2021.0
Developer Guide

Contents

1. Developer Guide for WorkZone Process 2021.0

With WorkZone Process you can automate work processes and you can work with processes

directly in Microsoft Outlook using WorkZone for Office or in WorkZone Client.

This guide

This guide describes how you can configure processes for WorkZone Process. It also covers more advanced topics about extending WorkZone Process with additional features and about integrating WorkZone Process from third party platforms.

Target groups

The main target groups of this guide are:

- Business analysts who want to create simple configurations of work processes and the forms applied in the processes.
- Business analysts who want to create simple configurations of work processes and the forms applied in the processes.
- Developers who want to create complex configurations and customizations.

Required skill level

To implement minor changes and additions in WorkZone Process, you do not need to be specialized within programming languages such as C# or to poses any other advanced development skill sets.

A basic understanding of the technologies mentioned below will enable you to start developing processes in Visual Studio and with a good overview of WorkZone Process work processes you can accomplish a lot by looking at the existing implementations.

The amount of prior experience that is required to work with process configurations for WorkZone Process depends on what you want to do.

- In general you must be familiar with Business Process Model and Notation (BPMN) and process modeling tools.
- You must also have a basic understanding of HTML and JavaScript.

Additional skill sets are recommended for the following tasks.

- To build workflows: Ability to work with Visual Studio and C#.
- To work with more advanced areas such as custom activity libraries: Experience with Windows Workflow Foundation.
- To build workflows forms: Knowledge and experience with HTML , Javascript, and libraries such as JQUery and Angular.
- To build your own form controls: experience with HTML, Javascript, Angular, and JQuery.

Related product documentation

- WorkZone Process User Guide
- WorkZone Process Administrator Guide
- WorkZone Configurator Administrator Guide

WorkZone links

- WorkZone documentation
- WorkZone support
- WorkZone website
- WorkZone portal

2. What's new

New features in WorkZone Process 2021.0

No changes in this release.

WorkZone Process 2020.3

New OAuth2 properties

All container interfaces, such as the init, edit, smart task, and case activity interfaces have been extended with three new properties **string odataUri**, **string processUri**, and **object AuthorizationHeader**. See Init form container interface, Edit form container, Smart tasks container, and Case activity container.

Use these properties when you build forms that will run on an environment that is configured with OAuth authentication.

WorkZone Process 2020.2

No changes in this release.

WorkZone Process 2020.1

No changes in this release.

WorkZone Process 2020.0

- The SDK now includes a sample SmartPost dispatcher, which can be used as a starting point for create a customized dispatcher. See Create a SmartPost dispatcher.

WorkZone Process 2019.3

No changes.

WorkZone Process 2019.2

No changes.

WorkZone Process 2019.1

Plugins

The **SimpleMergeDocumentsToPdf** activity now supports the ability to specify a plugin that allows additional properties to be set on the resulting PDF document metadata used in the **ExternalCommunication** package. See About plugins and Extend packages using plugins.

SmartPost

PowerShell script to start a SmartPost process

A new sample PowerShell script that shows how you can start a SmartPost process. The script replaces the C# example. See Start a SmartPost process using a script.

OpenCase parameter

A new **OpenCase** parameter that opens a closed case so that it is possible to send SmartPost messages. The parameter is only used if you want to integrate from another system.

Case activities

The documentation that describes how to create and deploy case activity lists based on DCR Graps has been moved to Case activities in the WorkZone Process Administrator Guide.

WorkZone Process 2019.0

No changes.

WorkZone Process 2018.2 SP1

No changes.

WorkZone Process 2018.2

No changes.

WorkZone Process 2018.1

No changes.

WorkZone Process 2018.01

- You can configure the SmartPost process comply with customized of locations of CVR and CPR numbers in the database. See Configure SmartPost PartyIdentifierSources (sending) and Configure SmartPost ContactAddressSources (receiving).

WorkZone Process 2018

- The topics on case activities have been revised.
- You can now create a DCR graph based on a default WorkZone template.
- You can now use Fiddler for debugging. See Enable Telerik Fiddler tracing.
- The Configure packages topic has been extended with a description of DataContextDefinition controls and a new details section control.
- The WorkZone Client form container now supports a Dirty Marking API. See Support for dirty marking in containers (Dirty Marking API).

WorkZone Process 2017 SP1

- New document filter control `<wzp:document-selector-filter>`.
  A new control for filtering documents in <wzp-multi-selector> and

<wzp:rollbackselector> has been introduced. The new <wzp:document-selector-filter> control replaces the <wzp:filter-selector> control, which is now obsolete. Configuration of new control is identical for SmartTask and InitForm.
For more information, see To add a filter control in the <wzp-multi-selector> and <wzp:rollbackselector> parent controls.

- New sequence mask control `<wzp:sequence-mask-selector-filter>`.
  A new control for selecting actors from sequence masks in <wzp-multi-selector> and <wzp:rollbackselector> has been introduced. The new <wzp:sequence-mask-selector-filter> control replaces the <wzp:sequence-mask-selector> control, which is now obsolete. Configuration of new control is identical for SmartTask and InitForm.
  For more information, see To add a sequence-mask control in the <wzp-multi-selector> and <wzp:rollbackselector> parent control.

- Obsolete filter controls and controller.
  The following filter controls and controllers are obsolete and should be removed from the html:

  - wzpInitFormFilterForSelectorCtrl
  - wzpSmartTaskFilterForSelectorCtrl
  - wzpInitFormSequenceMaskForSelectorCtrl
  - wzpSmartTaskSequenceMaskForSelectorCtrl

- New contact type filter: <wzp:contact-type-selector-filter>.
  A new control for filtering contacts and addreses by contact type for <wzp-multi-selector> and <wzp:rollbackselector> has been introduced.
  Configuration of new control is identical for SmartTask and InitForm.
  For more information, see To add the control in <wzp-multi-selector> and <wzp:rollbackselector> parent control

- The RollbackSelectors have been updated.

WorkZone Process 2017

Selector controls were upgraded

In WorkZone Process 2017 the selector controls in forms have been upgraded. For more information about the upgrade, see Upgrade selector controls from 2016 to 2016 R2.

Two elements were removed from the html forms:

`wzp-usertask-rollbackselector-panel-helper`: Previously this attribute was used in "`<ui:usertask ng-controller="ApproveTaskCtr" ui-intl="task.Submission." formname="taskform" wzp-usertask-rollbackselector-panel-helper>`". From now on, you can just use "<ui:usertask ng-controller="ApproveTaskCtr" ui-intl="task.Submission." formname="taskform" >"

`ng-controller="uiActionsCustomController"`: Previously this element was used in "<ui:actions capability="execute" ng-disabled="isNotValid()" ng-controller="uiActionsCustomController">". From now on, you can just use- "<ui:actions capability="execute" ng-disabled="isNotValid()" >"

Data model changes

The following new elements have been added to the registers and tables in the data model:

- WZP_CASE_ACTIVITY
- WZP_CASE_ACTIVITY_HISTORY
- WZP_SERVICE
- WZP_SERVICE_PARAMETER
- WZP_MAIL_NOTIFICATION

In this guide, the descriptions in the Database section was updated and the information is now divided into descriptions of tables and registers.

For more information about the new database elements, see Database.

Case activity graphs available for workflows

You can now use case activity graphs to model workflows in WorkZone for standard work processes as well as for ad-hoc tasks.

Case activity graphs enable you to model tasks in responsive workflows and the flow of tasks need not be known in advance. You can, for example, handle conditions such as these:

- The order of activities to be completed can vary.
- All possible activities need not be executed within each workflow.
- Activities must be repeated or disregarded depending on the outcome of other activities.

Integration

New integration features are now available. See Integration.

New form types

New form types have been introduced. Now the following form types are available:

Init Forms

Smarttasks forms

Edit forms

CaseActivity forms

For more information, see The forms concept.

3. Architecture

From this section you can get an overview of the WorkZone Process architecture and a basic understanding of the major components of the product.

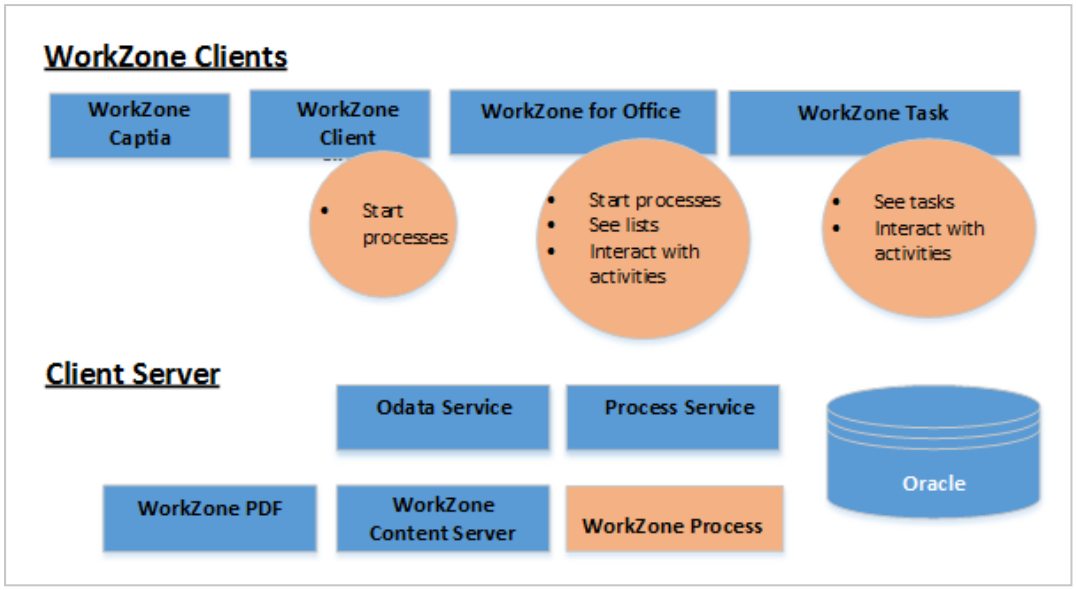
3.1 Overview

WorkZone Process is a service within the WorkZone environment.

The service extends WorkZone with workflow capabilities by adding services for workflows and forms that are used in WorkZone Process. On the client side, these services can be used to interact with users.

WorkZone Overview

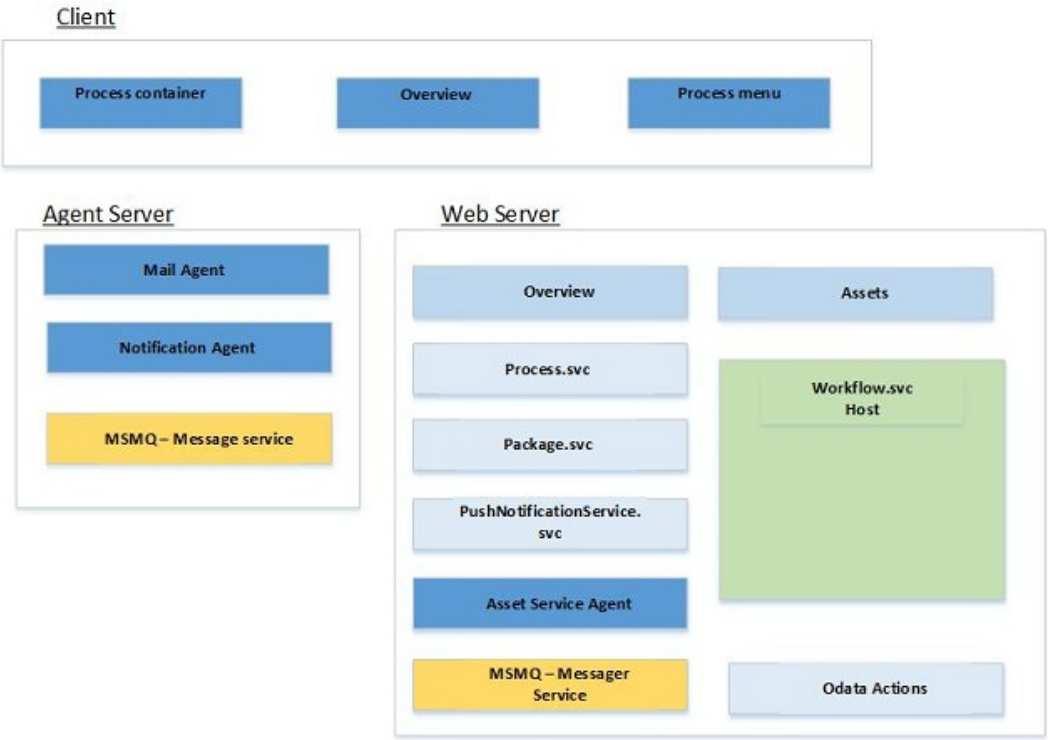
In the following overview, you can see the client and the server parts of the WorkZone environment. You can apply WorkZone Process forms and dialogs in the clients to, for example, view lists or start processes.



3.2 WorkZone Process components

The WorkZone Process overview below illustratest WorkZone Process components of an agent server and a web server installation of WorkZone Process.

WorkZone Process is composed of a set of components. In the following overview the components are grouped according to the installation method which can be either by an agent server or by a web server.



Client

Component	Description
Process container	Allows installation and configuration of all WorkZone Process components.
Overview	A single page application which allows viewing the status of all processes, the status of their tasks, and performing actions on both.
Process menu	Client-implemented context-sensitive menu option that is populated with processes available in the current context. By calling Process.svc//Definitions/{register}/{systemKey=null} the client gets available process for this context.

Agent server installations

The agent server installs Windows Services. Windows Services manages WorkZone Process related communication services.

Component	Description
Mail agent	Windows service that sends smart mails to actors.
Notification agent	Windows service that sends push notifications.
MSMQ	Receiving message queue that handles messages to the notification agent. For example, push notification messages for WorkZone Task.

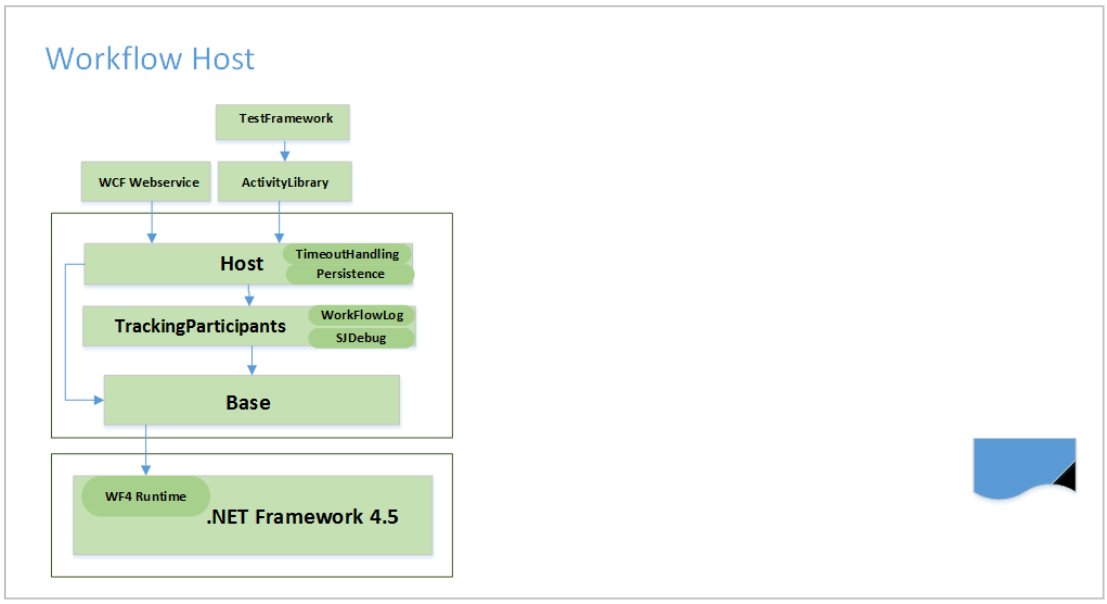
Web server installations

The web server installation requires Internet Information Services (IIS) such as rest service end points. The ISS must be activated from Windows NT. Also, a queue and a workflow host are required components.

Component	Description
`Workflow.svc`	REST service for workflows serviced by the workflow host.
`Package.svc`	REST service that handles package installation. The service allows loading/deployment of a package to a server.
`Process.svc`	Primary REST service for process, getting process information, starting a process, getting process forms, and doing process actions.
`PushNotificationService.svc`	REST service for registering push notification receivers.
Asset Service agent	Service that synchronizes the Assets folder of IIS. This ensures that package changes are loaded to the Assets folder.
MSMQ	Receiving message queue that handles messages to the notification agent. For example, push notification messages for WorkZone Task.

Component	Description
OData Actions	OData extensions that extend WorkZone Process registers by adding actions to them. This allows having one single interface for communication with OData and the WorkZone Process Host.

3.3 Host architecture



The Workflow Host

In a web server installation of WorkZone Process a web server host must be implemented to facilitate creation and processing of and communication with workflows.

Implementation

The Host is a DLL which is built as an extension to WF4 Runtime which is part of the .NET Framework 4.5. The WF4 Host creates instances of the workflows defined in the database and the workflows are implemented as XAML. The workflow instances are executed by runtime. The workflow host uses the webservice to communicate with the clients.

Persistence with timeouts

The WF4 Host has several classes of which two are going to be mentioned here:

- `Persistence`. Persistence saves the state of the workflow in the database, and unpersists the workflow when the method `ResumeBookmark` is called through the web service. For every workflow which is persisted, a timeout is set.
- `TimeoutHandling` unpersists workflows when they are set to expire, depending on the default duration defined for the workflow in **Workflow Configuration Management**.

The WCF Webservice

The workflow host is called when an end user starts and manages processes in the client. Communication between the client and the workflow host is facilitated by a Windows Communication Foundation (WCF) web service. If, for example, the type of workflow is §20, then the workflow host looks for that particular workflow type in the database, and – when found – starts a workflow instance.

The webservice is configured in IIS to start whenever IIS starts.

Activity library

`ActivityLibrary` is a collection of activities, that is, the building blocks of the workflows.

Tracking Participants

`TrackingParticipants` is a class which:

- keeps track of what happens with the running workflows. This is recorded in the in the database in the table `workflowlog`. You can access the workflow log in WorkZone Configuration Management.
- writes to SJDebug which is a diagnostic tool to track and log activities.

Base

The base contains all the basic elements that are used by the host, `ActivityLibrary` and `TrackingParticipants`. For example methods linking to SOM.

4. WZP Process Package Development

4.1 Configure packages

In WorkZone Process, you can model processes in process packages. A process package contains a process configuration and forms. You can use the WorkZone Process Package Loader to deploy a package to development, test, and staging environments.

A simple package contains the following basic configuration elements:

- **Package.xml** - A package configuration file that contains the configuration of the package forms and the processes.
- **UI**- A folder that contains the form view and controller. If you use it for further customizations, this folder can also contain localization files, layout files, and icons (seeForms) about form configuration.
  - **Init.htm**
  - The visual implementation of the **Process start** form.
  - **Init.js**
  - Validation logic and business rules for the **Process start** form.
  - **Task.htm**
  - The visual implementation of **Smart Task**.
  - **Task.js**
  - Validation logic and business rules for the **Smart Task** form.
- **Workflows** - Contains the package workflow (see Workflow process modelling for information on workflow modeling).

- **Workflows.xaml** - The workflow configuration.

The example below shows a simple package configuration file, which is configured with two forms and one workflow.

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<Package>
<Formularer>
  <FormDefinition>
  <FormGuid>{709f3330-9190-4cc9-a7d5-0b30edef0e6e}</FormGuid>
  <Name>Init.Submission</Name>
  <Default>J</Default>
  <ContentType>TEXT/HTML</ContentType>
  <ContentFile>ui\init.Submission.html</ContentFile>
  <ControllerFile>ui\init.Submission.js</ControllerFile>
  </FormDefinition>
  <FormDefinition>
  <FormGuid>{e31bed95-94f3-4c61-a37c-5460411621cd}</FormGuid>
  <Name>Task.Submission.Approve</Name>
  <Default>J</Default>
  <ContentType>TEXT/HTML</ContentType>
  <ContentFile>ui\task.Submission.Approve.html</ContentFile>
  <ControllerFile>ui\task.Submission.Approve.js</ControllerFile>
  </FormDefinition>
</Formularer>
<Workflows>
  <WorkflowDefinition>
     <Version>1.0.0.0</Version>
     <XamlFile>Workflows\Submission.xaml</XamlFile>
```

```
      <FormGuid>{709f3330-9190-4cc9-a7d5-0b30edef0e6e}</FormGuid>
      <AccessCode>ALLEEMNER</AccessCode>
      <Standard>J</Standard>
      <Processes>
        <ProcessDefinition>
        <ProcessGuid>{68D5E05E-C079-4A76-8CEB-B8EC44EDA56B}</ProcessGuid>
        <Type>MAIN</Type>
        <Name culture="en-GB">Basis Submission</Name>
        <Description>Basis process</Description>
        <DisplayOrder>666</DisplayOrder>
        <DurationUnit>D</DurationUnit>
        <DefaultDuration>15</DefaultDuration>
        <AccessCode>ALLEEMNER</AccessCode>
        <Package>Scanjour.Process.Basis</Package>
        </ProcessDefinition>
      </Processes>
    </WorkflowDefinition>
  </Workflows>
  </Package>
```

Form configuration

Each forms package must contain a `FormDefinition` node in the `Forms` section:

Basic form configuration

Example:

```
  <Forms>
    <FormDefinition>
      <FormGuid>{F3C3A448-F378-4AB9-8729-821941BBD9B0}</FormGuid>
```

```
    <Name>Init.Submission</Name>
    <Default>J</Default>
    <ContentType>TEXT/HTML</ContentType>
    <ContentFile>ui\init.Submission.html</ContentFile>
    <ControllerFile>ui\init.Submission.js</ControllerFile>
  </FormDefinition>
```

Where:

- `FormGuid` is the unique identifier of the form.
- `Name` is the name of the form.
- `Default` is "J". Do not change the value.
- `ContentType` is the type of content, usually "TEXT/HTML".
- `ContentFile` is the path to the file (in the zipped package) that contains the visual implementation of the form.
- `ControllerFile` is the path to the file that contains the logic of the form.

SmartTask form configuration

If it is a SmartTask form, the configuration must contain additional elements depending on which controls are used.

If the SmartTask form contains controls with dynamic data, the form configuration must contain a corresponding `DataContextDifinition` node.

Example:

```
  FormDefinition>
    <FormGuid>{21685432-EC7F-45FF-BB11-D1D4A7A04D16}</FormGuid>
    <Name>Task.Submission.Approve</Name>
    <Default>J</Default>
    <ContentType>TEXT/HTML</ContentType>
    <ContentFile>ui\task.Submission.Approve.html</ContentFile>
```

```
    <ControllerFile>ui\task.Submission.Approve.js</ControllerFile>
    <Data>
      <DataContextDefinition>
        <Name>ActiveActors</Name>
        <Query>WzpUserTasks?$expand=NameKey&amp;$select=InstanceId,NameKey_
        Value,TaskState_Value,NameKey/ID,NameKey/Summary,NameKey/NameType_
        Value,NameKey/NameCode&amp;$filter=InstanceId eq '{0}' and
        (TaskState_Value eq 'OPEN' or TaskState_Value eq
        'PENDING')&amp;$orderby=TaskOrder</Query>
        <MaxOfflinePages>10</MaxOfflinePages>
        <Parameters>
          <Parameter>InstanceId</Parameter>
        </Parameters>
      </DataContextDefinition>
    </Data>
  </FormDefinition>
```

Where:

- `Name` is the key of DataContext (specified for standard controls). If you add custom controls, you need to add a DataContextDefinition that corresponds to it).
- `Query` is the OData query used for collecting dynamic data.
- `MaxOfflinePages` is the maximum number of OData pages that are sent as offline data.
- `Parameters` is a list of parameters used in the query (see examples in the Basis package).

Standard DataContextDefinition controls

The table below lists `DataContextDefinition` standard controls:

Control	DataContextDefinition Name	Query and Parameters
wzp-process-log	ActionLog	<Query>WzpUserTasks?$select=NameCode/Summary,ProxyCode/Summary,ID,Importance_Value,TaskSchedule_Value,Title,Comment,NameCode_Value,NameOu_Value,ProxyCode_Value,ProxyOu_Value,Closed,Opened,Created,DueDate,NearDueDate,TaskAction_Summary,TaskState_Value,TaskAction_Value,TaskType_Value&$expand=NameCode,ProxyCode&$filter=Show eq true and InstanceId eq '{0}'&$orderby=TaskOrder</Query> Parameters> <Parameter>InstanceId</Parameter> </Parameters>
wzp:smart-task-details-section	MainPhases	<Query>WzpUserTasks?$filter=TaskId eq '{0}'&$expand=Phases,Root/Process,Root/File,Root,Instance&$select=Instance/RowId,Instance/Created,RootId,Phases/Closed,Phases/DueDate,Phases/Name_Summary,Phases/Number,Phases/Opened,Phases/Schedule_Value,Phases/State_Value,Root/Process/Name,Root/File/FileNo,Root/File/Title,Root/DueDate,Created</Query> <Parameters> <Parameter>TaskId</Parameter> </Parameters>
wzp:answers-and-comment	AnswerDocuments	Query>Records?$select=ID,Summary,DocumentType_Value,State_Value,Extension&$orderby=Mru/Favorite,Mru/Updated desc,Updated desc&$filter=FileKey_Value eq '{0}' and State_Value ne 'UP' and ExternalDocId ne ''</Query> <Parameters>

Control	DataContextDefinition Name	Query and Parameters
		`<Parameter>RegisterKey</Parameter>` `</Parameters>`

SharedDataContextDefinition

You can also create data context, which is shared by SmartTasks forms in a package.

Example:

```
<Forms>
  <FormSharedData>
    <SharedDataContextDefinition>
      <Name>ActionLog</Name>
      <Query>WzpUserTasks?$select=NameCode/Summary,ProxyCode/Summary,ID,Im
      portance_Value,TaskSchedule_Value,Title,Comment,NameCode_
      Value,NameOu_Value,ProxyCode_Value,ProxyOu_
      Value,Closed,Opened,Created,DueDate,NearDueDate,TaskAction_
      Summary,TaskState_Value,TaskAction_Value,TaskType_
      Value&amp;$expand=NameCode,ProxyCode&amp;$filter=Show eq true and
      InstanceId eq '{0}'&amp;$orderby=TaskOrder</Query>
      <MaxOfflinePages>3</MaxOfflinePages>
      <Parameters>
        <Parameter>InstanceId</Parameter>
      </Parameters>
    </SharedDataContextDefinition>
```

And then this shared data context can be used in any SmartTask definition in this package by the `Name` key.

Example:

```
<FormDefinition>
```

```
    <FormGuid>{B67DC731-9F6D-4A61-84F5-DEE028122D42}</FormGuid>
    <Name>Task.Submission.Rejected</Name>
    <Default>J</Default>
    <ContentType>TEXT/HTML</ContentType>
    <ContentFile>ui\task.Submission.Rejected.html</ContentFile>
    <ControllerFile>ui\task.Submission.Rejected.js</ControllerFile>
    <Data>
      <DataContextDefinition>
        <SharedName>ActionLog</SharedName>
      </DataContextDefinition>
    </Data>
  </FormDefinition>
```

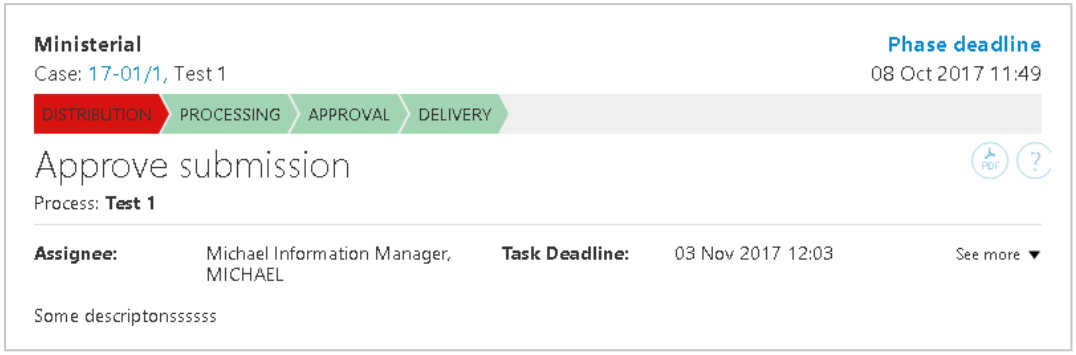
Details Section control

The SmartTask form contains a details section control. This section controls information about the SmartTask, online Help link, **Print** and **Pdf** buttons as well as the description.

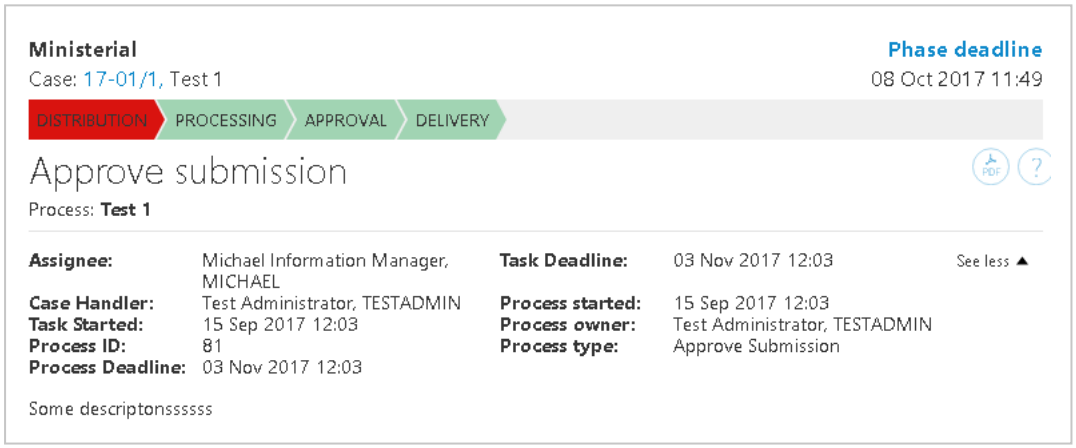
Example:

```
  <form autocomplete="off" name="taskform" class="wzp-usertask-form" ng-cloak>
      <div class="wzp-task-page">
        <wzp:smart-task-details-section
          source="context.context"
          title-label="TITLE"
          title-label-group="TASKHEARINGSUMMARY"
          help-link="Default.htm#Basis_package/Hearing_
          summary.htm%3FTocPath%3DWorkZone%2520Process%2520Basis%2520Package%7
          CBasis%2520hearing%2520processes%7C_____6">
        </wzp:smart-task-details-section>
```

If the SmartTask is part of a phase process, it also contains a phase bar and phase process information.



By default, the details information is shown in collapsed mode but it can be expanded to see more information.



This control requires the `DataContextDefinitions` “ActionLog” in the package.xml file, either directly or by `SharedDataContextDefinition`.

The query for this `DataContextDefiinition` is:

```
<Query>WzpUserTasks?$select=NameCode/Summary,ProxyCode/Summary,ID,Importanc
e_Value,TaskSchedule_Value,Title,Comment,NameCode_Value,NameOu_
Value,ProxyCode_Value,ProxyOu_
Value,Closed,Opened,Created,DueDate,NearDueDate,TaskAction_
Summary,TaskState_Value,TaskAction_Value,TaskType_
Value&amp;$expand=NameCode,ProxyCode&amp;$filter=Show eq true and
InstanceId eq '{0}'&amp;$orderby=TaskOrder</Query>
<Parameters>
  <Parameter>InstanceId</Parameter>
```

```
    </Parameters>
```

4.2 About plugins

It is possible to tune the behavior of certain processes using activities with support for plugins. In this release, the only process using plugins is the **ExternalCommunication** package with the SmartPost process.

The `SimpleMergeDocumentsToPdf` activity tests for the availability of a function with the signature:

```
/// <summary>
/// Update the record.
/// </summary>
/// <param name="updateRecord">The Pdf record to update</param>
/// <param name="inputRecords">The records used when merging the Pdf
record.</param>
/// <returns></returns>
bool UpdateRecord(string updateRecord, List<String> inputRecords);
```

Currently, this is the only activity that uses functions in a plugin but in the future other activities may support other functions that must also be defined in the plugin.

To create a plugin, you must create a C# class library project and define the interface. For example:

```
using System;
using System.Collections.Generic;

namespace WorkZone.<package>.Plugin
{
    /// <summary>
    /// Plugin to do updated to recently merged Pdf records.
    /// </summary>
    public interface IPostPdfMerge
    {
        /// <summary>
        /// Update the record.
        /// </summary>
        /// <param name="updateRecord">The Pdf record to update</param>
        /// <param name="inputRecords">The records used when merging the
Pdf record.</param>
        /// <returns></returns>
        bool UpdateRecord(string updateRecord, List<String>
```

```
inputRecords);
    }
}
```

The interface name is defined in the package where the process is defined. The class library assembly name is defined in the extension package.

The implementation must implement a constructor with the signature shown below and the function defined in the interface:

```
using System;
using System.Collections.Generic;
using System.Net;

namespace WorkZone.<package>.Plugin
{
    /// <summary>
    /// Implements Plugin for SimpleMergeDocumentsToPdf.
    /// </summary>
   public class PostPdfMerge : PluginBase, IPostPdfMerge
   {
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="oDataUri">The oData Uri.</param>
        /// <param name="credentials">The credentials used for
oData.</param>
        public PostPdfMerge(Uri oDataUri, ICredentials credentials) :
base(oDataUri, credentials)
        {
        }

        /// <summary>
        /// Update newly created Pdf document with additional metadata.
        /// </summary>
        /// <param name="updateRecord">The newly generated PDF
document</param>
        /// <param name="inputRecords">The documents contained in the
PDF</param>
        /// <returns>true if the update succeds.</returns>
        public bool UpdateRecord(string updateRecord, List<string>
inputRecords)
        {
        }
    }
}
```

The base interface exposes two methods, which allows the function to get an OData client context and a simple OData client.

```
ODataService GetOdataContext();

SimpleODataClient GetSimpleODataClient();
```

If the plugin needs to update custom properties that are not known to the OData context the `SimpleODataCLient` must be used.

4.3 Extend packages using plugins

With the introduction of plugins to certain activities, you will need to update an already installed package with an extension package. You define a plugin using a node in the `WorkflowDefinition` section that specifies the name of the assembly and the name of the interface:

```
<PluginAssembly>Assemblyname.dll</PluginAssembly>
<PluginInterface>IUpdateMetaData</PluginInterface>
```

The Package Loader supports that you can extend existing packages by installing a package extension.

The package must include a new node named `Extension` in the XML definition. The `Extension` node specifies which package that the extension affects:

```
<PackageDefinition>
    <Name>CommunicationExtension</Name>
    <Extension>ExternalCommunication</Extension>
    <Version>19.1.1.0</Version>
    <Description>Package contains ExternalCommunication
extensions.</Description>
 </PackageDefinition>
```

When the `Extension` node is met, the specified version of the package is matched against the version specified. The two versions must identical for the extension package to load.

The package.xml may contain sections to add or modify the following sections:

- <Assemblies>
- <Assets>
- <Forms>

The `<Workflows>` section only allows a small part of a WorkflowflowDefinition to be modified. The XAML part of the workflow cannot be modified so in order to identify a workflow, a new node named `TypeName` is specified, which together with the `Version` identifies the workflow. Also, it is possible to specify which dll contains the plugin. (The assembly must be added in the `<Assemblies>` section in order for the workflow to find it).

```
<WorkflowDefinition>
      <Version>19.1.1.0</Version>
      <TypeName>WorkZone.ExternalCommunication.SendSmartPost</TypeName>
      <PluginAssembly>WorkZone.FSMI.Plugin.dll</PluginAssembly>
```

In the `<Processes>` section `<ProcessDefinition>`, the following nodes can be modified:

- Name
- Description
- DisplayOrder
- DurationUnit
- DefaultDuration
- NearDuration
- AccessCode
- Access

The ProcessGuid is mandatory and must match the value of the original package.

Most important – it is possible to define more process parameters in the <Parameters> section.

```
<ProcessDefinition>
          <ProcessGuid>{23B9498E-BCA5-4746-98A0-
71E03CD6963C}</ProcessGuid>
          <Parameters>
          <!—specify new parameters -->
          </Parameters>
        </ProcessDefinition>
```

5. Process designer

5.1 Configure phase processes

5.1.1 The phase process

The phase process is a way to get an overview of processes, which consists of several steps, and indicate where in the steps a certain business process is. A phase process has one active phase and uses actions (bookmarks) to select the active phase.

Building a phase process

To assist in building phase processes as a XAML workflow a number of activities is available in the toolbox:

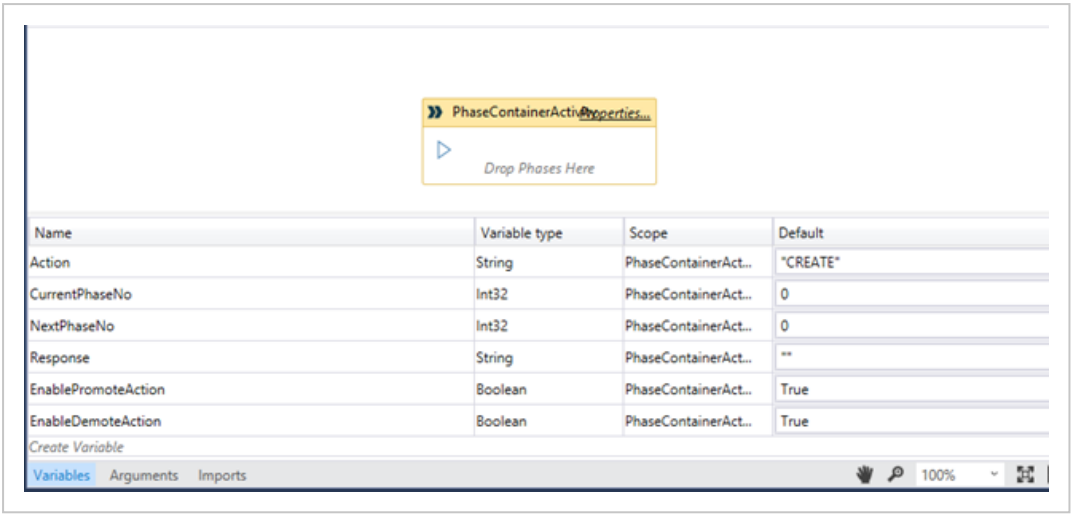
PhaseProcess activity

The main container for a phase process. This is a placeholder for the PhaseContainer activity and it has a number of predefined variables which are used by the PhaseContainer to manage phases.

PhaseContainer activity

The PhaseContainer is responsible for creating phases in the database and to respond to actions (bookmarks) supported by the phase activity. The Phase container has a visual interface that allows Phase activities to be dropped in the container.

Below is a snapshot of the PhaseContainer and the predefined variables in the PhaseProcess after the PhaseProcess has been dropped in the design surface in Visual Studio.



Edit workflow properties

Click the Properties button in the Phase container form, you can edit workflow properties such as Display name.[DRAFT]

Phase activity

The Phase activity is a composite activity that supports deadlines and can execute activities when actions (bookmarks) happens.

Activity placeholders are available for the following actions:

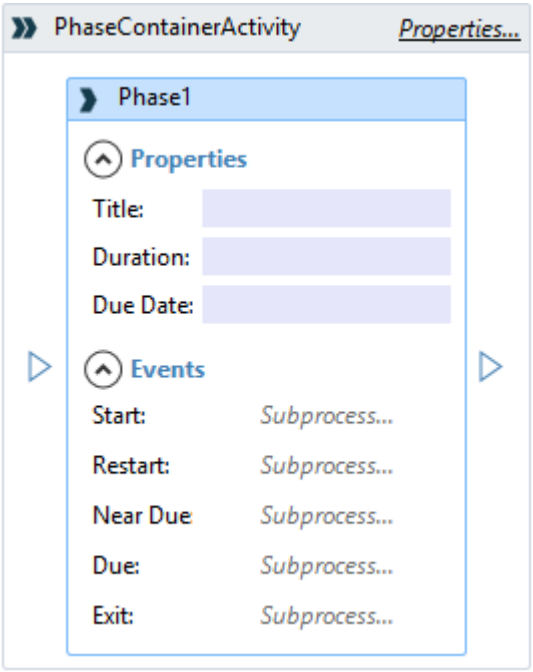
Start	The activities are executed when the phase becomes active at start or from a previous phase.
Restart	The activities are executed when the phase becomes active from a subsequent phase.
NearDue	The activities are executed when a deadline is about to happen.
Due	The activities are executed when a deadline is about to happen.
Exit	The activities are executed when the phase is left.

Note:

The above activities allow the Phase process to inform about an event. However, the phase process is about to change so the activities are executed in a NoPersistZone.

This ensures that the phase transition will not be delayed because the workflow is persisted.

Below is a snapshot after a phase has been dropped into the PhaseContainer activity:

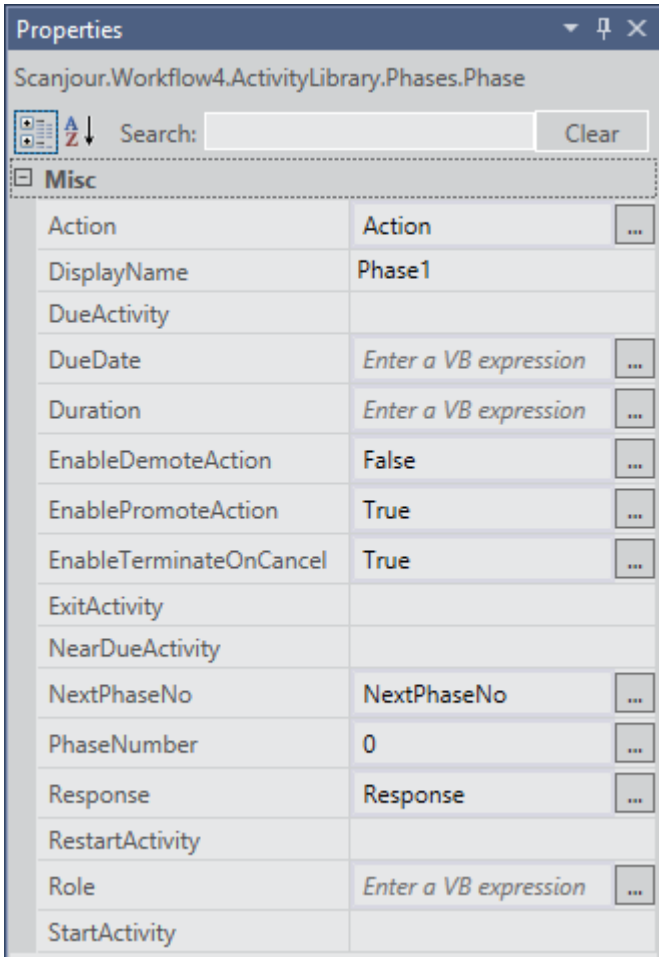


In the current version of the designer, a number of properties of the Phase activity must be bound manually to the variables in the PhaseContainer:

Action	Informs the Phase activity what action is to be executed.
NextPhaseNo	Informs about the next active phase. You can jump back and forward between phases.
Response	Informs the phase container about the action (bookmark) that the phase has just executed.
PhaseNumber	The number of the phase. Must reflect the order of the phase in the designer. First phase is number 0 (zero).
DisplayName	The Phase title used in the designer. The title is not used for the Outlook overview. See Defining the process in the package.xml file.
DueDate	If the phase has a fixed deadline the DueDate can be specified. Normally the DueDate is bound to an InArgument of the workflow to allow the deadline to be specified when the process is started.
Duration	If the phase has a fixed duration the Duration can be specified. If a DueDate is also specified it takes precedence over the Duration.

The Properties EnableDemoteAction and EnablePromoteAction are maintained by the PhaseContainer activity and prevents a demote action to be allowed in the first phase.

Below is a snapshot of the properties of phase 0 after the variables have been inserted.



The phase activity supports the following events:

CREATE:	Creates information about the phase in the database.
PROMOTE:	The phase is becoming active as result of a PROMOTE action.
DEMOTE:	The phase is becoming active as result of a DEMOTE action.
UPDATE:	The phase is updated (new deadline etc.)
CANCEL:	The process is canceled.

Note:

If the ExitOnDeadline property is set to True the icon for the process in the Outlook overview will indicate that the process was canceled. If it is set to False the icon will NOT indicate that the process was canceled.

The Phase Activity uses the following activities, which are visible in the toolbox but are assumed to work inside a Phase activity:

SimplePhase activity

The activity is responsible for creation of the phase in the database and defining the bookmarks needed to support allowed actions on the phase.

UpdatePhase action

Updates the phase action in the database after timer events has occurred in the Phase activity.

UpdatePhase state

Updates the Phase state after an event (bookmark) has occurred in the Phase activity.

Schedule activity

Used to calculate the next Phase deadline based in the DueDate and Duration properties.

IsStringNullOrWhiteSpace

Use to control the flow inside the Phase activity.

Configuring the activity placeholders

When the phase process has been designed with the correct number of phases, the activity placeholders can be populated with activities.

A number of activities are available:

CreateProcess

The activity can start a process from this package or from the Basis or Extended package.

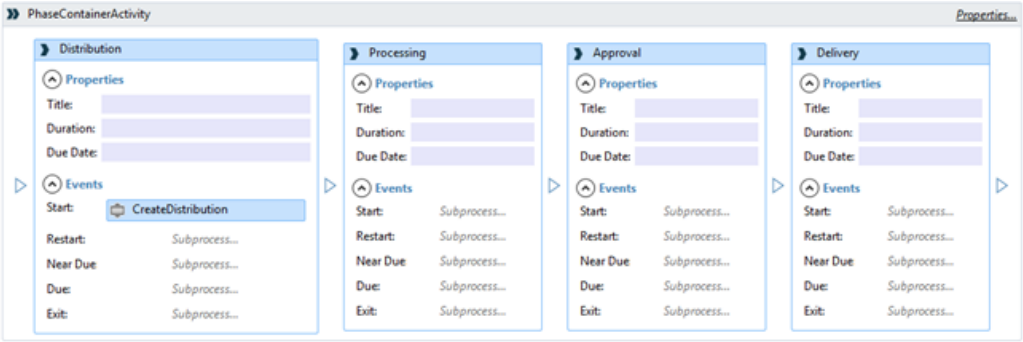
SimpleUserTask

This activity can be used to notify someone via a mail about the event which happened. It's important that the IsNotification property is set to true, so no bookmarks are defined.

Sequence

Adding a Sequence activity allow you to build actions from the activities available in the toolbox. Note, though, that the activites are executed in a NoPersistZone so you cannot use activities which relies on bookmarks during the execution.

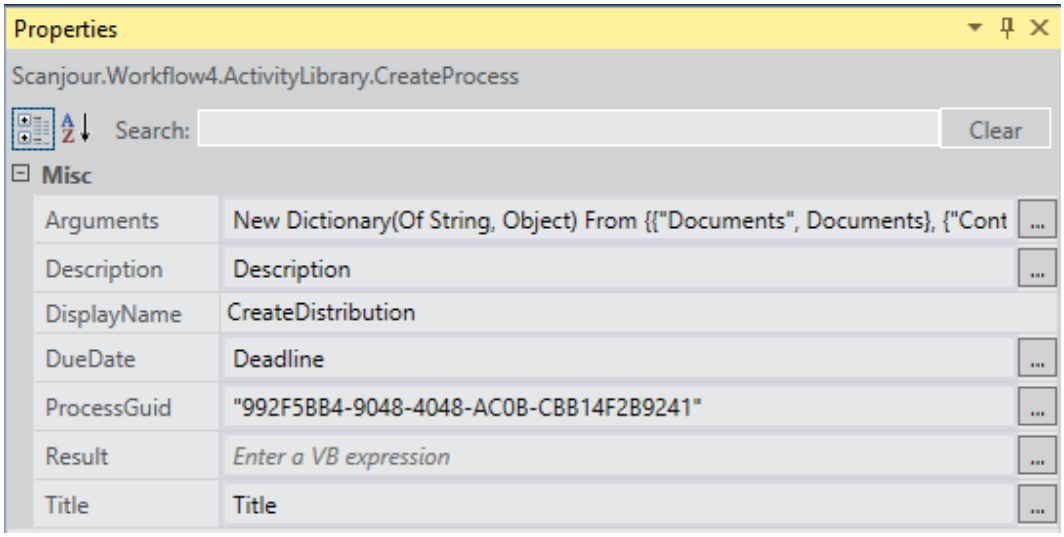
In the snapshot below the CreateProcess has been added to the start event of the first phase to start a distribution process from the Extended package.



When the StartProcess activity is added the properties must be specified in the **Properties:** window. It is important to specify all mandatory arguments of the process in the Arguments property as a Dictionary.

You may use fixed values or you may use arguments from the phase process in the dictionary.

Below is a snapshot of the properties for the Distribution process.



Defining the process in the package.xml file

Currently the designer does not support properties (Title, DueDate and Duration) so these properties must be set on the Properties pane in visual studio.

A number of other characteristics of the process needs to be defined in the process package under the WorkflowDefinition section. Some of the sections below is also described in the Workzone Process Package Development.

The section contains the following:

Version	The Workflow version. Allows breaking changes to be introduced to the workflow by changing the major or the minor version number.
XamlFile	The name of the XAML file containing the workflow.
FormGuid	The Guid of the corresponding Init form used to start the workflow. The Init form is defined in the Forms section of the Package.xml file.
EditFormGuid	The Guid of the corresponding Init form used to start the workflow. The Init form is defined in the Forms section of the Package.xml file.
AccessCode	An access code which may restrict which departments or users who can use the process.
Context	The context in which the process can be started. See the Context description below.
PhaseLabels	The localized Phase names used in the Outlook overview. Must contain the installed cultures in Workzone Content Server, but may contain more.
ActionLabels	The supported Actions (bookmarks). The Actions are defined by the

Phase activity and ,ay be copied from any Phase process.

Both PhaseLabels And ActionLabels are inserted as domains in the Custom_Domain register. The type is autogenerated WZP<wf_id>P and WZP<wf_id>A where <wf_id> is the workflow id in the wzp_workflow register.

Below is a sample of the WorkflowDefinition section of the package.xml file.

```
<WorkflowDefinition>
  <Version>4.3.0.0</Version>
  <XamlFile>Workflows\Ministerial.FivePhase.xaml</XamlFile>
  <FormGuid>{A147CAE0-4B6C-4576-B7C9-1F25CFEBB18B}</FormGuid>
  <EditFormGuid>{01C88CD3-3D36-4BE8-B6BB-8F2135603BEC}</EditFormGuid>
  <AccessCode></AccessCode>
  <Standard>J</Standard>
  <Context>
     <Register>FILE</Register>
     <EntityFilter>Closed eq null and not(Instances/any(p:
     p/Process/ProcessGuid eq '992f5bb4-9048-4048-ac0b-cbb14f2b9241' and
     (p/WorkflowStatus eq 'Persisted' or p/WorkflowStatus eq 'Running')))
     </EntityFilter>
     <TagFilter></TagFilter>
  </Context>
  <PhaseLabels>
     <Phase number="1">
        <Label culture="en-GB">Distribution</Label>
        <Label culture="da-DK">Fordeling</Label>
     </Phase>
     <Phase number="2">
        <Label culture="en-GB">Processing</Label>
        <Label culture="da-DK">Behandling</Label>
     </Phase>
```

```
      <Phase number="3">
        <Label culture="en-GB">Approval</Label>
        <Label culture="da-DK">Godkendelse</Label>
      </Phase>
      <Phase number="4">
        <Label culture="en-GB">Delivery</Label>
        <Label culture="da-DK">Aflevering</Label>
      </Phase>
    </PhaseLabels>
    <ActionLabels>
      <Action name="CANCEL">
        <Label culture="en-GB">Cancelled</Label>
        <Label culture="da-DK">Afbrudt</Label>
      </Action>
      <Action name="CLOSE">
        <Label culture="en-GB">Phase process ended</Label>
        <Label culture="da-DK">Fase proces afsluttet</Label>
      </Action>
      <Action name="DEMOTE">
        <Label culture="en-GB">Move to previous phase</Label>
        <Label culture="da-DK">Ryk til forrige fase</Label>
      </Action>
      <Action name="INIT">
        <Label culture="en-GB">Startet</Label>
        <Label culture="da-DK">Påbegyndt</Label>
      </Action>
      <Action name="NEARDUE">
        <Label culture="en-GB">Reminder date reached</Label>
        <Label culture="da-DK">Påmindelsesdato nået</Label>
```

```
        </Action>
        <Action name="PROMOTE">
          <Label culture="en-GB">Move to next phase</Label>
          <Label culture="da-DK">Ryk til næste fase</Label>
        </Action>
        <Action name="UPDATE">
          <Label culture="en-GB">Due date changed</Label>
          <Label culture="da-DK">Tidsfrist ændret</Label>
        </Action>
        <Action name="OVERDUE">
          <Label culture="en-GB">Schedule overdue</Label>
          <Label culture="da-DK">Tidsfrist overskredet</Label>
        </Action>
    </ActionLabels>
```

The context section

The context section of the WorkflowDefinition specifies the conditions that enable the process can be started. It specifies on which entity the process can by started (FILE or RECORD) and an additional filter in form of an OData query.

In the example above, the Distribution process can be started on a case which is not closed and which has no other distribution processes running.

The Tag filter is currently not used.

Deadline rules

5.1.2 Deadline principles

In WorkZone Processyou can configure phase process using various deadline rules. The deadline rules control if a phase is within the deadline, close to the deadline (near due) or if the deadline has passed (overdue).

You can also configure deadline based events. See Phase events.

You can see the state of the phase process indicated by colored icons in the WorkZone Process process overview:

- **Green**: Within deadline.
- **Yellow**: Near due.
- **Red**: Overdue.

Deadlines are calculated based on various units which are defined on the process itself.

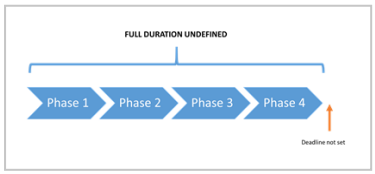
The following units are available:

Unit	Description
Days	Based on a 7 day week.
Work days	Based on a 5 day week.
Søgnedage	Danish term for work days minus public holidays as defined by the Danish Parliament.
Hours	Based on hours. You can configure a phases of a phase process using various rules by setting the **Duration** parameter on a phase.
Phase without duration and deadline	**Duration** is set to empty. The deadlines of the phase are dependent of the surrounding phases.
Phase with fixed duration	Duration is set to a fixed value, for example `10` for 10 of the calculation model units.
Phase with relative duration	Duration is set to a relative value (percentage), for example `50%`.
Phase deadline on specific date	The phase deadline is set by the user or by a defined rule.
Relative phase deadline from process start	Duration is set to for example `+2`, meaning two units from the start of the phase process.
Relative phase deadline from process deadline	Duration is set to for example `-2`, meaning two days from the phase process deadline.

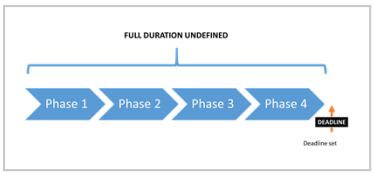
You must assign a deadline to the phase process.

Example: See examples of phase processes configured with deadline rules below.

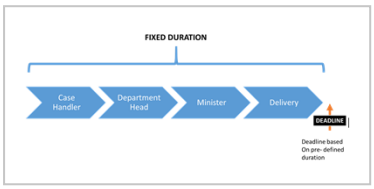
No deadline



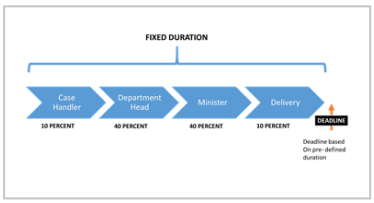
Deadline defined



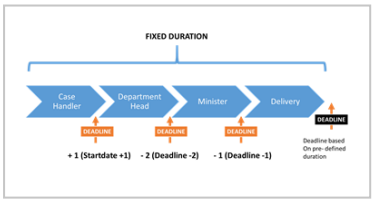
Fixed duration



Phases distributed by percent



Phases distributed relative to start time and deadline



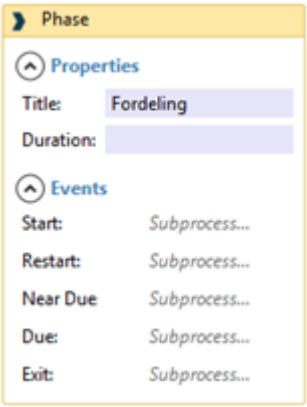
Phase events

Every phase has multiple possible events. You can configure a phase to activate one or more processes.

In the event properties, configure the event by dragging one or more processes to the event.

Phase	Event is triggered by:

Start	Promoting to the phase.
Exit	Promoting from the phase.
Restart	Demoting to the phase.
Near due	The phase has reached near due.
Due	The phase is overdue (the deadline has passed).



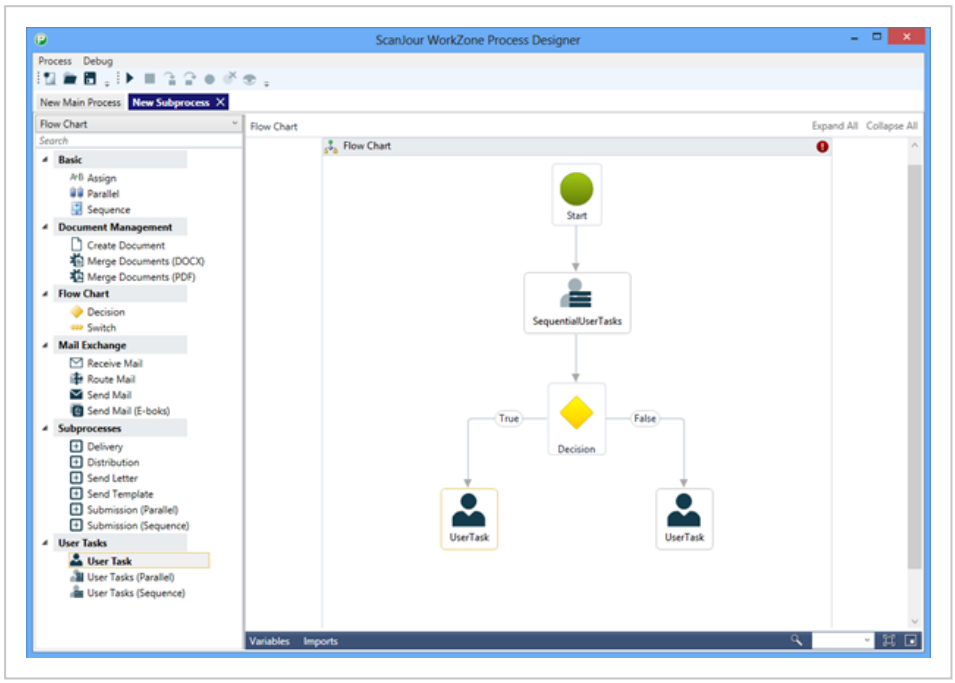
5.2 Configure sub processes

5.2.1 Workflow process modelling

You can model workflows using the WorkZone Process Designer. This tool is a standard process modeling tool consisting of a toolbox with activities and flow control elements and a canvas for process modeling.

Create a new process

1. Click **New proces** in the toolbox ribbon.



2. Drag activities and flow control elements from the toolbox ribbon to the process canvas.
3. Connect the elements using arrows. Arrows define the flow of the process.
4. Click on an activity to configure its properties.
5. Save the modeled process as XML.
6. Pack modeled processes together with their forms as a complete process package (for more information on packages, see Configure packages).

Standard activities

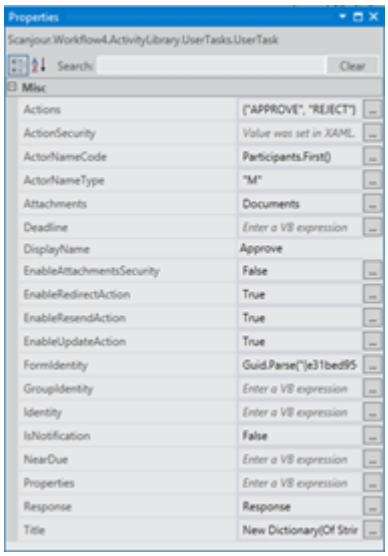
The WorkZone ProcessDesigner has a variety of standard activities that you can use for modeling processes. The activity package contains, among others, the activities listed below. For the full list, refer to the API documentation .

> **Note:** This list is dynamic and will be updated. You can make a request for additional generic activities to the product team.

Icon	Activity	Element	Description
	User task	User tasks	Initiates a smart task for user interaction.
	User tasks (Parallel)	User tasks	A series of parallel smart tasks with interaction from multiple users at the same time. Typically used for processes of the type Hearing.
	User tasks (Sequential)	User tasks	Sequentially ordered smart tasks with user interaction. Typically used for processes of the types Distribution and Approval.
	TaskInfo	User tasks	Writes a log entry which is visible in the Processes overview. Used for writing states for end users. E.g. used in Smart Post to constantly update delivery status in the overview.
	Create PDF	Document	PDF activity which can merge multiple documents into one PDF, adding bookmarks and watermark to the final PDF. (Currently used only in Smart Post. Will be made available in the standard product).
	Merge document	Document	Document activity merging context relevant fields into a DOCX document. Compatible with merge fields in WorkZone for Office. (Currently used only in Smart Post. Will be made available in the standard product).
	Load Register Object	Data	Loads a WorkZone entity to be used in the process.
	Update Register Object	Data	Updates a workzone entity.
	Insert Register Object	Data	Creates a WorkZone entity.
	Start process	Process	Starts a new process.
	Create Process History	Sub Processes	Creates a history document on the case. The document will contain meta data on the process execution.
	Send to Smart Post	Smart Post	Sends to Smart Post which routes Digital Post messages to eBoks or Straalfors Connect. (Requires

an extra license for Smart Post).

Example: Configuration of properties on a user task.



6. Workflow development

6.1 Activities

Building custom activities

Code Activities

If new activities are needed these may be developed as code activities based on the activity classes provided in Windows Workflow Foundation.

Normally, activities are subclassed from `CodeActivity` or `CodeActivity<T>`. Both activites do the main work in an overridden function:

```
protected override void Execute(CodeActivityContext context)
```

If the activity needs to access information from the WorkZone Process datamodel it is done through calls to the OData service provided by WorkZone Content Server 2014.

The WorkZone Process Workflow Host exposes OData through a `WorkflowHostExtension` which makes the job easier for the activity developer.

To get an ODataService, use the following code:

```
ODataService ctx = context.GetExtension<WorkflowHostExtension>().ODataContext
();
```

The ODataService is per default impersonated to the calling user. If, for some reason, you want to access registers and avoid the normal access code protection, you can get an ODataService which is not impersonated but runs as the user which is used by the workflow host:

```
ODataService ctx = context.GetExtension<WorkflowHostExtension>().ODataContext
(false);
```

Phase activities

In WorkZone Process 2014 R2 HF01 you have the following options for configuring phase activities.

Configuring phase names in multiple languages

Configuring Phase Start Time and End Time logic

- Duration in Days
- Relative Deadlines

Configuring phase events

You can configure phase events on:

- Start
- Exit
- Restart
- Near Due
- Due

Configuring several sub processes for each event. The events will be executed in sequence of configuration.

6.2 Testing workflows

Intro...

Mention ETW Tracking for testing

Test	Description
Host Test	
Activity Test	
Test Utils	

Workflow activity logging

Tracking participants

The Workflow Host is equipped with three different tracking participants which log activities in different places.

- `WorkflowLogTrackingParticipant` logs entries in `wzp_workflow_log`.
- `EWTTrackingParticipant` logs events which can be monitored in the event viewer, if it is enabled.
- `SjDebugTrackingParticipant` logs entries in sjDebug.

The tracking participant can be enabled/disabled in the `web config` file in the `Scanjour.Workflow4.Host.Settings` section. Only the first tracking participant is enabled by default. The two other participants are for troubleshooting.

```
<Scanjour.Workflow4.Host.Settings>
<setting name="EnableSjDebugTrackingParticipant" serializeAs="String">
<value>False</value>
</setting>
<setting name="EnableLogTrackingParticipant" serializeAs="String">
<value>True</value>
</setting>
<setting name="EnableEtwTrackingParticipant" serializeAs="String">
<value>False</value>
</setting>
</Scanjour.Workflow4.Host.Settings>
```

Tracking can be disabled for individual workflows for various reasons (for example, workflows never causing any problems or very "noisy" workflow). This is done in the same section of the `web.config` file. In the example below, all logging of workflows whose typename begins with `TestPackage.` and `Test` has been disabled.

```
<Scanjour.Workflow4.Host.Settings>
<setting name="DisableLoggingInWorkflows" serializeAs="Xml">
<value>
<ArrayOfString xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<string>TestPackage.</string>
<string>Test.</string>
</ArrayOfString>
</value>
</setting>
</Scanjour.Workflow4.Host.Settings>
```

To prevent the `wzp_workflow_log` from growing infinitely, an oracle batch job is installed that will delete log entries for completed workflows after 3 days – only leaving terminated and faulted workflows in the table. By default, everything possible is being logged. However, you can tune the logging on package level and on workflow level by defining a tracking profile to the package in the `PackageDefinition` section or the `Workflowdefinition`section.

The tracking profile is an XML description which defines what must be logged:

```
<PackageDefinition>
<Name>Basis</Name>
<Version>3.1.0.0</Version>
<Description>Basic submission</Description>
<TrackingProfile>Workflows\Basic.xml</TrackingProfile>
</PackageDefinition>
<WorkflowDefinition>
<Version>3.1.0.0</Version>
<XamlFile>Workflows\Submission.xaml</XamlFile>
<FormGuid>{709f3330-9190-4cc9-a7d5-0b30edef0e6e}</FormGuid>
```

```
    <AccessCode></AccessCode>

    <Standard>J</Standard>

    <TrackingProfile>Workflows\Sparse.xml</TrackingProfile>

    <WorkflowDefinition>
```

Tracking Profiles

A tracking profile controls the 7 different tracking records which are supported:

WorkflowInstanceRecords

Reports changes in the workflow instance state, for example: **Started**, **Idle**, **Unloaded**, **Resumed**.

ActivityScheduledRecords

Reports about every activity being scheduled for execution in the workflow. You may need to limit the reporting of scheduled activities because complex workflows may produce a lot of entries.

ActivityStateRecords

Reports the state of an executing activity, including the arguments and variables value in the activity. These records may be very useful for understanding what is going on in a workflow. However, you may need to limit which activities report `ActivityStateRecords`.

BookmarkResumptionRecords

Reports the bookmarks resumed on the workflow. These are important to understand when and why a workflow is resumed.

CancelRequestedRecords

Informs about activities being canceled, for example, a delay activity being canceled.

FaultPropagationQueries

Reports about unexpected errors in activities and the call stack in the workflow. This is important when you investigate the nature of an error.

CustomTrackingRecords

Reports information the developer of the activities has deemed important in order to understand what the activity is doing and why. This is always the result of a careful consideration from the developer of the activity. The following tracking profile is a tracing profile that tracks everything in a workflow. The `*` indicates that everything must be tracked.

```
<?xml version="1.0" encoding="utf-8"?>
<tracking>
<profiles>
<trackingProfile name="Basic">
<workflow activityDefinitionId="*">
<workflowInstanceQueries>
<workflowInstanceQuery>
<states>
<state name="*"/>
</states>
</workflowInstanceQuery>
</workflowInstanceQueries>
<activityScheduledQueries>
<activityScheduledQuery activityName="*" childActivityName="*" />
</activityScheduledQueries>
<activityStateQueries>
<activityStateQuery activityName="*">
<states>
```

```
<state name="*"/>
</states>
<arguments>
<argument name="*"/>
</arguments>
<variables>
<variable name="*"/>
</variables>
</activityStateQuery>
</activityStateQueries>
<bookmarkResumptionQueries>
<bookmarkResumptionQuery name="*" />
</bookmarkResumptionQueries>
<cancelRequestedQueries>
<cancelRequestedQuery activityName="*" childActivityName="*" />
</cancelRequestedQueries>
<faultPropagationQueries>
<faultPropagationQuery faultSourceActivityName="*"
faultHandlerActivityName="*" />
</faultPropagationQueries>
<customTrackingQueries>
<customTrackingQuery name="*" activityName="*" />
</customTrackingQueries>
</workflow>
</trackingProfile>
</profiles>
</tracking>
```

Below is a sample tracking profile which only tracks certain activities in a submission workflow, which disables all `ActivitySchedulesRecords` and limits the activities reporting `ActivitStateRecords`.

```
<?xml version="1.0" encoding="utf-8"?>
<tracking>
<profiles>
<trackingProfile name="Basic">
<workflow activityDefinitionId="*">
<workflowInstanceQueries>
<workflowInstanceQuery>
<states>
<state name="Started"/>
<state name="Unloaded"/>
<state name="Resumed"/>
<state name="Completed"/>
<state name="Faulted"/>
</states>
</workflowInstanceQuery>
</workflowInstanceQueries>
<activityScheduledQueries>
</activityScheduledQueries>
<activityStateQueries>
<activityStateQuery activityName="SequentialUserTask">
<states>
<state name="Executing"/>
</states>
<arguments>
<argument name="*"/>
</arguments>
```

```
<variables>
<variable name="*"/>
</variables>
</activityStateQuery>
<activityStateQuery activityName="ParallelUserTask">
<states>
<state name="Executing"/>
</states>
<arguments>
<argument name="*"/>
</arguments>
<variables>
<variable name="*"/>
</variables>
</activityStateQuery>
<activityStateQuery activityName="UserTask">
<states>
<state name="Executing"/>
</states>
<arguments>
<argument name="*"/>
</arguments>
<variables>
<variable name="*"/>
</variables>
</activityStateQuery>
<activityStateQuery activityName="SimpleUserTask">
<states>
<state name="Executing"/>
```

```
</states>
<arguments>
<argument name="*"/>
</arguments>
<variables>
<variable name="*"/>
</variables>
</activityStateQuery>
<activityStateQuery activityName="UserTaskTimerActivity">
<states>
<state name="Executing"/>
</states>
<arguments>
<argument name="*"/>
</arguments>
<variables>
<variable name="*"/>
</variables>
</activityStateQuery>
<activityStateQuery activityName="ValidateUserTaskActivity">
<states>
<state name="Executing"/>
</states>
<arguments>
<argument name="*"/>
</arguments>
<variables>
<variable name="*"/>
</variables>
```

```
    </activityStateQuery>

    </activityStateQueries>

    <bookmarkResumptionQueries>

    <bookmarkResumptionQuery name="*" />

    </bookmarkResumptionQueries>

    <cancelRequestedQueries>

    </cancelRequestedQueries>

    <faultPropagationQueries>

    <faultPropagationQuery faultSourceActivityName="*"
    faultHandlerActivityName="*" />

    </faultPropagationQueries>

    <customTrackingQueries>

    <customTrackingQuery name="*" activityName="*" />

    </customTrackingQueries>

    </workflow>

    </trackingProfile>

    </profiles>

    </tracking>
```

Defining a tracking profile suiting a specific workflow requires knowledge about what the workflow does and where interesting information about the execution is located. The syntax is easy to learn, and it is described both in books and on the internet. A lecture on how to specify the various queries is not in the scope of this description. More information is available at http://msdn.microsoft.com/en-us/library/ee513989(v=vs.110).aspx.

Other tracking records

A number of other tracking records appear in the log, and they cannot be controlled from a tracking profile. The reason is that the records inform about events which must not be hidden from the responsible person.

HostTrackingrecords

Informs about workflows that have been terminated by a user, or which end because of errors inside the workflow.

WorkflowInstanceTerminateRecords

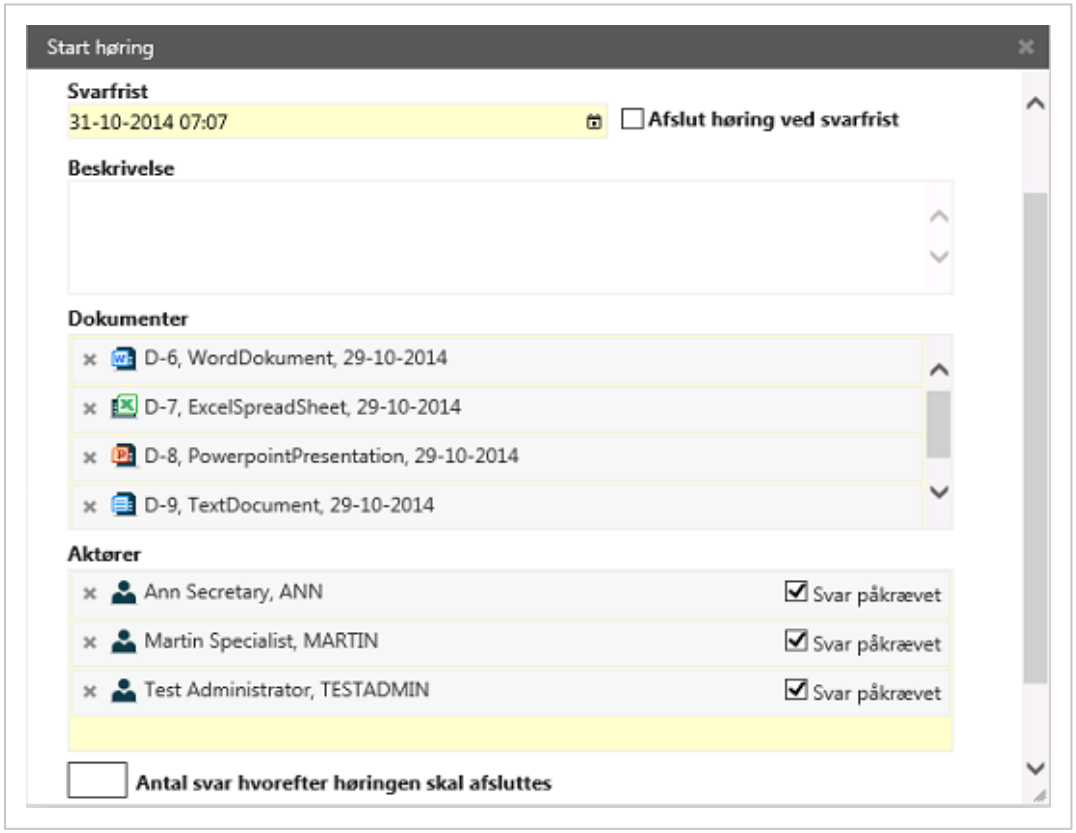
Informs about the reason why a workflow is terminated.

WorkflowInstanceUnhandledexceptionRecord

Informs about unhandled exceptions in the workflow.

6.3 Document flow

Documents participating in a process, for example, a hearing or a submission, are selected in the InitForm in the document control:



When the InitForm starts the process, the documents are handed to the process in the form of a `Record[]` of type `Scanjour.Process.Client.Lite.Record[]` in an `InArgument` called `Documents`:

Name	Direction	Argument type	Default value
Deadline	In	DateTime	DateTime.Now + TimeSpan.Frc
Documents	In	Record[]	*Enter a VB expression*
MandatoryActors	In	Contact[]	*Enter a VB expression*
OptionalActors	In	Contact[]	*Enter a VB expression*
ExitOnDeadline	In	Boolean	False
ExitOnThreshold	In	Int32	0

The workflow passes these arguments to the `ParallelUserTask` or `SequentialUserTask` activities in the activity properties.

The most important properties are **Attachments** and **Properties**:

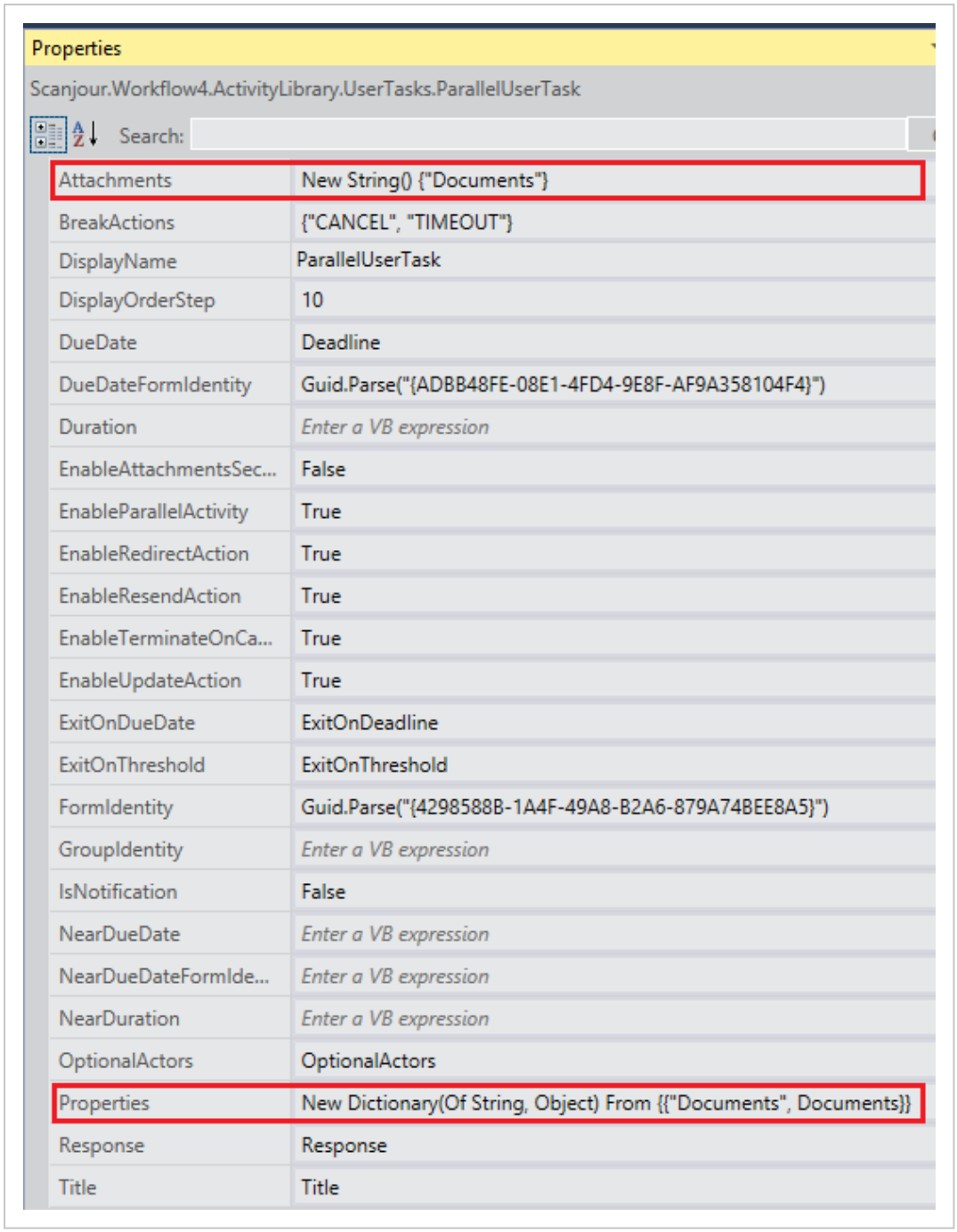
- **Attachments**

  Indicates which documents should be attached to the mail. The **Attachments** property is a `string[]` which stores the names of the properties where the documents are listed.

  Example: In the simple case where there are only documents in one property, the **Attachments** property is hardcoded with `New String() {"Documents"} (VB syntax)`.

- **Properties**

  Holds the `Record[]` in a dictionary that the user task converts to a `UserTaskPropertyCollection`. The **Properties** property is a dictionary with one entry, which is the documents array: `New Dictionary(Of String, Object) From {"Documents", Documents}`.



When using this way of specifying the documents, you will also be able to create more sophisticated processes that have more document controls in the InitForm.

Example: If you have documents and references where you want to attach only the documents to the mail, the references would still be the documents, but the **Properties** would be: `New Dictionary(Of String, Object) From {{"Documents", Documents},{"References", References}}`.

When a user task is created in the database table `wzp_user_task`, the properties are stored together with the user task. The user task goes through all properties defined in the properties collection and completes the information in the record structure. It furthermore adds other properties that are needed when the user task is shown in Outlook. The database contains the following properties:


```
{
"Documents":
{"key":"Documents","type":"Scanjour.OData.Client.Lite.WorkZone.Record[]",
"value":[{"TypeName":"Som.Record","MediaResource":null,"ID":"6",
"Actions":[],
"Properties":[
{"key":"ID","type":"System.String","value":"6"},
{"key":"FileKey_Value","type":"System.String","value":"81"},
{"key":"State_Value","type":"System.String","value":"UÅ"},
{"key":"RecordType_Value","type":"System.String","value":"DOK"},
{"key":"DocumentType_
Value","type":"System.String","value":"Word.Document.12"},
{"key":"Title","type":"System.String","value":"WordDokument"},
{"key":"Summary","type":"System.String","value":"D-6, WordDokument, 29-10-
2014"}],
"SubEntries":[],
"Feeds":[]},
{"TypeName":"Som.Record","MediaResource":null,"ID":"7",
"Actions":[],
"Properties":[
{"key":"ID","type":"System.String","value":"7"},
{"key":"FileKey_Value","type":"System.String","value":"81"},
{"key":"State_Value","type":"System.String","value":"UÅ"},
{"key":"RecordType_Value","type":"System.String","value":"DOK"},
```

```
{"key":"DocumentType_
Value","type":"System.String","value":"Excel.Sheet.12"},
{"key":"Title","type":"System.String","value":"ExcelSpreadSheet"},
{"key":"Summary","type":"System.String","value":"D-7, ExcelSpreadSheet, 29-
10-2014"}],
"SubEntries":[],
"Feeds":[]},
{"TypeName":"Som.Record","MediaResource":null,"ID":"8",
"Actions":[],
"Properties":[
{"key":"ID","type":"System.String","value":"8"},
{"key":"FileKey_Value","type":"System.String","value":"81"},
{"key":"State_Value","type":"System.String","value":"UÅ"},
{"key":"RecordType_Value","type":"System.String","value":"DOK"},
{"key":"DocumentType_
Value","type":"System.String","value":"PowerPoint.Show.12"},
{"key":"Title","type":"System.String","value":"PowerpointPresentation"},
{"key":"Summary","type":"System.String","value":"D-8, Powerpoint, 29-10-
2014"}],
"SubEntries":[],
"Feeds":[]},
{"TypeName":"Som.Record","MediaResource":null,"ID":"9",
"Actions":[],
"Properties":[
{"key":"ID","type":"System.String","value":"9"},
{"key":"FileKey_Value","type":"System.String","value":"81"},
{"key":"State_Value","type":"System.String","value":"UÅ"},
{"key":"RecordType_Value","type":"System.String","value":"DOK"},
{"key":"DocumentType_Value","type":"System.String","value":"txtfile"},
```

```
{"key":"Title","type":"System.String","value":"TextDocument"},
{"key":"Summary","type":"System.String","value":"D-9, TextDocument, 29-10-
2014"}],
"SubEntries":[],
"Feeds":[]}
]
},
"FileNo":{
"key":"FileNo","type":"System.String",
"value":"14-10/2"
},
"Officer":{
"key":"Officer","type":"System.String",
"value":"TESTADMIN"
},
"OfficerName":{
"key":"OfficerName","type":"System.String",
"value":"Test Administrator, TESTADMIN"
},
"Register":{
"key":"Register","type":"System.String",
"value":"FILE"
},
"RegisterKey":{
"key":"RegisterKey","type":"System.String",
"value":"81"
},
"InstanceId":{
"key":"InstanceId","type":"System.String",
```

```
"value":"a8a7a50c-d10b-4871-8890-e290ac525659"
},
"TaskId":{
"key":"TaskId","type":"System.String",
"value":"22"
}
}
```

When a user task mail is rendered, a number of `UserTaskResponse` are included in the mail. Here the property information is part of the `UserTaskResponse`, and the properties are updated to honor any restrictions that access codes may impose on the user who receives the mail.

A user task response looks as shown below:

(For readability, the type information shown below is removed).

```
{"$type":"Scanjour.Workflow4.Base.UserTaskResponse,
Scanjour.Workflow4.Base, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=null",
<responsetemplates>
<responsetemplate name="Actions">
"Action":null,
"Comment":null,
"Properties":
"Documents":
"key":"Documents","type":"Scanjour.OData.Client.Lite.WorkZone.Record[]",
"value":[
"ID":"6",
"Actions":[],
"Properties":[
"key":"ID","type":"System.String","value":"6"},
"key":"FileKey_Value","type":"System.String","value":"81"},
```

```
"key":"State_Value","type":"System.String","value":"UÅ"},
"key":"RecordType_Value","type":"System.String","value":"DOK"},
"key":"DocumentType_
Value","type":"System.String","value":"Word.Document.12"},
"key":"Title","type":"System.String","value":"WordDokument"},
"key":"Summary","type":"System.String","value":"D-6, WordDokument, 29-10-
2014"}],
"SubEntries":[],
"Feeds":[]},
"ID":"7",
"Actions":[],
"Properties":[
"key":"ID","type":"System.String","value":"7"},
"key":"FileKey_Value","type":"System.String","value":"81"},
"key":"State_Value","type":"System.String","value":"UÅ"},
"key":"RecordType_Value","type":"System.String","value":"DOK"},
"key":"DocumentType_Value","type":"System.String","value":"Excel.Sheet.12"},
"key":"Title","type":"System.String","value":"ExcelSpreadSheet"},
"key":"Summary","type":"System.String","value":"D-7, ExcelSpreadSheet, 29-10-
2014"}],
"SubEntries":[],
"Feeds":[]},
"ID":"8",
"Actions":[],
"Properties":[
"key":"ID","type":"System.String","value":"8"},
"key":"FileKey_Value","type":"System.String","value":"81"},
"key":"State_Value","type":"System.String","value":"UÅ"},
"key":"RecordType_Value","type":"System.String","value":"DOK"},
```

```
"key":"DocumentType_
Value","type":"System.String","value":"PowerPoint.Show.12"},
"key":"Title","type":"System.String","value":"PowerpointPresentation"},
"key":"Summary","type":"System.String","value":"D-8,
PowerpointPresentation, 29-10-2014"}],
"SubEntries":[],
"Feeds":[]},
"ID":"9",
"Actions":[],
"Properties":[
"key":"ID","type":"System.String","value":"9"},
"key":"FileKey_Value","type":"System.String","value":"81"},
"key":"State_Value","type":"System.String","value":"UÅ"},
"key":"RecordType_Value","type":"System.String","value":"DOK"},
"key":"DocumentType_Value","type":"System.String","value":"txtfile"},
"key":"Title","type":"System.String","value":"TextDocument"},
"key":"Summary","type":"System.String","value":"D-9, TextDocument, 29-10-
2014"}],
"SubEntries":[],
"Feeds":[]}]},
"FileNo":
"key":"FileNo","type":"System.String","value":"14-10/2"},
"Officer":
"key":"Officer","type":"System.String","value":"TESTADMIN"},
"OfficerName":
"key":"OfficerName","type":"System.String","value":"Test Administrator,
TESTADMIN"},
"Register":
"key":"Register","type":"System.String","value":"FILE"},
```

```
"RegisterKey":
"key":"RegisterKey","type":"System.String","value":"81"},
"InstanceId":
"key":"InstanceId","type":"System.String","value":"a8a7a50c-d10b-4871-8890-
e290ac525659"},
"TaskId":
"key":"TaskId","type":"System.String","value":"21"}},
"Answers":null,
"Identity":null}
</responsetemplate>
</responsetemplates>
```

Information about the documents is the information on the rendering time of the mail. In order for the mail to obtain actual values for the documents, the user task has maintained the list of documents in a child table to the `wzp_user_task` table named `wzp_user_task_attachments`.

This table has the following information:

- `task_id`: The `task_id` of the user task.
- `record_id`: The `record_key` of the document.
- `property_name`: The property name that the record is part of.
- `priority`: The order in the property.
- `attach`: Is set to **True** if the record will be attached to the mail.

The user task mail can obtain information about the current list of documents and their titles from OData through the register `wzp_user_task_attachment` because the record table is an extension table in this register:

`http://sjunittest/OData/WzpUserTaskInserts?&$filter=TaskId eq'<task_id>'` and `PropertyName eq 'Documents'&$expand=Records&$select=RecordId,Records/Title&$orderby=Priority`

This allows the mail to show an up-to-date list of documents in the mail while still honoring the access code protection of the documents.

7. The forms concept

7.1 Upgrade selector controls from 2016 to 2016 R2

In WorkZone Process 2016 R2 changes have been implemented for methods to select options in forms. Follow the guidelines below to upgrade forms:

1. In forms that are created with a JavaScript controller, go to the line `angular.module('wzp', ['….` and remove `'ui.select2'`,`'ui.selector'`, `'ui.forward'`,`'ui.rollbackselector'`. Then, if it does not exist already, add the appropriate wzp controls.

2. In all HTML forms, replace old controls with new ones as follows:

Old element	New element	Comments
ui:selector	wzp:selector	Use this for single selectors such as multiple:false, or use it if you don't need the possibility to edit items.
ui:selector	wzp-multi-selector	Use this for editable multiple selectors.
ui:filter-selector	wzp:filter-selector	
ui:sequence-mask-selector	wzp:sequence-mask-selector	
ui:forward	wzp:forward	
usertask-rollbackselector-panel-helper	wzp-usertask-rollbackselector-panel-helper	Use this for HTML attributes
ui:rollbackselector-panel	wzp:rollbackselector-panel	
ui:rollbackselector	wzp:rollbackselector-panel	
ui-checkboxselector	wzp-checkboxselector	Use this for HTML attributes in the ui:selector element

3. In all HTML forms replace the old ng-controller with new ones:

Old controller	New controller
SelectODATACtrl	wzpSelectODATACtrl
SelectODATAWithFilterCtrl	wzpSelectODATAWithFilterCtrl
InitFormFilterForSelectorCtrl	wzpInitFormFilterForSelectorCtrl
InitFormSequenceMaskForSelectorCtrl	wzpInitFormSequenceMaskForSelectorCtrl
SelectUserTaskDocumentsCtrl	wzpSelectUserTaskDocumentsCtrl
SelectUserTaskActorsCtrl	wzpSelectUserTaskActorsCtrl
CustomEditDocumentController	wzpCustomEditDocumentController
SmartTaskFilterForSelectorCtrl	zpSmartTaskFilterForSelectorCtrl

4. Remove the attribute `wzp-ad-selector-change-label` and its value.
5. Remove change-title-variable attribute and its value.
6. For the new `wzp:filter-selector` control, change the attribute `ng-controller-name` to `ng-controller`.
7. For instances of `wzp-multi-selector` or `wzp:rollbackselector` that contain instances of `wzp:filter-selector` or `wzp:sequence-mask-selector`, change the class attribute to `class="newline wzp-task-documentlist wzp-select-with-filter"`.
8. Separate each `wzp-multi-selector` by `<div class="wzp-task-editdocument">`.
9. Set `always-editable="true"` for the instance of `wzp-multi-selector` or `wzp:rollbackselector` that you want only in editable mode. An example would be the use of these selectors in Init forms.

7.2 Forms

WorkZone Process forms are based on HTML and JavaScript, which are well-known domains allowing a large degree of flexibility.

The basic concept is that a central form can be displayed on all clients using the WorkZone container interface, which is by default supported by WorkZone for Office and WorkZone Client.

As a minimum, a form consists of a view and a controller. The view is the visual part of the form displayed in WorkZone Process, which is configured in HTML using a set of basis controls. The controller contains the validation and business logic of the form, which is implemented in JavaScript.

> **Prerequisite:** Modeling of forms requires basic understanding of HTML and Javascript as well as the JavaScript libraries Angular, JQuery and WorkZone Process Basis library.

Form view

You can configure views using simple HTML elements defined in Basis.js.

Example: Submission Basis is a standard part of WorkZone Process.

```
<!doctype html>
<html lang="en">
<head>
    <title>WorkZone Process</title>
    <meta http-equiv="X-UA-Compatible" content="IE=edge;chrome=1" />
    <meta http-equiv="x-dns-prefetch-control" content="off" />
    <meta http-equiv="content-type" content="text/html; charset=utf-8"
/>
    <meta name="viewport" content="initial-scale=1.0, minimum-
scale=1.0, maximum-scale=1.0, user-scalable=no" />

    <link rel="stylesheet" href="Basis/css/app.css" />
    <script src="Basis/js/jquery.js"></script>
    <script src="Basis/js/angular.js"></script>
    <script src="Basis/js/basic.js"></script>

    <script localizationfile="" src="Basis/js/{0}.js"></script>
    <script localizationfile="" src="Basis/js/init.Submission.
```

```
{0}.js"></script>
</head>
<body ui-Intl="init.Submission.">
    <div class="wzp-page" ng-cloak>
        <form autocomplete="off" name="submissionForm" ui-startprocess
ng-controller="FormCtr" novalidate>
            <ui:title labelGroup="INITSUBMISSION"
label="TITLE"></ui:title>
            <ui:text ng-model="dataSource.Title"
labelgroup="INITSUBMISSION" label="PROCESSTITLE" max-length="256"
required class="newline"></ui:text>
            <ui:datetime name="deadlineControl"
labelgroup="INITSUBMISSION" ng-model="dataSource.Deadline"
label="DEADLINE" class="newline"></ui:datetime>
            <ui:text ng-model="dataSource.Description"
labelgroup="INITSUBMISSION" label="DESCRIPTION" rows="4" max-
length="3999" class="newline"></ui:text>

            <div class="wzp-task-editdocument">
                <wzp-multi-selector ng-controller="wzpSelectODATACtrl"
                                    ng-model="dataSource.Documents"
                                    pre-selected-
values="PreSelectDocuments"
                                    labelgroup="CONTROL"
                                    label="DOCUMENTS"
                                    placeholder="SELECT_DOCUMENTS"
                                    options="{
                                        register: 'Records',
                                        filter: searchInCurrentCase(),
                                        orderby:documentMruFilter(),
                                        freetextfield:'Summary',
                                        openItem:{
                                            icon:'content',
                                            title:'metadata',
                                            actionRegister:'Record'
                                        }
                                    }"
                                    always-editable="true"
                                    class="newline wzp-task-documentlist"
                                    ng-
disabled="defaultValuesNotInitialized()">
                    <wzp-upload-document ng-
model="dataSource.Documents"></wzp-upload-document>
                </wzp-multi-selector>
            </div>
            <div class="wzp-task-editdocument">
                <wzp-multi-selector ng-controller="wzpSelectODATACtrl"
                                    ng-model="dataSource.Actors"
                                    pre-selected-
values="PreSelectParties"
```

```
                                    labelgroup="INITSUBMISSION"
                                    label="ACTORS"
                                    placeholder="SELECT_ACTORS"
                                    options="{  showSelected:true,
                                     register: 'WzpFileUserRights',
                                     text: ['ID', 'Summary','NameType_
Value','NameCode_Value'],
                                     filter: addContactFilter(),

expand:'NameKey,NameKey/AddressKey',
                                     freetextfield:'tolower(Summary)',
                                     openItem:
{icon:'metadata',title:'metadata' , actionRegister:'Contact'} ,
                                     orderby :'Summary',
                                     iconType:'contacts'}"
                                    always-editable="true"
                                    class="newline wzp-task-
documentlist wzp-select-with-filter"
                                    ng-
disabled="defaultValuesNotInitialized()"
                                    required>
                    <wzp:sequence-mask-selector-filter ng-
model="twoWayBindings.ActorSequenceMasks"
                                                        parent-ng-model-
variable="dataSource.Actors">
                    </wzp:sequence-mask-selector-filter>
                </wzp-multi-selector>
            </div>
            <div class="wzp-bottom">
                <ui:help link="#WZP_UserGuide/Start_basis_
submission.htm%3FTocPath%3DWorkZone%2520Process%2520Basis%2520Package%7
CBasis%2520submission%2520processes%7C_____3"></ui:help>
                <div class="wzp-buttonset">
                    <ui:button label="START" action="submit()" ng-
disabled="isNotValid() || !defaultValuesInitialized"></ui:button>
                    <ui:button label="CANCEL" action="cancel()" ng-
disabled="defaultValuesNotInitialized()"></ui:button>
                </div>
            </div>
        </form>
    </div>
</body>
</html>
```

WorkZone Process includes a variety of HTML elements which you can use for modeling views. This table displays some of the basic HTML elements used in the example above.

HTML element	Description
ui:title	The title of the form.
ui:text	A free text field where you for example can set the hight and number of characters of the control.
ui:datetime	Specifies a date and time control.
ui:select	A dynamic list element which you can use for listing for example the WorkZone entities Document, Case and Contact. You can configure the list with various properties, for example sorting, which can define specific filters for valid values. You can open the WorkZone entities directly from the list.
ui:help	Points to context sensitive help. For customized solutions this can point to a given URL.
ui:button	Executes an action on the form.

See a list of basic elements in the API documentation where properties are specified.

7.3 Init form

Actor sequences in smart task Init forms

Actor sequence can be used for selecting actors in the `InitForm\SmartTask` selector control for actors (or a similar register) with the new `wzp:sequence-mask-selector` control.

Follow these steps to configure this option.

1. Add the css class `wzp-select-with-filter` to `wzp:selector` or `wzp:rollbackselector`.
2. Add the `wzp:sequence-mask-selector` control inside the parent selector with the attributes described in the table below:
3.

Attribute name	Description	Example or comments
`ng-model`	A pointer to the source model property, which this	`twoWayBindings.ActorSequenceMasks`

	control is bound to. Should be unique.	
`ng-controller`	The name of the controller.	`"wzpInitFormSequenceMaskForSelectorCtrl"` for InitForm, `"wzpSmartTaskSequenceMaskForSelectorCtrl"` for SmartTask
`parent-ng-model-variable`	Should be equal to the `'ng-model'` attribute of the parent `wzp.selector` or `wzp.rollbackselector`, and contains `'.'`	`dataSource.Actors`

Example: A `wzp:selector` with a `wzp:sequence-mask-selector` control for an Init form:

```
<wzp:selector ng-controller="wzpSelectODATACtrl"
ng-model="dataSource.Actors"
labelgroup="SUBMISSIONFORM"
label="ACTORS"
placeholder="SELECT_ACTORS"
options="{
showSelected:true,
register: 'WzpFileUserRights',
text: ['ID', 'Summary','NameType_Value','NameCode_Value'],
filter: addcontactfilter(),
expand:'NameKey,NameKey/AddressKey',
```

```
freetextfield:'tolower(Summary)',
openItem:{icon:'metadata',title:'metadata' , actionRegister:'Contact'} ,
orderby :'Summary',
iconType:'contacts'}"
class="newline wzp-select-with-filter"
required
ng-disabled="formisdisabled">
<wzp:sequence-mask-selector
ng-controller="wzpInitFormSequenceMaskForSelectorCtrl"
ng-model="twoWayBindings.ActorSequenceMasks"
parent-ng-model-variable="dataSource.Actors"
class="newline">
</wzp:sequence-mask-selector>
</wzp:selector>
```

Example: A `wzp:rollbackselector` with a `wzp:sequence-mask-selector` control for a Smart Task:

```
<wzp:rollbackselector
ng-model="twoWayBindings.Actors"
ng-controller="CustomEditDocumentController"
default-data-context-name="ActiveActors"
item-convertor-name="converterFromOdataToSelectorForActors"
save-result-convertor-name="updateActorsFromSomethingConverter"
options="{showSelected:true,
openItem:{icon:'metadata',title:'metadata' , actionRegister:'Contact'} ,
iconType:'contacts',
datacontextName: 'ValidActors'
}"
readonly="noCapability('execute','online')"
labelgroup="CONTROL"
```

```
    label="ACTORS"
    placeholder="SELECT_PARTIES"
    hide-indicator="twoWayBindings.editActorControlIsHidden"
    class="newline wzp-task-documentlist body-column wzp-select-with-filter">
    <wzp:sequence-mask-selector
    ng-controller="wzpSmartTaskSequenceMaskForSelectorCtrl"
    ng-model="twoWayBindings.ActorSequenceMasks"
    ng-hide="noCapability('execute','online')"
    parent-ng-model-variable="twoWayBindings.Actors"
    class="newline">
    </wzp:sequence-mask-selector>
    </wzp:rollbackselector>
```

7.3.1 Init form container interface

1. Interface object representing the Init Form container will be known in JavaScript as window.wzp.container.
2. The Init form container object must have the following fields and methods:
    - string `baseUri`

      Base URI of web site containing services required by form, namely: OData and Process control services.
    - string `locale`

      Current locale of the client (e.g. `en-GB`, `da-DK`).
    - string `processDefinition`

      Global unique identifier of definition of process to be started.
    - object `context`

Description of the context where process to be started.

- string `register`

  name of the context entity register.

- string `key`

  identifier of the context entity within the register.

- string[] `tags`

  additional descriptors of the context.

- void `openItemContent` (string register, string systemKey, string title)

  Opens item`s content in an app/client responsible for showing the content. Where register is SOM register.

- void `openItemMetadata` (string register, string systemKey)

  Opens item`s metadata in an app/client responsible for showings the metadata. Where register is the SOM register.

- void close (bool success, string processid)

  Tells the container that init form should be closed, supplies overall result (success parameter) and identifier of newly started process.

- object `notifications`

  Service providing the unified interface for displaying and further handling of notifications.

  - void `info` (string message) tells user something important.
  - void `error` (string message) notifies user about error.
  - void `warning` (string message) warns user about something.
  - bool `confirm` (string message) asks user to make a binary decision.

- void `contentLoaded` (bool success)

  Informs container that form and its content was loaded successfully or not.

- bool `showTitle` (string message)

  Method that allow container to show form Title on the level of container presentation. Should return true , is container will show title, or return false , if showing title is responsibility of form itself. Message is form Title (already localized).

- bool `showHelp` (string url)

  Method that allows container to open help url on the level of container presentation. Url is localized. Should return true, is container want to show help itself, or return false, if showing help is responsibility of form itself.

- object `formats`

  Description of different formats. Allows container to customize visual presentation of data. If not defined corresponding formats will be taken from localization resources of basis package. Date\Time format should use JQuery Date\Time specification. (See below)

  - string `longdate`

    definition of Long Date format

  - string `shortdate`

    definition of Short Date format

  - int `timezone`

definition of client timezoneoffset value in minutes. TimezoneOffset should be calculated as UTC-localtime, in minutes.

- string `shorttime`

  definition of Short Time format

- string `longtime`

  definition of Long Time format

- string odataUri

  URI of oData service required by form. If not provided, default baseUri property is used as fallback.

  Example:

  `http(s)://[endpoint]/odata/`

- string processUri

  URI of Process service required by form. If not provided, default baseUri property is used as fallback.

  Example:

  `http(s)://[endpoint]/Process.Process.svc/`

- object AuthorizationHeader

  The AuthorizationHeader object must contain an Authorization property with the token for OAuth authentication.

  Example:

  ```
  AuthorizationHeader
  {
  Authorization: "Bearer
  ```

eyJhbGciOiJSUzI1NiIsImtpZCI6Ijg3NTlhMmViNDEwZjI1NTE1ODMw
ZWQxZWU2MDhlZmY2IiwidHlwIjoiSldUIn0.eyJuYmYiOjE1OTM2MDM1
NTUsImV4cCI6MTU5MzYwNzE1NSwiaXNzIjoiaHR0cDovL2RiMDEvb2F1
dGgyIiwi"
}

7.4 Edit form

Edit forms allow you to edit properties of an existing process.

Each package should have at least one edit form definition, a default edit form HTML and a controller (js) file.

Definition:

```
<FormDefinition>
    <FormGuid>{ EditForm Guid}</FormGuid>
    <Name>Edit.Default</Name>
    <Default>J</Default>
    <ContentType>TEXT/HTML</ContentType>
    <ContentFile>ui\edit.Default.html</ContentFile>
    <ControllerFile>ui\edit.Default.js</ControllerFile>
  </FormDefinition>
```

The same GUID should be defined for each workflow definition:

```
<WorkflowDefinition>
  <Version>6.0.0.0</Version>
  <XamlFile>Workflows\Submission.xaml</XamlFile>
  <EditFormGuid>{ EditForm Guid }</EditFormGuid>
```

Default Edit form html file

The edit form allows you to modify title, description, deadline (DueDate), and priority of an existing process.

```
<!doctype html>
<html lang="en">
<head>
  <title>WorkZone Process Edit Page</title>
  <meta http-equiv="X-UA-Compatible" content="IE=edge;chrome=1" />
  <meta http-equiv="x-dns-prefetch-control" content="off" />
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="initial-scale=1.0, minimum-scale=1.0,
  maximum-scale=1.0, user-scalable=no" />
  <link rel="stylesheet" href="Basis_6.0.0.0/css/app_6.0.0.0.css" />
  <script src="Basis_6.0.0.0/js/jquery_6.0.0.0.js"></script>
  <script src="Basis_6.0.0.0/js/angular_6.0.0.0.js"></script>
  <script src="Basis_6.0.0.0/js/basic_6.0.0.0.js"></script>
  <script localizationfile="" src="Basis_6.0.0.0/js/{0}_6.0.0.0.js"></script>
  <script localizationfile="" src="Basis_6.0.0.0/js/init.Submission.{0}_
  6.0.0.0.js"></script>
</head>
<body ui-intl="init.Submission.">
  <div class="wzp-page" ng-cloak>
     <form autocomplete="off" name="editForm" novalidate wzp-edit-process >
       <ui:title labelgroup="SUBMISSIONFORM" label="EDITFORM"></ui:title>
       <ui:text ng-model="dataSource.Title" labelgroup="HEARINGFORM"
       label="PROCESSTITLE" max-length="256" required class="newline"
       ></ui:text>
       <div class="twoRowBlock">
```

```
            <ui:datetime name="deadlineControl"
              labelgroup="EDITFORM"
              ng-model="dataSource.DueDate"
              label="DEADLINE"
              class="twoRowDate">
            </ui:datetime>
            <wzp:selector ng-controller="wzpSelectODATACtrl"
              ng-model="dataSource.Priority"
              labelgroup="CONTROL"
              label="PRIORITY"
              options="{
              multiple:false,
              allowClear:false,
              minimumInputLength:0,
              minimumResultsForSearch:-1,
              query: PriorityQuery }"
              placeholder="SELECT_PRIORITY"
              class="twoRowSelector">
            </wzp:selector>
          </div>
          <ui:text ng-model="dataSource.Description"
          labelgroup="SUBMISSIONFORM" label="DESCRIPTION" rows="4" max-
          length="3999" class="newline" ></ui:text>
          <div class="wzp-bottom">
            <ui:help link="#Prcs_overview/Use_the_process_overview.htm#Edit_
            process_
            details%3FTocPath%3DThe%2520Processes%2520overview%7CUse%2520the%
            2520Processes%2520overview%7C_____5"></ui:help>
            <div class="wzp-buttonset">
```

```
            <ui:button label="SAVE" action="submit()" ng-disabled="isNotValid
            ()"></ui:button>
            <ui:button label="CANCEL" action="cancel()"></ui:button>
          </div>
        </div>
      </form>
    </div>
  </body>
  </html>
```

> **Note:** The main angular directive for an edit form is **wzp-edit-process**. For more information, see API Forms.

Default Edit form controller (JS) file

```
  "use strict";
  (function (window, angular, undefined) {
  angular.module('wzp', ['ngResource', 'ngUtilities', 'localize',
  'common.services', 'wzp.filters', 'settings', 'ui.helpers', 'ui.title',
  'ui.text', 'ui.datetime', 'ui.sortable', 'ui.button', 'ui.help', 'wzp.edit-
  process','wzp.controls']);
    window.init = function (wzpContainer) {
      var editFormContainer = new window.EditFormContainer(wzpContainer);
      angular.module('wzp').constant('wzpContainer', editFormContainer);
      angular.bootstrap(window.document, ['wzp'])
    }
```

```
  })(window, window.angular);
```

The code for the EditFormContainer class is defined in the basis.js file in the Basis package.

7.4.1 Edit form container

The Edit form container API corresponds to the API of the Init form container except for the following:

- context.key value equal WzpWorkflowInstances . ID value

In the WorkZone Process **Overview** , only the Edit form container is available.

See Init form container interface.

7.4.2 Smart tasks container

The Smart tasks container combine API elements from the Init forms API and the Smart task container API extended with the following fields and methods:

- the context.key value equals the Smart Task.ID value
- the processDefinition value should be empty
- bool useSmartContainer should equal true
- should contain the capabilities property from the Smart task container API
- void reload ()
  Method that allows a container reload iframe with a Smart task form on form request
- void blockUI () and void unblockUI()
  Methods that allow a form to ask for a Container block\unblock frame.

See Init form container interface.

7.4.3 Smart task container interface

User task container is an extended init form container. It means that it should implement the same interface as init form container (except for the context field) plus:

- Object data

  Contains routines for getting data from the data context. Encapsulates online / offline data management from the form.

  - json `get`(string contextName, string filter)

    Returns single page of OData response (in online mode) or complete set of offline data for specified context. See the description of json type below. Parameter "Filter" is additional ODATA like string that should be added to original Odata request (this functionality can be changed in future)

  - feed `getFeed`(string contextName, string filter)

    Returns feed object with a first page of OData response (in online mode) or complete set of offline data for specified context. See the interface of feed type below. Parameter "Filter" is additional ODATA like string that should be added to original Odata request (this functionality can be changed in future)

- bool `executeAction`(string name, json data)

  Executes action on the user task and supplies data. Should return false in case of error (after showing error). False as response after error made possible to do some other actions in this Task.

- json capabilities

  Provides a JSON list of array of string values in json format. For now, there are the following capabilities:

  - print – having the capability in the list means that printing is supported.
  - online - having the capability in the list means that container is in online mode.
  - execute – having the capability in the list means that an action can be executed.

  The list should be maintained along emerging new capabilities.

- json `getContextData` ()

  Returns data for Context section of `smarttask Metadata XML`. See the description of json type below.

- bool `executeUserTask`()

  Executes user task and should change User Task status. Should return false in case of error (after showing error). False as response after error made possible to do some other actions in this Task.

- bool `executeNonFinalAction` (string name, json data)

  Executes an action which does not change the state of the user task and supplies data. In case of an error, it will return `false`. After this action it is possible to do other actions in this task.

Interface of feed type

- json `result`

  Contains response object for the certain part of the feed.

- feed `next`()

  Returns continuation of the feed or *null* if there is no more data.

- bool `hasMorePages`

  defines if is there any more data in the feed that can be returned by calling `next` () function.

Passing json data

JSON data is passed to / from container in a string form.

Smart task container initialization sequence

1. Preprocess HTML:
    - Replace `<BASE/>` tag if local assets cache is used.
    - Parse smart task metadata XML contained in the body of smart task.
    - Initialize container instance with context and data contexts taken from parsed metadata.
2. Create web-browser / iframe and load HTML content there.
3. When the content is loaded, attach the container instance to web-browser / iframe if necessary, and invoke

    ```
    window.init(container)
    ```

    in the context of web-browser / iframe. For Outlook (If WebBrowser control is used), window.external should be used as a container argument.

Smart task metadata XML schema

See descriptions of each response template under Response Templates.

```
<context register="wzp_user_task" key="21">
***
</context>
<responsetemplates>
  <responsetemplate name="Actions>
  ***
  </responsetemplate>
  <responsetemplate name="Forward>
  ***
  </responsetemplate>
```

```
    <responsetemplate name="Update>
    ***
    </responsetemplate>
  </responsetemplates>
  <data>
    <datacontext name="ForwardActors" query="***">
    ***
    </datacontext>
    <datacontext name="ActionLog" query="***">
    </datacontext>
    <datacontext name="AnswerDocuments" query="***'">
    </datacontext>
  </data>
  <smarttask>
  <context register=”wzp_usertask” key=”123”/>
  <data>
  <datacontext name=”Cases” query=”Files”>
  <!-- offline data goes here -->
  </datacontext>
  <datacontext name=”Contacts” query=”Contacts?$select=ID,Summary”>
  <!-- offline data goes here -->
  </datacontext>
  </data>
  </smarttask>
```

The way smart task meta data and base address is present in HTML (required for preprocessor, take into an account that HTML is not XML, so you shouldn’t parse the whole smart task form using XML parser):

```
  <!doctype html>
```

```
<html>
<head>
<base href="..."/>
...
</head>
<body>
...
<script language="text/xmldata" id="metadata">
<smarttask>
...
</smarttask>
</script>
</body>
</html>
```

Response Templates

Actions

The response template "`Actions`" contains the following fields:

- **Action**: The action the response is issuing.
- **Comment**: The comment that follows the action.
- **Properties**: The user task properties.
- **Answers**: An optional `Record[]` containing the response documents.
- **Identity** : The user task identity.

Forward

The response template "`Forward`" contains the following fields:

- **Action**: The action the response is issuing.
- **Comment**: The comment that follows the action.
- **Properties**: The user task properties.
- **Answers**: An optional `Record[]` containing the response documents.
- **Identity**: The user task identity.
- **Actor**: The actor that the user task is forwarded to.

Update

The response template `"Update"` contains the following fields:

- **Action**: The action the response is issuing.
- **Comment**: The comment that follows the action.
- **Properties**: The user task properties with updated documents.
- **Answers**: An optional `Record[]` containing the response documents.
- **Identity**: The user task identity.
- **Attachments**: The properties containing updated attachments.
- **Actors**: The updated actor list with new/obsolete/reordered actors.

New methods in the Smart Task Container Interface

In object data, two new methods have been added for WorkZone Process 2014 R2.

- feed `getOdataFeed` (string query)

  Returns a feed object with a first page of OData response for the OData request provided as a parameter (in online mode). In offline mode, it should return an empty feed.

- json `getOdataEntry` (string query)

Returns a json object as a result of the OData response for the OData request provided as a parameter (in online mode). In offline mode, it should return the result null.

For WorkZone Process 2017, new properties and methods have been added.

Properties

- `SupportAsync`

  if True, the container is working in Async mode and all functions are called with callback as last the parameter.

- `supportExecuteAllActions`

  if True, the container has the executeAllActions function.

- `supportPreselectedValues`

  if True, the container supports pre-select Values functionality.

Functions

- `executeAllActions`(string noFinalName,Json noFinalData, string finalName, JSOn finalData,)

  Executes non-final actions at first and then executes final actions.

- `getPreselectedValues`()

  Returns a Json String as a Dictionary of Keys to Arrays of IDs for the pre-selected values functionality. It is allowed to preselect values in selector controls. For example, {"PreSelectParties":[{"ID":"381"}, {"ID":"201"}],"PreSelectDocuments":[{"ID":"435"},{"ID":"436"}, {"ID":"494"}]}".

7.5 Case activity form

The case activity form is a specific type of a smart task form and it should define the following properties for the js-controller:

```
window.init = function (wzpContainer) {
  wzpContainer.isCaseActivityForm = true;
  wzpContainer.haveLockTakeOverFunctionality = false;
  angular.module('wzput').constant('wzpContainer', wzpContainer);
  angular.bootstrap(window.document, ['wzput']);
};
```

7.5.1 Case activity container

The Case activity container is a specific occurrence of the Smart task container. It is actually an asynchronous Smart task container interface with a lot of functionality that has been implemented internally. The functionality is also used for forms in WorkZone Client smart tasks.

See Smart tasks container.

7.6 Containers

7.6.1 Standard container

7.6.2 Building custom form containers

Changes in the interaction between Container and form controls

In WorkZone Process 2014 R2, interaction between the container and form controls has been changed in order to support the error handling interface. A new `'wzpContainerHelper'` module has been implemented and this must be used instead of `'wzpContainer'`.

> **Important:** The `'wzpContainerHelper'` module has the same structure as the container, but it will contain additional functionality for event handling and dynamic capability (not implemented yet).

Changes in the form controllers

1. You should still inject the WorkZone Process control in the form module in the form controller as before: `angular.module('wzput').constant ('wzpContainer', wzpContainer);`
2. You no longer need to inject the addition `'localContainer'` (and related js-definitions): ~~angular.module('wzput').constant('localContainer', localContainer);~~
3. You must add the new angular service `'wzpContainerHelper'`: `angular.module('wzput',['ngResource', 'ngProgressLite', 'ngUtilities', 'localize', 'wzpContainerHelper', 'wzp.filters', 'settings' … ]);`
4. Use `'wzpContainerHelper'` in all controls and controllers to work with the Container API instead of `'wzpContainer'`, which is obsolete.

Support of custom containers

If your form needs to use some custom properties of the functions, then either extend `'wzpContainerHelper'` or create a new custom module and use this instead of `'wzpContainerHelper'`.

Example: The extension `'wzpContainerHelper'` module.

```
 angular.module('wzp').config(function ($provide) {
 //decorator to wrap uiHelper service
$provide.decorator('wzpContainerHelper', function ($delegate) {
// override or add any functions in uiHelper
 $delegate.CustomFunction = function (options, value) {
```

```
// put overwriting code here
 this.wzpContainer.customFunction(options, value)
 }
return $delegate;
})
 });
```

Example: Definition of the custom `ContainerHelper` module.

```
angular.module('wzpContainerHelper').factory('wzpContainerHelperCustom',
['wzpContainerHelper', 'wzpContainer', function (wzpContainerHelper,
wzpContainer) { wzpContainer.customFunction = function (){
return wzpContainer.customFunction();
}
return wzpContainer;
}]);
```

Support for dirty marking in containers (Dirty Marking API)

Smarttasks can track a user's changes and inform the WorkZone Client form container about it, if the container supports the Dirty Marking API.

To set up the container to support the Dirty Marking API, you need to:

- Set the **supportDirtyFlag** property to true, `supportDirtyFlag = true`.
- Apply the **changeDirtyState(isDirty)** method. The method will be called with `isDirty = true`, when the smarttask recives some changes. If the changes were deleted or changed back, the same method will be called with the flag `isDirty = false`.

WorkZone Process smarttasks support track changes in the following controls:

- Document and Actors multi-selectors
- Comment field

- Answers selectors
- Forward\Reject To\Conditional selectors

7.7 Controls

7.7.1 Upgrade selector controls from 2016 to 2016 R2

In WorkZone Process 2016 R2 changes have been implemented for methods to select options in forms. Follow the guidelines below to upgrade forms:

1. In forms that are created with a JavaScript controller, go to the line `angular.module('wzp', ['….` and remove `'ui.select2','ui.selector', 'ui.forward','ui.rollbackselector'`. Then, if it does not exist already, add the appropriate wzp controls.
2. In all HTML forms, replace old controls with new ones as follows:

Old element	New element	Comments
ui:selector	wzp:selector	Use this for single selectors such as multiple:false, or use it if you don't need the possibility to edit items.
ui:selector	wzp-multi-selector	Use this for editable multiple selectors.
ui:filter-selector	wzp:filter-selector	
ui:sequence-mask-selector	wzp:sequence-mask-selector	
ui:forward	wzp:forward	

usertask-rollbackselector-panel-helper	wzp-usertask-rollbackselector-panel-helper	Use this for HTML attributes
ui:rollbackselector-panel	wzp:rollbackselector-panel	
ui:rollbackselector	wzp:rollbackselector-panel	
ui-checkboxselector	wzp-checkboxselector	Use this for HTML attributes in the ui:selector element

3. In all HTML forms replace the old ng-controller with new ones:

Old controller	**New controller**
SelectODATACtrl	wzpSelectODATACtrl
SelectODATAWithFilterCtrl	wzpSelectODATAWithFilterCtrl
InitFormFilterForSelectorCtrl	wzpInitFormFilterForSelectorCtrl
InitFormSequenceMaskForSelector Ctrl	wzpInitFormSequenceMaskForSelector Ctrl
SelectUserTaskDocumentsCtrl	wzpSelectUserTaskDocumentsCtrl
SelectUserTaskActorsCtrl	wzpSelectUserTaskActorsCtrl
CustomEditDocumentController	wzpCustomEditDocumentController
SmartTaskFilterForSelectorCtrl	zpSmartTaskFilterForSelectorCtrl

4. Remove the attribute `wzp-ad-selector-change-label` and its value.
5. Remove change-title-variable attribute and its value.

6. For the new `wzp:filter-selector` control, change the attribute `ng-controller-name` to `ng-controller`.
7. For instances of `wzp-multi-selector` or `wzp:rollbackselector` that contain instances of `wzp:filter-selector` or `wzp:sequence-mask-selector`, change the class attribute to `class="newline wzp-task-documentlist wzp-select-with-filter"`.
8. Separate each `wzp-multi-selector` by `<div class="wzp-task-editdocument">`.
9. Set `always-editable="true"` for the instance of `wzp-multi-selector` or `wzp:rollbackselector` that you want only in editable mode. An example would be the use of these selectors in Init forms.

7.7.2 Form basic controls

WorkZone Process has a standard library containing a number of Basic forms controls. These are embedded in the Basic.js package. For more information about the form controls, see API Documentation

ui:title
ui:text
ui:datetime
ui:select
ui:help
ui:button
ui:startprocess
ui:actions
ui:action
wzp:forward
ui:usertask
ui:label
ui:comment
ui:link
ui:repeatview
ui:checkbox
ui:integer

7.7.3 Editable controls in smarttasks

About components

To fully enable the functionality of editing the list of documents and/or actors (or any other dynamic list related to a task) in a smart task, the controls must be used together. In the view file (HTML) of the smart task, follow these steps:

1. Add the `wzp:rollbackselector` control for each list that must be editable.
2. Add one `wzp:rollbackselector-panel` control for managing general save and cancel actions for these controls.
3. Optionally, use a specific expand-panel construction for collapsing/expanding `wzp:rollbackselector`. (See Configure expanding/collapsing a wzp:rollbackselector section).
4. In the controller file (JS) of the smart task implementation, add the module `wzp.controls` to the list of modules in the smart task controller:

```
angular.module('wzput', ['ngResource', 'ngProgressLite',
'ngUtilities', 'localize', 'wzp.filters', 'settings', 'ui.help',
'wzp.controls', 'ui.actions', 'ui.action', 'ui.usertask',
'ui.comment', 'ui.label', 'ui.link', 'ngProgressLite',
'ui.helpers', 'ui.repeatview');
```

The wzp.rollbackselector control

This control allows showing items from a dynamic collection related to a smart task and changing them (add, delete and reorder), and after that saving or canceling these changes with or without completing the current smart task.

Data context

Each `wzp:rollbackselector` control needs two `DataContextDefinition` to get access to dynamic data. One for preselected values, and one for possible choices. The collection can return different sets of properties, but it should be converted to the same item's collections by the converters of the control.

Examples

To edit a list of documents, these two data contexts can be used.

- For a document already selected for the current smart task:

```
<DataContextDefinition>
<Name>AttachedDocuments</Name>
<Query>WzpUserTaskInserts?$filter=TaskId eq '{0}' and Attach eq
true&amp;$expand=Records&amp;$select=RecordId,Records/Title,Records/State_
Value,Records/RecordType_Value,Records/DocumentType_
Value,Records/Summary&amp;$orderby=Priority</Query>
<MaxOfflinePages>3</MaxOfflinePages>
<Parameters>
<Parameter>TaskId</Parameter>
</Parameters>
</DataContextDefinition>
```

- For a document that can be added to the smart task:

```
<DataContextDefinition>
<Name>AnswerDocuments</Name> <Query>Records?$select=ID,Summary,DocumentType_
Value,State_Value&amp;$orderby=ID,Summary&amp;$filter=FileKey_Value eq '{0}'
and State_Value ne 'UP' and ExternalDocId ne '' </Query>
<MaxOfflinePages>10</MaxOfflinePages>
<Parameters>
<Parameter>RegisterKey</Parameter>
</Parameters>
</DataContextDefinition>
```

To edit a list of documents, these two data contexts can be used.

- For representing actors for all active smart task for the current process:

```
<DataContextDefinition>
```

```
<Name>ActiveActors</Name>
<Query>WzpUserTasks?$expand=NameKey&amp;$select=InstanceId,NameKey_
Value,TaskState_Value,NameKey/ID,NameKey/Summary,NameKey/NameType_
Value,NameKey/NameCode&amp;$filter=InstanceId eq '{0}' and (TaskState_
Value eq 'OPEN' or TaskState_Value eq
'PENDING')&amp;$orderby=TaskOrder</Query>
<MaxOfflinePages>10</MaxOfflinePages>
<Parameters>
<Parameter>InstanceId</Parameter>
</Parameters>
</DataContextDefinition>
```

- For actors that can be added the smart task:

```
<DataContextDefinition>
<Name>ForwardActors</Name>
<Query>WzpFileUserRights?$select=ID,Summary,NameType_Value,NameCode_
Value&amp;$expand=NameKey,NameKey/AddressKey&amp;$orderby=Summary&amp;$fi
lter=FileKey eq '{0}' and NameKey/AddressKey/Email ne ''</Query>
<MaxOfflinePages>10</MaxOfflinePages>
<Parameters>
<Parameter>RegisterKey</Parameter>
</Parameters>
</DataContextDefinition>
```

Add filters in the wzp.controls

You can enable predefined filtering options for the `wzp.selector` control in the init form or the `wzp.rollbackselector` control in the smarttask form.

Filtering options for the Init form

To add a filter control in the `wzp.selector` control in the Init form, follow these steps:

1. Add or modify the `wzp.selector` control (“parent” selector):

    a. Change the `ng-controller` value to `"wzpSelectODATAWithFilterCtrl"`.

    b. Add the css class “`wzp-select-with-filter`” for correct styling.

    c. Add the attribute `wzp-ad-selector-change-label` with the following expression:

    ```
    {{<twoWayBindings>.<DocumentLabelWithFilterValue>}}",
    ```

    where the variable “`twoWayBindings.DocumentLabelWithFilterValue`” is equal to the `"change-title-variable"` attribute in the nested `wzp:filter-selector` control.

    d. Add the attribute `wzp-ad-selector-filter-variable`.

    This should contain a variable equal to the `"ng-model"` attribute in the nested `wzp:filter-selector` control.

2. Add the `wzp:filter-selector` control inside the parent selector with these parameters:

Attribute	Description	Example or comment
ng-model	A pointer to the source model property, which this control is bound to. It should be equal to the `'wzp-ad-selector-filter-variable'` attribute of the parent `wzp.control` and contains `'.'`.	`twoWayBindings.DocumentFilters`
ng-controller	Name of controller	`wzpInitFormFilterForSelectorCtrl`
predefined-	Register of parent	`Record`

Attribute	Description	Example or comment
filter-register	selector	
change-title-variable	A pointer to the 2-way binding variable for updating the parent label. It should be equal to the expression in the `'wzp-ad-selector-change-label'` attribute of the parent `wzp.control`.	`twoWayBindings.DocumentLabelWithFilterValue`

3. Define the `<twoWayBindings>` object in scope of the `ui.startprocess` controller of the Init form.

   Example: `$scope.twoWayBindings = {};`

This is an example of a `wzp.control` with a filter control for documents:

```
<wzp:selector

ng-model="dataSource.Documents"

ng-controller="wzpSelectODATAWithFilterCtrl"

labelgroup="ADV_SHARED"

label="DOCUMENTS"

placeholder="SELECT_DOCUMENTS"

options="{

register: 'Records',

filter: searchInCurrentCase(),

freetextfield:'Summary',

openItem:{icon:'content',title:'metadata',actionRegister:'Record'} }"

class="newline wzp-select-with-filter"

ng-disabled="formisdisabled"
```

```
wzp-ad-selector-change-label="
{{twoWayBindings.DocumentLabelWithFilterValue}}"
wzp-ad-selector-filter-variable="twoWayBindings.DocumentFilters">
<wzp:filter-selector
ng-controller="wzpInitFormFilterForSelectorCtrl"
ng-model="twoWayBindings.DocumentFilters"
predefined-filter-register="Record"
change-title-variable="twoWayBindings.DocumentLabelWithFilterValue"
class="newline">
</wzp:filter-selector>
</wzp:selector>
```

Filtering options for the smarttask form

To enable filtering options in the smarttask form, you need to perform steps similar to the steps for the `wzp.selector` for the Init form but with the following differences:

1. For the `wzp.rollbackselector` control (“parent” selector control instead of `wzp.selector`):
    a. Use the `ng-controller` for the `wzp.rollbackcontrol`. No special `ng-controller` is necessary.
    b. Add `query: getQueryWithFilter` in the `options` attribute.
2. For the `wzp:filter-selector` control inside the parent selector, define the following additional parameters:
    a. `ng-controller="wzpSmartTaskFilterForSelectorCtrl"`.
    b. `ng-hide="noCapability('execute','online')"` to hide the filter control in offline/read-only mode.

Example: `wzp.rollbackselector` with a filter control for documents:

```
<wzp:rollbackselector
ng-model="Documents"
ng-controller="wzpCustomEditDocumentController"
default-data-context-name="AttachedDocuments"
item-convertor-name="converterFromOdataToSelectorForAttachments"
save-result-convertor-name="updateDocumentsFromAttachmentConverter"
options="{
register: 'Records',
openItem:{ actionRegister:'Record'},
datacontextName: 'DocumentsContext',
query: getQueryWithFilter
}"
readonly="noCapability('execute','online')"
labelgroup="CONTROL"
label="DOCUMENTS"
placeholder="SELECT_DOCUMENTS"
class="newline wzp-task-documentlist wzp-select-with-filter"
wzp-ad-selector-change-label="
{{twoWayBindings.DocumentLabelWithFilterValue}}"
wzp-ad-selector-filter-variable="twoWayBindings.DocumentFilters"
>
<wzp:filter-selector
ng-controller="wzpSmartTaskFilterForSelectorCtrl"
ng-hide="noCapability('execute','online')"
ng-model="twoWayBindings.DocumentFilters"
predefined-filter-register="Record"
change-title-variable="twoWayBindings.DocumentLabelWithFilterValue"
class="newline">
</wzp:filter-selector>
```

```
</wzp:rollbackselector>
```

Control specification

Control specification should be placed in `div` with css style class definition. The default css class is `wzp-task-editdocument`.

Examples

- Editing documents

```
<div class="wzp-task-editdocument">
<wzp:rollbackselector
ng-model="Docs"
ng-controller="wzpCustomEditDocumentController"
default-data-context-name="AttachedDocuments"
item-convertor-name="converterFromOdataToSelectorForAttachments"
save-result-convertor-name="updateDocumentsFromAttachmentConverter"
options="{ register: 'Records', openItem:{ actionRegister:'Record'},
datacontextName: 'AnswerDocuments'}"
readonly="noCapability('execute','online')"
labelgroup="CONTROL"
label="DOCUMENTS"
placeholder="SELECT_DOCUMENTS"
required
class="newline wzp-task-documentlist">
</wzp:rollbackselector>
</div>
```

- Editing actors

```
<div class="wzp-task-editdocument expand-panel-body" ng-
hide="!actorEditVisible">
<wzp:rollbackselector
```

```
ng-model="Actors"
ng-controller="wzpCustomEditDocumentController"
default-data-context-name="ActiveActors"
item-convertor-name="converterFromOdataToSelectorForActors"
save-result-convertor-name="updateActorsFromSomethingConverter"
options="{showSelected:true, openItem:{icon:'metadata',title:'metadata' ,
actionRegister:'Contact'} , iconType:'contacts', datacontextName:
'ForwardActors'}"
readonly="noCapability('execute','online')"
labelgroup="CONTROL"
label="ACTORS"
placeholder="SELECT_PARTIES"
class="newline wzp-task-documentlist body-column">
</wzp:rollbackselector>
</div>
```

Controller and converter functions

The Basis package contains a custom controller for the `wzp.rollbackselector` control for editing the documents list and the actors list.

It contains two pairs of converters (one for documents and one for actors):

- A converter for presenting information from `DataContextDefinition` which has a format that is suitable for the `wzp.rollbackselector` control (the names of the converter functions are used in the attribute `item-convertor-name` of the `wzp.rollbackselector`).
- A converter for saving changes (the names of the converter functions are used in the attribute `save-result-convertor-name` of the `wzp.rollbackselector`).

Also, the controller should have an init part, which initializes converters and overwrites the `openSelectedItemHandler` delegate.

```
function init() {
$scope.$$childTail.itemConvertor = $scope
[$scope.$$childTail.itemConvertorName];
$scope.$$childTail.openSelectedItemHandler = documentOpenSelectItemHandler;
$scope.$$childTail.saveResultConvertor = $scope
[$scope.$$childTail.saveResultConvertorName];
};
```

Controller structure

```
angular.module('wzp.rollbackselector').controller
('CustomEditDocumentController',['$scope',function ($scope)
{

/* Converters for Document*/
$scope.converterFromOdataToSelectorForAttachments = function (data, options,
helpers) {};
$scope.updateDocumentsFromAttachmentConverter = function (data, result,
hasChangesDelegate) { };
/* Converters for Actors*/
$scope.converterFromOdataToSelectorForActors = function (data, options,
helpers) {}
$scope.updateActorsFromSomethingConverter = function (data, result,
hasChangesDelegate) {};
/* overwrite openSelectedItemHandler delegate */
function documentOpenSelectItemHandler(itemData, handlerName) {
if (itemData.actionregister === 'Record') {
if (!itemData.text && !itemData.type)
return;
```

```
} $scope.$$childTail.baseOpenSelectedItemHandler(itemData, handlerName);
};
/* Function to init wzp.rollbackselector with convectors. Mandatory for
custom controllers. */

function init() { };
init();
}]);
```

Examples of an item converter

These converters convert items from the data parameter to an array of objects using control options (options parameter) and some static methods (helpers parameter).

The structure of the data object depends on the ODATA request's result, defined by the corresponding `DataContextDefinition`.

A returned array of items should have the following properties: `id`, `text`, `register`, `icon`, `type`, `namecode`, `actionregister`.

If `wzp.rollbackselector` has any customizations, then the properties of returned items should be aligned with them.

Example of item converter for editing documents

```
$scope.converterFromOdataToSelectorForAttachments = function (data,
options, helpers) {
var documents = [];
$.each(data, function (key, value) {
var recordProperty = { DocumentType_Value: '', State_Value: '' };
if (!!value.Records && value.Records.length > 0) {
recordProperty = value.Records[0];
}
var document = {
```

```
id: value.RecordId,
text: recordProperty.Summary,
register: options.register,
icon: helpers._getIcon(options, { DocumentType_Value:
recordProperty.DocumentType_Value, State_Value: recordProperty.State_Value
}),
type: (options.register == 'Records' ? recordProperty.DocumentType_Value :
''),
namecode: (options.register == 'Contacts' ? recordProperty.NameCode : ''),
actionregister: (!!options.openItem && !!options.openItem.actionRegister) ?
options.openItem.actionRegister : options.register
}
helpers._protectedDocumentTitleFix(document);
documents.push(document);
});
return documents;
};
```

Example of item converter for editing actors

```
$scope.converterFromOdataToSelectorForActors = function (data, options,
helpers) {
var actors = [];
$.each(data, function (key, value) {
var actorPropery = value.NameKey;
var actor =
{
id: actorPropery.ID,
text: actorPropery.Summary,
register: options.register,
```

```
icon: helpers._getIcon(options, actorPropery),
type: '',
namecode: actorPropery.NameCode,
actionregister: (!!options.openItem && !!options.openItem.actionRegister) ?
options.openItem.actionRegister : options.register
};
actors.push(actor);
});
return actors;
}
```

Example of a result converter for editing documents

```
$scope.updateDocumentsFromAttachmentConverter = function (data, result,
hasChangesDelegate) {
if (!checkforChanges(hasChangesDelegate)) return result;
var documents = [];
$.each(data, function (index, value) {
var document = {
$type: "Scanjour.OData.Client.Lite.WorkZone.Record,
Scanjour.OData.Client.Lite", TypeName: "Som.Record", MediaResource: null,
ID: value.id
}
documents.push(document);
})
var _propertyToUpdateName = 'Documents';
result.Attachments = [_propertyToUpdateName];
result.Properties[_propertyToUpdateName].value = documents;
return result;
};
```

Where :

- data – changed collection provided by the control.
- Result – the populated template response part.
- hasChangesDelegate – delegate to define if any changes were done. If no changes were done, then there is no need to save anything (but there can be changes here).

For a description of response populating, see Populating document changes.

Example of a result converter for editing actors

```
$scope.updateActorsFromSomethingConverter = function (data, result,
hasChangesDelegate) {
if (!checkforChanges(hasChangesDelegate)) return result;
var actors = [];
$.each(data, function (index, value) {
var actor = {
$type: "Scanjour.OData.Client.Lite.WorkZone.Contact,
Scanjour.OData.Client.Lite", TypeName: "Som.Contact", MediaResource: null,
ID: value.id
};
actors.push(actor);
});
result.Actors = actors;
return result
};
```

For a description of the parameters, see Example of item converter for editing documents.

For explanation about response populating, see Populating actor changes.

Example of result converter for editing actors

```
$scope.updateActorsFromSomethingConverter = function (data, result,
hasChangesDelegate) {
if (!checkforChanges(hasChangesDelegate)) return result;
var actors = [];
$.each(data, function (index, value) {
var actor = {
$type: "Scanjour.OData.Client.Lite.WorkZone.Contact,
Scanjour.OData.Client.Lite", TypeName: "Som.Contact", MediaResource: null,
ID: value.id
};
actors.push(actor);
});
result.Actors = actors;
return result
};
```

For a description of the parameters, see Example of item converter for editing documents.

For explanation about response populating, see Populating actor changes.

Wzp:rollbackselector-panel control

This control presents a separated panel with a save and a cancel button which give the ability to save or cancel changes done with all `wzp:rollbackselector` controls at the same time.

The panel is invisible by default, and becomes visible when any changes are done by any `wzp:rollbackselector`.

The **Cancel** button cancels changes in all `wzp:rollbackselector` controls.

The **Save** button saves changes in all `wzp:rollbackselector` controls in one update request.

Control specification

The control should be placed in `div` with a css style class definition. The default css class is `edit-panel`.

The attribute `ng-hide="noCapability('execute','online')"` makes the panel visible only when a smart task has both '`execute`' and '`online`' capability at the same time.

Example

```
<div class="edit-panel" ng-hide="noCapability('execute','online')">
<wzp:rollbackselector-panel
readonly="noCapability('execute','online')"
labelgroup="CONTROL"
cancel-button-label="CANCEL"
cancel-button-hide="false"
save-button-label="SAVE"
capability="execute"
usertask-response-template-name="Update"
class="newline">
</wzp:rollbackselector-panel>
</div>
```

Configure expanding/collapsing a wzp:rollbackselector section

Using this html structure and styles, and some javascript+ angular code, a `wzp:rollbackselector` can be placed in expand-colapse panel.

HTML

```
<div class="expand-panel-header" ng-hide="actorEditVisible">
```

```
<div class="button-column" ng-click="actorEditSwitch()">
<div></div>
</div>
<div class="wzp-control-labeled body-column">
<label>{{$root.i18n('CONTROL', 'ACTORS')}}</label>
</div>
</div>
<div class="wzp-task-editdocument expand-panel-body" ng-
hide="!actorEditVisible">
<div class="button-column" ng-click="actorEditSwitch()">
<div></div>
</div>
<wzp:rollbackselector
* * *
labelgroup="CONTROL"
label="ACTORS">
</wzp:rollbackselector>
</div>
```

Where

`{{$root.i18n('CONTROL', 'ACTORS')}}` should be equal to the label group and label attribute of the `wzp:rollbackselector` control.

JavaScript + Angular code

You can add new properties in the smart task controller (in angular.module ('`ui.usertask`').controller('`ApproveTaskCtr`',) body for switching between expanded and collapsed states of the section:

```
$scope.actorEditVisible = false;
$scope.actorEditSwitch = function () {
```

```
$scope.actorEditVisible = !$scope.actorEditVisible;

}
```

Response Template population

Response structure

You can save any changes in document or actor collections (or any other changes for smart tasks) by performing an "`Update`" process action for smart tasks with a correctly populated ResponseTemplate.

For this action you must use the ResponceTemplate with the name "`Update`" form SmartTask Metadata ResponseTemplates .

The current structure of this ResponseTemplate is (in JSON format) as follows:

```
{

"$type": "Scanjour.Workflow4.Base.UserTaskUpdateResponse,

Scanjour.Workflow4.Base, Version=4.1.0.0, Culture=neutral,

PublicKeyToken=null",

"NearDueDate": "\/Date(-62135596800000)\/",

"DueDate": "\/Date(-62135596800000)\/",

"Attachments": null,

"Actors": null,

"OptionalActors": null,

"Action": null,

"Comment": null,

"Properties": {

"$type": "Scanjour.Workflow4.Base.UserTaskProperties,

Scanjour.Workflow4.Base, Version=4.1.0.0, Culture=neutral,

PublicKeyToken=null",

"Documents": {
```

```
"$type": "Scanjour.Workflow4.Base.UserTaskProperty,
Scanjour.Workflow4.Base, Version=4.1.0.0, Culture=neutral,
PublicKeyToken=null",
"key": "Documents",
"type": "Scanjour.OData.Client.Lite.WorkZone.Record[]",
"value": [
{
"$type": "Scanjour.OData.Client.Lite.WorkZone.Record,
Scanjour.OData.Client.Lite, Version=4.1.0.0, Culture=neutral,
PublicKeyToken=null",
"TypeName": "Som.Record",
"MediaResource": null,
"ID": "7",
"Actions": [ ],
"Properties": [ { "$type": "Scanjour.OData.Client.Lite.PropertyMember,
Scanjour.OData.Client.Lite, Version=4.1.0.0, Culture=neutral,
PublicKeyToken=null", "key": "ID", "type": "System.String", "value": "7" }
],
"SubEntries": [ ],
"Feeds": [ ]
}
]
},
"FileNo": {},
"Officer": {},
"OfficerName": {},
"Register": {},
"RegisterKey": { },
"InstanceId": { },
"TaskId": { }
```

```
},
"Answers": null,
"Identity": null
}
```

Populating actor changes

To save changes in the actor list, you must save the new collection of changed items in the `Actors` (or `OptionalActors`) property of the `responseTemplate` object.

These collections have the type `Som.Contact[]`, so an element of an array must be in the following format (in JSON):

```
{
$type: "Scanjour.OData.Client.Lite.WorkZone.Contact,
Scanjour.OData.Client.Lite", TypeName: "Som.Contact",
MediaResource: null,
ID:id
};
```

Where `id` is the `ID` of a contact from the `wzp:rollbackselector` items.

Populating document changes

To save changes in the document list, you must save the new collection of changed items using this sequence of actions:

1. Save the name of the collection with changed values from the Properties collections as an array of a string:

   Attachments = [“`Documents`”];

2. From an array with new items of the type `Som.Record` in the following format, enter:

   ```
   {
   ```

```
$type: "Scanjour.OData.Client.Lite.WorkZone.Record,
Scanjour.OData.Client.Lite", TypeName: "Som.Record",
MediaResource: null,
ID: id
},
```

Where `id` is the `ID` of `Record` from `wzp:rollbackselector` items.

3. Save this array in Properties["`Documents`"].value.

Add a filter control in The <wzp-multi-selector> and <wzp:rollbackselector> parent controls

To add a filter control in the `<wzp-multi-selector>` and `<wzp:rollbackselector>` parent controls, follow these steps:

1. Add or modify the `<wzp-multi-selector>` and `<wzp:rollbackselector>` controls ("parent" selector):
   a. Change the `ng-controller` value to "`wzpSelectODATAWithFilterCtrl`".
   b. Add the css class "`wzp-select-with-filter`" for correct styling.
   c. Add the attribute `wzp-ad-selector-filter-variable`, that should contain a variable equal to the "`ng-model`" attribute in the nested `<wzp:document-selector-filter>` control.
2. Add the `<wzp:document-selector-filter>` control inside the parent selector with "`ng-model`" attribute equal to the '`wzp-ad-selector-filter-variable`' attribute of the parent `<wzp-multi-selector>` and `<wzp:rollbackselector>` and contains '.'.
3. Define the `<twoWayBindings>` object in scope of the controller of the form.

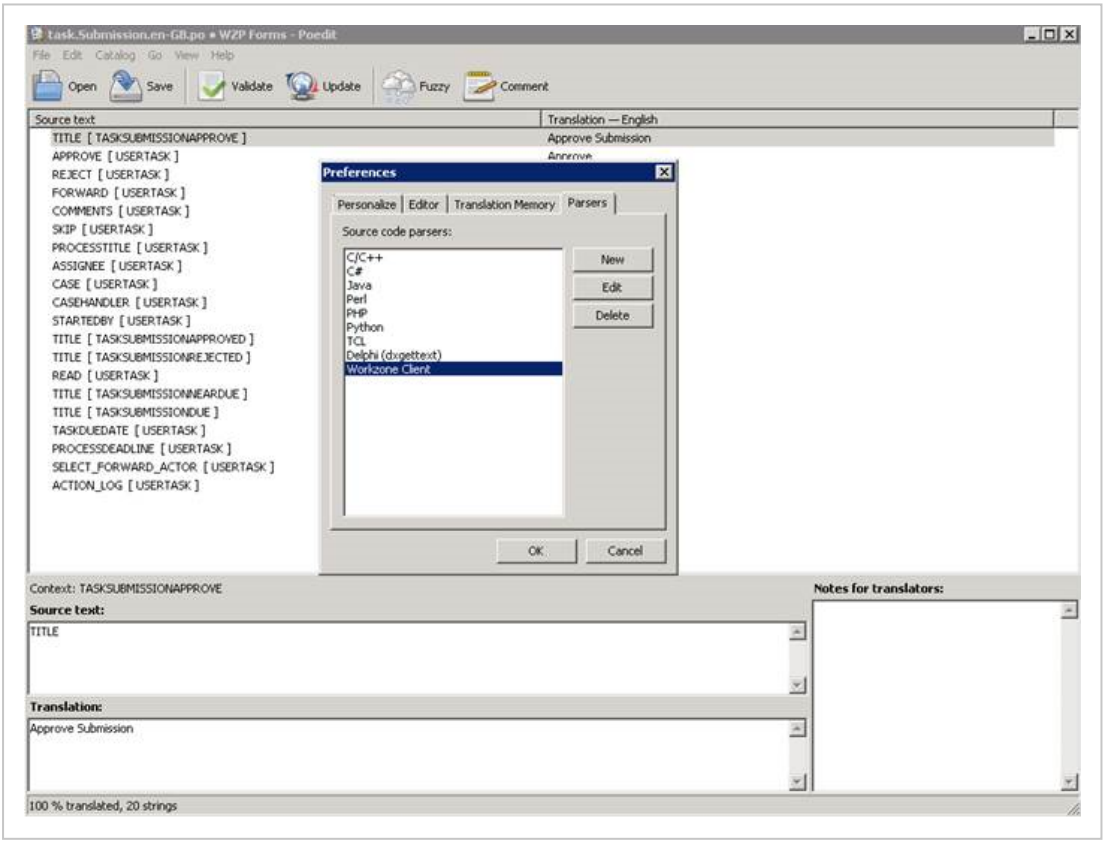
Example: `$scope.twoWayBindings = {};`

7.8 Form localization

Form Localization Concept for localization <body ui-Intl="js/init.Submission."> PO File format. POEDITOR phantomJS.. Settings ->

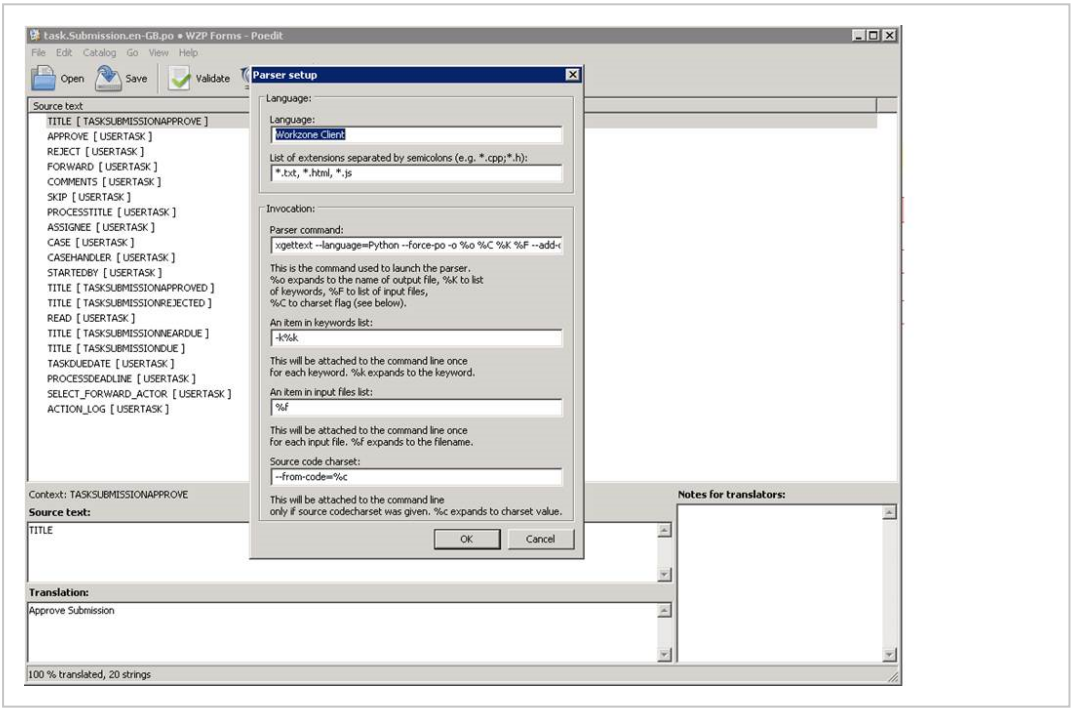
Configuring POEditor to work with WZP forms localization resources

1. Download and install POEditor (http://poedit.net/).
2. Open **File** > **Preferences** > **Parsers**.



3. Add new parser with the following settings:

Language	<ParserName> (ex. Workzone Client)
List of extensions separated by semicolons	*.txt, *.html, *.js
Parser command	xgettext --language=Python --force-po -o %o %C %K %F --add-comments=Reference
An item in keywords list	-k%k
An item in input files list	%f
Source code charset	--from-code=%c



Add or remove new string key in localization resources

1. Check out the read-only flag from the `*.terms.txt` resource and `*.po` files.
2. Add a new string in the format `_("KEYGROUP", "STRINGKEY")` in the source file `*.terms.txt`.
3. Open each `po` file in POEditor.
4. Click **Update**.
5. Check your changes in the update summary and click **OK**.

6. Add the localized string value for the new key value.
7. Save your changes.

8. Processes overview

8.1 Filtering

The WorkZone Process 4.0 overview has the following filters:

WorkZone Process

Domain Restriction filters

The filters are known under three different names:

- "Predefined filters" in WorkZone Client
- "Domain restrictions" in oData
- "Domain" in SOM

In the UI, the list of the domain restriction filters is accessible from **Lists** in context menu. The list of filters is dynamic, meaning its content is based on data from the content server, and it is populated when the overview is started.

In case you need a new domain restriction filter, you must add a new domain to the `wzp_workflow_instance` register. You must refresh the overview to view the new filter.

In WorkZone Process 2014, only 6 domain restriction filters are available:

- `workflows_running` – Active processes
- `workflows_closed` – Closed processes

- `workflows_own_by_me` – My processes
- `workflows_own_by_my_ou` – My unit`s processes
- `workflows_pending_my_action` – Processes pending me
- `workflows_deadline_exceeded` – Processes with exceeded deadline

In order to open the overview with one of the filters applied, add the `filter` url parameter and use one of the filters described above and prefix it with `fixed_` as a value.

Example: `http://DSN/Process/Overview?filter=fixed_workflows_own_by_me`

Register filter

The filter forces the overview to show all processes (running, closed, and cancelled) started on a register (**File**, **Record**, or **Contact**). In WorkZone Process 2014, only the **File** register filter is supported, meaning that only processes started on a specific case can be shown.

In order to show process started on a case, specify the following two parameters in the overview url:

- `register` – possible values: FILE.
- `registerId` – possible values: Any case key.

```
http://DSN/Process/Overview?register=FILE&registerKey=201
```

Default filter and filter precedence

If nothing is specified in the url parameters, the **My processes** domain restriction filter is applied. It is equivalent to calling the overview with the following url parameter: `filter=fixed_workflows_own_by_me`.

In the WorkZone Process 2014 implementation of the overview, only one filter can be applied at a time. If both the domain restriction filter and the register filter are specified in the url, then the register filter is applied, causing the domain restriction filter to be disregarded.

Filter parameters

You can open the Overview in WorkZone Process with various views depending on the parameters that you apply in the url.

Note, that only one filter can be used at a time.

The following table provides an overview of the parameters used to create the links in the WorkZone Process Overview.

Filter	Open WorkZone Process Overview	URL parameters	Example
Case	Open with all processes for a case.	register + registerKey - ID of a case	http://host/Process/Overview/?register=FILE&registerKey=321
Process	Open for a specific process. The process node is expanded.	processID - identification of a process	http://host/Process/Overview/?processID=ebbd337e-69a4-42f8-868c-c1d2e112eef8
Task	Open for a specific user task.	smartTaskId - ID of a user task	http://host/Process/Overview/?smartTaskID=182
Overview list	Open with a selected item from the Lists context menu in the overview.	filter - a list of all values for a parameter can be obtained by request	http://host/Process/Overview?filter=fixed_a_shared_workflows_4_closed (closed processes) or http://host/Process/Overview?filter=fixed_a_shared_workflows_2_own_by_me" (my processes)

8.2 Create a SmartPost dispatcher

You can create a customized SmartPost dispatcher that allows you to send documents through a new channel. This is useful if the current channels are not sufficient, or if the customer has a custom system for sending messages or a proxy/gateway that routes traffic to e-Boks or Strålfors.

This topic describes how to develop a custom dispatcher using the SDK that is part of any WorkZone process installation. The SDK is located in the SDK subfolder in Process installation folder. The default location is: `C:\Program Files (x86)\KMD\WorkZone\Process\SDK\SDK.zip`.

To assist you in developing a custom dispatcher, you can use a sample dispatcher, which is included in the SDK. The sample dispatcher uses an Exchange server to send documents if the recipient has a legal email address. The sample dispatcher is located in the `SDK\exampleprojects\ExchangeDispatcher` folder.

You develop dispatcher as a .NET framework application preferably using the C# language. You can build the sample project using Microsoft Visual Studio 2017 or 2019.

Compile and install the sample dispatcher

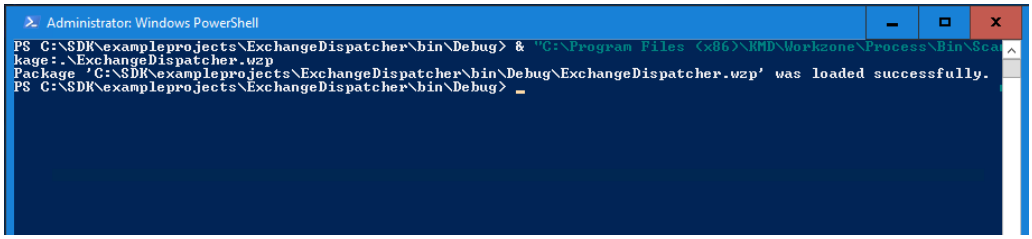
> **Prerequisite:** The steps in these instructions must be performed by an AD domain user who is created as a WorkZone user and has been assigned the PROCESSADM access code.

You compile the dispatcher in Visual Studio by opening the project named **Workzone.Dispatcher.Exchange.csproj**.

When you have compiled the sample project, you can find a WorkZone Process package named **ExchangeDispatcher.wzp** in the output folder, for example in `bin\debug`.

Load the package

Use the package loader to load the package in to WorkZone.



After loading a package, you need to recycle the WzpSv capplication pool.

```
Restart-WebAppPool WzpSvc
```

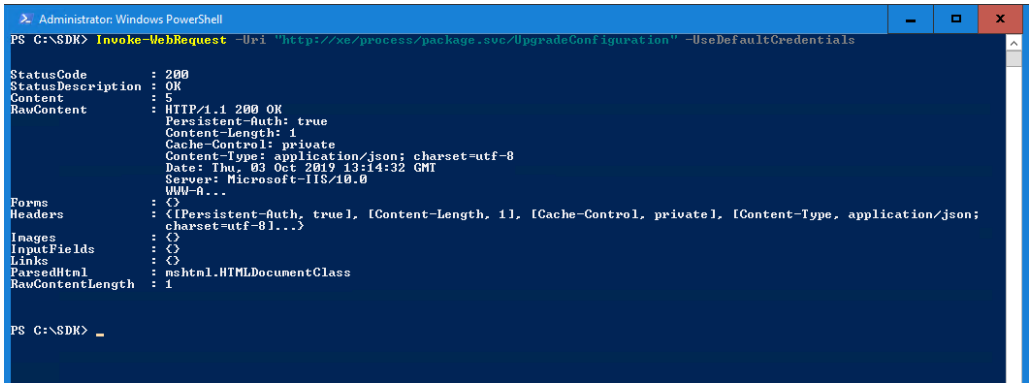
See Configure packages.

Upgrade configuration

The configuration for the dispatcher is created by calling the method **UpgradeConfiguration** on the process package service:

```
http://<database>/process/package.svc/UpgradeConfiguration
```

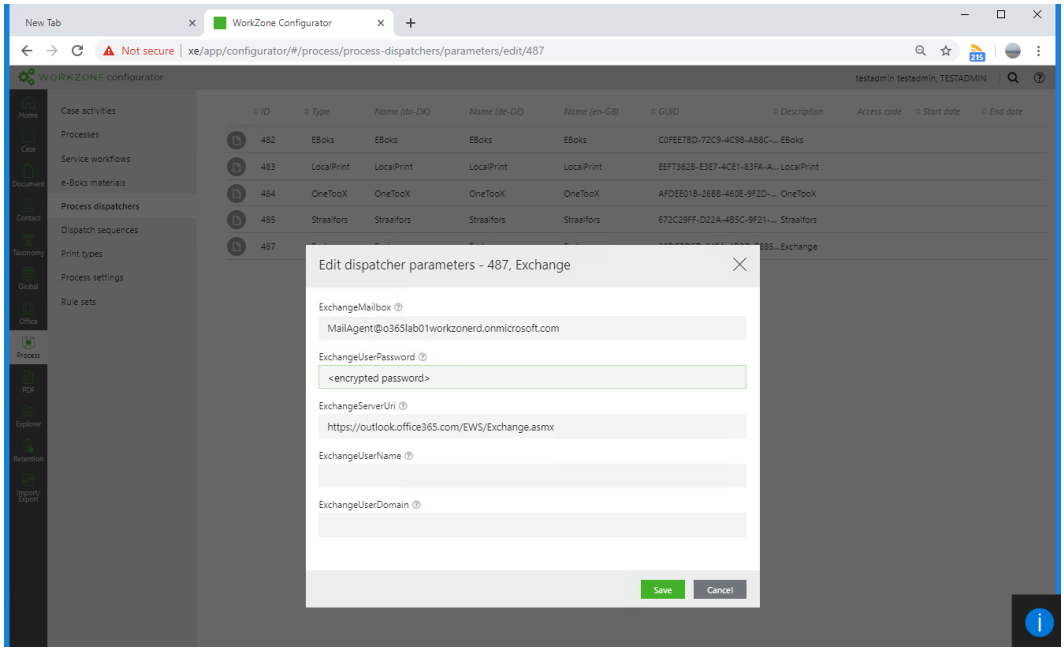
You can use a browser or PowerShell:



The return value (5 in the example) corresponds to the number of dispatchers that are upgraded. If the number is less than zero, it indicates an error. You can find the description of the error in the application event log with source Scanjour Workflow Host.

Configure the dispatcher

After successful installation and upgrade, you can configure the new dispatcher in WorkZone Configurator:



The sample dispatcher uses the same Exchange library as the rest of SmartPost.

See also Configure dispatchers in the WorkZone Process Administrator Guide.

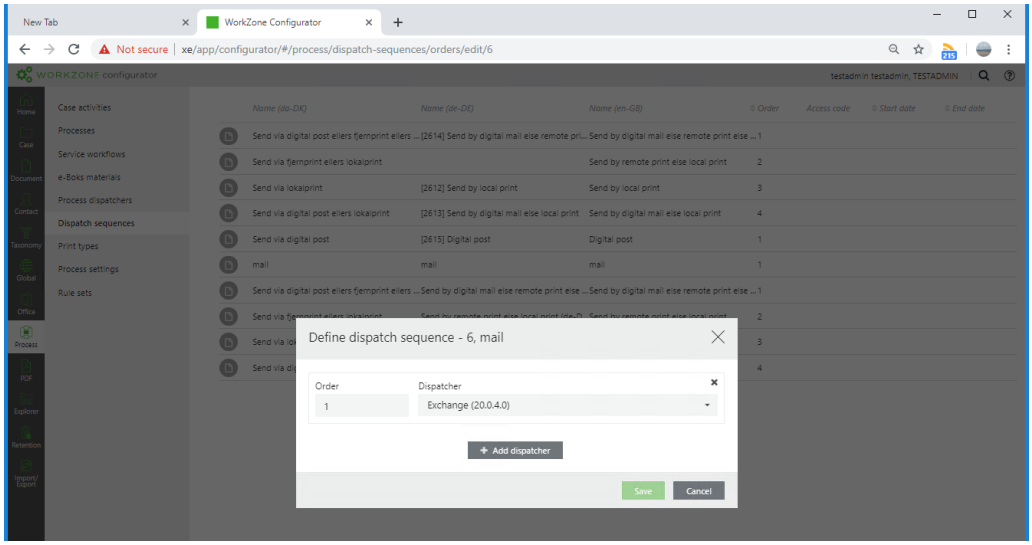
Test the sample dispatcher

If you want to test the sample dispatcher in practice, it is important to configure the dispatcher to be able to send emails and then you must provide an encrypted password.

> **Note:** If the user that is configured for sending smartmails is reused, then the encrypted password string can be found in the configuration file for the `Scanjour.Process.MailAgent.dll.config` in the `c:\program files (x86)\kmd\workzone\process\bin` folder.

Add the sample dispatcher to a dispatch sequence

To use the sample dispatcher for sending documents, it must be part of a dispatch sequence.

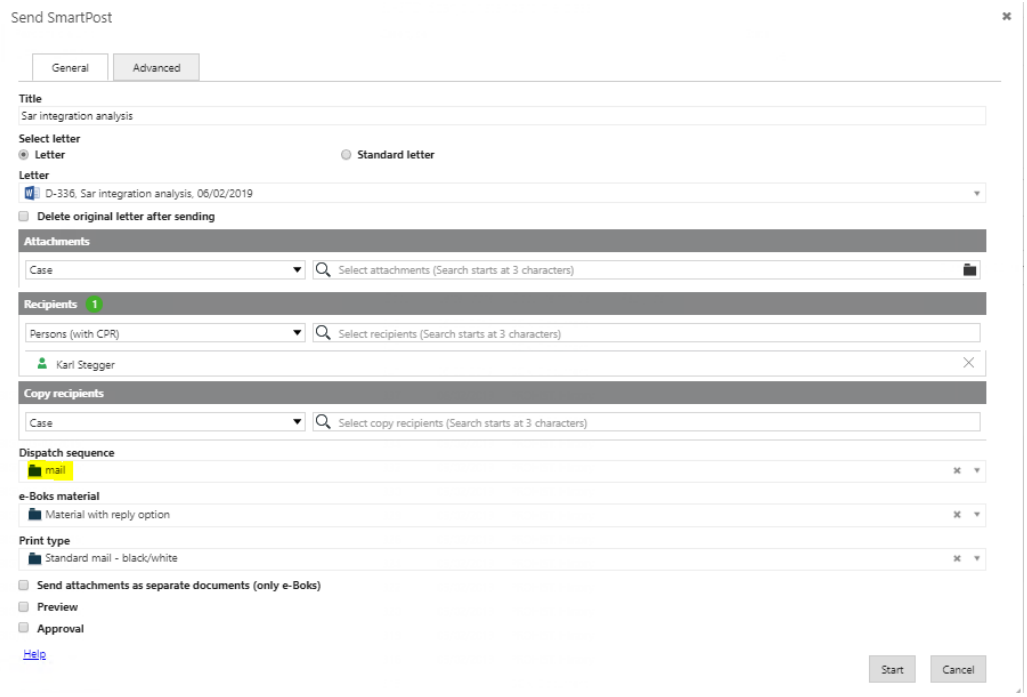


In this example, a new dispatcher sequence named *mail* is created. It only contains the sample dispatcher.

See also Configure dispatch sequences in the WorkZone Process Administrator Guide.

Use the sample dispatcher

When the dispatcher sequence has been defined, it is possible to start a SmartPost process using the dispatcher sequence, in this example named *mail*:



Test the sample dispatcher

If you want to test the sample dispatcher in practice, it is important that you configure the dispatcher to send emails and then you must provide an encrypted password.

> **Note:** If the user that is configured for sending smartmails is reused, then the encrypted password string can be found in the configuration file for the `Scanjour.Process.MailAgent.dll.config` in the `c:\program files (x86)\kmd\workzone\process\bin` folder.

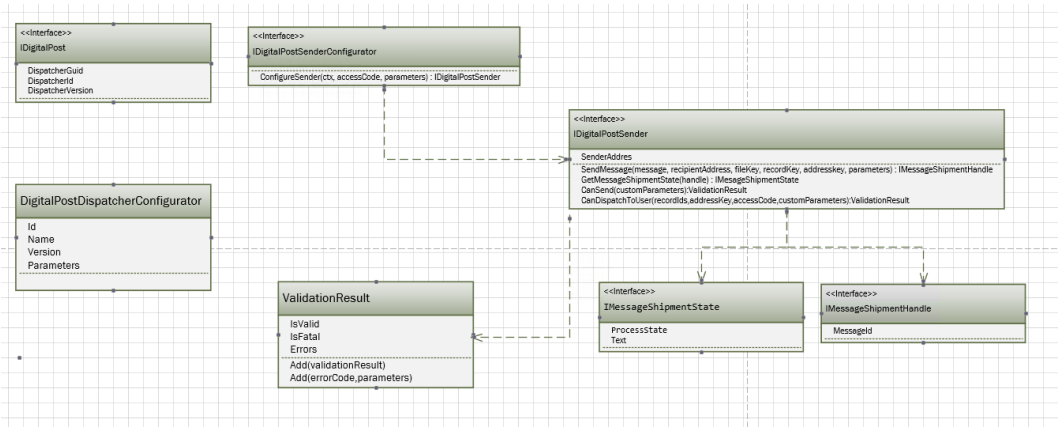
8.3 SmartPost dispatcher classes, interfaces, and attributes

The implementation of a dispatcher requires developing classes that implement specific interfaces and export attributes that enable them to be discovered by the dispatcher framework.

The **Workzone.Dispatcher.Base.dll** assembly contains the definition of the interfaces and all helper classes that the interfaces need.

The .NET Framework assembly **System.ComponentModel.Composition.dll** contains the attribute class used for export. These assemblies must be referenced from the Visual Studio project that builds the new dispatcher.

The class diagram below shows classes and interfaces that must be implemented in a dispatcher:



Implement IDigitalPost interface

The main interface in a dispatcher is **IDigitalPost** interface. Implementing this interface describes the version and unique ID of the dispatcher.

Properties

Name	Description	Example value
DispatcherGuid	A GUID that identifies the dispatcher.	6fbb4267-2edd-495a-ae55-1075743b9286
DispatcherId	A property that can hold the ID of the dispatcher. The value of this property is normally not used or provided by the dispatcher.	
DispatchVersion	A version string that matches the latest version of WorkZone Process.	20.0.0.0

For the class that implements this interface to be discovered by the dispatcher framework, it is required to add an Export attribute to the class that exports the **IDigitalPost** type (use System.ComponentModel.Composition.ExportAttribute).

Implement IDigitalPostSenderConfigurator interface

The **IDigitalPostSenderConfigurator** interface only has one method that returns an initialized class that implements **IDigitalPostSender**.

Methods

Name	Description
ConfigureSender	Returns an instance of a class that implements **IDigitalPostSender**. Normally, this method reads the settings and passes values to the sender class in the constructor. The

Name	Description
	parameters for this method are: • oDataService handle that can be used for accessing the database. • AccessCode (not used). • Extra parameters applied to the SmartPost workflow at start-up.

For the class that implements this interface to be discovered by the dispatcher framework it is required to add an Export attribute to the class exporting the **IDigitalPostSenderConfigurator** type.

Example:

```
[Export(typeof(IDigitalPost))]
[Export(typeof(IDigitalPostSenderConfigurator))]
0 references | Anders Egeberg, 13 days ago | 1 author, 2 changes
public class SenderConfigurator : IDigitalPost, IDigitalPostSenderConfigurator
{
    7 references | Anders Egeberg, 14 days ago | 1 author, 1 change
    public string DispatcherGuid => GlobDef.Guid.ToString().ToUpper();

    7 references | Anders Egeberg, 14 days ago | 1 author, 1 change
    public string DispatcherVersion => GlobDef.Version;

    23 references | Anders Egeberg, 14 days ago | 1 author, 1 change
    public string DispatcherId { get; set; }

    9 references | Anders Egeberg, 13 days ago | 1 author, 2 changes
    public IDigitalPostSender ConfigureSender(ODataService odataService, string accessCode, IEnumerable<CustomParameter> parameters = null)
    {
        IDictionary<string,string> arguments = new Dictionary<string, string>();
        arguments.Add(ExchangeSettings.ExchangeMailbox.GetSetting(odataService, DispatcherId));
        arguments.Add(ExchangeSettings.ExchangeServerUri.GetSetting(odataService, DispatcherId));
        arguments.Add(ExchangeSettings.ExchangeUserDomain.GetSetting(odataService, DispatcherId));
        arguments.Add(ExchangeSettings.ExchangeUserName.GetSetting(odataService, DispatcherId));
        arguments.Add(ExchangeSettings.ExchangeUserPassword.GetSetting(odataService, DispatcherId));
        return new Sender(odataService, arguments);
    }
}
```

In the Exchange dispatcher example, the **SenderConfigurator** class both implements **IDigialPost** and **IDigitalPostSenderConfigurator**. The **ConfigureSender** method reads the settings into a dictionary and passes this dictionary to the constructor of the **ConfiureSender** class (this class implements **IDigtalPostSender**) together with the **oDataService**.

Implement IDigitalPostSender interface

The class that implements **IDigitalPostSender** is the class that has the logic for dispatching messages and validating the dispatcher configuration as well as the

dispatches before dispatching to a specific recipient.

Methods

Name	Description
SendMessage	Sends a message to a recipient. A prebuilt message is handed over to the method but it is also possible to use other data about the recipient, case, or document using the keys supplied in the arguments. The **oDataService** supplied in the **DigitalPostSenderConfigurator::ConfigureService** can be used for querying the WorkZone database. The arguments for the method are: • message—the message that has been built for dispatching. • recipientAddress—a postal address built from the address data in WorkZone. • fileKey—the file key that the workflow has been started from. • recordKey—the record key that contains the letter to be sent. • addressKey—the address key of the recipient. • parameters—extra parameters supplied to the SmartPost process at start-up. The method must return an instance of a class that implements the **IMessageShipmentHandle** interface. This value can be used by the dispatcher framework to query for the state of the dispatch using the **GetMessageShipmentState** method.

Name	Description
GetMessageShipmentState	Returns the state of the message from the dispatcher client. • handle is a dispatch handle formerly returned from **SendMessage**. The return value is an instance of a class that implements the **IMessageShipmentState** interface.
CanSend	Validates if the dispatcher is able to send messages using the current configuration. For example, if the dispatcher needs a network connection to send a message, this method will validate that the connection is successful. This method is called before the SmartPost process is started to prevent starting a SmartPost process that is not able to send messages. The return value is **ValidationResult** that contains a set of errors.
CanDispatchToUser	Validates if it is possible to send a document (record) to a specific user. The arguments are: • recordIds of the documents that are planned to be sent. • addressKey of the recipient. • accessCode (not used). • customParameters of the SmartPost process. Return value is a **ValidationResult** that contains a set of errors.

Properties

Name	Description	Type
SenderAddress	The dispatcher framework adds the **SenderAddress** to the message object that is passed to **SendMessage**.	PartyAddress including postal address.

Exception handling

If an exception is thrown during the dispatch, the dispatcher framework resolves the action to take by comparing the exception with the values in the **wzp_filtering_entry** table. If a matching entry is found, the row in **wzp_error_manager** determines how to act on the error. Possible actions are:

- **TryNextSource**—It is not possible to send to this user. If there is another dispatcher in the current dispatcher sequence, it should be used instead.
- **NotifyUserAndAbort**—The dispatch failed and it should not be sent using another dispatcher.
- **Retry**—A temporary communication error occurred. Retry is done after a delay (the number of retries and the delay is also specified in the table).

Implement IMessageShipmentHandle interface

This interface is used by the framework to keep track of messages that have been sent.

Properties

Name	Description	Type
MessageId	A message ID used to identify the message.	String

Implement IMessageShipmentState interface

The **IMessageShipmentState** interface is used to get the state of a message that has been sent. It is assumed that the dispatcher implementation can retrieve information about the progress of the dispatch to the recipient.

Properties

Name	Description	Type
ProcessState	The current state of the dispatch.	MessageShipmentProcessState
Text	A description of the state that can be used in logs and reports.	String

Use of ValidationResult class

The **ValidationResult** class is used as return values on the **CanDispatchToUser** and **CanSend** methods on the **IDigitalPostSender**. The methods and properties on the class cannot be overridden.

The errors added to a **ValidationResult** are localized in the database. To create an error message that can be displayed to the user, the code must be added to **wzp_error_message** and localized versions of the message must be added to **wzp_error_message_lang**.

Properties

Name	Description	Type
IsValid	Used by the dispatcher framework to determine if the result is OK. If no errors are added, the result is regarded as valid.	Bool
IsFatal	If this property is true, the dispatch will fail even if the dispatch sequence includes another	Bool

Name	Description	Type
	dispatcher.	
Errors	The list of errors that prevents the dispatcher from sending the message.	IEnumerable<MessageEntry>

Methods

Name	Description
Add (ValidationResult)	Merges a validation result into the current instance.
Add (errorCode,parameters)	Used to add a new error message to the list of errors. • errorCode—An enum value from any enumeration. This value must exist in the WorkZone database in the **wzp_user_message** table (in uppercase). • parameters—An array of values that can be merged into the localized message.

Example of definition and use of error message

The exchange example dispatcher has defined an enumeration of errors called CustomInfoEnum.

```
public enum CustomInfoEnum
{
    NoEmailAddress,
    IllegalEmailAddress,
    UnresolveableEmailAddress,
    NoSenderEmail,
    IllegalSenderEmailAddress,
    UnresolveableSenderEmailAddress
}
```

The SQL script `messages.sql` inserts the values in wzp_user_message and wzp_user_message_lang. This is the part of the script where NoEmailAddress is defined:

```
--NOEMAILADDRESS
delete from WZP_USER_MESSAGE_LANG where code = 'NOEMAILADDRESS';
delete from WZP_USER_MESSAGE where code = 'NOEMAILADDRESS';

insert into wzp_user_message(CODE)
select 'NOEMAILADDRESS' from DUAL;

insert into WZP_USER_MESSAGE_LANG(ROW_ID, CODE, MESSAGE, CULTURE_NAME, CULTURE_SOURCE)
select WZP_USER_MESSAGE_LANG$ROW_ID.nextval, 'NOEMAILADDRESS', 'Modtageren med adresselbnr {0} har ikke nogen email adresse', 'da-DK', 'da-DK' from DUAL;

insert into WZP_USER_MESSAGE_LANG(ROW_ID, CODE, MESSAGE, CULTURE_NAME, CULTURE_SOURCE)
select WZP_USER_MESSAGE_LANG$ROW_ID.nextval, 'NOEMAILADDRESS', 'The recipient does with addresskey {0} does not have an email address', 'en-GB', 'en-GB' from DUAL;
```

Note that the error message contains a `'{0}'` that indicates where the ID of the recipient is placed.

In the CanDispatchToUser method, it is validated if the user has an email address – otherwise the error message is added.

```
public ValidationResult CanDispatchToUser(List<string> recordIds, string addressKey, string accessCode = null, IEnumerable<CustomParameter> customParameters = null)
{
    var result = new ValidationResult();
    result.IsFatal = false;

    var email = GetEmail(addressKey);
    if (string.IsNullOrWhiteSpace(email)) result.Add(CustomInfoEnum.NoEmailAddress, new[] { addressKey });
    else if (!IsValidEmail(email)) result.Add(CustomInfoEnum.IllegalEmailAddress, new[] {email});

    return result;
}
```

Implement DigitalPostDispatcherConfigurator class

If the dispatcher has settings that a user should be able to maintain in the WorkZone Configurator, you must create a **DigitalPostDispatcherConfigurator** class in the dispatcher. This class does not implement a specific interface but must expose these 4 properties:

Properties

Name	Description	Type
Id	The GUID of the dispatcher.	String
Name	The Name of the dispatcher.	String
Version	The version number string of the dispatcher	String
Parameters	The parameters that can be configured for the dispatcher. **Parameters** is an enumeration of tuples of 5 values that describes each	IEnumerable< Tuple<string,bool,string,string,string,string> >

Name	Description	Type
	configuration parameter: 1. Name of the setting (string) 2. Is the setting required (bool) 3. Type of the setting value (string) 4. Default value (string) 5. Description (string) The description is shown when you hover the mouse over the question mark in the WorkZone Configurator.	

The **DigitalPostDispatcherConfigurator** class is recognized by the dispatcher framework in a slightly different way than the other dispatcher classes. It must Export a string (not a type) with the value "DigitalPostDispatcherConfigurator".

Example: The sample Exchange dispatcher has 6 settings that are declared as shown below.

```
[Export(DigitalPostDispatcherConfigurator)]
0 references | Anders Egeberg, 14 days ago | 1 author, 2 changes
public class DispatcherConfigurator
{
    private const string DigitalPostDispatcherConfigurator = nameof(DigitalPostDispatcherConfigurator);

    private readonly List<Tuple<string, bool, string, string, string>> _parameters = new List<Tuple<string, bool, string, string, string>>
    {
        new Tuple<string, bool, string, string, string> (ExchangeSettings.ExchangeMailbox.ToString(), true,
            "STRING","foo@foo.com", "Mailbox to send from"),
        new Tuple<string, bool, string, string, string> (ExchangeSettings.ExchangeUserPassword.ToString(), true,
            "STRING","Password", "Password to access the mailbox"),
        new Tuple<string, bool, string, string, string> (ExchangeSettings.ExchangeServerUri.ToString(),false, "STRING","",
            "url end-point for the exchange asmx service. If empty autodiscover will be used to resolve the endpoint."),
        new Tuple<string, bool, string, string, string> (ExchangeSettings.ExchangeUserName.ToString(),false, "STRING","",
            "Name of the user account used for accessing the exchange service. If empty the mailbox will be used."),
        new Tuple<string, bool, string, string, string> (ExchangeSettings.ExchangeUserDomain.ToString(), false, "STRING","",
            "Domain of the user account used for accessing the exchange service. If empty the mailbox will be used.")

    };

    0 references | Anders Egeberg, 15 days ago | 1 author, 1 change
    public string Id => GlobDef.Guid.ToString();

    0 references | Anders Egeberg, 15 days ago | 1 author, 1 change
    public string Name => GlobDef.Name;

    0 references | Anders Egeberg, 15 days ago | 1 author, 1 change
    public string Version => GlobDef.Version;

    0 references | Anders Egeberg, 15 days ago | 1 author, 1 change
    public IEnumerable<Tuple<string, bool, string, string, string>> Parameters => _parameters;
}
```

The **SenderConfigurator** class (that implements **IDigitalPostSenderConfigurator**) reads the setting values by calling DispatcherUtility.GetSetting(oDataService, enumValue.ToString(), dispatcherId) using an extension method.

8.4 Deploy a SmartPost dispatcher

You deploy the dispatcher by creating a WorkZone package that contains the dispatcher assembly and a package.xml file that describes the package.

The example exchange dispatcher has a package.xml as a project file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Package>
  <PackageDefinition>
    <Name>Exchange dispatcher</Name>
    <Version>20.0.5.0</Version>
    <Description>Package contains exchange dispatcher.</Description>
  </PackageDefinition>
  <Assemblies>
    <AssemblyDefinition>
      <DispatchFile>Assemblies\WorkZone.Dispatcher.Exchange_20.0.5.0.dll</DispatchFile>
      <PdbFile>Assemblies\WorkZone.Dispatcher.Exchange_20.0.5.0.pdb</PdbFile>
    </AssemblyDefinition>
  </Assemblies>
</Package>
```

This build target in the project file copies the binaries and the package, and zips the result into ExchangeDispatcer.wzp:

```
<Target Name="PostBuildEvent">
  <Copy SourceFiles="$(ProjectDir)package.xml" DestinationFolder="$(ProjectDir)package" />
  <Copy SourceFiles="$(TargetDir)WorkZone.Dispatcher.Exchange_20.0.5.0.dll;$(TargetDir)WorkZone.Dispatcher.Exchange_20.0.5.0.pdb" DestinationFolder="$(ProjectDir)package\assemblies" />
  <Exec Command="powershell.exe -noprofile Compress-Archive -DestinationPath '$(ProjectDir)ExchangeDispatcher.zip' -Path '$(ProjectDir)package\*'" />
  <Exec Command="powershell.exe -noprofile Move-Item -Path '$(ProjectDir)ExchangeDispatcher.zip' -Destination '$(TargetDir)ExchangeDispatcher.wzp' -Force" />
</Target>
```

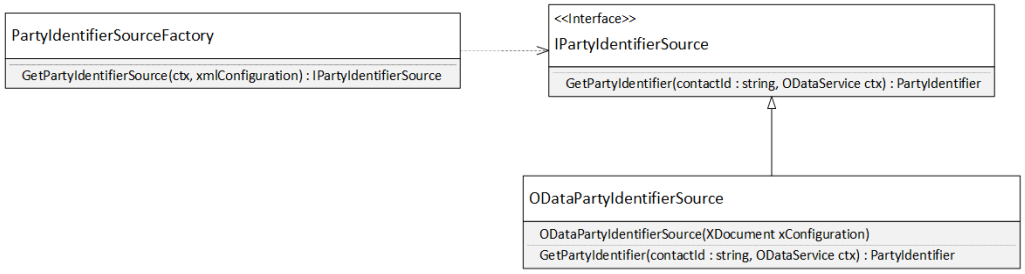
8.5 Configure SmartPost PartyIdentifierSources

SmartPost uses **PartyIdentifierSources** instance to look up sender information such as CVR or CPR numbers.

This topic describes the implementation and function of the **PartyIdentifierSources** instance and how you can customize the configuration and/or extend it.

Design

The diagram below shows the core types which constitute the design of the **Party IdentifierSources**. The design complies with the factory design pattern.



Factory pattern implementation

IPartyIdentifierSource is the interface that exposes the specific implementation of a **Party Identifier Source** instance. The method **GetPartyIdentifier** will be called by the application, when a party identifier (CPR or CVR number) is to be retrieved from a contact identifier (contactID). The OData service context is provided as a parameter to support database access in the implementation.

ODataPartyIdentifierSource is a concrete implementation of the **IPartyIdentifierSource** interface. See ODataPartyIdentifierSource for more details.

The **PartyIdentifierSourceFactory** can create instances, which implement the **IPartyIdentifierSource** interface. The actual construction is done through a

configuration, which is described in Configuration of the GetPartyIdentifier method. The configuration is provided to the factory as an XML element.

Utility method(GetPartyIdentifier)

The application requests that the factory provides a **Party Identifier Source**, which the application can then use to retrieve the party identifier from a contact identifier. The party identifier is retrieved in various ways depending on the name type of it. It is therefore useful to have one **Party Identifier Source** instance per name type that can provide a party identifier.

All this is contained in the static **GetPartyIdentifier(string, ODataService)**method in the **IdentifierSourceUtilities** class. Figure 2 shows an example of how this method can be used.

Example: Use of the GetPartyIdentifier method in the IdentifierSourceUtilities class

```
private void WritePartyIdentifier(string nameKey)
{
    var ctx = GetODataService();
    var contact = ctx.Contacts.Where(o => o.NameKey == nameKey).SingleOrDefault();
    if (contact == null)
    {
        Console.WriteLine("No such contact!");
    }
    else
    {
        var partyIdentifier = IdentifierSourceUtilities.GetPartyIdentifier(contact, ctx);
        if (partyIdentifier == null)
        {
            Console.WriteLine("The contact does not have a CPR or CVR number");
        }
        else
        {
            Console.WriteLine($"The contact contains a {partyIdentifier.IdentifierType} with the code '{partyIdentifier.Token}'.");
        }
    }
}
```

Configuration of the GetPartyIdentifier method

The method is configured by the XML specified in the **Process settings** in WorkZone Configuration Management (**Operation** > **Process Settings**) or in the WZP_SETTING entity named **PartyIdentifierSources** (the module name is “WorkZone”). This XML is read and interpreted by the **GetPartyIdentifier** method. See Utility method (GetPartyIdentifier).

Example: The standard configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<party-identifier-sources>

                <party-identifier-source name-type="C"
class="WorkZone.Dispatcher.Base.ODataPartyIdentifierSource">
                                <register-name>Contacts</register-name>
                                <query-template>?$filter=ID eq
'{}'&amp;$select=NameCode,NameType_Value</query-template>
                                <field-name>NameCode</field-name>
                </party-identifier-source>

                <party-identifier-source name-type="E"
class="WorkZone.Dispatcher.Base.ODataPartyIdentifierSource">
                                <register-name>Contacts</register-name>
                                <query-template>?$filter=ID eq
'{}'&amp;$select=NameCode,NameType_Value</query-template>
                                <field-name>NameCode</field-name>
                </party-identifier-source>

                <party-identifier-source name-type="J"
class="WorkZone.Dispatcher.Base.ODataPartyIdentifierSource">
                                <register-name>Contacts</register-name>
                                <query-template>?$filter=ID eq
'{}'&amp;$select=TaxIdNo,NameType_Value</query-template>
                                <field-name>TaxIdNo</field-name>
                </party-identifier-source>

                <party-identifier-source name-type="V"
class="WorkZone.Dispatcher.Base.ODataPartyIdentifierSource">
                                <register-name>Contacts</register-name>
                                <query-template>?$filter=ID eq
'{}'&amp;$select=TaxIdNo,NameType_Value</query-template>
                                <field-name>TaxIdNo</field-name>
                </party-identifier-source>

</party-identifier-sources>
```

The XML is interpreted as described below.

The following applies to the entire XML:

- No namespaces are considered.
- No schema is specified since the XML details depends on the implementations of the **Party Identifier Sources**.

For the root element, the following rules applies:

- The name of the root element is not important.
- All attributes on the root element are ignored.

The child elements of the root element are searched for the element with an attribute named **name-type** and which value matches the string parameter to the **GetPartyIdentifier** method. If no such match was found, then the method returns null. Otherwise the found child element is parsed to the factory, which now is responsible of creating a **Party Identifier Source** for that name type based on the XML element.

Configuration of a factory

As mentioned above, the XML element (configuration element) that matches the requested name type is parsed to the factory, in order to make it produce a **Party Identifier Instance**. The factory does this by reading the class attribute from the configuration element. The value of the class attribute must be the fully qualified class name of the requested **Party Identifier Source** implementation and the class must implement the interface of the **IPartyIdentifierSource**.

The factory then searches the class for a constructor, which matches one of the following signatures:

- ctor(ODataService, XElement)
- ctor(XElement, ODataService)
- ctor(XElement)
- ctor(ODataService)
- ctor()

Where ODataService (FQCN = Scanjour.Process.OData.Client.Proxy.ODataService) is an OData access to the database and XElement (FQCN = System.Xml.Linq.XElement) is the XML element found by the factory.

The search is performed in the order shown above. Whenever a constructor is found, the parameters are provided and the constructor is called, so the **Party Identifier Source instance** is created and eventually returned by the factory.

The ODataService makes it possible for the **Party Identifier Source** constructor to search for additional information in the database.

The XElement can be used to retrieve implementation-specific configuration to the constructor.

ODataPartyIdentifierSource

The **ODataPartyIdentifierSource** class is a general-purpose implementation of the **IPartyIdentifierSource** interface.

The **ODataPartyIdentifierSource** can access any register in the database that is made available through OData. The register, the query, and where the value for the **Party Identifier** are described below.

Configuration

The **ODataPartyIdentifierSource** is configured by the XML element, which is provided by the factory.

Example: A configuration example of the **ODataPartyIdentifierSource**

```
<party-identifier-source name-type="C" class="WorkZone.Dispatcher.Base.ODataPartyIdentifierSource">
                <register-name>Contacts</register-name>
                <query-template>?$filter=ID eq '{}'&amp;$select=NameCode,NameType_Value</query-template>
                <field-name>NameCode</field-name>
</party-identifier-source>
```

The attributes on the **party-identifier-source** are not used by the class, but have already been used by the factory. The XML element works more like a placeholder for the three inner XML elements.

It is the three inner XML elements, which configure the **ODataPartyIdentifierSource**.

register-name	The name of the register on which the OData query will take offset.
query-template	The template which is used to form the query. When the **GetPartyIdentifier(string, ODataService)** method is invoked, then two empty curled braces ({}) will be replaced by the contact identifier, which is the first parameter in the method.
field-name	The name of the field in the result which content will be returned by the method. The field is expected to contain the code of the Party Identifier. E.g. NameCode.

The final OData query will be formed in the following way:

```
{base-uri}{register-name}{partial-query}
```

Where

`{base-uri}` is the URI to the data source – for example, http://db01/OData/

`{register-name}` - The content of the register-name element, for example Contacts.

`{partial-query}` - The content of the query-template element after the curled braces has been replaced by the name key. For example a query template can be:

```
?$filter=ID eq '{}'&$select=NameCode,NameType_Value
```

Remember that & in XML must be written as `&` - see example above.

If the name key is 36, the `{partial-query}` will then be:

```
?$filter=ID eq '36'&$select=NameCode,NameType_Value
```

Based on the above examples the final query will be:

```
http://db01/OData?$filter=ID eq '36'&$select=NameCode,NameType_Value
```

From the result of the query, the **NameCode** of the first entity will be used as **Party Identifier**.

Customized implementation

If the provided **ODataPartyIdentifierSource** is insufficient for making a customization, then a customized implementation will probably solve it.

To do so, you must make an assembly containing your customized **Party Identifier Source**. Add the assembly to the WorkZone Process package, and change the configuration in WZP_SETTINGS, so your class is used by the factory to create your **Party Identifier Source**.

Follow these steps:

1. Create a Class Library project for the purpose. Beware of dependencies to other projects.
2. Make the project reference the **WorkZone.Dispatcher.Base** assembly.
3. In your project, create a file containing an empty class.

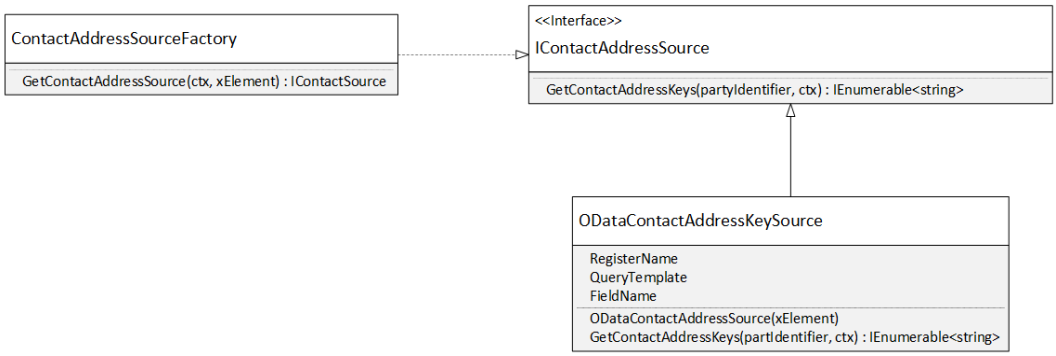
4. Make your file use the **WorkZone.Dispatcher.Base** namespace.
5. Make the class implement the **IPartyIdentifierSource** interface.
6. Make a constructor to the class, that complies to one of the constructors described in Configuration of a factory.
7. If required, then use the constructor to retrieve configuration information from the XML element or directly from the database using the ODataSerivice provided.
8. Implement the **GetPartyIdentifier** method, so it complies with the interface.
9. Write some tests that verifies your implementation.
10. Compile and include your assembly in the WorkZone Process package.
11. Change the configuration in WorkZone Configuration Management or WZP_SETTINGS so that your new Party Identifier Source is used by the correct name type(s) and to make the constructor receive the correct XML element (if required).
12. Use Visual Studio to generate a new assembly with an updated set of proxy classes. The content of the new assembly must take offset in your customized data dictionary.
13. Make your installation substitute the existing assembly with the newly generated assembly. Do this by copying the new assembly to "C:\Program Files (x86)\KMD\WorkZone\Process\Web\Services\Bin"
14. Make an IISRESET.
15. Test your creation.

8.6 Configure SmartPost ContactAddressSources

SmartPost uses a **Contact Address Sources** instance when a message is received from e-Boks, and the sender must be associated with the document (record) that is

created at receipt of the SmartPost message. The connection between the message and the sender is made through addresses. The **Contact Address Sources** instance and its configuration point out the addresses that identify the sender and associate the sender with the document.

This topic describes the implementation and function of the



Contact Address Sources instance and how to customize it by configuring or extending it.

Design

The diagram shows the core types which constitute the design of the **Contact Address Source** instance. The design complies to the factory design pattern.

Factory pattern implementation

IContactAddressSource is the interface that exposes the specific implementation of a **Contact Address Source** instance. The **GetContactAddressKeys** method will be called by the application, when a set of keys for addresses of the contacts is to be retrieved from a party identifier (CVR or CPR numbers). The OData service context is provided as a parameter to support the database access from the implementation.

ODataContactAddressSource is a concrete implementation of the **IContactAddressSource** interface. See ODataContactAddressSource for more details.

The **ContactAddressSourceFactory** can create instances, which implement the **IContactAddressSource** interface. The actual construction is done according to a

configuration, which is described in Configuration of the GetContactAddressKeys method. The configuration is provided to the factory as an XML element.

Utility method (GetContactAddressKeys)

The application can ask the factory to provide a **Contact Address Source** instance, and then the application can use the **Contact Address Source** instance to retrieve the address keys that are related to a party identifier. The way contact addresses are retrieved varies depending on the party identifier type, for example CVR or CPR, and there it is useful to have at least one set of **Contact Address Source** instances per party identifier type where each can provide a set of address keys.

All this is contained in the static **GetContactAddressKeys**(PartyIdentifier, ODataService) method in the **IdentifierSourceUtilities** class.

Example: Use of the **GetContactAddressKeys** method in the **IdentifierSourceUtilities** class

```
private void WriteContactAddressKeys(PartyIdentifier partyIdentifier)
{
    var ctx = GetODataService();
    var contactAddressKeys = IdentifierSourceUtilities.GetContactAddressKeys(partyIdentifier, ctx);

    Console.WriteLine($"The party {partyIdentifier} refers to the following address keys:");
    foreach(var contactAddressKey in contactAddressKeys)
    {
        Console.WriteLine(contactAddressKey);
    }
}
```

Configuration of the GetContactAddressKeys method

The method is configured by the XML specified in the **Process settings** in WorkZone Configuration Management (**Operation** > **Process Settings**) or in the WZP_SETTING entity named **ContactAddressKeySources** (the module name is "WorkZone"). This configuration is XML that is read and interpreted by the **GetContactAddressKeys** method, see Utility method (GetContactAddressKeys).

Example: The standard configuration of the factory

```
<contact-address-sources>

  <contact-address-source party-identifier-type="CVR" class="WorkZone.Dispatcher.Base.ODataContactAddressSource">
    <register-name>Addresses</register-name>
    <query-template>?$filter=Name/TaxIdNo eq '{}'&amp;$select=ID</query-template>
    <field-name>ID</field-name>
  </contact-address-source>

  <contact-address-source party-identifier-type="CVR" class="WorkZone.Dispatcher.Base.ODataContactAddressSource">
    <register-name>Addresses</register-name>
    <query-template>?$filter=Name/NameCode eq '{}'&amp;$select=ID</query-template>
    <field-name>ID</field-name>
  </contact-address-source>

  <contact-address-source party-identifier-type="CPR" class="WorkZone.Dispatcher.Base.ODataContactAddressSource">
    <register-name>Addresses</register-name>
    <query-template>?$filter=Name/NameCode eq '{}'&amp;$select=ID</query-template>
    <field-name>ID</field-name>
  </contact-address-source>

</contact-address-sources>
```

The XML is interpreted as described below.

The following applies to the entire XML:

- No name spaces are considered.
- No schema is specified because the XML details are up to the actual implementations of the **Contact Address Sources** instance.

For the root element, the following rules applies:

- The name of the root element is not important.
- All attributes on the root element are ignored.

When the **GetContactAddressKeys** method is invoked, the XML is interpreted and the method is executed in the following way:

1. **Selection of child elements**

   The child elements of the root element are searched for the element having an attribute named **party-identifier-type**, which is compared to identifier type (CVR or CPR) of the provided party identifier. The comparing is case insensitive. If the content of the **party-identifier-type** attribute matches the provided party identifier, then the child element is accepted. All other child elements are ignored.

2. **Creation of Contact Address Sources** instance

For each of the accepted child elements a **Contact Address Source** instance is created. See Configuration of a factory.

3. **Invocation of created Contact Address Sources instance**

   When the instance has been created, the **GetContactAddressKeys** method is invoked, which returns a set of address keys. The set of address keys depends in the implementation.

4. **Collection of address keys**

   Because several **Contact Address Source** instances can be created and invoked in the same call to the **GetContactAddressKeys** method in the **IdentifierSourceUtilities** class, several non-empty sets of address keys can be returned. The **GetContactAddressKeys** method collects the address keys contained in these sets in a `System.Generic.Collections.HashSet<string>`. In this way, it is ensured that the same address keys remain unique.

5. **The final result**

   Finally, the **GetContactAddressKeys** method returns an enumeration of the collected address keys.

Configuration of a factory

As mentioned in section 2.3, the XML element (configuration element) that matches the requested party type is parsed to the factory, to make it produce a **Contact Address Source** instance. The factory does that by reading the class attribute from the configuration element. The value of the class attribute must be the fully qualified class name of the requested **Contact Address Source** implementation, and the class must implement the **IContactAddressSource** interface.

The factory then searches the class for a constructor, which matches one of the following signatures:

- ctor(ODataService, XElement)
- ctor(XElement, ODataService)
- ctor(XElement)
- ctor(ODataService)
- ctor()

Where `ODataService (FQCN = Scanjour.Process.OData.Client.Proxy.ODataService)` is an OData access to the database and XElement (`FQCN = System.Xml.Linq.XElement`) is the XML element found by the factory.

The search is done in the shown order. Whenever a constructor is found, the parameters are provided and the constructor is called, so that the **Contact Address Source** instance is created and eventually returned by the factory.

The ODataService makes it possible for the C**ontact Address Source** constructor to search additional information in the database.

The XElement can be used to retrieve implementation specific configuration to the constructor.

ODataContactAddressSource

The **ODataContactAddressSource** class is a general-purpose implementation of the **IContactAddressSource** interface.

The **ODataContactAddressSource** can access any register in the database that is made available through OData. The register, the query, and where the address keys values are located, are described below.

Configuration

The **ODataContactAddressSource** is configured by the XML element, which is provided by the factory. An example of an XML element for the **ODataContactAddressSource**.

Example: A configuration example of the **ODataContactAddressSource**

```
<contact-address-source party-identifier-type="CPR" class="WorkZone.Dispatcher.Base.ODataContactAddressSource">
  <register-name>Addresses</register-name>
  <query-template>?$filter=Name/NameCode eq '{}'&amp;$select=ID</query-template>
  <field-name>ID</field-name>
</contact-address-source>
```

The attributes on the contact-address-source are not used by the class, but have already been used by the factory. The XML element works more like a placeholder for the three inner XML elements.

It is the three inner XML elements, which configure the **ODataContactAddressSource**.

register-name	The name of the register on which the OData query will take offset.
query-template	The template that is used to form the query. When the **GetContactAddressKeys(PartyIdentifier, ODataService)** method is invoked, then two empty curled braces ({}) will be replaced by the party identifier code, such as the actual CVR or CPR number, which is the first parameter to the method.
field-name	The name of the field on the result, which content will be returned by the method. The field is expected to contain the address keys of the address entity, which is connected to the provided party identifier, for example ID.

The final OData query will be formed in the following way:

```
{base-uri}{register-name}{partial-query}
```

Where

`{base-uri}` is the URI to the data source – for example http://db01/OData/

`{register-name}` - The content of the register-name element, for example Addresses.

`{partial-query}` - The content of the query-template element after the curled braces has been replaced by the name key. For example, a query template can be:

```
?$filter=Name/NameCode eq '{}'&$select=ID
```

(Remember that & in XML must be written as `&` - see Figure 4 for an example.)

If the party identifier code is '180582-3042' then the `{partial-query}` will then be:

```
?$filter=Name/NameCode eq '180582-3042'&$select=ID
```

Taken the above examples the final query will be:

```
http://db01/OData?$filter=Name/NameCode eq '180582-3042'&$select=ID
```

From the result of the query, the ID of all the returned entities will be collected.

Customized implementation

If the provided **ODataContactAddressSource** is insufficient for making a specific customization, then a customized implementation will probably solve it.

To do so you must make an assembly containing your customized **Contact Address Source**. Add the assembly to the WorkZone Process package and change the configuration in `WZP_SETTINGS`, so that your class is used by the factory to create your **Contact Address Source** instance.

Follow these steps:

1. Create a Class Library project for the purpose. Beware of dependencies to other projects.
2. Make the project reference the WorkZone.Dispatcher.Base assembly.
3. In your project create a file containing an empty class.
4. Make your file use the WorkZone.Dispatcher.Base namespace.
5. Make the class implement the **IContactAddressSource-interface**.
6. Make a constructor to class that complies to one of the constructors described in Configuration of a factory.
7. If required, then use the constructor to retrieve configuration information from the XML element or directly from the database using the ODataService provided.
8. Implement the **GetContactAddressKeys** method, so it complies to the interface.
9. Write tests that verifies your implementation.
10. Compile and include your assembly in the WorkZone Process package.

11. Change the configuration in WorkZone Configuration Management or WZP_SETTINGS so your new Contact Address Source is used by the correct party identifier type and so the constructor receives the correct XML element (if required).
12. Use Visual Studio to generate a new assembly with an updated set of proxy classes. The content of the new assembly must take offset in your customized data dictionary.
13. Make your installation substitute the existing assembly with the newly generated assembly. Do this by copying the new assembly to “C:\Program Files (x86)\KMD\WorkZone\Process\Web\Services\Bin"
14. Make an IISRESET.
15. Test your creation.

9. Integration

9.1 Start a SmartPost process using a script

You can start a SmartPost process by calling the Process service and posting arguments in JSON format. You can use the following parameters:

Name	Description	Example
Cred	Credentials of the WorkZone used to invoke the request. If is not supplied, the user is prompted for credentials.	
WzUrl	The main URL to WorkZone.	http://db01
WzODataUrl	The sub URL to the OData site under WorkZone The default value is 'OData'.	'OData'
FileKey	The FileKey of the case to run the SmartPost from	932835
Title	The title of the SmartPost process started, This is the title shown in the Processes overview. If the title is not supplied, the title of the document is used.	
DefinitionId	The unique ID (GUID) of the SmartPost process to use. The default value is '23b9498e-bca5-4746-98a0-71e03cd6963c'.	
Description	Optional argument. Used to supply a description for the SmartPost process.	
Deadline	If a SmartPost message is sent for preview or approval, a deadline can be set. If the deadline for the preview or approval is	

Name	Description	Example
	exceeded, a reminder is sent to the previewer or approver. This argument defines the deadline. The default value is 'tomorrow'.	
Subject	The subject is the title of the message that is handed over to the dispatcher. When sending to e-Boks, the subject is the title of the message shown to the end user. If the subject argument is not supplied the Title will be used.	
RecordID	The RecordID of the document to send. The argument is mandatory.	
AttachmentRecordIds	Contains the recordid's of the attachments. The attached documents must exist in the WorkZone database.	
IsApproval	Switch parameter. This argument indicates that the dispatch will be forwarded to the case handler for approval. `IsApproval` and `IsPreview` are mutually exclusive so only one of them or none of them can be true.	
IsDeleteOriginal	Switch parameter. If supplied, the original document is deleted after dispatch is done.	
Ispreview	Switch parameter. This argument indicates that the messege is sent to the process owner for preview before the dispatch	

Name	Description	Example
	continues. `IsPreview` and `IsApproval` are mutually exclusive so only one of them or none of them can be true.	
RecipientAddressKeys	Addresskeys of the recipients. At least one must be specified.	
CopyRecipientAddressKeys	Addresskeys of the copy recipients.	
DispatcherSequenceId	The ID of the dispatcher sequence used for dispathing the document. The argument is mandatory.	
CloseCase	A switch parameter. If supplied, the case will be closed after dispatch.	
OpenCase	A switch parameter. If supplied and the case is closed, it will be opened before starting the SmartPost process.	
CaseState	The new state of the case, which is applied after the dispatch is done. If not supplied, the state of the case remains unchanged. It must be a legal value in Custom domain 'SAGTILST'.	
Access	Sets an access restriction on the SmartPost process. It must be a legal AccessCode. If the argument is not supplied, the default access code configured for the process is used.	
Importance	Argument used to set the priority of the SmartPost process. The possible values are '1-HIGH', '2-NORMAL' and '3-LOW'. The	

Name	Description	Example
	Default value is '2-NORMAL'.	
CustomDispatcherParameters	This parameter can be used for supplying dispatcher specific parameters as a Hashtable.	
MaterialId	The e-Boks material ID to use for the dispatch. The parameter is only used when using an e-Boks dispatcher and in this case, the argument is mandatory.	
RemotePrintTypeId	The PrintTypeID for the dispatch. The parameter is only used when using a remote print dispatcher and in this case, the argument is mandatory.	

Example:

```
.\Send-SmartPost.ps1 -WzUrl http://xe -FileKey 221 -Title "Test
title" -RecordId 240 -RecipientAddressKeys 301 -
DispatcherSequenceId 70
```

Below is an sample PowerShell script:

```
param(
   [PSCredential]$Cred=(Get-Credential -Message "Workzone user login"),
   [parameter(Mandatory=$True)][string]$WzUrl,
   [parameter(Mandatory=$True)][string]$FileKey,
   [parameter(Mandatory=$True)][string]$Title,
   [string]$DefinitionId="23b9498e-bca5-4746-98a0-71e03cd6963c",
   [string]$Description=$Title,
   [DateTime]$Deadline=[DateTime]::Now.AddDays(1),
   [string]$Subject=$Title,
   [parameter(Mandatory=$True)][string]$RecordId,
   [string[]]$AttachmentRecordIds=@(),
   [switch]$IsApproval,
   [switch]$IsDeleteOriginal,
   [switch]$Ispreview,
   [string[]]$RecipientAddressKeys,
   [string[]]$CopyRecipientAddressKeys,
   [string]$DispatcherSequenceId,
   [switch]$CloseCase,
   [switch]$OpenCase,
   [string]$CaseState,
   [string]$Access,
   [string]$Importance="2-NORMAL",
   [HashTable]$CustomDispatcherParameters=@{},
   [string]$MaterialId,
```

```
   [string]$RemotePrintTypeId
   )

Function Start-WzpProcess
{
   param(
   [PSCredential]$Cred=(Get-ScriptCredential ),
   [string]$WzUrl=(Get-ModuleVar -Name "WzUrl"),
   [string]$EntityType="File",
   [DateTime]$Deadline=([DateTime]::Now.AddDays(1)),
   [parameter(Mandatory=$True)][string]$EntityId,
   [parameter(Mandatory=$True)][string]$DefinitionId,
   [parameter(Mandatory=$True)][string]$Title,
   [string]$Description,
   [parameter(Mandatory=$True)][HashTable]$Properties,
   [string]$Acces,
   [string]$Importance,
   [string]$ParentId,
   [string]$Owner,
   [string]$Subject,
   [switch]$UseHttps
   )

   $UniqParam="1$([System.Random]::new([Datetime]::Now.Millisecond).next
(100000000,999999999))"

   [string]$Uri =
"$($WzUrl)/Process/Process.svc/Processes/${EntityType}/${EntityId}?uniqpa
ram=${UniqParam}"

   $Body=@{
     DefinitionId=$DefinitionId;
     Title=$Title;
     Description=$Description;
     Deadline=$Deadline;
     Properties=$Properties;
     Access=$Acces;
     Importance=$Importance;
     ParentId=$ParentId;
     Owner=$Owner;
     Subject=$Subject} | ConvertTo-Json -Depth 10


   $Result = Invoke-WebRequest -Uri $Uri -Body $Body -Method Post -
Credential $Cred
   return $Result.Content
}
```

```
   if (![string]::IsNullOrWhiteSpace($MaterialId))
{$CustomDispatcherParameters.MaterialId = $MaterialId}
   if (![string]::IsNullOrWhiteSpace($MaterialId))
{$CustomDispatcherParameters.RemotePrintTypeId  = $RemotePrintTypeId }

   $Properties=@{
     Subject=@{Name="Subject";Type="System.String";Value=$Subject};
     RecordId=@{Name="RecordId";Type="System.String";Value=$RecordId};
     AttachmentRecordIds=@
{Name="AttachmentRecordIds";Type="System.String
[]";Value=$AttachmentRecordIds};
     IsApproval=@
{Name="IsApproval";Type="System.Boolean";Value=$IsApproval.ToString()};
     IsDeleteOriginal=@
{Name="IsDeleteOriginal";Type="System.Boolean";Value=$IsDeleteOriginal.
ToString()};
     IsPreview=@
{Name="IsPreview";Type="System.Boolean";Value=$IsPreview.ToString()};
     Deadline=@
{Name="Deadline";Type="System.DateTime";Value=$Deadline};
     CustomDispatcherParameters=@
{Name="CustomDispatcherParameters";Type="System.Collections.Generic.Dic
tionary``2[[System.String, mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089],[System.Object, mscorlib,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]],
mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089";Value=$CustomDispatcherParameters};
     RecipientAddressKeys=@
{Name="RecipientAddressKeys";Type="System.String
[]";Value=$RecipientAddressKeys};
     CopyRecipientAddressKeys=@
{Name="CopyRecipientAddressKeys";Type="System.String
[]";Value=$CopyRecipientAddressKeys};
     DispatcherSequenceId=@
{Name="DispatcherSequenceId";Type="System.String";Value=$DispatcherSequ
enceId};
     CloseCase=@
{Name="CloseCase";Type="System.Boolean";Value=$CloseCase.ToString()};
     CaseState=@
{Name="CaseState";Type="System.String";Value=$CaseState};
     OpenCase=@
{Name="OpenCase";Type="System.Boolean";Value=$OpenCase.ToString()};
   }

   Start-WzpProcess -EntityId $FileKey -DefinitionId $DefinitionId -
Title $Title -Description $Description -Subject $Subject -Properties
$Properties -Importance $Importance -Cred $cred -WzUrl $WzUrl -Acces
$Access
```

10. Web services

The webservice is a WCF webservice which allows the clients to start and communicate with the workflows. The table below describes the operations available with the webservice.

10.1 Creating workflows

Get available processes

1. Get all process definitions to a specific case:

   `http://demo2.connectzone.dk/Process/Process.svc/Definitions/{REGISTER}/{SYSTEMKEY}?tags={TAGS}`

   `KEY` is the key of the specific case.

Get information for starting a process

1. Get the template (JSON object) of the process I want to start.
2. To find `Guid` (definition ID), see Get available processes.

   `http://demo2.connectzone.dk/Process/Process.svc/Definitions({DEFINITIONID})/StartupInfo`

Get a form

1. Use `GET` to start a process on the current case.

   The JSON object from Get information for starting a process is filled in and sent back using POST on the case which has the matching key (`KEY`).

   ```
   http://demo2.connectzone.dk/Process/Process.svc/Definitions
   ({DEFINITIONID})/Form?assetRoot={ASSETROOT}&culture={CULTURE}
   ```

Start a process

1. Use POST to start a process.

   ```
   http://demo2.connectzone.dk/Process/Process.svc/Processes/
   {REGISTER}/{SYSTEMKEY}
   ```

10.2 Workflow service

Webservice

The webservice is a WCF webservice which allows the clients to start and communicate with the workflows. The table below describes the operations available with the webservice.

Operation	Signature	Description
CreateWorkflowFromXml	Guid CreateWorkflowFromXml (string workflowData);	Creates a new workflow instance, based on the parameter `workflowData`. The parameter must be loadable as strings in XmlDocument, and interpreted as a value of the

Operation	Signature	Description
		type `workflowCreationData`.
CreateWorkflow	Guid CreateWorkflow (WorkflowCreationData workflowData);	Creates a new running workflow instance based on `workflowCreationData`. The value returned is a unique identifier for the created workflow instance. `WorkflowCreationData` contains specifications such as workflow type and version.
ResumeBookmark	BookmarkResumptionResult ResumeBookmark(Guid instanceId, string bookmarkName, object value);	This method is invoked by user actions in the client, or the expiration of a process. The parameters it needs is the workflow instance ID, the bookmark ID, and an object value, for example the ID of a proxy user. The value returned indicates whether the call succeeded or not: Success, NotFound, NotReady.
GetWorkflowStatus	WorkflowStatus GetWorkflowStatus(Guid instanceId);	The value returned is the current status of the

Operation	Signature	Description
		workflow: Created, Running, Persisted, Completed or Faulted.

Host

The table below describes the operations available with the host.

Operation	Signature	Description
CreateWorkflowFromXml	`Guid CreateWorkflowFromXml(string workflowData, sjSession session);`	Creates a new workflow instance, based on the parameter `workflowData`. It must be loadable as strings in XmlDocument, and interpreted as a value of the type `workflowCreationData`. The parameter `session` is an impersonated SOM session.
CreateWorkflow	`Guid CreateWorkflow (WorkflowCreationData workflowData, sjSession session);`	Creates a new running workflow instance based on `workflowCreation`

Operation	Signature	Description
		`Data`. The value returned is a unique identifier for the created workflow instance. `WorkflowCreationData` contains specifications such as workflow type and version. The parameter `session` is an impersonated SOM session.
CreateWorkflow	`Guid CreateWorkflow (WorkflowCreationData workflowData, sjSession session, Action<WorkflowApplicationCompletedEventArgs> CompletedCallback)`	The same as the previous CreateWorkflow, but it also includes the paramters CompletedCallback which is called when the workflow is completed.
ResumeBookmark	`BookmarkResumptionResult ResumeBookmark(Guid instanceId, string bookmarkName, object value,`	This method is invoked by user actions in the

Operation	Signature	Description
	`sjSession session);`	client, or the expiration of a process. The parameters it needs is the workflow instance ID, the bookmark ID, and an object value, for example the ID of a proxy user. The parameter `session` is an impersonated SOM session. The value returned indicates whether the call succeeded or not: `Success`, `NotFound`, `NotReady`.
ResumeBookmark	`BookmarkResumptionResult ResumeBookmark(Guid instanceId, string bookmarkName, object value, sjSession session, Action<WorkflowApplicationCompletedEventArgs> CompletedCallback)`	The same as the previous ResumeBookmark, but it also includes the parameter `CompletedCallback` which is called when the workflow is completed.

Operation	Signature	Description
GetWorkflowStatus	`WorkflowStatus GetWorkflowStatus (Guid instanceId);`	The value returned is the current status of the workflow: `Created`, `Running`, `Persisted`, `Completed` or `Faulted`.
Initialize	`void Initialize(sjSession session)`	Initializes the workflow host and loads the data from the db which is necessary for creating workflows. Creates workflow descriptors, that is, the known workflows in the database, type, version, and XAML that describes the workflow types as an internal data structure. In addition, it sets up timers for pending timeouts in currently persisted

Operation	Signature	Description
		workflows.
GetOutputs	`IDictionary<string, object> GetOutputs(Guid instanceId, sjSession session)`	Gets the output argument `IDictionary<string, object>` for a given completed workflow instance. Limitation: outputs can only be gotten for workflows which have been completed by this instance of the workflow host. In all other cases, null will be returned. The reason is that this type of output is not stored in the database.
GetWorkflowInformation	`RunningWorkflowInformation GetWorkflowInformation(Guid instanceId, sjSession session)`	Returns all available information on a workflow instance.
GetKnownWorkflows	`IEnumerable<WorkflowType> GetKnownWorkflows`	Gets all workflow types available,

Operation	Signature	Description
		that is, those workflow types where the host is able to create workflow instances.
InjectWorkflowDescriptor	void InjectWorkflowDescriptor(Activity activity, sjSession session)	Makes an activity known to the workflow host, so that workflow instances can be created from it. The parameter `activity` is the workflow which is to be made known to the host. `session` is a SOM session.
InjectWorkflowDescriptor	void InjectWorkflowDescriptor(string xaml, string typeName, Version version, sjSession session)	Makes a workflow described by XAML known to the workflow host, so that workflow instances can be created from it. The parameter `xaml` is the XAML

Operation	Signature	Description
		string, `typeName` is the type name by which the workflow is to be known. `version` is the versionm, and`session` is a SOM session.

Interface types

The interface types are represented by classes. Below, these classes are described:

- WorkflowCreationData
- WorkflowStatus
- RunningWorkflowInformation
- WorkflowDescriptor
- WorkflowType.

WorkflowCreationData

```
public WorkflowType WorkflowType { get; private set; }
```

`WorkflowType` gets the workflow type.

```
public string WorkflowTitle { get; private set; }
```

`WorkflowTitle` gets the workflow title.

```
public string WorkflowParent { get; private set; }
```

`WorkflowParent` gets the parent workflow?

```
    public string ProcessId { get; private set; }
```

`ProcessID` gets the ID of the running process.

```
    public string Description { get; private set; }
```

`Description` gets the description of the process.

```
    public string AssociatedRegister { get; private set; }
```

`AssociatedRegister` gets the name of the case register, record (document), contact or address.

```
    public string AssociatedRegisterKey { get; private set; }
```

`AssociatedRegisterKey` gets the register key.

```
    public DateTime EndDate { get; private set; }
```

`EndDate` gets the workflow end date.

```
    public IDictionary<string, object> Arguments { get; private set; }
```

`IDictionary` gets the values of the workflow in arguments.

WorkflowStatus

The table describes the values that WorkflowStatus can get:

Value	Description
Created	The workflow instance has been created but is not yet running.
Running	The workflow instance is running.

Value	Description
Persisted	The workflow instance is idle and persisted.
Completed	The workflow instance is completed.
Faulted	The workflow instance has been terminated by an unhandled exception.

RunningWorkflowInformation

```
public WorkflowDescriptor Descriptor { get; }
```

`Descriptor` gets the descriptor of a running workflow. See description of WorkflowDescriptor.

```
public string Owner { get; }
```

`Owner` gets the name code of the process owner.

```
public WorkflowStatus Status { get; }
```

`Status` gets the status of the workflow. See description of WorkflowStatus.

```
public Exception ExceptionThrown { get; }
```

`ExceptionThrown` gets the exception description when the workflow is faulted.

```
public IDictionary<string, object> Outputs { get; }
```

`IDictionary<string, object>` gets the output argument for a given completed workflow instance.

```
public DateTime PendingTimerExpiration { get; }
```

`PendingTimerExpiration` gets the expiration date and time of a pending workflow, if a timer is running for the workflow.

```
public bool TimerRunning { get; private set; }
```

`TimerRunning` tells if there is a timer on the workflow, and if it is not yet expired.

```
public string ServerName { get; private set; }
```

`ServerName` gets the name of the server that the host will be running on if the timer on the workflow expires.

WorkflowDescriptor

```
public Activity WorkflowType { get; private set; }
```

`WorkflowType` gets the workflow type as a .NET System.Activities.Activity.

```
public Version Version { get; private set; }
```

`Version` gets the version of the workflow as a .NET System.Version.

```
public string WorkflowKey { get; private set; }
```

`WorkflowKey` gets the key in the `workflow` register.

```
public string AssemblyKey { get; private set; }
```

`AssemblyKey` gets the workflow assembly key in the `workflow_assembly` register.

WorkflowType

```
public string TypeName { get; private set; }
```

`TypeName` gets the name of the workflow type.

```
    public Version Version { get; private set; }
```

`Version` gets the version of the workflow.

10.3 OData actions

OData custom actions are implemented on the following registers:

WzpWorkflowInstance

WzpUserTask

WzpWorkflowInstance

The following custom actions are used:

- **Promote**: Available on phase workflows and will promote to the next phase.
- **Demote**: Available on phase workflows and will demote to the previous phase.
- **Cancel**: Available on any workflow and will cause the workflow to complete.

WzpUserTask

The following custom actions are available on any workflow implementing a `user_task`:

- **Approve**: Approves the user task.
- **Reject**: Rejects the user task.
- **Skip**: Skips the user task.
- **Forward (string nameType, string nameCode)**: Creates a new user task for the contact with (`name_type` and `name_code`).
- **Reschedule (DateTime dueDate)**: Changes the user task due date to the specified due date.

Usage of oData custom actions

Usage in C# using the Scanjour.Process.Odata.Client

Given an entity of a `WzpWorkflowInstance` the actions can be issued from C# code in the following way:

```
string id = 3301;

WzpWorkflowInstance instance = (from i in ctx.WzpWorkflowInstances where i.ID
== id select i).Single()

if (ctx.IsActionAvailable(instance, "Promote") ctx.ExecuteAction(instance,
"Promote", null);
```

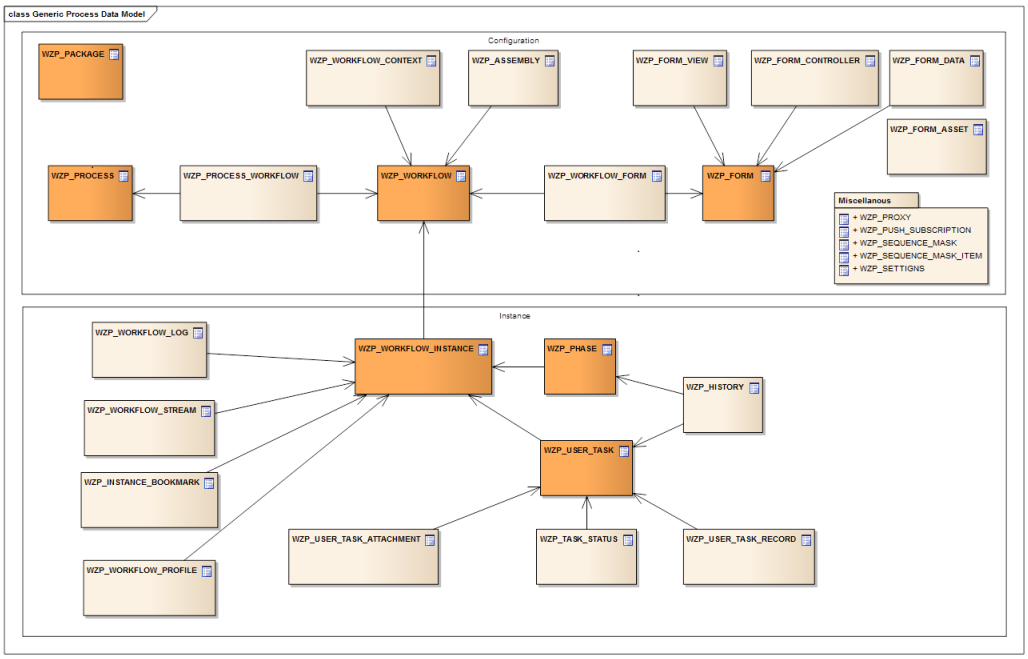
Usage in JavaScript

Read more about the usage of OData actions here:

http://msdn.microsoft.com/en-us/library/hh859851(v=vs.103).aspx

11. Database

The WorkZone Process data model is illustrated by the following diagram.



Click here to open a PDF version of the data model in a new window.

11.1 Process configuration registers

WZP_PACKAGE

Description: Defines installed packages in the system. New versions of the same package overwrites the package definition.

Table	Type	Description
WZP_PACKAGE	Main	Defines the packages known to the system.

WZP_PROCESS

Description: Defines processes in the packages.

Table	Type	Description
WZP_PACKAGE	Parent	The package the process originates from.
WZP_PROCESS	Main	The process definition.
WZP_PROCESS_ WORKFLOW	Child	Mapping between process and workflow. Workflows have a start and an end date. Only one workflow can be active at a given time.
WZP_PROCESS_ PARAMETER	Child	Parameter required for a process. Currently used for service processes.

Domains:

Table	Description
Name	**Description**
Active	All active processes.
Active service	All active service processes.
Active workflow	All active processes which are not service processes.

WZP_WORKFLOW

Description: Defines known workflows in the packages.

Table	Type	Description
WZP_WORKFLOW	Main	The workflow definition with Xaml code.
WZP_PACKAGE	Extension	The package that the workflow originates from.
WZP_WORKFLOW_CONTEXT	Extension	The context in which the workflow is valid.
WZP_WORKFLOW_PROFILE	Extension	The profile used to log workflow execution.
WZP_WORKFLOW_FORM	Child	Forms used by the workflow.
WZP_WORKFLOW_INSTANCE	Child	Actual workflow instances.

WZP_PROCESS_WORKFLOW

Description: Defines mapping between processes and workflows. Workflows have a start and an end date. Only one workflow can be active at a given time

Table	Type	Description
WZP_PROCESS	Parent	The process.
WZP_WORKFLOW	Parent	The workflow.
WZP_PROCESS_WORKFLOW	Main	The mapping between processes and workflows.

Domains:

Name	Description
Active	All active processes.
Active service	All active service processes.
Active workflow	All active processes which are not service processes.

WZP_PROCESS_PARAMETER

Description: Defines the parameters used by a process. Used when service processes are defined.

Table	Type	Description
WZP_PROCESS	Parent	The process that the parameter is defined for.
WZP_PROCESS_ PARAMETER	Main	The parameter definitions.

WZP_SERVICE

Description: Defines the services process instances. Setup in CCM Operation/Process services.

Table	Type	Description
WZP_PROCESS	Parent	The process.
WZP_SERVICE	Main	The service.
WZP_SERVICE_ PARAMETER	Child	The service parameters.

WZP_SERVICE_PARAMETER

Description: Parameter values to defined service processes.

Table	Type	Description
WZP_SERVICE	Parent	The service.
WZP_SERVICE_ PARAMETER	Main	The service parameters.

WZP_ASSEMBLY

Description: Defines assemblies used by a package. Downloaded at startup by the workflow host.

Table	Type	Description
WZP_PACKAGE	Parent	The package that the assembly belongs to.
WZP_ASSEMBLY	Main	The assemble and assembly code

11.2 Process configuration tables

WZP_PACKAGE

Description: Package definitions.

Column	Label	Description
package_id	ID	Unique ID.
name	Package name	Package name required="y".
version	Package version	Package version required="y".
description	Package description	Package description.
tracking profile	Tracking profile	The package tracking profile xml code.
created	Created	Creation date/time. Automatically created.
create user	Created by	Created by.
updated	Modified	Update time. Automatically created.
update user	Updated by	Updated by.

WZP_PROCESS

Decription: Process definitions.

Column	Label	Description
process_id	ID	Unique ID
process_guid	Process GUID	Process GUID
name	Name	Process name
description	Description	Process description
type	Process type	Process type (MAIN/SUB)
display_order	Process display order	Process display order
duration_unit	Duration Unit	Phase duration unit
default_duration	Default duration	Default duration in days
default_time	Default time	Default time of day for duration expiration

Column	Label	Description
near_duration	Near duration percentage	Near duration percentage. 0 is no near duration defined. 90% means near duration happens when 90% of the duration is gone.
access_code	Access code	Access code. Access codes must exist for affected users in the access code register
start_date	Start date	Process valid from this date
end_date	End date	Process valid to this date
standard	Standard process	Standard process - cannot be changed
locked	Execution type	Process execution type. Must be created in the custom domain register under custom domain WZP-LOCK
package_id	Package ID	Package ID the process is part of
access	Execution type	Process execution type. Must be created in the custom domain register under custom domain.
process_tags	Process tags	Tags for process usage

WZP_PROCESS_WORKFLOW

Description: Process/workflow map

Column	Label	Description
row_id	Key	Key
process_id	ID of process	ID of the process
wf_id	ID of associated workflow	ID of the workflow
start_date	Start date	Process/workflow map valid from this date
end_date	End date	Process/workflow map valid to this date
access_code	Access code	Access code. Access codes must exist for affected users in the access code register.

WZP_ASSEMBLY

Description: Store for the known workflow assemblies in the system.

Column	Label	Description
assembly_id	Key	Key
name	Assembly name	Name of the assembly dll
version	Assembly version	Assembly version
package_id	Package ID	Package ID that the assembly is part of
assembly	Assembly code	The assembly binary code
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
updated	Modified	Update time. Automatically created.
update_user	Updated by	Updated by

WZP_PROCESS_NAME

Description: Localized name table for WZP process name.

Column	Label	Description
row_id	Key	Key
process_id	Process ID	Process ID
text	Name	Process name
culture_name	Language	Language code
culture_source	Language sourde	Culture source from which this entry has been created from
edited	Name edited	Name edited

WZP_PROCESS_DESC

Description: Localized description table for WZP process description.

Column	Label	Description
row_id	Key	Key
process_id	Process ID	Processs ID
text	Process description	Process description
culture_name	Language	Language code
culture_source	Language source.	Culture source from which this entry has been created from
edited	Name edited	Name edited

WZP_PROCESS_PARAMETER

Description: Service workflow parameters.

Column	Label	Description
row_id	ID	Unique ID.
process_id	ID	Unique process ID.
name	Parameter name	The name of the parameter.
description	Description	Parameter description.
type	Parameter type	The parameter type (STRING, INTEGER, PASSWORD).
mandatory	Mandatory	The parameter value is mandatory.
argument	InArgument	The parameter is InArgument to the workflow.

WZP_SERVICE

Description: Service workflow definitions.

Column	Label	Description
service_id	ID	Unique ID.
process_id	ID	Unique process ID.
name	Parameter name	The name of the service workflow.
interval	Monitor interval	
restart interval	Monitor interval	

Column	Label	Description
enabled	Service enabled	Service is enabled
hostnames	List of hosts	The hostnames where the service should run. If blank only one instance is running.

WZP_SERVICE_PARAMETER

Description: Service workflow parameters.

Column	Label	Description
row_id	ID	Unique ID.
service_id	ID	Unique ID.
name	Parameter name	The name of the parameter.
description	Description	Parameter description.
type	Parameter type	The parameter type.
mandatory	Mandatory	The parameter value is mandatory.
argument	InArgument	The parameter is InArgument to the workflow.
value	Parameter value	The value of the parameter.
created	Created	Creation date/time. Automatically created.
updated	Modified	Update time. Automatically created.

WZP_WORKFLOW

Description: Store for the known workflow xaml descriptions in the system.

Column	Label	Description
wf_id	Key	Key
package_id	Package ID	Package ID workflow is part of
typename	Workflow type name	Name of the workflow (Xaml filename)
version	Xaml version	Xaml version (Xaml version)
xaml	Xaml code	The workflow Xaml code

Column	Label	Description
activity_version	Activity version	Version of activity library
standard	Standard workflow	Standard workflow / cannot be changed
access_code	Access code	Access code. Access codes must exist for affected users in the access code register.
phase_label	Phase label	Label text for phase information. Label texts must be created in the label text register with type WZP-PHASE
task_label	Label	Label text for task information. Label texts must be created in the label text register with type) WZP-TYPE
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
updated	Modified	Update time. Automatically created.
update_user	Updated by	Updated by.

WZP_WORKFLOW_CONTEXT

Description: Store for the workflow context specification.

Column	Label	Description
wf_id	Key	Key
register	Register	Register
entity_filter	Entity Filter	OData filter on entity to check if it can be a context for workflow
tag_filter	Tag Filter	Logical expression on context tags for validation of the context against the workflow

11.3 Process forms registers

WZP_FORM

Description: User interface forms used in workflows.

Table	Type	Description
WZP_FORM	Main	Forms used as process user interface.
WZP_FORM_VIEW	Extension	Storage for the view part of the form.
EXP_FORM_ CONTROLLER	Extension	Storage for the controller part of the form

WZP_FORM_DATA

Description: Specification of data associated with the form.

Table	Type	Description
WZP_FORM	Parent	Forms used as process user interface
WZP_FORM_DATA	Main	Data context for forms

WZP_FORM_ASSET

Description: User interface assets shared between different forms.

Table	Type	Description
WZP_FORM_ASSET	Main	User interface assets shared between forms

WZP_WORKFLOW_FORM

Description: Specification of which forms used in workflows.

Table	Type	Description
WZP_WORKFLOW_ FORM	Main	The mapping between workflow and forms

Table	Type	Description
WZP_FORM	Extension	The init form (group:form)
WZP_FORM	Extension	The edit form (group:edit_form)

11.4 Process forms tables

WZP_FORM

Description: Forms used as process user interface.

Column	Label	Description
form_id	Form Identity	Form Identity
form_guid	Global Form Identity	Global Form Identity
package_form_guid	Global Form Identity	Global Form Identity 4.2 and later
package_id	ID	Unique ID
package_version	Package version	Package version
name	Form Name	Form Name
default	Is Default	Default form to be used for starting workflows

WZP_FORM_VIEW

Description: Storage for the 'view' part of the form

Column	Label	Description
form_id	Form Identity	Form Identity
content_type	Content Type	Content type used in HTTP header
content	Content	View content

WZP_FORM_CONTROLLER

Description: Storage for the 'controller' part of the form

Column	Label	Description
form_id	Form Identity	Form Identity
content	Content	JavaScript content

WZP_FORM_DATA

Description: Data context for forms

Column	Label	Description
row_id	...	Internal unique id
form_id	Form Identity	Form Identity
name	Name of the context	Name of the data context
max_offline_pages	Max offline page	Max OData pages to be retrived for offline usage
query	Query	OData Query
parameters	Query parameters	OData Query parameyters

WZP_WORKFLOW_FORM

Description: Relation between workflows and forms

Column	Label	Description
row_id	...	Internal unique ID
wf_id	Workflow Identity	The workflow to be started using specified form
form_id	Form Identity	The form used to start the workflow
default	Is Default	Is the form default for starting the workflow
edit_form_id	Edit Form Identity	The form used to edit the workflow

WZP_FORM_ASSET

Description: Storage for shared user interface assets

Column	Label	Description
asset_id	Asset Identity	Asset Identity
key	Asset Key	File-system friendly asset key
content_type	Content Type	Content type used in HTTP header
content	Asset Content	Base64 encoded content of the asset
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
updated	Modified	Update time. Automatically created.
update_user	Updated by	Updated by

11.5 Process instance registers

WZP_WORKFLOW_INSTANCE

Description: Known workflow instances in the system.

Table	Type	Description
FILE	Parent	The file the instance is running under.
WZP_WORKFLOW	Parent	The workflow which is instanciated.
WZP_WORKFLOW_ INSTANCE	Main	The workflow instance.
V_WZP_LOCK_INSTANCE	Extension	The Instance lock.
V_WZP_ROOT	Extension	Instance root flag.
V_WZP_RUNNING_ INSTANCE	Extension	The instance count for root instances.
WZP_INSTANCE_ BOOKMARK	Extension	Currently available bookmarks.
WZP_INSTANCE_ ARGUMENT	Extension	Instance arguments to start instance.
WZP_WORKFLOW_ INSTANCE_ELAB	Extension	Instance free text.
WZP_PROCESS	Extension	Instance process.

Table	Type	Description
WZP_PHASE	Child	Phases (for phase instance).
WZP_USER_TASK	Child	All UserTasks (group:task).
WZP_USER_TASK	Child	Open user. Tasks only (group:opentask).
WZP_CASE_ACTIVITY	Child	Case activities (for case activity instance).
WZP_SERVICE_ PARAMETER	Child	Instance parameters (for service instance).
V_WZP_OPEN_USER_TASK	Extension	Current open user task only.
V_WZP_MY_OPEN_TASK	Extension	Open user task for me.
V_WZP_MY_OU_OPEN_ TASK	Extension	Open user task for my ou.
V_WZP_TASK_ERROR	Extension	Most important error in instance.
WZP_HISTORY	Child	User task history.
V_WZP_WORKFLOW_ INSTANCE_RECORD	Child	Records used in tasks.

WZP_WORKFLOW_INSTANCE_ELAB

Description: Known workflow instances in the system. Used for free text searches.

Table	Type	Description
WZP_WORKFLOW_ INSTANCE	Parent	The workflow instance.
WZP_WORKFLOW_ INSTANCE_ELAB	Main	The instance elaborating text

WZP_LOCK_INSTANCE

Description: The known workflow instances locks in the system.

Table	Type	Description
V_WZP_LOCK_ INSTANCE	Main	Instance lock.

WZP_WORKFLOW_STREAM

Description: Workflow stream storage.

Table	Type	Description
WZP_WORKFLOW_ INSTANCE	Parent	The instance.
WZP_WORKFLOW_ STREAM	Parent	The persisted data for stream arguments.

WZP_WORKFLOW_INSTANCE_RECORD

Description: Workflow instance records.

Table	Type	Description
RECORD	Parent	The record metadata for documents.
V_WZP_ WORKFLOW_ INSTANCE_RECORD	Main	The documents used for the instance.

WZP_WORKFLOW_LOG

Description: Workflow instance acticity log.

Table	Type	Description
WZP_WORKFLOW_ LOG	Main	The instance execution log.

11.6 Process instance tables

WZP_INSTANCE_ARGUMENT

Description: The workflow instances InArguments.

Column	Label	Description
row_id	workflow instance ID	Workflow instance ID.
dictionary	InArguments	Workflow InArguments dictionary

WZP_WORKFLOW_INSTANCE

Description: The known workflow instances in the system

Column	Label	Description
row_id	...	Internal unique id
instance_id	workflow instance Id	Workflow instance Id
parent_id	Workflow parent instance	Workflow parent instance id
service_id	Service ID	Service ID
root_id	workflow root instance Id	Workflow root instance Id
wf_id	Workflow ID	Workflow ID
process_id	Process ID	Process ID
title	Title	Title
description	Description	Remark
register	Register	Register
register_key	Register key	Register key
file_key	File key	register key when register = FILE
persistence_state	State	Workflow persistance state
server_name	Server name	Server name
workflow_status	Workflow status	Workflow status
created		Creation date/time. Automatically created.
create_user	Created by	Created by
updated	Modified	Update time. Automatically created.
closed	User task closed date	The time the workflow was completed.
pending_time	Due date	Time when workflow is to be

Column	Label	Description
		resumed.
due_date	Workflow close date	Time when workflow is supposed to be ended.
owner	Created by	Name of the user who created the item. Automatically updated.
owner_ou	Owner department at creation time	Owner department at creation time.
importance	Instance importance	Workflow instance importance. Must be created in the custom domain register under custom domain WZP-PRIO.
access	Execution type	Process execution type. Must be created in the custom domain register under custom domain WZP-LOCK.
access_code	Indblik	Dummy placeholder to overcome a bug in som when foreignAccessCheck is used in the wzp_workflow_task register
log cleared	Log cleared date	The time the workflow log was cleared.

WZP_WORKFLOW_LOG

Description: Workflow activity log

Column	Label	Description
row_id	Key	Internal unique id
server_name	Server Name	Server Name
process_id	Windows process Id	Windows process Id
event_datetime	Timestamp	TimeStamp
instance_id	Workflow Instance Id	Workflow Instance Id
workflow_type	Workflow Type	Workflow Type
activity_type	Activity Id	Activity Id
activity_name	Activity Type	Activity Type

Column	Label	Description
action_type	Action Type	Action Type
record_type	Record Type	Record Type
properties	Properties	Activity Properties

WZP_WORKFLOW_STREAM

Description: Workflow Stream Storage

Column	Label	Description
stream_id	Stream ID	Stream key
instance_id	Workflow Instance ID	Reference to the workflow instance owning the stream
title	Stream Content Title	Name of file the stream was produced from
extension	Stream Content Extension	Extension of file the stream was produced from
content_type	Stream content type	MIME content type of the stream content.
content	Stream Content	Content of the stream

WZP_INSTANCE_BOOKMARK

Description: The workflow instances bookmarks

Column	Label	Description
instance_id	workflow instance ID	Workflow instance ID
bookmarks	Bookmarks	Enabled bookmarks

WZP_WORKFLOW_INSTANCE_ELAB

Description: Search table for workflow_instance

Column	Label	Description
instance_id	Instance ID	Instance ID of workflow
elab_text	elab	Describing identification of the

Column	Label	Description
		workflow_instance

V_WZP_LOCK_INSTANCE

Description: The known workflow instances locks in the system

Column	Label	Description
instance_id	workflow instance ID	Workflow instance ID
locked	Locked	Locked while loaded
update_status	Workflow status at update	Workflow status at update

V_WZP_WORKFLOW_INSTANCE_RECORD

Description: Records attached to the workflow instance

Column	Label	Description
instance_id	Instance ID	Records attached to the workflow instance
record_id	Record identity	Record identity

WZP_HISTORY

Description: The known workflow tasks in the system

Column	Label	Description
row_id	Key	Unique task key
instance_id	Instance ID	Instance ID
root_id	Root workflow instance ID	Root workflow instance ID
wf_id	Workflow key	Workflow key
user_task_id	User Task ID	User Task ID
name_key	Contact number	Contact system key
name_code	Responsible	Name code of person/department to respond
name_type	Contact type	Name type of person/department to

Column	Label	Description
		respond
name_ou	Name ou	Ou of contact to respond if contact is employee or department
proxy_code	Proxy	Name code of employee actually responding
proxy_ou	Proxy ou	Ou of employee actually responding"
proxy_code	Proxy	Name code of employee actually responding
register	Register	Register
register_key	Register key	Register key
task_label	Label	Label text for task information. Label texts must be created in the label text register with type = WZP-TYPE"
task_action	Action	Task action
show	Show	Include in list
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
comment	Comment	Remark

WZP_WORKFLOW_PROFILE

Description: Store for the known workflow xaml descriptions in the system.

Column	Label	Description
wf_id	Key	Key
tracking_profile	Tracking profile	The package tracking profile xml code

V_WZP_ROOT

Description: Workflow_instance root indicator.

Column	Label	Description
instance_id	Instance ID	Instance ID of workflow.
is_root	Is root process	Is root process.

V_WZP_RUNNING_INSTANCE

Description: Workflow_instance running children indicator.

Column	Label	Description
instance_id	Instance ID	Instance ID of workflow.
running_instances	Number of running MAIN and SUB	Number of running MAIN and SUB.

V_WZP_WORKFLOW_INSTANCE_RECORD

Description: Records attached to the workflow instance.

Column	Label	Description
instance_id	Instance ID	Instance ID
record_id	Record identity	Record identity

V_WZP_USER_TASK_APPROVER

Description: Determine which users have participated in a process.

Column	Label	Description
instance_id	Instance ID	Instance ID
name_key	Contact number	Contact system key
name_code	Responsible	Name code of person/department to respond
name_type	Contact type	Name type of person/department to respond.
elab_text	elab_text	Elaborating text

11.7 Process task registers

WZP_USER_TASK

Description: Known workflow user tasks in the workflow instances.

Table	Type	Description
WZP_WORKFLOW_INSTANCE	Parent	The workflow instance the user task belongs to.
WZP_WORKFLOW_INSTANCE	Parent	The root workflow instance (group:root).
WZP_FORM	Parent	The init and edit forms for the user task.
WZP_USER_TASK	Main	The user task.
WZP_USER_TASK_TITLE	Child	The localized title.
WZP_USER_TASK_RECORD	Child	Attachment documents (group:attachment).
WZP_USER_TASK_RECORD	Child	The answer documents (group:answer).
WZP_HISTORY	Child	The user task history.

WZP_OPEN_USER_TASK

Description: Known workflow user tasks in the workflow instances.

Table	Type	Description
WZP_WORKFLOW_INSTANCE	Parent	The workflow instance the user task belongs to.
V_WZP_OPEN_USER_TASK	Main	Open user tasks.

WZP_USER_TASK_ATTACHMENT

Description: Records attached to the user task.

Table	Type	Description
WZP_USER_TASK	Parent	The user task.
WZP_USER_TASK_ATTACHMENT	Main	The attached documents.
RECORD	Extension	Document metadata.

WZP_USER_TASK_INSERT

Description: Records attached to the user task including deleted records.

Table	Type	Description
WZP_USER_TASK	Parent	The user task.
WZP_USER_TASK_ ATTACHMENT	Main	The attached documents.
RECORD	Child	The document metadata

WZP_TASK_STATUS

Description: Known user task status in the workflow instances. Used to create user tasks without any actor.

Table	Type	Description
WZP_WORKFLOW_ INSTANCE	Parent	The workflow instance the user task belongs to.
WZP_WORKFLOW_ INSTANCE	Parent	The root workflow instance (group:root).
WZP_TASK_STATUS	Main	The user task.

WZP_HISTORY

Description: The history of the known workflow tasks in the system.

Table	Type	Description
WZP_WORKFLOW_ INSTANCE	Parent	The workflow instance the user task belongs to.
WZP_HISTORY	Main	The user task history.

WZP_PHASE

Description: The phases for a MAIN workflow.

Table	Type	Description
WZP_WORKFLOW_INSTANCE	Parent	The workflow instance the user task belongs to.
WZP_PHASE	Main	The phases in MAIN process workflows.

11.8 Process task tables

WZP_PHASE

Description: Workflow phases of a mainflow

Column	Label	Description
row_id		
instance_id	workflow instance Id	Workflow instance Id
number	Phase number	Phase number
label	Phase label	Label text for phase information. Label texts must be created in the label text register with type = WZP-PHAS
name	Phase name	Phase name
state	State	Phase state. Must be defined in the custom_Must be created in the custom domain register under custom domain WZP-TSTA
action	Action	Phase action
schedule	Schedule	Phase schedule
role	Phase role	Phase role.
near_duration	Phase near duration	Phase near duration
duration	Phawse duration	Phase duration
opened	Phase opened date	The last time the phase was opened
closed	Phase closed date	The last time the phase was closed.

Column	Label	Description
near_due_date	Due date	The period available to nearly complete the phase.
due_date	Due date	The period available to complete the phase.
calculated	Calculated	DueDate is calculated

WZP_TASK_STATUS

Description: The known workflow tasks in the system

Column	Label	Description
task_id	Key	Unique task key
instance_id	Workflow instance identity	Workflow instance identity
root_id	Root workflow instance identity	Root workflow instance identity
name_key	Contact number	Contact system key
name_code	Responsible	Name code of actor
name_type	Contact type	Name type of actor
name_ou	Organization unit of actor	Organization unit of actor
task_order	Display order	Display order
task_state	State	Workflow task state. Must be created in the custom domain register under custom domain WZP-TSTA
task_label	Label	Label text for task information. Label text must be created in the label text register with type = WZP-TYPE
task_action	Action	Task action
task_schedule	State	Workflow task schedule. Must be created in the custom domain register under custom domain WZP-TTIM
task_error	Error type	Workflow task error type: Must be created in the custom domain register under custom domain WZP-TERR
due_date	Due date	The period available to complete the user task

comment	Comment	Remark
show	Show	Include in list
opened	User task opened date	The time the user task was opened
closed	User task closed date	The time the task was closed.
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
updated		Update time. Automatically created.
update_user	Updated by	Updated by
title	Title	User Task title

WZP_USER_TASK_TITLE

Description: Localization table for user task title

Column	Label	Description
row_id	Key	Key
task_id	User Task ID	User Task ID
text	Name	User Task title
culture_name	Language	Language code
culture_source	Label source	Culture source from which this entry has been created from

WZP_USER_TASK_ATTACHMENT

Description: Records attached to the user task

Column	Label	Description
row_id	Key	Key
task_id	Task identity	Task identity
record_id	Record identity	Record identity
record_type	Type	Record type. Must be created in the custom domain register under custom domain WZP-RTYP

property_name	Property name	The name of the property the document originates from
priority	Document order	Document order
attach	Attach	Attach document to mail

WZP_USER_TASK_RECORD

Description: Records attached to the user task

dbname:"wzp_user_task_attachment"

Column	Label	Description
row_id	Key	Key
task_id	Task identity	Task identity
record_id	Record identity	Record identity
record_type	Type	Record type. Must be created in the custom domain register under custom domain WZP-RTYP
property_name	Property name	The name of the property the document originates from
priority	Document order	Document order
attach	Attach	Attach document to mail

WZP_USER_TASK

Description: The known workflow tasks in the system

Column	Label	Description
task_id	Key	Unique task key
instance_id	Workflow instance identity	Workflow instance identity
root_id	Root workflow instance identity	Root workflow instance identity
group_id	Task group identity	Task group identity
form_id	Smart task form identity	Smart task form identity
name_key	Contact number	Contact system key

name_code	Responsible	Name code of actor
name_type	Contact type	Name type of actor
name_ou	Organization unit of actor	Organization unit of actor
proxy_code	Proxy	Name code of employee actually responding
proxy_ou	Proxy ou	Ou of employee actually responding
task_order	Display order	Display order
task_state	State	Workflow task state. Must be created in the custom domain register under custom domain WZP-TSTA
task_label	Label	Label text for task information. Label texts must be created in the label text register with type = WZT-TYPE
task_action	Action	Task action
task_schedule	State	Workflow task schedule. Must be created in the custom domain register under custom domain WZP-TTIM
task_error	Error type	Workflow task error type. Must be created in the custom domain register under custom domain WZP-TERR
access	Execution type	Execution type. Must be created in the custom domain register under custom domain WZP-LOCK.
mandatory	Mandatory	Mandatory actor
near_duration	Phase near duration	User task near duration
duration	Phase duration	User task duration
near_due_date	Due date	The period available to nearly complete the user task.
due_date	Due date	The period available to complete the user task.
calculated	Calculated	Due date and near due date is calculated
comment	Comment	Remark
show	Show	Include in list
is_notification	Is notification?	Flag specifying if task doesn't require action from actor.

severity	Severity	Severity
importance	Instance importance	Workflow instance importance. Must be created in the custom domain register under custom domain WF4_PRIO
update_code	Update code	Update access code. Access codes must exist for affected users in the access code register
properties	Properties	Serialized JSON object containing properties of the user task.
opened	User task opened date	The time the user task was opened.
closed	User task closed date	The time the task was closed.
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
updated		Update time. Automatically created.
update_user	Updated by	Updated by
title	Title	User Task title

V_WZP_OPEN_USER_TASK

Description: The known open workflow tasks in the system

Column	Label	Description
instance_id	Workflow instance identity	Workflow instance identity
task_state	State	Workflow task state. Must be created in the custom domain register under custom domain WZP-TSTA
multiple	Multiple	Multiple open tasks in instance
name_type	Contact type	Name type of actor
name_code	Responsible	Name code of actor
name_ou	Organization unit of actor	Organization unit of actor

WZP_HISTORY

Description: The known workflow tasks in the system.

Column	Label	Description
row_id	Key	Unique task key
instance_id	Instance ID	Instance ID
root_id	Root workflow instance ID	Root workflow instance ID
wf_id	Workflow key	Workflow key
user_task_id	User Task ID	User Task ID
name_key	Contact number	Contact system key
name_code	Responsible	Name code of person/department to respond
name_type	Contact type	Name type of person/department to respond
name_ou	Name ou	Ou of contact to respond if contact is employee or department
proxy_code	Proxy	Name code of employee actually responding
proxy_ou	Proxy ou	Ou of employee actually responding
register	Register	Register
register_key	Register key	Register key
task_label	Label	Label text for task information. Label texts must be created in the label text register with type = WZP-TYPE
task_action	Action	Task action
show	Show	Include in list
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
comment	Comment	Remark

WZP_HISTORY_DESC

Description: Localization table for commen

Column	Label	Description
row_id	Key	Key
task_id	User Task ID	History ID

text	Name	The Comment
culture_name	Language	Language code
culture_source	Label sourc	Culture source from which this entry has been created from

V_WZP_MY_OPEN_TASK

Description: Total number of open workflow tasks for me in the system.

Column	Label	Description
instance_id	Workflow instance identity	Workflow instance identity
number_open_task	Number of open tasks for me	Number of open tasks for me

V_WZP_MY_OU_OPEN_TASK

Description: Total number of open workflow tasks for my ou in the system.

Column	Label	Description
instance_id	Workflow instance identity	Workflow instance identity
number_open_task	Number of open tasks for my ou	Number of open tasks for my ou

V_WZP_TASK_ERROR

Description: The known open task error in workflow instances.

Column	Label	Description
instance_id	Workflow instance identity	Workflow instance identity
multiple	Multiple	Multiple open tasks in instance
task_error	Error type	Workflow task error type. Must be created in the custom domain register under custom domain WZP-TERR.

11.9 Miscellaneous registers

WZP_PROXY

Description: Define delegates for actors.

Table	Type	Description
	Type	**Description**
WZP_PROXY	Main	Define delegates for users.
NAME	Extension	Actor information (group:actor)
V_OU_EMPLOYEE_DOMAIN	Extension	Delegate information (group:proxy)

WZP_FILE_USER_RIGHT

Description: Determine which users have read access for a given file.

Table	Type	Description
Table	**Type**	**Description**
V_WZP_FILE_USER_RIGHT	Main	Users with read access to a file

WZP_SETTINGS

Description: WorkZone Process settings.

Table	Type	Description
WZP_SETTINGS	Main	The key / value settings

WZP_SEQUENCE_MASK

Description: WorkZone Process sequence masks.

Table	Type	Description
WZP_SEQUENCE_MASK	Main	The sequence masks

Table	Type	Description
WZP_SEQUENCE_ MASK_ITEM	Child	The sequence mask members.

WZP_PUSH_SUBSCRIPTION

Description: Defines which IPad or IPhone devices are eligible for push subscriptions.

Table	Type	Description
EMPLOYEE	Parent	The user information
WZP_PUSH_ SUBSCRIPTION	Main	The subscription

WZP_MAIL_NOTIFICATION

Description: Defines which users are receiving mail notifications.

Table	Type	Description
WZP_MAIL_ NOTIFICATION	Main	The mail notification users.

11.10 Miscellaneous tables

WZP_PROXY

Description: N/A

Column	Label	Description
row_id	Row ID	System key
name_key	Contact number	Contact system key
proxy_key	Contact number	Contact system key
proxy_role	Role	Proxy role. Must be created in the custom domain register under custom domain WZP-PRXY

V_WZP_FILE_USER_RIGHT

Description: Determine which users have read access for a given file

Column	Label	Description
file_key	Key in file	Case key, key (system ID) to the file register
name_key	Contact number	Contact system key
name_code	Responsible	Name code of person/department to respond
name_type	Contact type	Name type of person/department to respond
elab_text	elab_text	elaborating text

WZP_SETTINGS

Description: N/A

Column	Label	Description
row_id	Row ID	
module	module	module
key	Setting key	
access_code	Access code	Access code. Access codes must exist for affected users in the access code register
value	Setting value	

WZP_PUSH_SUBSCRIPTION

Description: N/A

Column	Label	Description
row_id	Row ID	
device_token	Device token	Device token
user_name	Subscriber	Username
subscription_date	Created	Creation date/time. Automatically

Column	Label	Description
		created.
Locale_id	Locale ID	The locale used on the device.

WZP_MAIL_NOTIFICATION

Description: Workzone mail notification

Column	Label	Description
name_key	Contact number	Contact system key
receive_mail_ Notification	Notification	Receive mail notifications

WZP_SEQUENCE_MASK

Description: N/A

Column	Label	Description
sequence_id	Sequence ID	
name	module	module
description	Setting key	
access_code	Access code	Access code. Access codes must exist for affected users in the access code register
update_code	Update code	Update access code. Access codes must exist for affected users in the access code register
shared	Shared sequence	Shared sequence
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
updated	Modified	Update time. Automatically created.
update_user	Updated by	Updated by

WZP_SEQUENCE_MASK_ITEM

Description: N/A

Column	Label	Description
row_id	Row id	
sequence_id	Sequence id	
name_key	Contact number	Contact system key
order	Sequence item order	Sequence item order
date_offset	Item duration offset	Item duration offset

11.11 Case activity registers

WZP_CASE_ACTIVITY

Description: Case activities for a case from DCR graph events.

Table	Type	Description
WZP_WORKFLOW_INSTANCE	Parent	The workflow instance the smarttask belongs to.
WZP_WORKFLOW_INSTANCE	Parent	The root workflow instance (group:root).
WZP_FORM	Parent	The case activity init form.
WZP_CASE_ACTIVITY	Main	The case activities.
WZP_CASE_ACTIVITY_TITLE	Child	The localized case activity title.
WZP_CASE_ACTIVITY_HISTORY	Child	The case activity history.

WZP_CASE_ACTIVITY_HISTORY

Description: Case activity history.

Table	Type	Description
WZP_CASE_ACTIVITY_HISTORY	Main	The case activity history.

11.12 Case activity tables

WZP_CASE_ACTIVITY

Description: The known DCR graph events in a DCR graph.

Column	Label	Description
instance_id	Workflow instance identity	Workflow instance identity.
root_id	Root workflow instance identity	Root workflow instance identity.
activity_type	Activity type	activity type. Must be created in the custom domain register under custom domain WZP-ATYP.
activity_name	Activity name	Activity name
description	Description	Activity description
comment	Comment	Remark
value	Activity value	Activity value entered at execute.
role	Actor role	Actor role. Must be created in the custom domain register under custom domain WZP-ROLE.
form_id	Activity form identity	Activity form identity.
name_key	Contact number	Name key of respondee.
name_code	Responsible	Name code of respondee.
name_type	Contact type	Name type of respondee.
name_ou	Organization unit of respondee	Organizational unit of respondee
proxy_code	Proxy	Name code of employee actually responding.
proxy_ou	Proxy ou	Ou of employee actually responding.
activity_state	State	Activity state. Must be created in the custom domain register under custom domain WZP-ASTA.
activity_schedule	State	Activity schedule. Must be created in the custom domain register under custom domain WZP-TTIM.
activity_error	Error type	Activity error type. Must be created in

Column	Label	Description
		the custom domain register under custom domain WZP-TERR.
access	Execution type	Execution type. Must be created in the custom domain register under custom domain WZP-LOCK.
near_duration	Phase near duration	Smarttask near duration.
duration	Phase duration	Smarttask duration.
near_due_date	Due date	The period available to nearly complete the smarttask.
due_date	Due date	The period available to complete the smarttask.
calculated	Calculated	Due date and near due date is calculated.
severity	Severity	Severity
importance	Instance importance	Workflow instance importance. Must be created in the custom domain register under custom domain WF4_PRIO.
update_code	Update code	Update access code. Access codes must exist for affected users in the access code register.
executed	User task opened date	The time the user task was opened.
created	Created	Creation date/time. Automatically created.
create_user	Created by	Created by
updated	Modified	Update time. Automatically created.
update_user	Updated by	Updated by

WZP_CASE_ACTIVITY_TITLE

Description: Localization table for activity title.

Column	Label	Description
row_id	Key	Key
activity_id	Activity ID	Activity ID

text	Name	User task title
culture_name	Language	Language code
culture_source	Language source	Culture source from which this entry has been created from***
edited	Name edited	Name edited

WZP_CASE_ACTIVITY_HISTORY

Description: The executed action events in a DCR graph.

Column	Label	Description
row_id	Key	Unique history key
activity_id	Key	Unique activity key
activity_action	Action	Activity action. Must be created in the custom domain register under custom domain WZP-AACT.
activity_schedule	State	Activity schedule. Must be created in the custom domain register under custom domain WZP-TTIM
comment	Comment	Remark
value	Activity value	Activity value entered at execute.
proxy_code	Proxy	Name code of employee actually responding.
proxy_ou	Proxy ou	Ou of employee actually responding.
created	Created	Creation date/time. Automatically created.

11.13 SmartPost registers

WZP_IDENTIFIER_SOURCE

Description: Configuration of where to locate identifiers of various party types.

Table	Type	Description
WZP_exception_class	Main	

WZP_EXCEPTION_CLASS

Description: Configuration of exceptions and actions to exceptions.

Table	Type	Description
WZP_exception_class	Main	

WZP_EBOKS_MATERIAL

Description: A material entity reflects a material defined by the e-Boks Administration Portal. A material type relates to one or more subscription groups. Any messages sent to e-Boks must be of one (and only one) material.

Table	Type	Description
WZP_EBOKS_MATERIAL	Main	
WZP_EBOKS_MATERIAL_NAME	Child	

WZP_NAME_EXTENSION

Description: An extension of the NAME table. The table keeps track of contact subscriptions and MRU (Most Recently Used) lists.

Table	Type	Description
WZP_NAME_EXTENSION	Main	
WZP_INDENTIFIER_SOURCE	Extension	

WZP_EBOKS_SUBSCRIPTION

Description: Direct access to the EBOKS_SUBSCRIPTION table used for unsubscribing.

Table	Type	Description
WZP_EBOKS_SUBSCRIPTION	Main	

WZP_SHIPMENT_TYPE

Description: The configured dispatch types.

Table	Type	Description
WZP_SHIPMENT_TYPE	Main	
WZP_SHIPMENT_TYPE_NAME	Child	
WZP_SHIPMENT_TYPE_ORDER	Child	

WZP_SHIPMENT_TYPE_ORDER

Description: The configured dispatch type order.

Table	Type	Description
WZP_DISPATCHER	Parent	
WZP_SHIPMENT_TYPE_ORDER	Main	

WZP_REMOTE_PRINT_TYPE

Description: The configuration of a remote print type.

Table	Type	Description
WZP_REMOTE_PRINT_TYPE	Main	
WZP_REMOTE_PRINT_TYPE_NAME	Child	

WZP_SMARTPOST_LOG

Description: The SmartPost dispatch log. Used to build the SmartPost history log.

Table	Type	Description
WZP_SMARTPOST_LOG	Main	

Table	Type	Description
WZP_SMARTPOST_RECIPIENT	Child	
WZP_SMARTPOST_ATTACHMENT	Child	

WZP_SMARTPOST_RECIPIENT

Description: The SmartPost dispatch recipient log. Used to build the SmartPosthistory log.

Table	Type	Description
WZP_SMARTPOST_RECIPIENT	Main	

WZP_SMARTPOST_ATTACHMENT

Description: The SmartPost dispatch attachment log. Used to build the SmartPost history log.

Table	Type	Description
WZP_SMARTPOST_ATTACHMENT	Main	
RECORD	Extension	

WZP_DISPATCHER

Description: Communication channel.

Table	Type	Description
WZP_DISPATCHER	Main	
WZP_DISPATCHER_NAME	Child	

WZP_DISPATCHER_PARAMETER

Description: Configuration of dispatchers.

Table	Type	Description
WZP_DISPATCHER	Parent	

Table	Type	Description
WZP_DISPATCHER_PARAMETER	Main	

11.14 SmartPost tables

WZP_FILE_EXTENSION

Description: Table that contains the status before OpenCaseScope changes it.

Column	Label	Description
file_key	Surrogate key in SP.	Case number. Surrogatekey (systemident.) to the FILE register.
prev_closed	Original closed date	Original closed state before updated by constructor in OpenCaseState in SmartPort
prev_update_code	Original update code	Original update code before updated by constructor in OpenCaseState in SmartPort

WZP_IDENTIFIER_SOURCE

Description: Configuration of where to locate identifiers for various party types.

Column	Label	Description
party_identifier_ source_key	Party Identifier Source Key	Unique identifier for the table entities.
name_type	Name Type	The name type from the NAME entity
e_boks_source	e-boks Source	The source for e-Boks CVR/CPR numbers.
att_source	Attention Source	The source for making attention fields in attached attention.xml file.
id	ID	Unique identifier for the exceptions classes.
exception_type_ name	ExceptionTypeName	ExceptionTypeName.
priority	ID	Priority of the exception if more than one matches.
exception_ message_filter	ExpressionMessageFilter	Filter that must match all exceptions of this class.

Column	Label	Description
exception_scope_string	ExpressionScopeString	The name of the scope where the exception is legal.
presentation_string_template	PresentationScopeString	Presentation string template. It may contain matches from regular expression.
error_code	ErrorCode	Error code extracted from exception.
description	Description	Description of the excption class .
exception_action_string	ActionString	Action to perform for this exception class.

WZP_EBOKS_MATERIAL

Description: A material entity reflects a material defined by the e-Boks Administration Portal. A material type relates to one or more subscription groups. Any messages sent to e-Boks must of one (and only one) material.

Column	Label	Description
material_key	Material key	Unique identifier for table entities.
external_id	External Identifier	Unique identifier of the material specified externally by e-Boks.
order	Display order	Display order.
replyable	Can be replied to.	Indicates whether it is possible for the end user to reply on messages containing this material.
access_code	Access code	Access code. The access codes to use must be registered for the relevant users in the ACCESS_CODE register.
material_key	Material key	Unique identifier for table entities.
external_id	External Identifier	Unique identifier of the material specified externally by e-Boks.
order	Display order	Display order.
access_code	Access code	Access code. Access codes to be used must be registered for the relevant users in the ACCESS_CODE register.
row_id	...	Internal unique ID.

WZP_EBOKS_MATERIAL_NAME

Description: Localized names for dispatch types.

Column	Label	Description
material_key	...	Material key..
text	Text	Localized name.
culture_name	Culture	Culture name.
culture_source	...	Culture source from which this entry has been created.
edited	...	Edited flag.

WZP_NAME_EXTENSION

Description: Extension to a contact. Used for avoiding free search trigger.

Column	Label	Description
name_code	Contact code	Contact code.
name_key	Navnelbnr	Surrogate key (system ID) in the NAME register.
name_type	Contact type	Contact type.
tax_id_no	CVR No	CVR No.
tax_id_prod_no	P No	Production unit ID.
last_material_key	Last used material key	The last used material key.

WZP_EBOKS_SUBSRIPTION

Description: Extension to a contact. Used for unsubscribing manually.

Column	Label	Description
system_id	Dispatch system	e-Boks dispatch system ID
recipient_id	ident	Recipient ID.
recpient_type	ident type	Recipient type.

Column	Label	Description
content_type	content type	Content type.
is_subscribing	is subscribing	is subscribing.

WZP_SHIPMENT_TYPE

Description: The configured dispatch types.

Column	Label	Description
shipment_key	Key for shipment type	Unique identifier for the table entities.
order	Display order	Display order.
access_code	Access code	Access code. Access codes to be used must be registered for the relevant users in the ACCESS_CODE register.

WZP_SHIPMENT_TYPE_NAME

Description: Localized name for dispatch types.

Column	Label	Description
row_id	...	Internal unique ID.
shipment_key	...	Dispatch type key.
text	Text	Localized name.
culture_name	Culture	Culture name.
culture_source	...	Culture source from which this entry has been created.
edited	...	Edited flag.

WZP_SHIPMENT_TYPE_ORDER

Description: Link between the dispatch types and their applied channel types.

Column	Label	Description
row_id	Unique identity of the link.	Internal unique ID.

Column	Label	Description
shipment_key	Dispatch type key	The key to the dispatch type that is linked to a dispatch channel type by this entity.
order	Channel order	Channel order.
channel_type		Foreign key to the channel entity. It must be created in the custom domain register under custom domain WZP-SC.
dispatcher_id		Foreign key to wzp_dispatcher.

WZP_REMOTE_PRINT_TYPE

Description: The different constellations of remote print configurations.

Column	Label	Description
key	Remote print type	A configuration of a remote print
porto_category_key		Foreign key to the porto_category entity. The key must be created in the custom domain register under custom domain WZP-MCAT.
returned_letter_ handling_key		Foreign key to the returned_letter_handling entity. The key must be created in the custom domain register under custom domain WZP-RLH.
urgency_level_key		Foreign key to the urgency_level entity. The key must be created in the custom domain register under custom domain WZP-UL.
simplex_duplex_key		Foreign key to the simplex_duplex entity. The key must be created in the custom domain register under custom domain WZP-SD.
print_color_option_ key		Foreign key to the print_color entity. The key must be created in the custom domain register under custom domain WZP-PCO.
envelope_type_key		Foreign key to the envelope_type entity. The key must be created in the custom domain register under custom domain WZP-ET.
access_code	Access code	Access code. Access codes to be used must

Column	Label	Description
		be registered for the relevant users in the ACCESS_CODE register.
order	Display order	Display order.

WZP_REMOTE_PRINT_TYPE_NAME

Description: Localized name table for WorkZone Process name.

Column	Label	Description
row_id	Key	Key
key	Remote print type	A configuration of a remote print.
text	Name	Name of the remote print configuration.
culture_ name	Language	Language code.
culture_ source	Language source	The source language of the entry.
edited	Name edited	Name edited.

WZP_SMARTPOST_LOG

Description: The SmartPost log

Column	Label	Description
row_id	...	Internal unique ID.
instance_id	Workflow instance ID	Workflow instance ID.
approval	Approve before sending	Specifies that the document must be approved before sending.
shipment_key	Shipment type key	Dispatch type key.
material_key	Material key	The selected material
remote_print_type_ key	Remote print type	The selected remote print type.
delete_original	Delete original after sending.	Specifies that the original document should be deleted after sending.
overall_status	Overall status	Message sending overall status.

Column	Label	Description
overall_status_date	Overall status date	Date when the overall status was collected.

WZP_SMARTPOST_RECIPIENT

Description: RecipIents of the SmartPost dispatches.

Column	Label	Description
row_id	Key	Key
instance_id	Workflow instance ID	Workflow instance ID
record_key	Record identity	Record identity
recipient	Contact identity	Contact identity
role	Role	Rolle for aktdeltager.
shipment_channel	Dispatched by	Name of the shipping channel
external_id	External ID	The ID of the dispatch from the dispatch channel
shipment_state	Shipment state	The state of the dispatch.
shipment_date	Shipment date	Dispatch date

WZP_SMARTPOST_ATTACHMENT

Description; Attachments to the SmartPost dispatches.

Column	Label	Description
row_id	Key	Key
instance_id	Workflow instance ID	Workflow instance ID
record_key	Record identity	Record identity

WZP_DISPATCHER

Description: Communication channel

Column	Label	
dispatcher_id	Dispatcher ID	Unique identifier for table entities.

Column	Label	
guid	DLL Guid	A GUID to identify the associated DLL
description	Dispatcher Decsription	A description to explain the purpose of the dispatcher
assembly	Dispatcher Assembly	The Assembly name
access_code	Access code	Access code. Access codes to be used must be registered for the relevant users in the ACCESS_CODE register.

WZP_DISPATCHER_NAME

Description: Localized name for dispatchers.

Column	Label	Description
row_id	...	Internal unique iD.
dispatcher_id	...	Dispatcher ID.
text	Text	Localized name.
culture_name	Culture	Culture name.
culture_source	...	Culture source from which this entry has been created.
edited	...	Edited flag.

WZP_DISPATCHER_PARAMETER

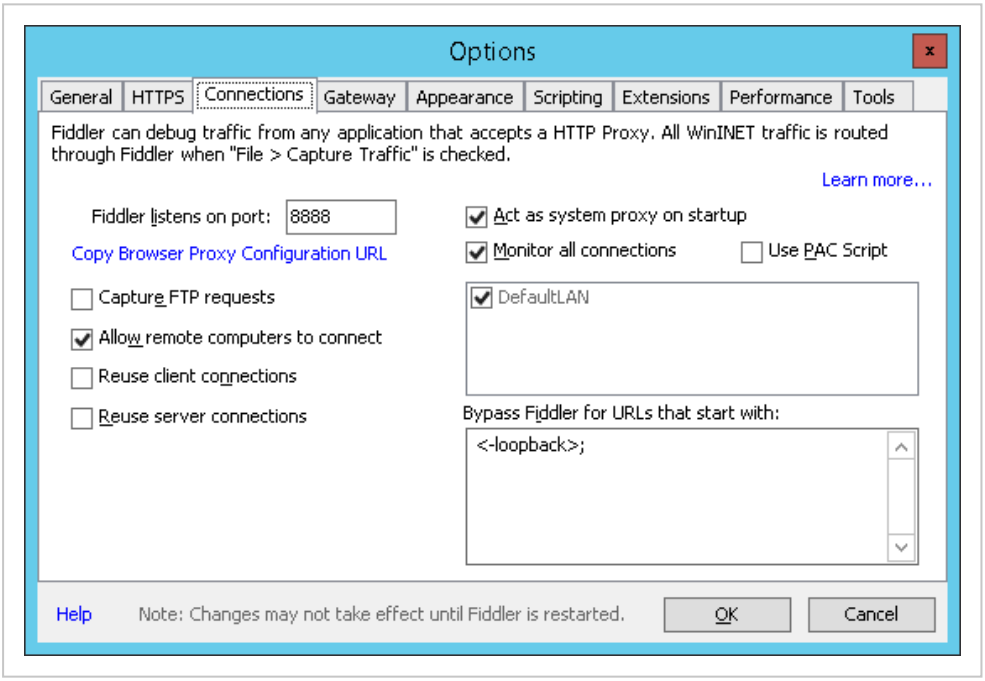
Column	Labe	Description
row_id	ID	Unique ID.
dispatcher_id	ID	Unique ID.
name	Parameter name	The name of the parameter.
description	Description	Parameter description.
type	Parameter type	The parameter type (STRING, INTEGER, PASSWORD).
mandatory	Mandatory	The parameter value is mandatory.
value	Parameter value	The value of the parameter.

12. Enable Telerik Fiddler tracing

You can use Fiddler for debugging.

Enable Fiddler

1. Start Fiddler.
2. Click **Tools** > **Options** > **Connections** tab.
3. Select the **Allow remote computers to connect** check box and make sure that the other check boxes are cleared.



4. If you made changes, restart Fiddler.

Uncomment Fiddler tracing in the web.config file

1. Close any web browsers.
2. Open the web.config file, and uncomment in the block that enables Fiddler tracing (the <system.net> block).

```
<!-- Enable Fiddler tracing using reverse proxy -->
<!--<system.net>
  <defaultProxy>
    <proxy bypassonlocal="False" usesystemdefault="True"
    proxyaddress="http://127.0.0.1:8888" />
  </defaultProxy>
</system.net>-->
```

3. Save the file.
4. Recycle WzpSvc application pool in IIS.

Important: When you are done with the debugging, you must roll back the settings in the **Options** dialog box in Fiddler, and comment out the Fiddler tracing block again. Otherwise, WorkZone Process cannot run unless Fiddler is started, and other applications may get information from the database.

13. Terms and conditions

Intellectual Property Rights

This document is the property of KMD. The data contained herein, in whole or in part, may not be duplicated, used or disclosed outside the recipient for any purpose other than to conduct business and technical evaluation provided that this is approved by KMD according to the agreement between KMD and the recipient. This restriction does not limit the recipient's right to use information contained in the data if it is obtained from another source without restriction set out in the agreement between KMD and the recipient or by law.

Disclaimer

This document is intended for informational purposes only. Any information herein is believed to be reliable. However, KMD assumes no responsibility for the accuracy of the information. KMD reserves the right to change the document and the products described without notice. KMD and the authors disclaim any and all liabilities.