Building Composite Applications with Visual Composer - Process-Oriented Applications

Applies to:

SAP NetWeaver Visual Composer 7.0.

Summary

This article is one in a series of Sergey Kozyrev that works as an application expert and project leader in the defense and security industry.
In his articles he will share from his experience of working with Visual Composer to create and simplify our SAP processes.

Author: Sergey Kozyrev

Company: Israeli defense and security industry

Created on: 10 March 2008

Author Bio

Sergey Kozyrev is an application expert and project leader in the Israeli defense and security industry

Table of Content

Introduction

Hello, I want to introduce myself - my name is Sergey Kozyrev and I'm working as an application expert and project leader in the defense and security industry.

In my coming series of blogs I want to share with you from our experience of working with Visual Composer to create and simplify our SAP processes.

Our organization has a wide implementation of SAP products. These implementations cover most of our IT needs and relate to different area, such as logistics, accounting, human resources, document management and so on.

In the first article Building Composite Applications with Visual Composer – General Overview we gave you a general explanation about our Visual Composer solution without any technical information. In the current articl**e** I want to discuss one of the specifications of our Visual Composer solution – building process-oriented application with Visual Composer.

The Specification of the Process-Oriented Applications

As the part of our Visual Composer solution we wanted to get a response for project creation flow presented by the PS (Project Systems) SAP module. The specification of this module is its high orientation to processes, e.g. transferring information between different project steps, performing signatures collection and so on. In our imagination we saw it as a collection of blocks when each of them is responsible for one specific action. For example, we wanted to have a block for project creation that at the end of the creation process will transfer the user to the project details window. The project details window will show the user the possible actions in the first stage of the project lifecycle. When the user will make one of these actions, e.g. DMS document linkage to the project, the application will transfer him to the block that is responsible for this process.

In general, the system can be viewed as a graph that has blocks as its vertices and its edges are the transportations between them.

This behavior defines three main specifications of the process-oriented applications:

- Not deterministic data flow
- Data transfer between different blocks
- The number of parameters to be transferred is not static

The first specification of the process-oriented applications is that the data flow is not deterministic. From some building block the user can move to several different other blocks. For example, from the document creation block the user can move to the project block, open the document search block or she can open the block that presents the created document data. This functionality is not standard in Visual Composer that applies other tools such as Guided Procedure to move between different models.

Another specification is a need to transfer data from one block to another. For example, if the user wants to link the DMS document to the project, the application has to deliver to the document linkage building block the information about the project. The user will need this information for the linkage operation and (possibly) to return back to the project he came from.

The next specification is that the number of parameters to be transferred is not static. Each building block requires different number of parameters. In the previous example, the document linkage block would have to get only one parameter – the project number, while the block that is opening the linked document from the project would need at least 4 parameters (document type, document number, document version and document part) that represent the document key. The standard Visual Composer solution recommending the use of EPCM events doesn't allow transferring changeable number of parameters.

The System Architecture Alternatives

During the design stage we discussed several alternatives for building process-oriented application.

The first alternative was to present each one of the building blocks by iView and put all the iViews to one Visual Composer model. This approach had a lot of advantages. First of all, it was the easiest one. We had to create only one Visual Composer model and to perform the transportation between the iViews by using a system hyperlink action that is one of standard system actions for the Visual Composer button. There is a possibility to transfer parameters between the iViews by adding the parameters to the URL string for the iView and is well described in the article How to Send Parameters to a Modeled iView

Unfortunately, this standard approach didn't suit our needs. First of all, try to think about a Visual Composer model with something like 20 iViews in the first level (not including nested iViews)! It doesn't sound like an application you want to deal with… The time to compile, deploy and debug can be un-proportionably huge. Secondly, the application will open each new iView in the new browser window and it doesn't suit for all of the transfers. For example, the user is on the first stage of the project and he wants to pass to the second stage. If the browser will open it in the separate window, the user will get **the same project** opened in two different stages together! It can lead to misunderstanding and wrong user behavior.

The other approach is to use GP (Guided Procedure), a standard SAP product that is the part of the NetWeaver 7.0. This solution allows an easy transfer between the iViews with supporting parameters. Unfortunately, we encountered some unsuitability of GP for our system. The most serious was the problem to use EPCM events with GP. The use of EPCM events is the part of Visual Composer environment and is implemented with Signal Out building block. You can use EPCM events for different purposes such as sending events between iViews on the same portal page or transferring data from different sources to the same table. Unfortunately GP refreshes the screen each time an EPCM event is sent. It means that it's rather problematic to combine Visual Composer iView with some other iView on the same screen. For example, in the project model we wanted to show to the user the project tree and near it the actions to perform on the current project stage. The actions were presented by the Visual Composer iView while the project tree was presented by WebDynpro application when both of them belonged to the same portal page. So each time when the user wanted to send an EPCM event from the WebDynpro application to the Visual Composer iView the whole screen was reloaded.

Defining the System Architecture

So we came to the conclusion that we need to develop another approach that will overcome all the above problems.

Our system architecture consists of several parts.

The first part is a JavaScript application that serves as a base for all Visual Composer, WebDynpro and other models. The JavaScript application is connected to the portal role and is started when the user is opening the role. It gets several parameters when the most important of them is the URL of the first Visual Composer model to open to the user. After the deployment all Visual Composer models become the part of portal content and can be found in the portal content tree under Visual Composer directory. They can be easily copied to other portal content areas and their URL can be extracted with simple Preview action. This model the user will see in the beginning when the role will be opened. The meaning of the JavaScript application is to change between Visual Composer models and transfer parameters between them.

The second part is the collection of Visual Composer and WebDynpro models. Each model is responsible for one specific action as it was designated in the beginning (project creation, project update, document view and so on). All of the models were deployed to the portal and are the part of the portal content.

The third part is the SAP table that stores the URLs of the models from the second part. For each model we stored its id, its description and its URL. The meaning of this table is to perform the change of the URL in one place only – in the table - without searching all the places in all the models where it can to appear. To understand the importance of this table let's imagine the following scenario. For example, we have Document Details model that presents all document information including its originals, basic data, classification and so on. You can move to this model from different places – from Document Search model when choosing one of the search results' documents, from Project model when choosing one of the project's documents and so on. Now you want to make some change in your Document Details model. First of all you'll do it in your test environment and then transfer to the production environment. During this transfer you need to perform export for the corrected model from the test to production environment and to change the addresses inside the model. If we wouldn't use the SAP database table to store the URLs, we had to change all the addresses for Document Details model call manually in all the places where it appears. If we forget at least one of the URLs to change, it will address us to the model in the test environment instead of production! The use of SAP database table allows not to perform any changes at all in this case.

Creating Departure Model

So how the model transfer is performed?

Let's see it on some example. Let's imagine that the process you want to perform is to find some document and open its content. The whole process can be divided into two models – the search model and the document content model. The proposed scenario is as followed – the user performs document search, then chooses one of the documents in the search results table and opens it to get additional document details.

The search model includes the input form InputSearch with parameters for search and action button, standard BAPI that performs document search named BAPI_DOCUMENT_GETLIST2 and the output table OutputSearch with the search results (see Figure 1).

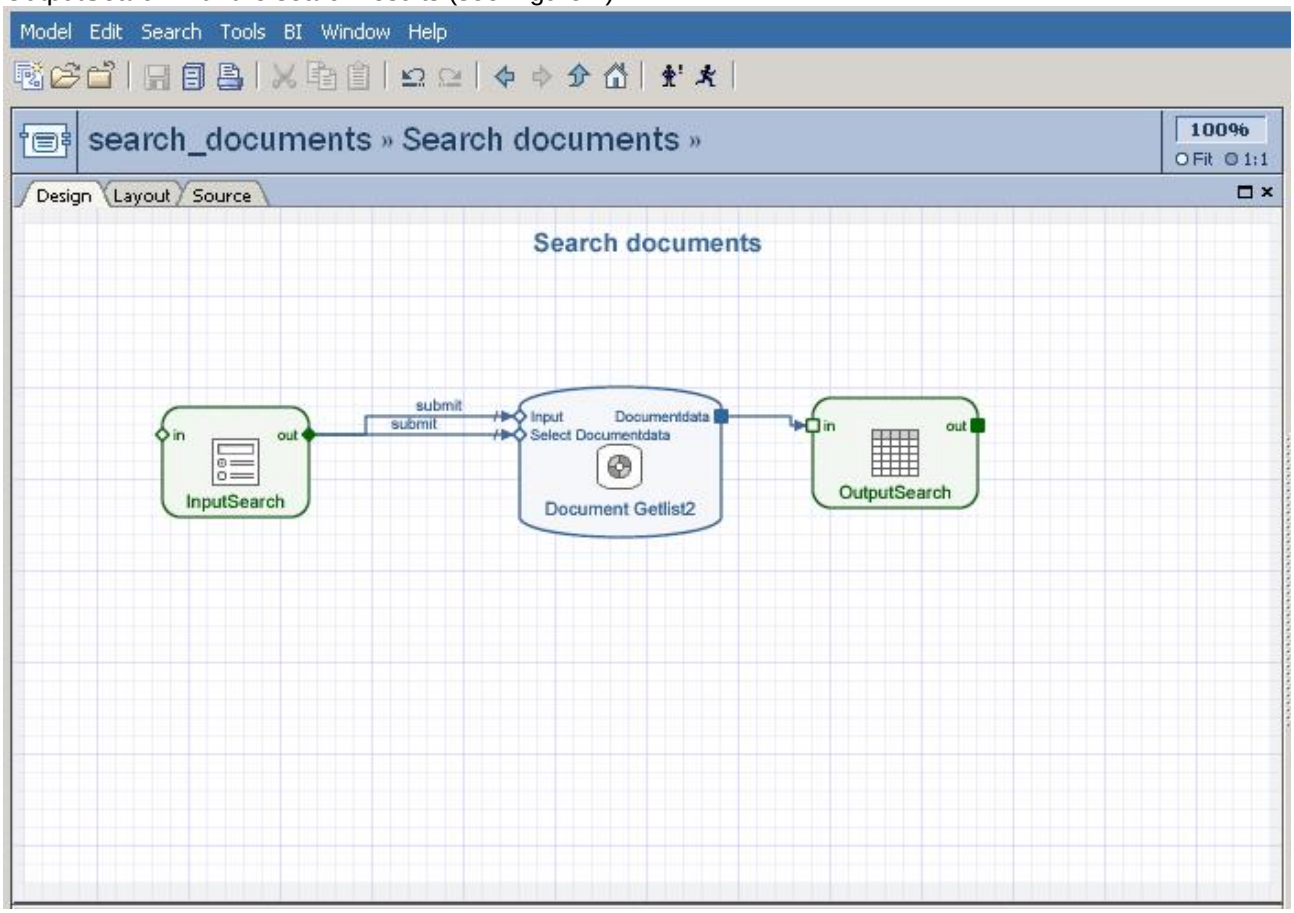

Figure 1. The search model

When the user chooses one of the rows of the OutputSearch table and clicks the button Open we want to open the document content model and to transfer it the key of the chosen document. First of all we should get the URL of the second model (the document content model). As you remember this URL is stored in the SAP table and can be easily obtained from it. We created a function module named ZPRTL_GET_NAV_LINKS_IVIEW that gets the model id and returns its URL. We should define a customer event OPENDOC for the Open button and call the ZPRTL_GET_NAV_LINKS_IVIEW function when OPENDOC event is raised (See Figure 2).

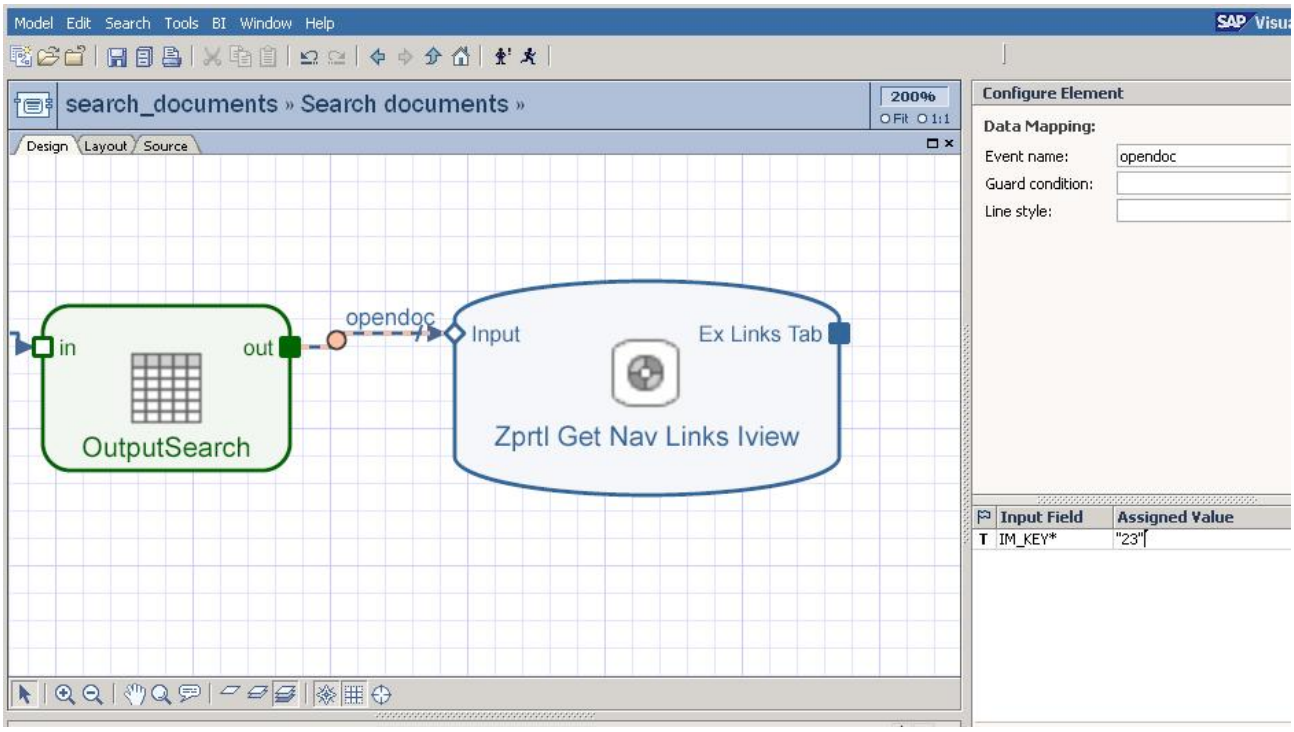

Figure 2. Function ZPRTL_GET_NAV_LINKS_IVIEW call

The next step that we have to do is to define a signal that will perform the transfer to the second model. We define a Signal Out named NEXTPAGE with the following parameters (See Figure 3):

- ADDRESS – this parameter will store the URL of the second model
- STR1 – this parameter will store document type
- STR2 – this parameter will store document number
- STR3 – this parameter will store document version
- STR4 – this parameter will store document part

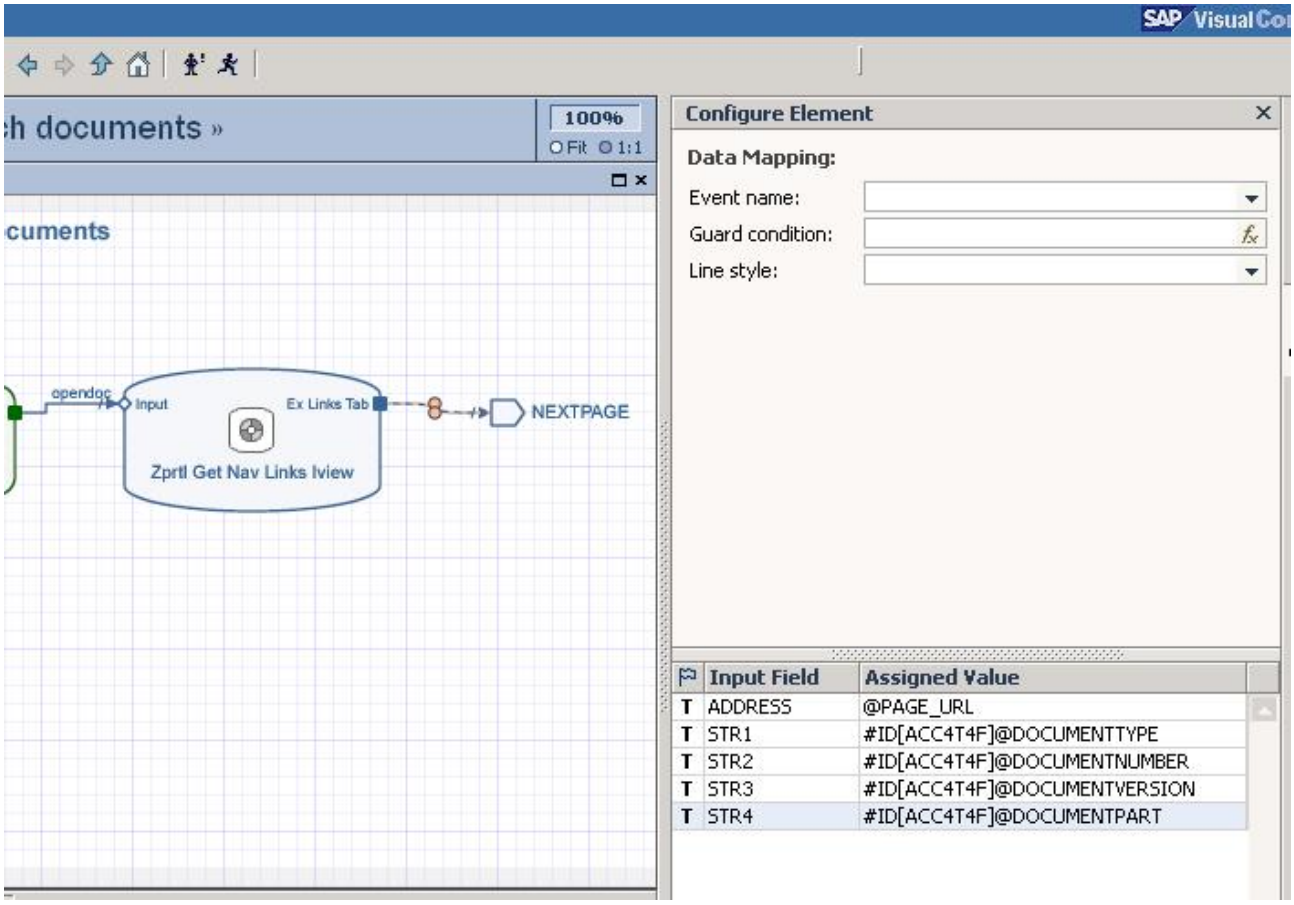

Figure 3. NEXTPAGE signal definition

Now we can connect between the output port of the ZPRTL_GET_NAV_LINKS_IVIEW function and the NEXTPAGE signal and to transfer parameters to it. We transfer URL to the ADDRESS parameter, document type to the STR1 parameter, document number to the STR2 parameter, document version to the STR3 parameter and document part to the STR4 parameter (taken from OutputSearch).

The final step that we need to perform is to deploy the model to the portal environment and to get its URL.

Creating Arrival Model

So in this step the departure model is created and deployed. Now we need to create the arrival model – in our example it's the document content model. We want from this model to get the document key sent from the document search model and to show the document information.

Firstly, we choose a Start point and drag it to our model. For the Start point we define 4 parameters named STR1, STR2, STR3 and STR4, exactly as in the departure model (See Figure 4).

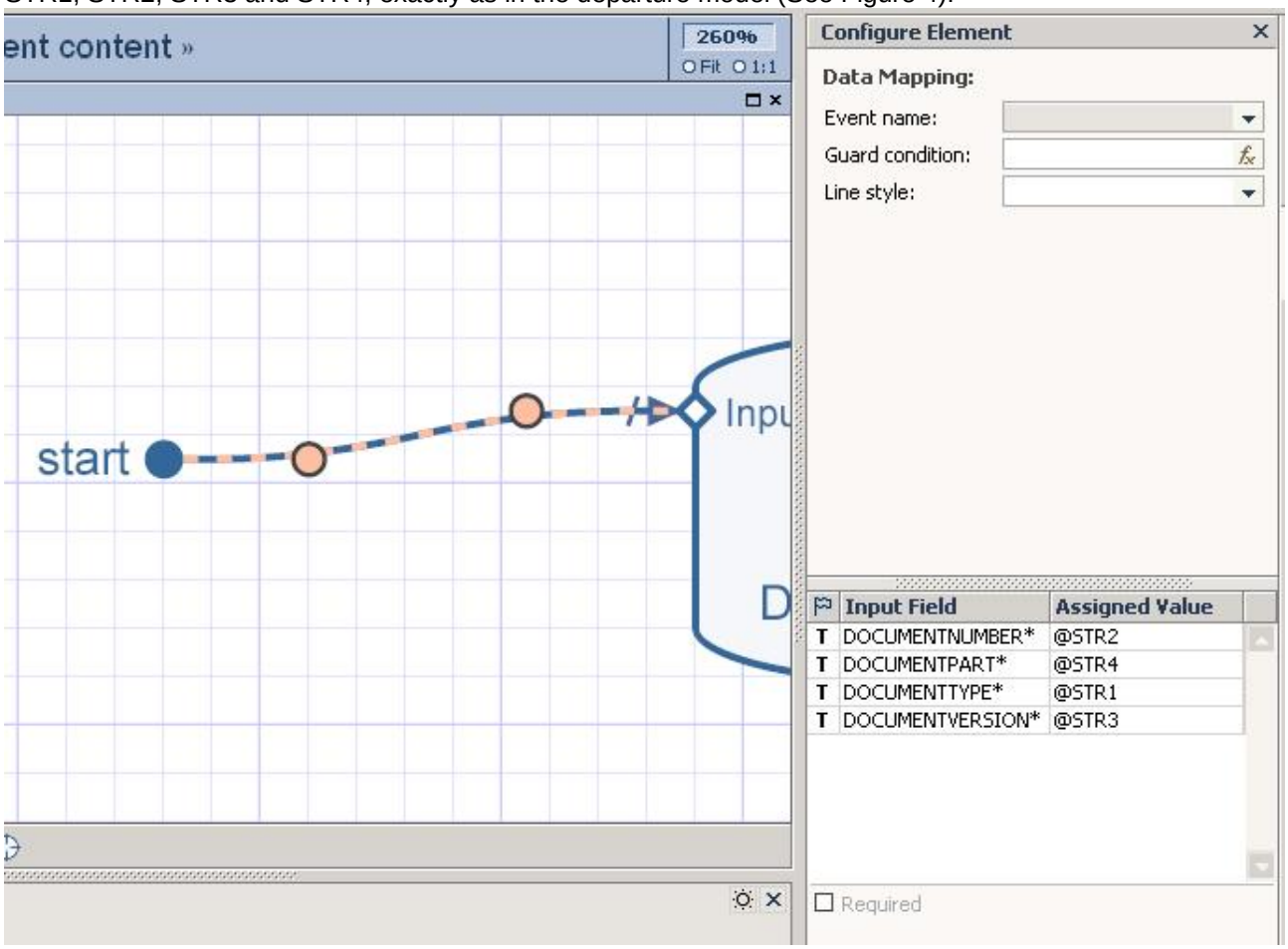

Figure 4. Start point definition

To get the document information by its key, we are using standard BAPI named BAPI_DOCUMENT_GETDETAIL2. So we connect it to the Start point and defining the output table for it named OutputDoc table. The table will include all the document information after running the BAPI (See Figure 5).

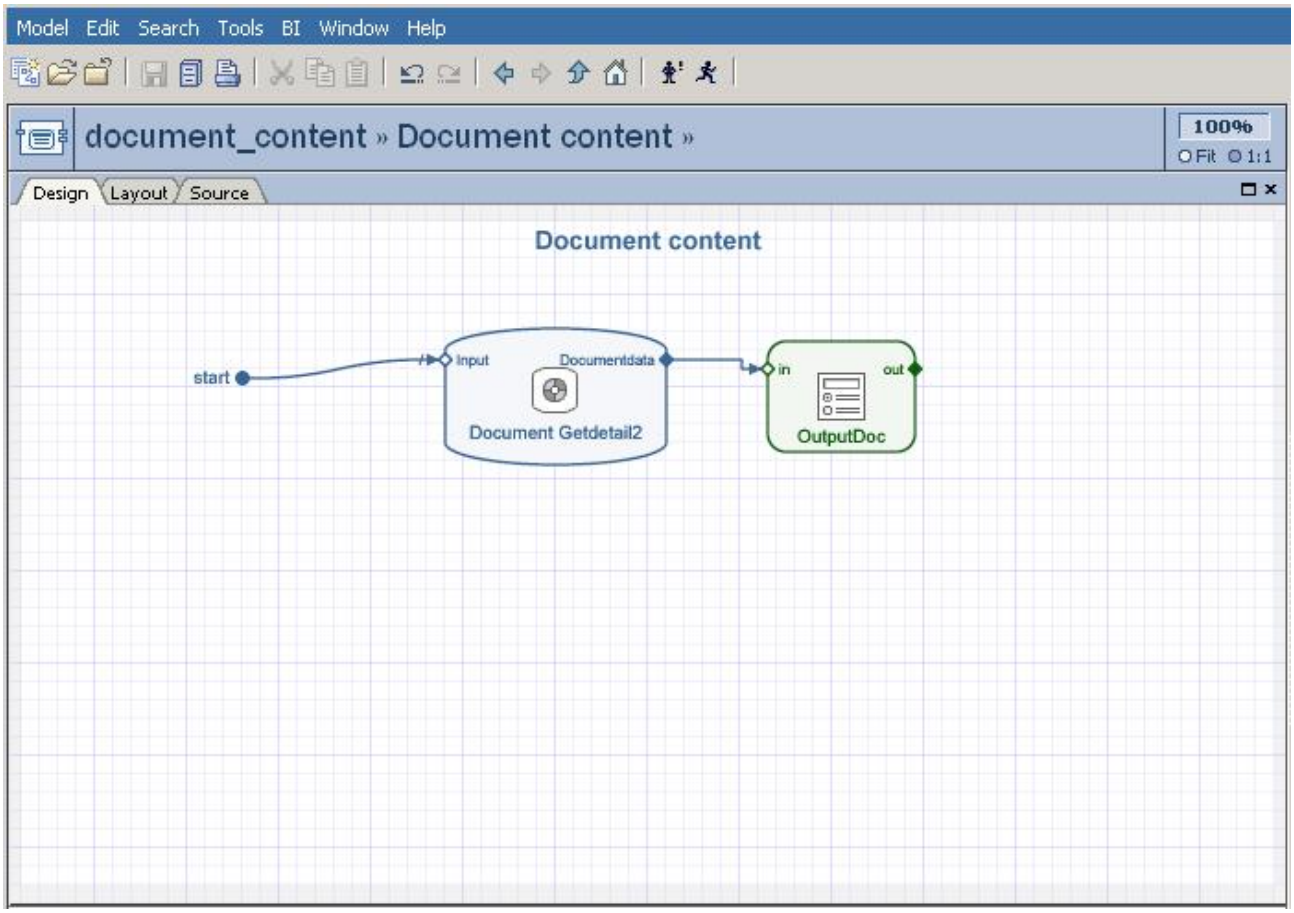

Figure 5. The document content model

The last step, like in the departure model, is the deployment to the portal environment.

Defining JavaScript Application

Ok, we did some work and now we have two models, the document search model that throws NEXTPAGE signal with some parameters and the document content model that gets the same parameters. But how the transfer is performed?

The main "responsibility" for the transfer should be dedicated to the JavaScript application written by our brilliant programmers.

JavaScript application is also deployed to the portal and gets the URL of the departure model as its parameter. It connects the URL to its main frame so that when the user calls for the JavaScript, it shows the given URL.

In our example we want to connect to it the URL of the document search model. In this step you can check yourself and try to open JavaScript and to see the first model (See Figure 6).

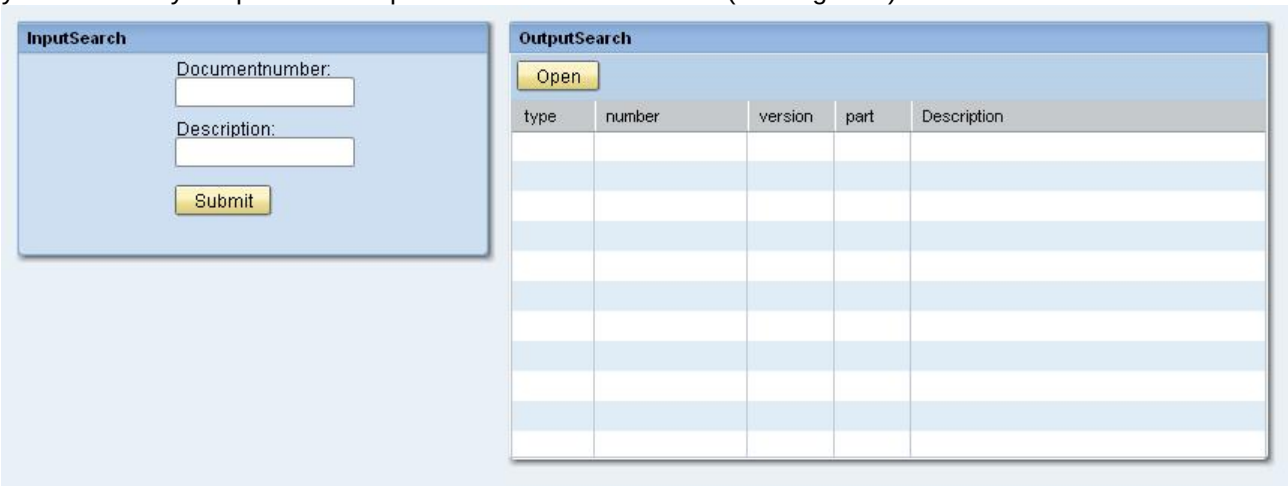

Figure 6. The document search model

Why do we need it? The answer is simple. We connected the departure model to the main frame of the JavaScript application to catch the NEXTPAGE signal sent by the user when he clicks on the Open button!

Each time the NEXTPAGE signal is thrown, we catch it and activate a relevant function. The function logic is as follows:

- Parse the parameters got from the signal. The function expects to receive the ADDRESS and unlimited number of STR1 ... STRn
- Create the address of the arrival model. The address is created by concatenating the ADDRESS parameter and STR1 … STRn parameters as described in this article
- Change the URL connected to the main frame of the JavaScript application to the new address

That's it…

Let's check our application. We run the JavaScript application that opens the document search model. Then we perform some search for the documents and get the search results (See Figure 7).

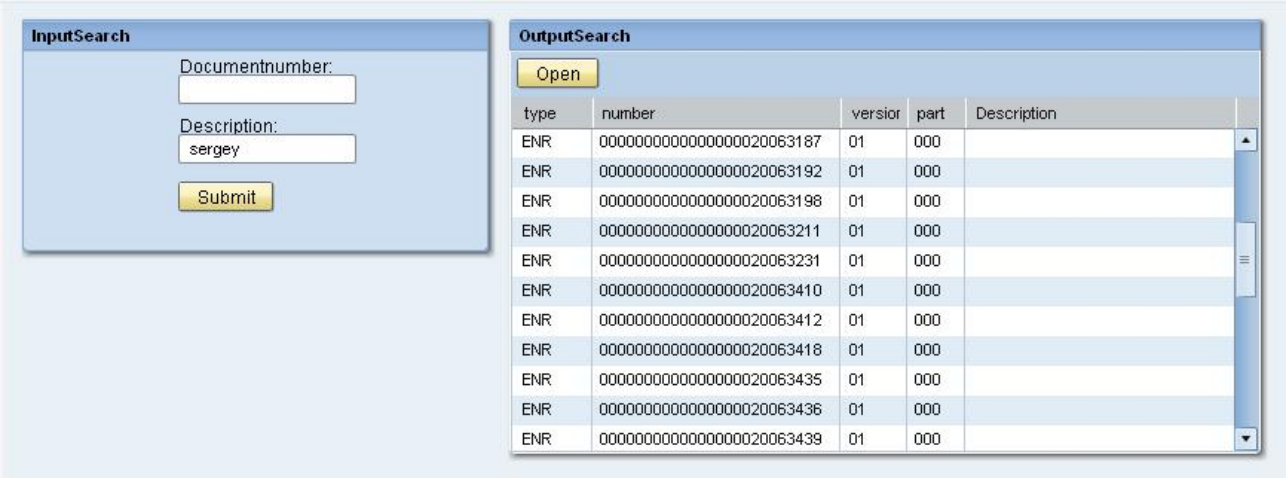

Figure 7. Running document search

In the next step we choose one of the documents in the output table and click on the Open button. In this step the document content model is loaded and we can see the document information (See Figure 8).

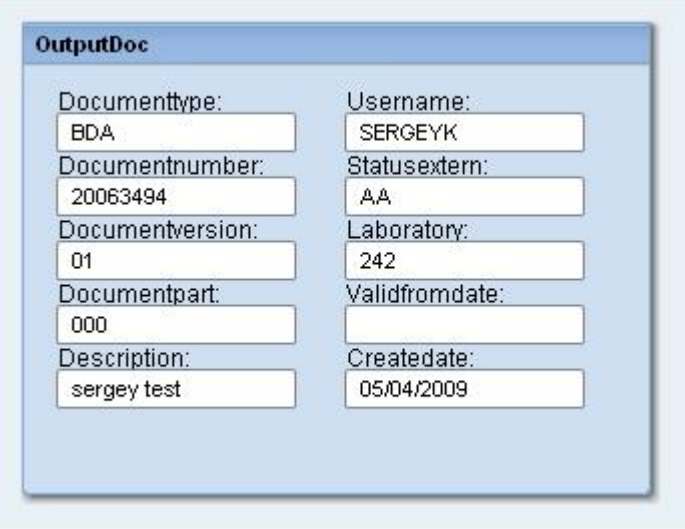

Figure 8. Document information

Advantages and Disadvantages

Let's try to understand what are the advantages and the disadvantages of the created application.

We can point out to several main features:

1. We succeeded to create an application that can transfer between models and deliver parameters from one model to another
2. We have a total control to open the arrival model in the same portal window or in the new one. The rule can be defined inside JavaScript and (optionally) transferred from Visual Composer iView as one of the parameters
3. We can throw EPCM events without refreshing the screen so the information entered by user doesn't disappear!
4. The application can transfer unlimited number of parameters between models
5. There is possibility to perform unlimited number of model transfers. Each transfer is just another call to the same JavaScript function!
6. The application structure is extremely flexible. It allows to define several iViews in the same screen and to play with their size and other properties (more in the next blogs…)

The main disadvantage is the development time. You need an acquaintance with several tools like Visual Composer and JavaScript to provide connection between them. But we think that the result is worth the invested work.

Summary and Looking Forward

In this blog we wanted to discuss one of the specifications of our Visual Composer solution – building process-oriented application with Visual Composer. We checked the alternatives for development of a process-oriented application with Visual Composer, described the system architecture and gave a detailed example have this architecture can be applied in our solution. In the next blogs we will continue to discuss each one of the specifications raised in the first article.

We will appreciate to get your comments and suggestions about the topics that are the most interesting for you and you want to see them in the next blogs.

Related Content

Visual Composer page at SDN

Visual Composer Forum

Copyright