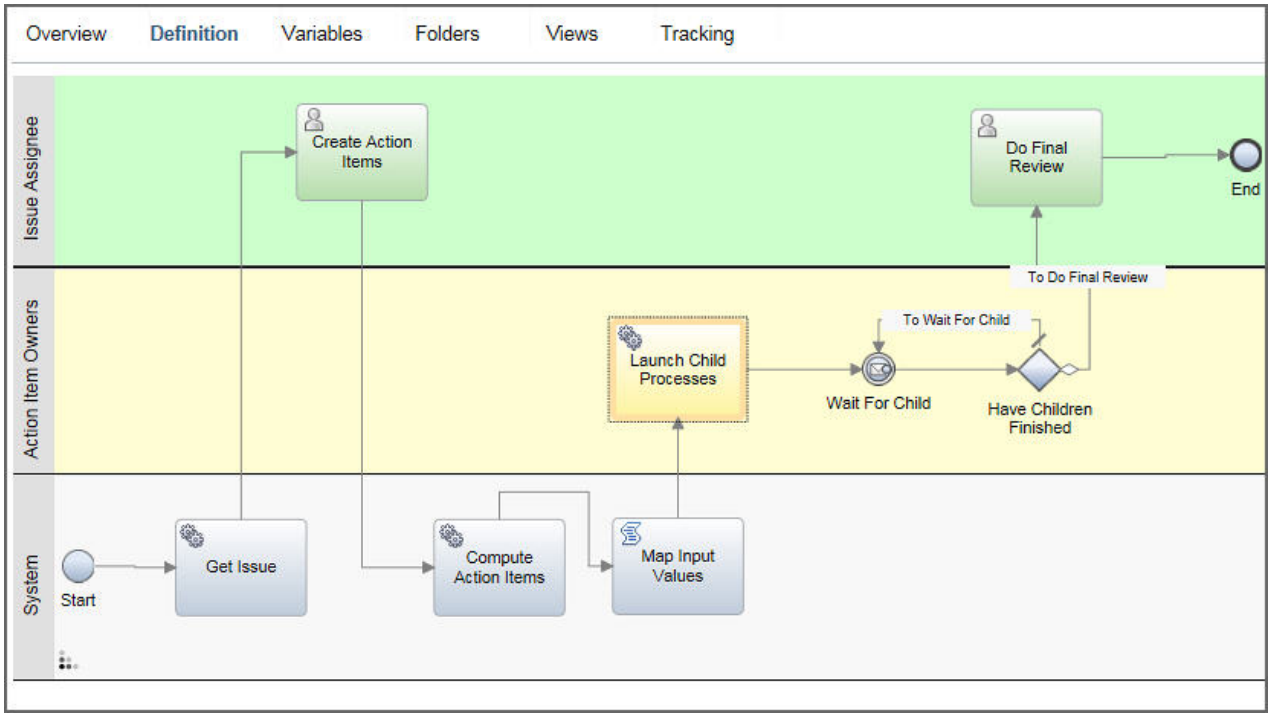
You can use the OPLaunchChildProcesses integration service in a hierarchical process where a parent object in one process can launch child processes for objects that are associated with the parent object. This example illustrates how to handle a large number, even hundreds, of child action items without compromising server performance.

Before you begin

Complete the tasks in "Defining basic hierarchical processes" on page 37. You build on that knowledge and data in this example.

About this task

The basic hierarchical processes example cannot throttle process execution for linked processes and is best suited for small data volumes. The advanced hierarchical processes example is designed for large data volumes. You use the BPM API to launch a child process for every value in the ChildInputs list variable, a private variable you create. The APIs can control how quickly the processes are launched, including a delay between processes to throttle execution if needed. Lastly, you can optionally extend the parent process to wait for child processes to complete before moving to the next activity.



Defining a parent process to use OPLaunchChildProcesses

Modify the parent process in the basic hierarchical process to use the OPLaunchChildProcesses integration service.

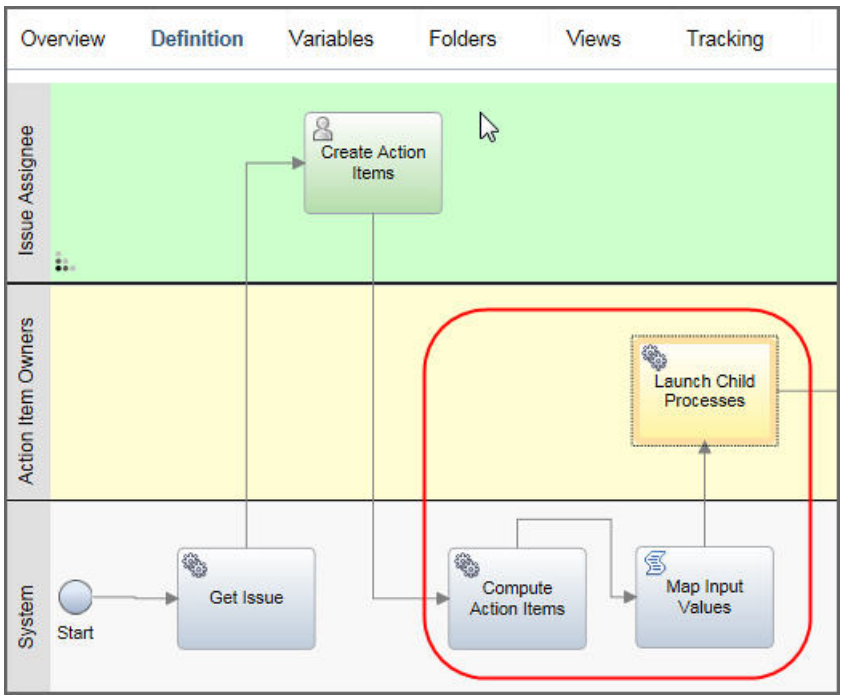
Procedure

1. Duplicate the parent process, `Issue Remediation`, that you created for the basic hierarchical processes example. Rename it to `Issue Remediation Adv`.
2. Duplicate the child process, `Complete Action Items`, that you created for the basic hierarchical processes example. Rename it to `Complete Action Items - Adv`.

3. Click the **Variables** tab.
   a) Create a private variable whose type is Map (System Data). Name it `childInputs`.
   b) Select **List** to make `childInputs` a List of Maps.
   c) Select **Has default** for Default Value.
4. Click the **Definition** tab. The process has three swimlanes:
   - Issue Assignee
   - Action Item Owners
   - System
5. In the Action Item Owners swimlane, select the `Complete Action Item` node.
   a) Rename the node to `Launch Child Processes`.
   b) Change it from a Linked process to a System Task.
   c) Click **General**. In **Behavior**, select None in **Loop type**.
   d) Click **Implementation**. In **Type**, select System Task.
   e) In **Implementation**, select the OPLaunchChildProcesses integration service.
   f) Click **Data Mapping**. In **Input Mapping**, define the following fields:
      - In **ProcessName**, type the name of the child process, `Complete Action Item - Adv`.
      - In **InputValues**, type `tw.local.childInputs .`
      - In **Delay**, type `1000`.
      - In **AddDependency**, type `true`.
6. In the System swimlane, create a Script activity node after Compute Action Items:
   a) Name it `Map Input Values`.
   b) Click **Script**.
   c) Enter a script like the following example. If you use the variable names for the parent and child processes as given in this example, you do not have to modify the script.

```
//initialize childInputs List of Maps
tw.local.childInputs = new tw.object.listOf.toolkit.TWSYS.Map();
//loop for every action item
for (var i=0; i<tw.local.actions.listLength; i++) {
    //get the action item
    var action = tw.local.actions[i];
    //initialize a map for this action item
    var map = new tw.object.toolkit.TWSYS.Map();
    //map values from the action to the input variable names of the child process
    map.put("actionItemId", action.base.id);
    //insert map to the childInputs list
        tw.local.childInputs.insertIntoList(i, map);
```

   d) Reconnect the arrows from Compute Action Items to now go to Map Input Values. Connect an outgoing arrow from Map Input Values to Launch Child Processes.



7. Save the process.

Results

The OPLaunchChildProcesses integration service launches the child process but does not wait for the child to be completed before it moves to the next activity. After every child process in the list has been launched, the OPLaunchChildProcesses integration service is completed and the parent process resumes and moves to the next activity node. This means that in the Issue Remediation example after Launch Child Processes, the Issue Assignee gets the task for Do Final Review, regardless of whether the child processes have completed or not. Also, if the OPLaunchChildProcesses AddDependency input is true, it makes a relationship between the parent process and child process. Without this relationship, the two processes have no connection after the child process has been launched. With the dependency relationship, the parent process cannot finish until all child processes are finished. Additionally, if you terminate the parent process, the child processes are also terminated. Depending on your use cases, this may be sufficient for your needs.

For information about relationships in IBM Business Process Manager, see https://www.ibm.com/support/knowledgecenter/SSFTN5_8.5.7/com.ibm.wbpm.wle.editor.doc/topics/tcrtrelation.html.

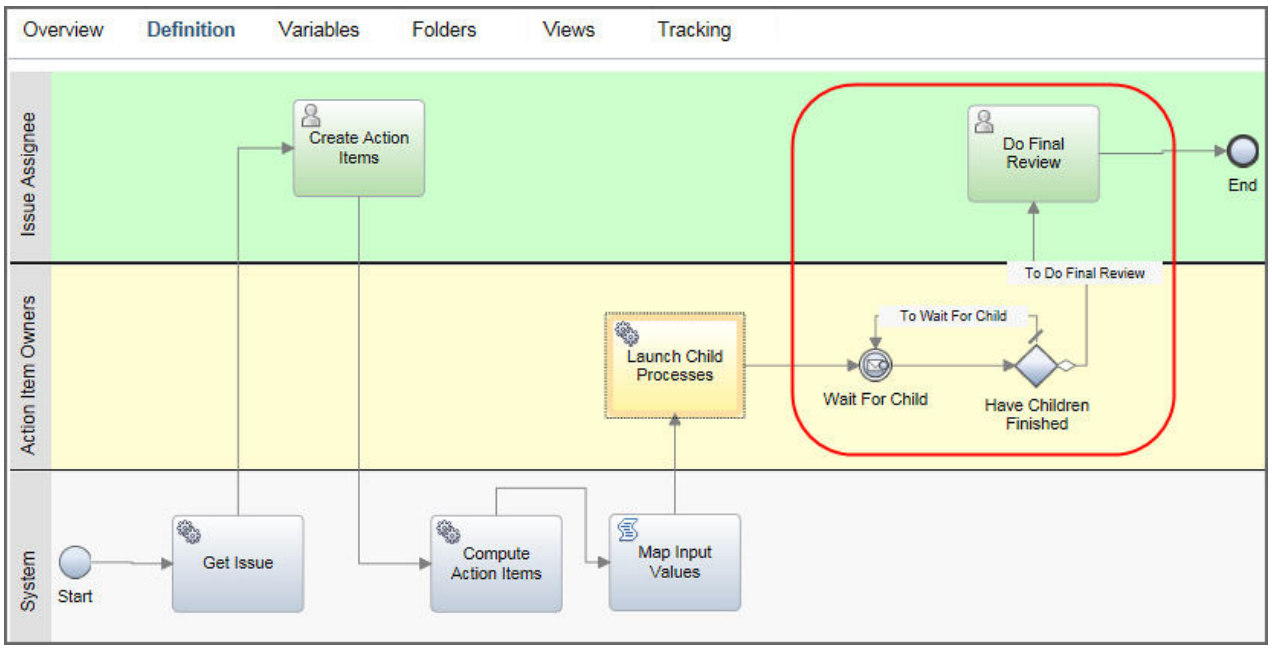
What to do next

You are finished with the advanced hierarchical example. You can optionally extend the parent process to wait for child processes to complete before moving to the next activity.

Extending a parent process to wait for child processes to complete

You can optionally extend the parent process to wait for child processes to complete before moving to the next activity. You define an undercover agent that enables child processes to a send message to the parent process when it is finished.

The following figure shows the extension that is explained in this part of the example.



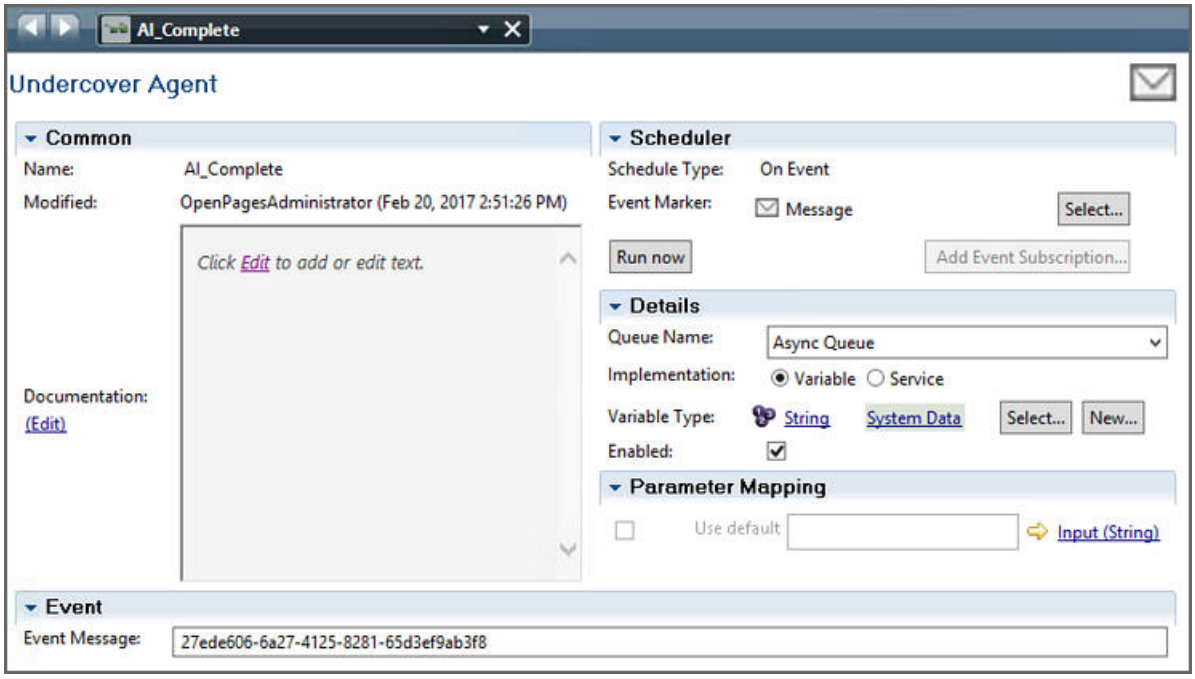
Defining the undercover agent

Define the undercover agent. An undercover agent defines a queue where processes can send and listen for messages. An undercover agent is attached to a message event or a content event in a process or business process definition and calls a service to handle the event.

For more information about undercover agents, see https://www.ibm.com/support/knowledgecenter/SSFTN5_8.5.7/com.ibm.wbpm.wle.editor.doc/topics/using_undercover_agents.html.

Procedure

1. Open the Process Designer desktop editor.
2. Open the process application with your processes.
3. In **Implementation**, click **Undercover Agent**.
   a) In **Name**, type `AI_Complete`.
   b) In **Schedule Type**, select On Event.
   c) In **Details**, select String in **Variable Type**.



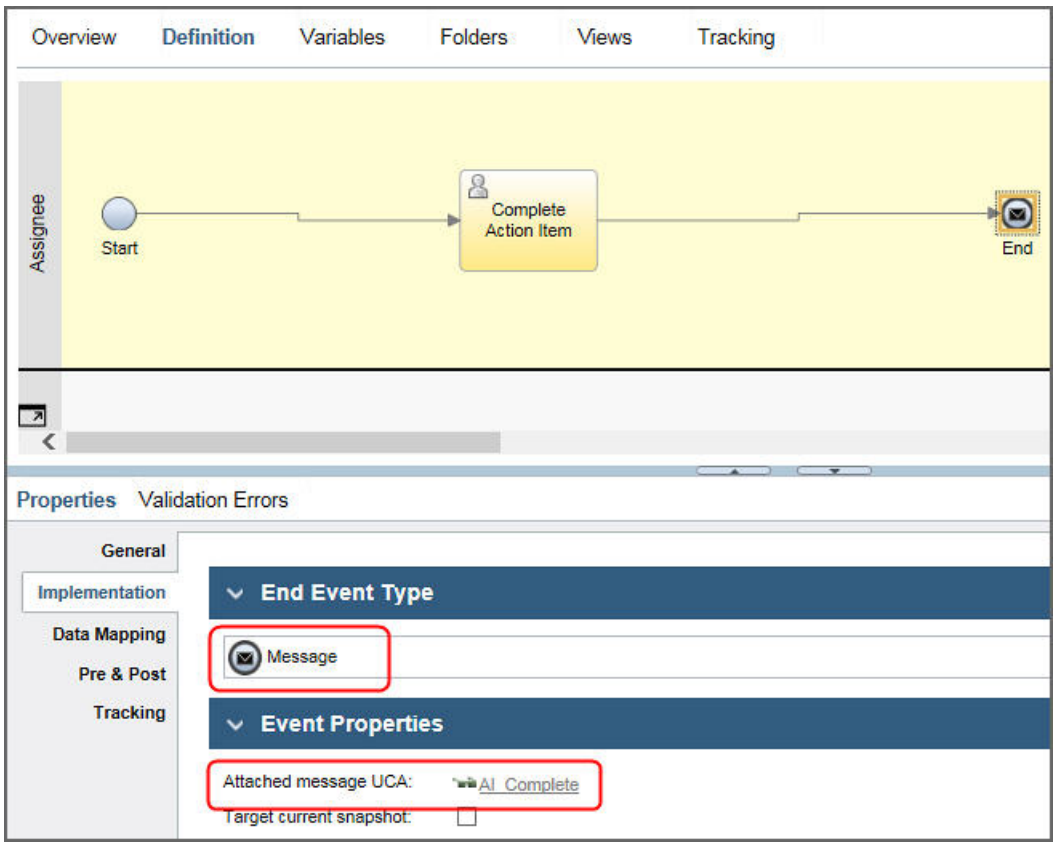
4. Save the undercover agent.

Changing the child process to send the undercover agent message

Update the child process to have a variable reference to the parent process ID and to create a message that is sent to the undercover agent.

Procedure

1. Open the Process Designer web editor.
2. Open the child process that is named `Complete Action Item - Adv`.
3. Click the **Variables** tab and add a private variable of type String (System Data). Name it `ParentPID`.
4. Click the **Definition** tab.
5. Click the End node.
6. In the **Properties** pane, click the **Implementation** tab.
   a) In **End event type**, select Message.
   b) In **Event Properties**, select the `AI Complete` undercover agent in **Attached message UCA**.
7. Click **Data Mapping**. In **Input Mapping**, define the following field:
   - In **InputValues**, type `tw.local.ParentPID`.



8. Save the process.

Extending a parent process to wait for undercover agent messages

Extend the existing parent process to wait for the child processes to be completed before it continues. This uses a counter to track when the child processes complete. An Intermediate Message Event (IME) is used to receive the undercover agent messages that are sent when each child process ends.

Using an undercover agent allows for asynchronous communication between different process instances. In this case, you can control the parent process by waiting for enough messages to be sent by the child processes as they end.

Procedure

1. Open the Process Designer web editor.
2. Open the parent process that is named `Issue Remediation Adv`.
3. Click the **Variables** tab.
   a) Create a private variable whose type is Integer (System Data). Name it `ChildrenFinished`. Select **Has Default** and set it to 0.
   b) Create a private variable whose type is String (System Data). Name it `currentProcessId`.
4. Click the **Definition** tab. This process has three swimlanes:
   - Issue Assignee
   - Action Item Owners
   - System
5. In the Action Item Owners swimlane, add two nodes after the `Launch Child Processes` System task.
6. First, add an Intermediate event.
   a) Name it `Wait For Child`.

b) Click **Implementation**.

c) In **Implementation**, select `AI_Complete` for **Attached message UCA**.

d) Select **Consume Message**.

e) Select **Durable subscription**.

f) Click **Data Mapping**, select Output (String) in **Correlation Variable**. In the mapping of Output =, type `tw.local.currentProcessId`.

g) Click **Pre&Post** and add the following assignment in **Post Assignments**:

```
tw.local.ChildrenFinished =  tw.local.ChildrenFinished + 1
```

7. Next, add an exclusive gateway.

a) Name it `Have Children Finished`.

b) Create an outgoing arrow from the exclusive gateway node back to the Intermediate Event.

c) Create a second arrow from the gateway to the next step, Do Final Review task.

d) Create an outgoing arrow from the Intermediate Event to the Gateway.

e) In **Implementation**, set the **Default flow** as `To Wait For Child`.

f) Set the To Do Final Review condition to the following expression:

```
1.    tw.local.ChildrenFinished ==
tw.local.actions.ListLength
```

8. Open the **Map Input Values** script node and click **Script**.

a) Insert the following line to the beginning of the script.

```
tw.local.currentProcessId = tw.system.currentProcessInstance.id;
```

b) Locate the line `map.put("actionItemId", action.base.id); .` Insert the following lines after it.

```
//map the process instance id to the child process
    map.put("ParentPID", tw.local.currentProcessId);
```

This passes the process instance id to the new input variable in the child process.

The final script would be:

```
tw.local.currentProcessId = tw.system.currentProcessInstance.id;
```

```
//initialize childInputs List of Maps
tw.local.childInputs = new tw.object.listOf.toolkit.TWSYS.Map();

//loop for every action item
for (var i=0; i<tw.local.actions.listLength; i++) {
    //get the action item
    var action = tw.local.actions[i];
    //initialize a map for this action item
    var map = new tw.object.toolkit.TWSYS.Map();
    //map values from the action to the input variable names of the child process
    map.put("actionItemId", action.base.id);
    //map the process instance id to the child process
    map.put("ParentPID", tw.system.currentProcessInstance.id);
    //insert map to the childInputs list
        tw.local.childInputs.insertIntoList(i, map);
}
```

9. Save the process.

Results

Now the Do Final Review task will not be assigned to the Issue Assignee until all the Launch Child Processes are finished.

Retrieving a list of child objects

You can retrieve a list of child objects and display them on a coach page. The user can view the list, select one or more children from the list, and update the parent object or any selected child objects. In this example, you find Risk objects under a Risk Assessment object and show them in a coach page table. You create a new client-side human service, define a variable for the list of child objects, and retrieve the child objects for the parent object. The OPHierarchicalQuery service will return more information than OPGetChildAssociations.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name for your user interface.
6. Click the **Variables** tab.
   a) Click **Private** and provide a name, such as, `ReturnedRisk`.
   b) Select **Is list**.
   c) Click **Select** and choose a business object, such as, OPSOXRisk.
7. Select the **Diagram** tab.
   a) Drag a **Service** from the palette onto the canvas. Name it `GetRisks`.
   b) On the **Implementation** pane, click **Call a service**.
   c) Click **Select** and choose OPHierarchicalQuery.
8. Click the **Data Mapping** pane, and provide entries for the following fields:
   a) In **systemTask**, specify whether the task accesses OpenPages as a system account or the current account.
   b) In **parentType**, enter the ID or full path of the parent object, such as, `"RiskAssessment"`. This can be a local variable.
   c) In **parentId**, enter the ID or full path of the parent object, such as, `tw.local.RA_ID`. This can be a local variable.
   d) In **isPrimary**, enter `false`.
   e) In **isCaseInsensitive**, enter `true`.
   f) In **fields**, enter `"[SOXRisk].*"`.
   g) In **isDirect**, enter `true`.
   h) In **Output Mapping**, select returnObjects (List of ANY) and type `tw.local.ReturnedRisk`.
9. Click the **Coaches** tab and select a template.
   a) Drag a table from the palette onto the canvas.
   b) In the table view, bind the `ReturnedRisk` variable to the table.
   c) Create the column header for the first column.
      - Drag an Output Text to the table.
      - Name the column header `Risk Name`.
      - Bind the column to the `ReturnedRisks.currentItem.risk.base.name` variable.
   d) Create the column header for the second column.

- Drag an Output Text to the table.
- Name the column header `Risk Description`.
- Bind the column to the `ReturnedRisks.currentItem.risk.base.description` variable.

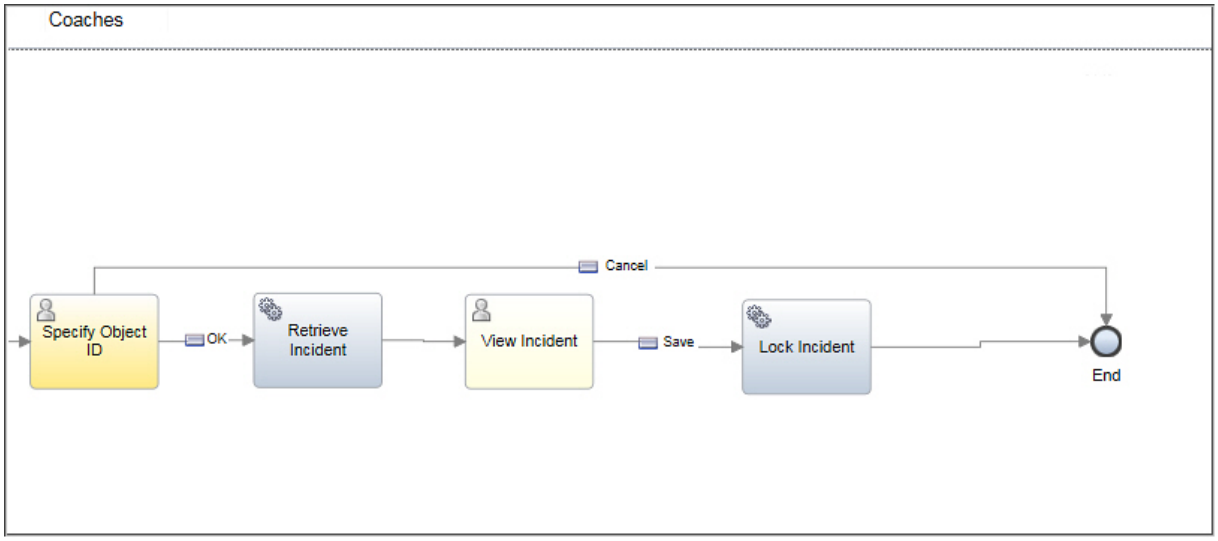
10. Save the coach service.

Locking and unlocking objects

You can lock and unlock objects in a business process.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name, such as, `Lock Incident`.
6. Click the **Diagram** tab and build the process.
7. At the point when you want to lock an object, drag a **Service** from the palette onto the canvas. Connect the arrows link from the previous node to the new service.



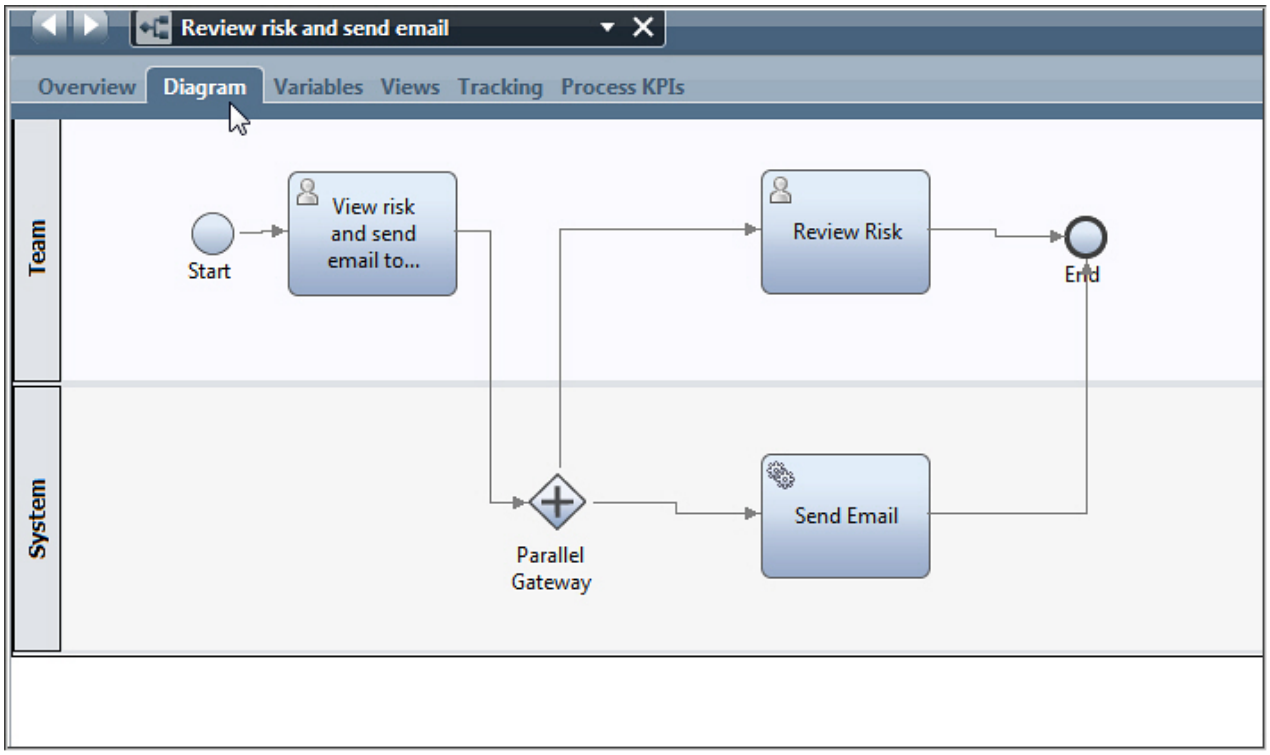
8. Click the service node, and select the **Implementation** pane.
9. In **Behavior** section, select **Call a Service** and then select the OPLockObject integration service from the OpenPages Platform Toolkit.
10. Select the **Data Mapping** pane and enter a specific object ID or `tw.local.objectId` in **objectId**,
11. Save the changes to the service.

    Note: To unlock an object, follow the same steps but use the OPUnlockObject integration service.

Sending email notifications

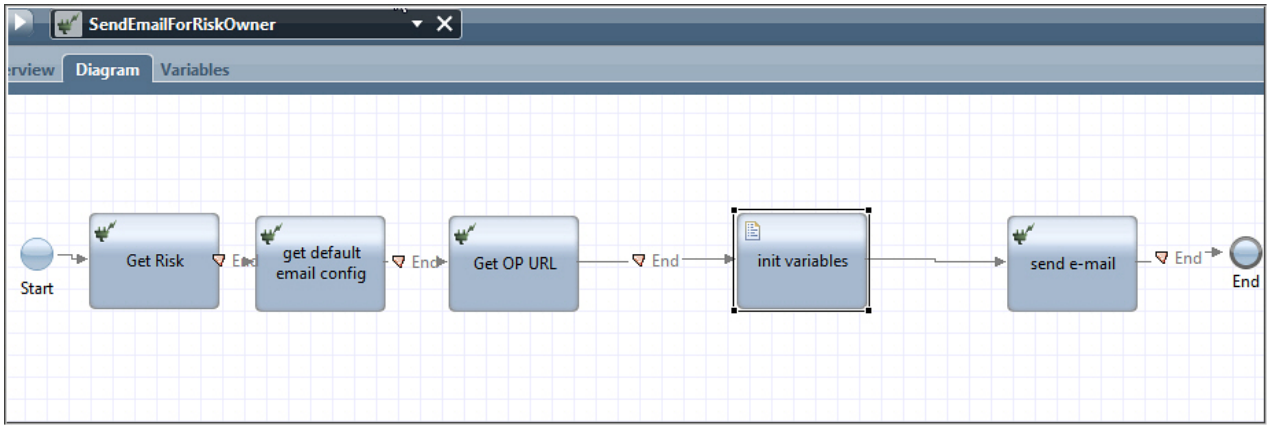
You can define business processes that will send email notifications at designated points in a business process. In this example, Risk objects are retrieved from OpenPages and an email is sent to the object's primary owner. The email contains a link to a review task in the body text.

The following figure shows the business process that is explained in this example.



A Parallel Gateway is used in this process because the email contains a URL link to a process task that does not yet exist. The Parallel Gateway starts the Send Email system task and simultaneously assigns a UI task (the task that the URL contains a link to). You must add a script to the system task that looks up the other task's ID.

The Send Email node calls a nested service, which is shown in the following figure.



The nested service completes the following functions:

- retrieves the object by using OPGetObject
- retrieves system default properties by using Email Get System Default Properties
- retrieves the URL to OpenPages by using OPGetBaseURL
- constructs the email subject and body text by using a script
- sends the email by using Email Send IBM BPM Email

Procedure

1. Open the Process Designer desktop editor.
2. Open the Process Designer.
3. Add dependencies to the OpenPages toolkits.
   a) Click **Toolkits** and select OpenPages Platform.
   b) Click **Toolkits** and select OpenPages Solutions.
4. Click **New** beside **Processes** > **Business Process Definition**, and name it `Review risk and send email`.
5. Create the first nodes in the business process (these nodes are not part of this example).
6. At the point when you want an email to be sent, drag a **Gateway** from the palette to the System swimlane.
   a) In the **Properties** pane, click the **General** tab.
   b) Select Parallel Gateway in **Gateway Type**.
   c) Leave **Outgoing Flow Percentages** on the **Simulation** tab empty (to be completed later automatically).
7. Drag an **Activity** from the palette to the Team swimlane. Name it `Review Risk`.
   a) In the **Properties** pane, click the **Implementation** tab. Select **User Task**. Click **Select** and choose the client-side human service representing the coach page used by the email recipient when clicking the link in the email.
   b) Click **Data Mapping**, and add the input and output mapping that is required for the coach page.
8. Drag an **Activity** from the palette to the System swimlane. Name it `Send Email`.
9. In the **Properties** pane, click the **Implementation** tab. Select **System Task**. Click **New** and create a new integration service. Name it `SendEmailForRiskOwner`.
10. Click **Variables** and define the following variables required by the integration service.
    - `smtpHost` (String)
    - `defaultFromAddress` (String)
    - `emailTo` (String)
    - `emailSubject` (String)
    - `emailBody` (String)
    - `riskObject` (OpSOXRisk)
    - `opBaseUrl` (String)
11. Drag a **Nested Service** from the palette onto the canvas. Define this node to retrieve the object. The object's owner will be the email recipient.
    a) In the **Properties** pane, click the **Implementation** tab, and select the OPGetObject OpenPages integration service.
    b) Click the **Data Mapping** tab. In **Input Mapping**, enter `tw.local.objectId` in **ObjectId**. In **Output Mapping**, enter `tw.local.riskObject` in **outObject(ANY)**.
12. Drag a **Nested Service** from the palette. This node retrieves system default properties that are used in the email, for example, an SMTP host and a default from address.
    a) In the **Properties** pane, click the **Implementation** tab, and select the Email Get System Default Properties system integration service. This is an integration service that is delivered by IBM Business Process Manager.
    b) Click the **Data Mapping** tab. In **Output Mapping** select `smtpHost` and `defaultFromAddress`.
13. Drag a **Nested Service** from the palette onto the canvas. This node retrieves the base URL.
    a) In the **Properties** pane, click the **Implementation** tab, and select the OPGetBaseURL OpenPages integration service.

b) Click the **Data Mapping** tab. In **Output Mapping**, enter `tw.local.opBaseUrl` in **baseURL (String)**.

14. Drag a **Server Script** from the palette onto the canvas. This node constructs the email subject and body text.
    a) In the **Implementation** pane, enter the following script:

```
// set log level
log.infoEnabled = true;

log.info("smtp host: " + tw.local.smtpHost);
log.info("defaultFromAddress: " + tw.local.defaultFromAddress);

// find e-mail address of the risk object owner, and set it to a variable
var user = tw.local.riskObject.OPSS_Rsk__Owner;
var user1 = tw.system.org.findUserByName(user);
log.info("Risk Owner: " + user1);
tw.local.emailTo = user1.attributes['Task Email Address'];
log.info("email to: " + tw.local.emailTo);

// set e-mail subject
tw.local.emailSubject = '[BPM] Requesting your review on ' + tw.local.riskObject.base.name;

// find task id of the coach page to review the risk object
var taskId = tw.system.currentProcessInstance.tasks[tw.system.currentProcessInstance.tasks.length-1]
   .id.split('\.')[1];
for(var i=0;i<tw.system.currentProcessInstance.tasks.length;i++) {
    if (tw.system.currentProcessInstance.tasks[i].subject.match(".*Review.*")) {
        taskId = tw.system.currentProcessInstance.tasks[i].id.split('\.')[1];
    }
}

// find process id
var procId = tw.system.currentProcessInstance.id.split('\.')[1];

// set e-mail content, which contains link to object detail page, BPM task page, and BPM process page
tw.local.emailBody = "<p>Please review Object Link : " + tw.local.opBaseUrl +
   "openpages/view.resource.do?fileId=" + tw.local.objectId + " and</p>";
tw.local.emailBody += "<p>Task Link : https://bpmserver.com:9443/ProcessPortal/
   launchTaskCompletion?taskId=" + taskId + " , finally</p>";
tw.local.emailBody += "<p>Process Link : https://bpmserver.com:9443/ProcessPortal/
   launchInstanceUI?instanceId=" + procId + "</p>";
```

15. Drag a **Nested Service** from the palette onto the canvas. This node sends the email.
    a) In the **Properties** pane, click the **Implementation** tab, and select the Email Send IBM BPM Email system integration service
    b) In **Input Mapping** , specify the following variables:
    - `tw.local.emailTo` in **to (String)**
    - `tw.local.defaultFromAddress` in **from (String)**
    - `tw.local.emailSubject` in **subject (String)**
    - `tw.local.emailBody` in **messageText (String)**
    - `"text/html"` in **contentType (String)**
    - `tw.local.smtpHost` in **smtpHost (String)**
16. Connect all the nodes with arrows.
17. Save the business process.

Terminating a running process

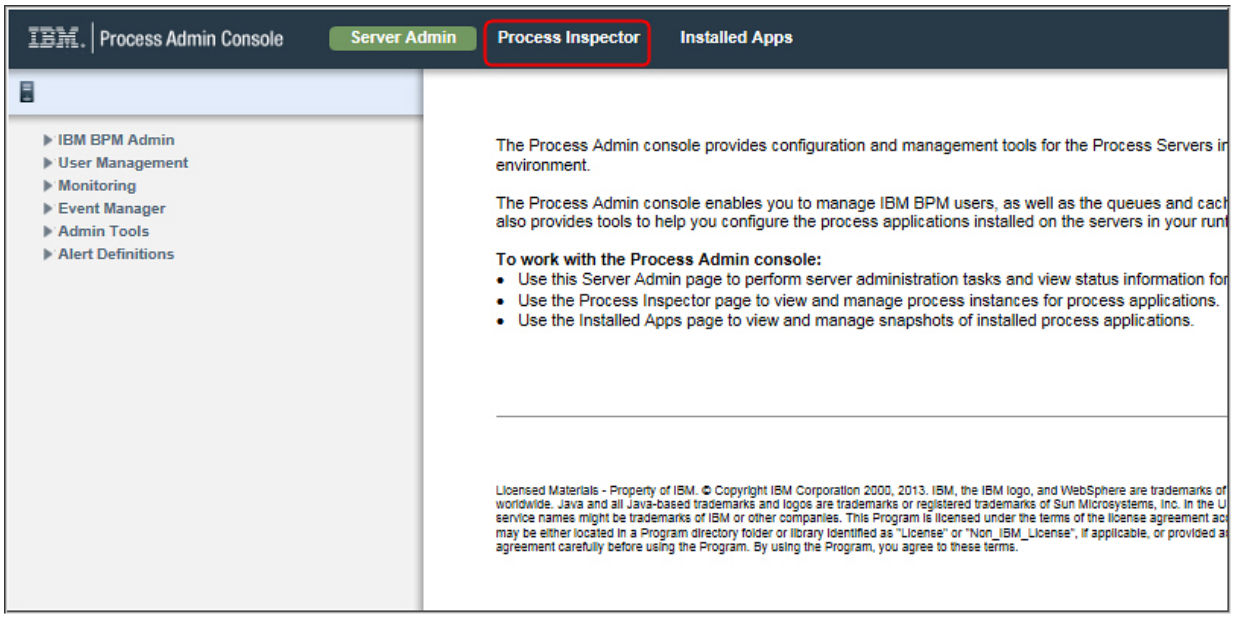
You can use the Inspector in IBM® Process Designer to terminate an instance of an erroneous running process. It can also be used to manage, test, and administer business process instances. You access the Inspector on the Process Admin Console.

For more information about the Inspector, see http://www.ibm.com/support/knowledgecenter/en/SSV2LR/com.ibm.wbpm.wle.editor.doc/topics/running_debugging_procs.html (http://www.ibm.com/

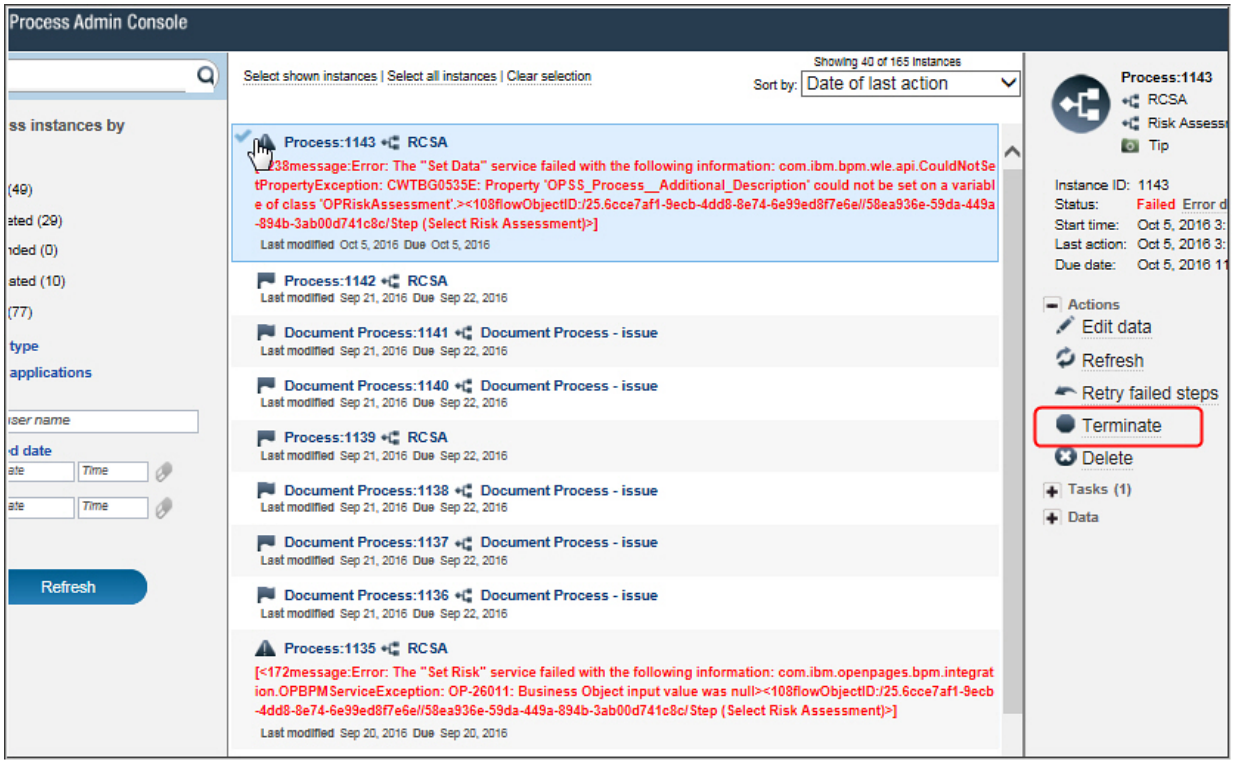
support/knowledgecenter/en/SSV2LR/com.ibm.wbpm.wle.editor.doc/topics/running_debugging_procs.html).

Procedure

1. Access the Process Admin Console.
2. Click **Process Inspector**.



3. Find the instance of the process you want to terminate.
4. Highlight the instance and click **Terminate**.



Chapter 6. Error messages and handling

Error messages and handling are managed by OpenPages integration services.

The OpenPages integration services can issue error messages for various reasons, such as, user input errors or configuration errors. When an error occurs, the system terminates the process instance and the integration service. However, you can overwrite the default behavior by adding error handling and error message codes to coach pages. For example, the services can show a specific error message or recover from the error under certain conditions.

For more information about error handling in IBM Business Process Manager, see http://www.ibm.com/support/knowledgecenter/SSFTDH_8.5.7/com.ibm.wbpm.wle.editor.doc/topics/handling_exceptions.html and (ftp://ftp.software.ibm.com/software/iea/content/com.ibm.iea.ibpmgr/ibpmgr/8.0/ProcessDesigner/BPM80_ErrorAndTerminationHandling.pdf).

Error messages issued by integration services

OpenPages integration services can issue the following error messages.

OP26001
Requested operation could not be understood by the OpenPages server (HTTP error code 400). Refer the BPM log file for more detail.

OP26002
Requested operation failed due to the authentication failure (HTTP error code 401).

OP26003
You do not have permission to perform requested operation (HTTP error code 403).

OP26004
Requested object not found by the OpenPages server (HTTP error code 404).

OP26005
Requested operation failed due to the resource conflict (HTTP error code 409).

OP26006
Requested range is not satisfiable (HTTP error code 416).

OP26007
Requested operation failed due to internal server error (HTTP error code 500). Refer the BPM log file for more detail.

OP26008
Connection refused by the OpenPages server.

OP26009
Requested operation failed by the timeout.

OP26010
Initializing connection to OpenPages server failed. Verify the OpenPages base URL configuration.

OP26011
Requested operation failed due to unsatisfiable arguments.

OP26012
Requested operation failed due to system task authentication error. Verify OpenPages system task credential configuration.

OP26013
Requested operation failed due to namespace binding error. Verify the namespace binding configuration.

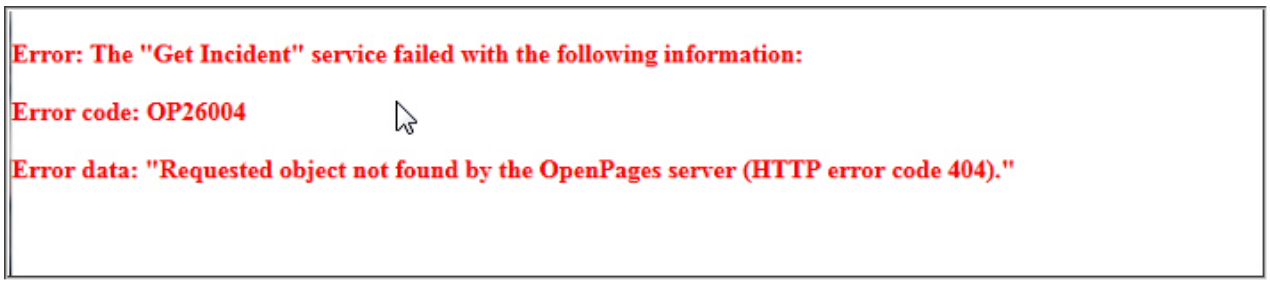
OP26014
OpenPages server response is not expected JSON format.

OP26999
Requested operation failed to access OpenPages. Refer to the BPM log file for more detail.

Adding error handling to a client-side human service

You can add error handling to client-side human services so that integration services display error messages to users.

For example, the following message is issued if an integration service tries to access an OpenPages object that does not exist.



If you do not add error handling to client-side human services, error messages are not displayed to users.

In this example, you create a client-side human service and add error handling to a coach page.

Procedure

1. Open the Process Designer web editor.
2. Create a process app.
3. Open the process app in the Process Designer.
4. Add dependencies to the OpenPages toolkits.
   a) Click **+** next to **Toolkits** and select OpenPages Platform.
   b) Click **+** next to **Toolkits** and select OpenPages Solutions.
5. Click **New** beside **User Interface** > **Client-Side Human Service**, and enter a name.
6. Click the **Diagram** tab.
7. Drag an **Event Handler** from the palette onto the canvas. The event handler does not require connecting arrows.
8. In the **Properties** pane, click the **Implementation** tab.
9. In **Triggering Event**, select **Error event**.
10. In **Behavior**, select either **Catch all errors** or **Catch specific errors** and specify an error that you want to test for. For a list of all error codes, see “Error messages issued by integration services” on page 55.
11. To implement the event handler, double-click the event handler activity in the diagram.
12. Drag a **Coach** from the palette onto the canvas.
13. Drag an **End Event** from the palette onto the canvas. In the **Properties** pane, click the **Implementation** tab. In **Event Type**, select **Error end event**. Optionally, if you want to handle the case as expected behavior and proceed to the parent process, leave it as **End event**.
14. Click the **Variables** tab, and add private variables to hold the error codes and data. Name them `code1` and `msg1`. Both variables are Strings.
15. In the **Properties** pane, select the **Pre&Post** tab. Add the following lines to the **Pre-Execution Script**:

```
tw.local.code1 = tw.error.code;
```

```
tw.local.msg1 = tw.error.data;
```

16. Click the **Coaches** tab. Add Output Text views to the coach page and bind them to `code1` and `msg1`.
17. Optionally, change the color of the error code and error message controls. For example, you can add an HTML attribute that displays the error message as red text.
18. Save the client-side human service.

Notices

This information was developed for products and services offered in the U.S.A.

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. This document may describe products, services, or features that are not included in the Program or license entitlement that you have purchased.

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
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Location Code FT0
550 King Street
Littleton, MA

01460-1250
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All 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 the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

Copyright

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. 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 IBM's application programming interfaces.

Trademarks

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corp., 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 www.ibm.com/legal/copytrade.shtml.

Index

P

R

S

T

U