Develop composite applications with SAP NetWeaver Composition Environment 7.1

Part 2 — User interfaces and the tools that help you create them

by Volker Stiehl

Volker Stiehl
Product Manager,
SAP AG

Volker Stiehl received a degree in computer science from the Friedrich-Alexander-University of Erlangen-Nürnberg, Germany. He was a consultant for distributed J2EE-based business solutions and integration architectures at Siemens for 12 years. In 2004 he joined SAP's product management team for a composite application development toolset, where he's also responsible for methodologies and best practices. He holds workshops in composite applications and is a regular speaker at conferences, such as SAPPHIRE, SAP TechEd, and JavaOne. You may reach him at volker.stiehl@sap.com.

Rapid change and a constant need for innovation form the foundation of today's business world. The need to continuously improve your business processes has become as much about survival as competitive advantage. For example, you may have the highest quality product in your market, but your customer is running out of patience with your slow customer service response. Your nearest competitor has an excellent product too, and its customer service response is quick and efficient. If you need to either increase customer satisfaction or risk losing your customer, then it's important to shorten the response time required for complaints coming through your call center. How do you increase customer satisfaction without a new, large-scale IT implementation? Enter the composite application.

SAP's composite applications are packaged applications that sit on top of existing enterprise solutions and reuse their functionality to form new collaborative business processes. As Web and enterprise services continue to grow in number, reusable, collaborative composite applications enable you to take advantage of the many functions already available with a limited software effort. To help you maximize their potential, SAP has introduced SAP NetWeaver Composition Environment (SAP NetWeaver CE), which presents the design and runtime environment for composite applications.

This three-part series of articles shows you the characteristics, architecture, and challenges of developing composite applications and how you address these issues. The first installment (*SAP Professional Journal*, May/June 2008) provides an introduction to composite applications, SAP NetWeaver CE, and SAP Composite Application Framework (SAP CAF). It addresses the bottom layer of the composite application architecture: services and business objects, as shown in **Figure 1** on the next page. (The first article also sets up the customer service complaint example.)

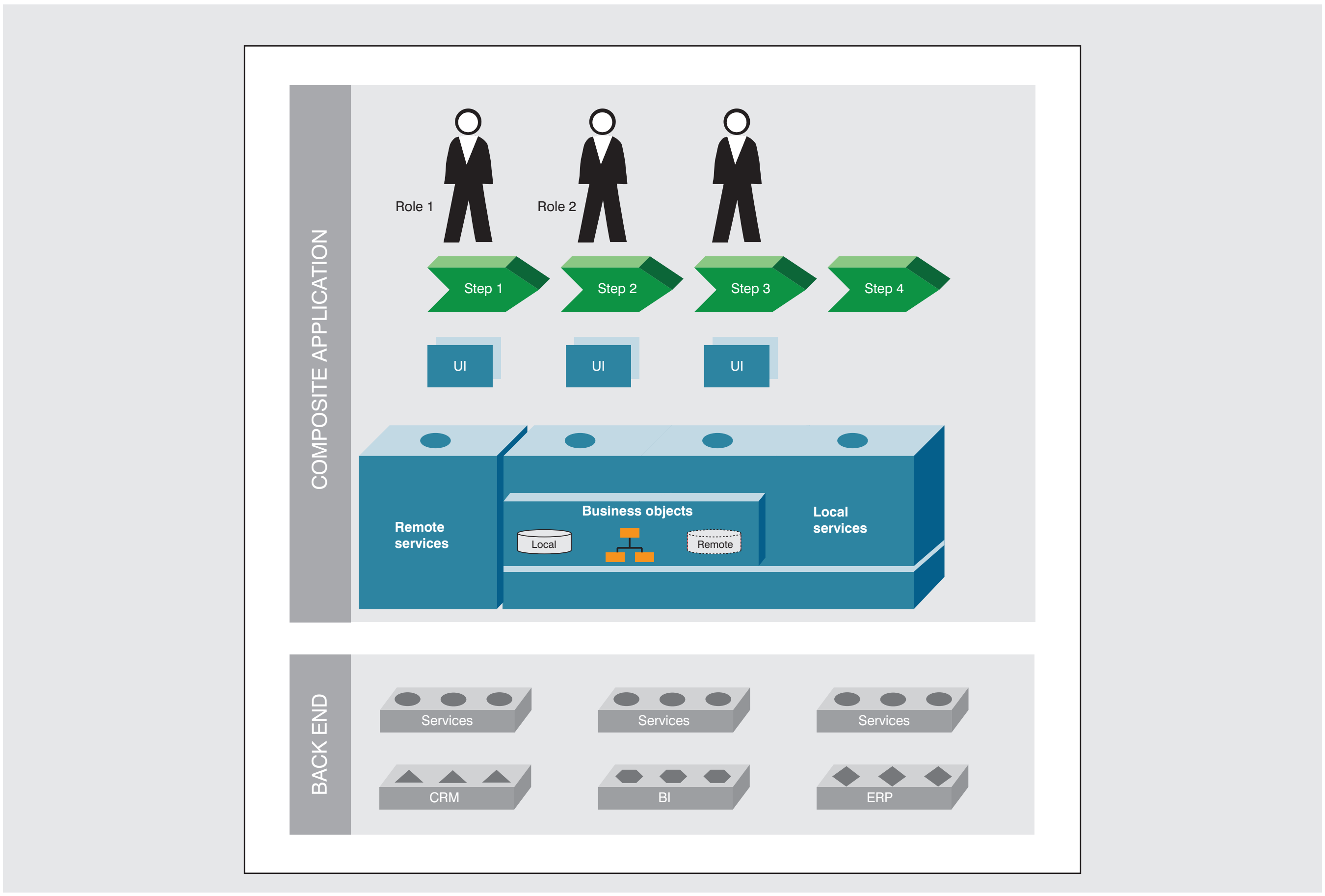


Figure 1 Architecture of a composite application

This article, the second installment, examines the next layer of the architecture: user interfaces (UIs), and the various tools that you can use to create them. It explores why SAP NetWeaver Visual Composer (referred to as simply Visual Composer in this article) is the tool for developing "lightweight" interfaces. The article also explains two techniques that you can use to consume Web services along with their advantages and disadvantages. Then, the article walks you through the tasks for creating UIs with Visual Composer. You will create three UIs: two for the call center agent and one for the complaint manager. The first UI is for the call center agent to enter a customer's complaint. The second UI is for the complaint manager to review the complaint. The third UI is for the call center agent to review the complaint manager's comments and respond to the complaint accordingly.

The article concludes with some advanced features that you can include in your composite applications.

The third and final installment in this series will consider the top layer processes (roles and steps) and help you pull them all together into new, innovative, collaborative business processes. It will explore modeling the processes with SAP Guided Procedures (SAP GP), which have a browser-based development environment and a robust runtime engine to take care of process execution and notifying participating users.

In this article, I assume that you have read the previous article in this series. This article builds upon that knowledge. As you learned in the previous article, a composite application development project can be broken down into five steps:

Why use Visual Composer instead of Web Dynpro?

Visual Composer is for “lightweight” UIs in which the delivered standard UI components suffice. It’s important to remember that with Visual Composer, you have to live with standard functionality; you cannot extend it. If you reach the limits of Visual Composer and still need a particular UI functionality that Visual Composer doesn’t have, then your only option is to start from scratch with Web Dynpro.

If you want to integrate UIs into SAP GP, it’s much easier in Visual Composer than it is in Web Dynpro. Your first composite application probably won’t (or probably shouldn’t) address highly complex use cases, so Visual Composer is the right tool at first. However, Visual Composer has its limitations —no programming possible, no nested tables, no F4 helps, no dynamic content, and no complex input validation — in other words, no use cases in which you want to programmatically alter the UI’s appearance. In addition, the skills required to use Web Dynpro are much more advanced than those for Visual Composer. It’s not just a small step; it’s a huge learning curve that developers have to surmount to use Web Dynpro.

Step 1: Search for services in the Enterprise Services Workplace (ES Workplace)

Step 2: Simplify the service interfaces with SAP CAF

Step 3: Create UIs with Visual Composer

Step 4: Model business objects with SAP CAF

Step 5: Model collaborative processes with SAP GP

Steps 1 and 2 were covered in Part 1 of this series, and Steps 4 and 5 will be examined in the final article. This article deals with Step 3.

Let’s begin with an explanation of why Visual Composer is the tool SAP recommends for developing composite applications and its ability to consume Web services, reducing the software effort required.

Developing composite applications using Visual Composer

Visual Composer plays a key role in the overall enterprise service-oriented architecture (enterprise SOA) story. It is a visual graphic modeling environment for UIs, making it well suited as a tool that helps to bridge the gap between business and IT people. (To understand why Visual Composer is the tool of choice here instead of SAP Web Dynpro, see the sidebar above.) With such a tool at hand, these teams can sit together and create a prototype of the UI within minutes. This ensures that the business people get what they really want. Early projects confirmed these benefits: Customers became excited about this new way of creating software. For the first time the business people felt like they were part of the project and that they were being heard. These advantages, which can be gained by using modeling environments, hold true for the other tools such as SAP GP and SAP CAF as well.

One of the key criteria for the development of Visual Composer was that it should address efficient UI modeling in an intuitive way. You don’t need any programming skills to work with Visual Composer, and the learning curve is low. This reduces the entry barrier as much as possible. Even untrained people can achieve initial results in a short time.

For a good foundation on Visual Composer, I recommend that you check out the information on the SAP Help Portal. Use the following link and subsequent menu path to learn more about Visual Composer modeling: http://help.sap.com; SAP NetWeaver → SAP NetWeaver CE. Click on the SAP NetWeaver Composition Environment Library link. In the tree on the left in the window that appears, expand the node SAP NetWeaver Composition Environment Library → Developer's Guide → Developing and Composing Applications → Modeling Composite Views with Visual Composer.

Note!

This article doesn't cover SAP NetWeaver Visual Composer in detail. You will find a thorough overview of it in Karl Kessler's articles "Get started creating SAP Enterprise Portal iViews with Visual Composer" and "Advanced techniques for enhancing your SAP Enterprise Portal iViews with Visual Composer" (*SAP Professional Journal*, November/December 2005). The basic concepts of this tool remain the same as they were in 2005, but some of the details have changed significantly. The version discussed in these articles produced iViews. The version of SAP NetWeaver Visual Composer that you use with SAP NetWeaver CE creates standalone applications. Look for articles on the latest version of SAP NetWeaver Visual Composer in future issues of this publication.

Another reason that Visual Composer makes sense as a development tool is the way in which it enables you to consume Web services.

Consuming Web services in Visual Composer

There are two techniques for consuming services in Visual Composer:

- **Calling the Web services via logical destination:** This is SAP's recommended approach because it guarantees the independence of your UIs from the IT landscape on which your application is running. As a consequence, you have to create logical destinations for each service that your UI calls. For example, if your UI consumes four services, you must implement four wrapper services in SAP CAF (as shown in the SAP CAF exercise in the first article) and then you publish them via logical destinations. Sometimes, that's just too much effort for a quick solution built for a few users. In these cases, the following alternative is an option.
- **Directly calling the Web services from the UI:** Although you know the drawbacks of direct service consumption (as discussed in the previous article), you can sometimes neglect them because of the limited scope of your solution. In this situation, direct service consumption definitely makes sense. It means that you don't have to implement an additional SAP CAF layer to wrap the services you want to consume. Although not recommended, I don't want to hold back this technique, especially if you want to consume services from the Services Registry (e.g., the Services Registry of the ES Workplace). You can easily do this with Visual Composer bringing a significant efficiency boost. If you use Visual Composer as a standalone tool for one-shot applications to solve urgent problems quickly or for a limited number of end users, it's called developing a composite view.

Why do I mention both of these solutions although only one is recommended? For a pure composite application, the independence of the back-end systems is of paramount importance. However, there are use cases, such as the composite view, where you can live with dependencies.

Now, you are ready to create the UIs for the call center agent and the manager.

Step 3: Create UIs with Visual Composer

There are several tasks that make up Step 3. First, you need to set up the Java server for service consumption and establish any necessary proxy settings. Then, you work with Visual Composer's Layout board to create the data, event, and screen flow for the application and its Design board to define the look and feel of the UI. Next, you must determine what services you need to model the UI and where you might find them. Then you obtain the UI components, add any data that doesn't have a service to the UI, and test the UI.

Before you start creating, or modeling, the first UI, you need to prepare the Java server so Visual Composer can consume services from the ES Workplace's Services Registry. For this, you will have to define the following three logical destinations (as you did in the first article):

- The UDDI-based registry

  Destination type: WSDL
  Destination name: UDDI_DESTINATION
  URL: http://sr.esworkplace.sap.com/uddi/wsdl/uddi_v3_service.wsdl
  System: Java
  Authentication: HTTP authentication
  User ID/password: (basic)
  UserID: sruser
  Password: eswork123

- The classifications

  Destination type: WSDL
  Destination name: CLASSIFICATION_DESTINATION
  URL: http://sr.esworkplace.sap.com/ClassificationService/CS?wsdl
  System: Java
  Authentication: HTTP authentication
  User ID/password: (basic)
  UserID: sruser
  Password: eswork123

- The system on which the services run (e.g., system HU2)

  Destination type: Services Registry
  System: ABAP
  System name: HU2
  Host name: iwdf1030
  Installation number: 0120003411
  Client: 800
  Authentication: HTTP authentication
  User ID/password: (basic)
  UserID: <*your account*>
  Password: <*your password*>

The proxy settings of your Java server are related to these entries. The Java server connects to the Services Registry and the back-end system to collect the appropriate service information. Whether a proxy is necessary for these connections or not is defined in the server's proxy settings. This is how you find the proxy settings:

1. Open SAP NetWeaver Administrator via URL http://<*host*>:<*port*>/nwa.

2. Click on SOA Management and then Technical Configuration.

3. Click on System Global Settings. This takes you to the editor for your proxy settings. Make the proxy entries as needed (e.g., host, port, exclusions).

Now you can begin to employ Visual Composer.

Working in Visual Composer

To use Visual Composer, you only have to understand the following two boards:

- On the Design board, you primarily focus on the data, event, and screen flow. You define on which services and data fields the UI works, how a service receives its input data (e.g., from a form), how to display the output data that a service call returns (e.g., via a table or a chart), and how events trigger actions such as calling services or leaving the screen.

- On the Layout board, you define in detail how a UI appears to the end user. Each piece of the UI used by the Design board, such as forms and tables, is refined on the Layout board. For example:

  - For a form, you define the UI controls you'd like to use for each chosen field (e.g., an input field, a drop-down list, a date picker, or a radio button).

  - For a table, you define the order of the fields, whether multi-selection is allowed, or whether the table is editable, and if so, which validation rules apply to the data entered.

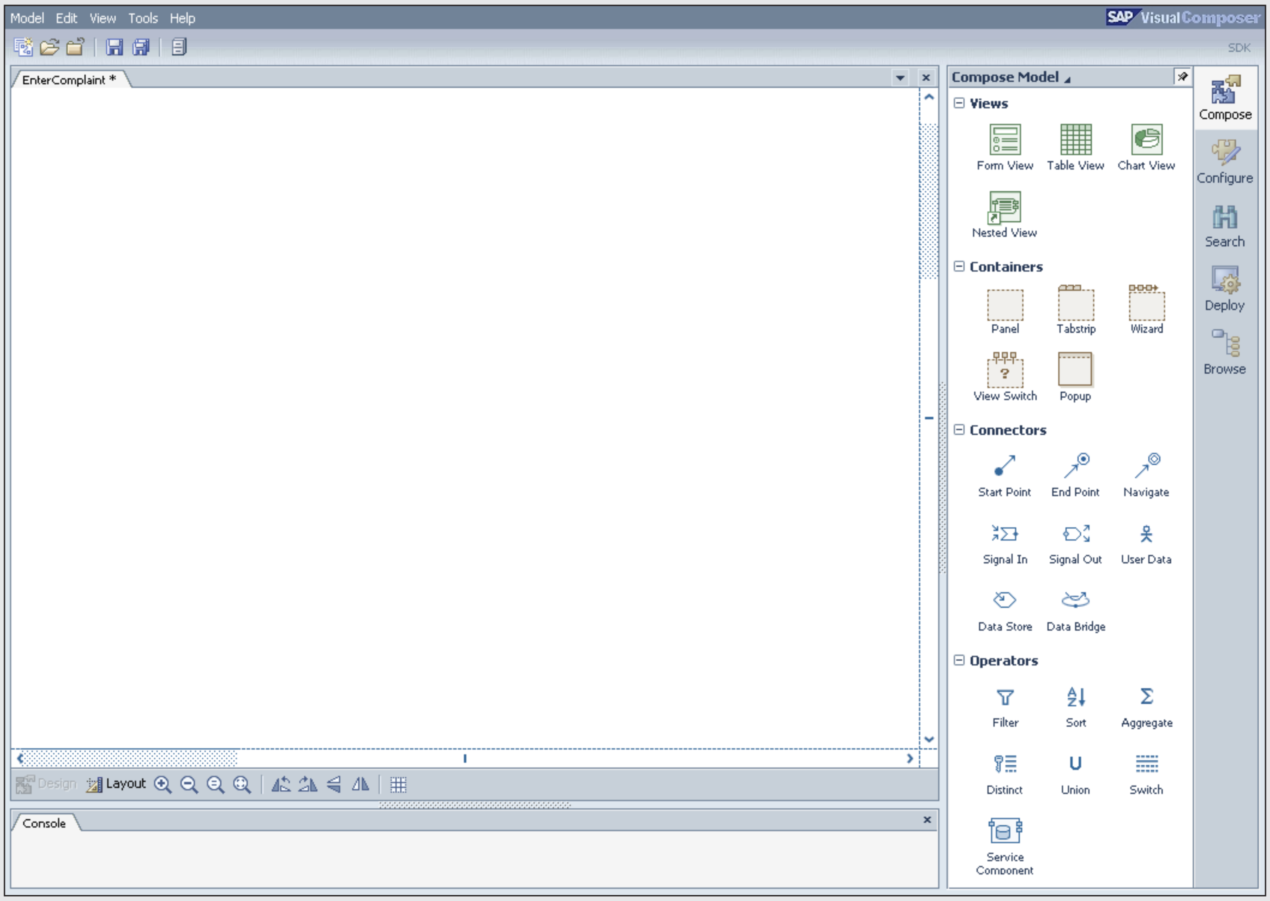


Figure 2 Beginning screen — the Design board — for SAP NetWeaver Visual Composer

This is how Visual Composer supports a clear separation between the screen's details and its associated data flow. The separation influences the development process significantly: You typically start on the Design board to sketch the primary data flow before you detail the UI's appearance on the Layout board. Let's see how this design philosophy looks in action.

Start Visual Composer via the URL http://<*server*>:<*port*>/VC/freestyle.jsp and create a new model for the call center agent (main menu item Model → New, pick Composite View as the model type, and leave all default values as they are). Name your model EnterComplaint, and assign it to a new development component "spj" (you create a new development component via the New button to the right of the development component's drop-down field). After you leave the dialog, you see the entry screen for Visual Composer (see **Figure 2**).

At the top of this screen is the main menu; below that is the main toolbar. On the right side of the screen, you see the task panel toolbar and to the left of the toolbar you have the context-sensitive task panel. Depending on the function you choose from the task panel toolbar, the contents of the task panel window may change. In the middle is the area for your modeling efforts: It's the workspace with its associated workspace toolbox directly adjacent at the bottom. At the bottom of this screen, you can switch between the Design and Layout boards. Finally, you see the console window, which contains error and diagnostic messages to help you in case of unexpected system behavior.

The next task in creating UIs for the composite application is to determine what services you need to access to model the UI for the call center agent.

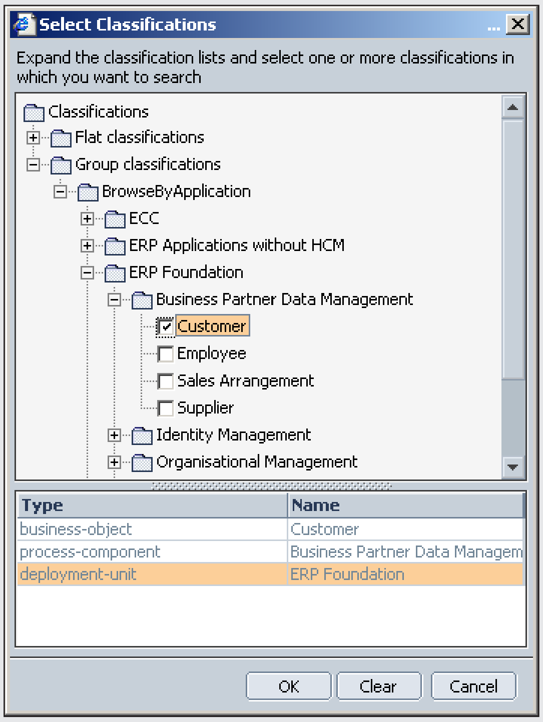
Searching for services

As a developer, you should think about the services you need for modeling the UI. The call center agent needs one service that searches for customers based on their names and a second service to retrieve the details of the customers listed in the search results. For the initial customer search, use the service directly from the ES Workplace (this is not the recommended way of using services in a composite application but it is useful for one-shot applications). For the detail retrieval, you should use your dummy service from the first article, readCustomerBasicData, through Visual Composer. Let's start with the following steps to search for the service in the ES Workplace:

1. Click on the Search button on the task panel toolbar. In the Search dialog that appears in the task panel, select Services Registry from the Search drop-down list. This automatically changes the entry in the Type drop-down list to Services Registry as well.

2. Because the Services Registry contains services grouped by classifications, you have to tell Visual Composer which classifications you want to search. To accomplish this, follow the Advanced Search link.

3. A dialog pops up listing all the classifications, as shown in **Figure 3**. These classifications have already been retrieved from the ES Workplace's Services Registry and let you know whether the configuration of your connection is working. If you get an error message (e.g., Visual Composer can't establish the connection), double-check your entries (proxy settings, logical destinations) in the SAP NetWeaver Administrator. Since you're only looking for services related to the Customer business object, expand the classifications tree, select Customer, and click on OK.

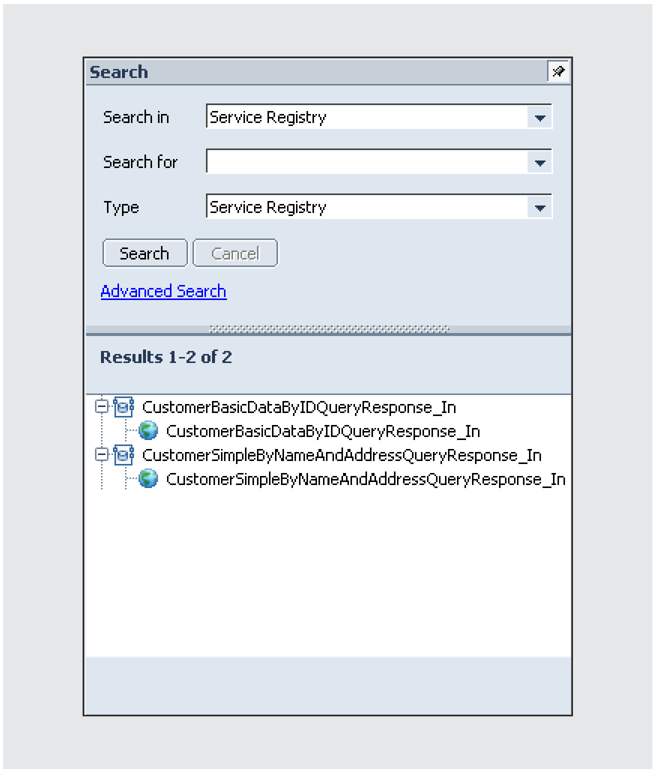


Type	Name
business-object	Customer
process-component	Business Partner Data Managem
deployment-unit	ERP Foundation

Figure 3 Selecting classifications in Visual Composer

Note!

It is also possible to search for services without selecting a classification. To do this, you simply enter a search string (perhaps with wildcards) in the Search for field and then click on Search (e.g., customer* to search for all customer-related services).



Results 1-2 of 2

CustomerBasicDataByIDQueryResponse_In
CustomerBasicDataByIDQueryResponse_In
CustomerSimpleByNameAndAddressQueryResponse_In
CustomerSimpleByNameAndAddressQueryResponse_In

Figure 4 Listing all customer-related services

4. Click on Search to get a list of all customer-related services, as shown in **Figure 4**.

5. Make sure that you are on the Design board and drag the service named CustomerSimpleByName AndAddressQueryResponse_In into the workspace. When you do this, Visual Composer retrieves the input and output parameters of this particular service call and displays them within the service, as shown in **Figure 5**, based on the service's Web Services Description Language (WSDL) description.

6. Next, search for your self-developed enterprise service dummy, readCustomerBasicData. In the first article, you created a logical destination for it: CafCustomerBasicDataDest. This destination shows up in the Search in drop-down list. Select it and see how the Type drop-down list automatically changes to WSDL. Click on Search and your service appears in the Results area. Drag and drop your service into the workspace underneath the first service as well. Your screen should now look like the one in **Figure 6.**

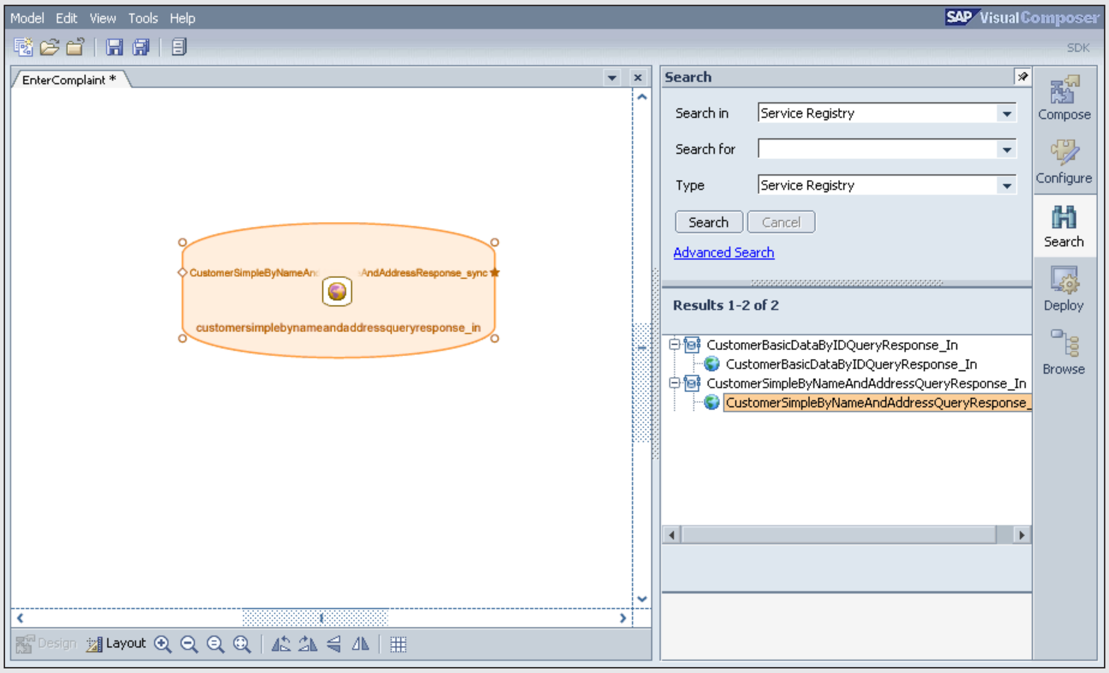
Once you have determined which services you need for the UI, you have to obtain the appropriate UI components.

Obtaining the UI components

Both the searching and retrieving details services are ready to derive the UI components from their WSDL descriptions. To determine how to connect UI components to services, and vice versa, you need to clearly understand how your UI ought to behave. In the example, the call center agent enters the customer's name in a form field and clicks on the Submit button to call the service. The system displays the values that the service returns in a table. Next, the call center agent picks the customer from that table for whom he or she needs the detailed data and clicks on it. So, each time you select a customer from the table, you need to call the readCustomerBasicData service and display the detailed data returned by that service in a second form. You can model this process with Visual Composer:

1. Every service has input and output parameters. Input parameters are indicated by a circle on the left side of each service icon, and output parameters by an appropriate icon on the right side of the service icon. They are called input plugs and output plugs, respectively. To model an input form, move your mouse to the input plug of your CustomerSimpleByNameAndAddressQueryRes ponse_In service and drag and drop a connection line from this plug to somewhere else in the workspace; then, release the mouse button. A context menu pops up and shows you all your options. In this case, you want to create an input form, so select Form View from the context menu, as shown in **Figure 7** on page 34.

2. Next, a dialog automatically pops up, listing all input parameters for the associated service. Now you can see the disadvantage of direct service consumption: You are confronted with the entire complex interface of the enterprise service. By default, Visual Composer selects all parameters



Search

Search in: Service Registry

Search for:

Type: Service Registry

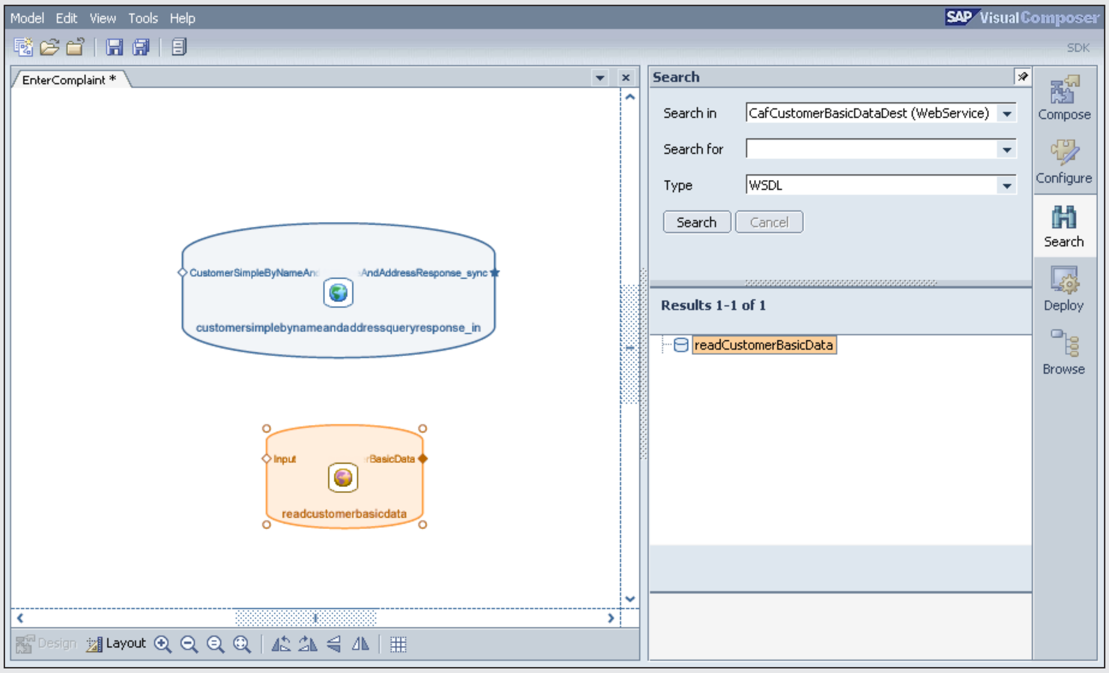
Search | Cancel

Advanced Search

Results 1-2 of 2

CustomerBasicDataByIDQueryResponse_In
CustomerBasicDataByIDQueryResponse_In
CustomerSimpleByNameAndAddressQueryResponse_In
CustomerSimpleByNameAndAddressQueryResponse_

Figure 5 Dragging a service into the workspace



Search

Search in: CafCustomerBasicDataDest (WebService)

Search for:

Type: WSDL

Search | Cancel

Results 1-1 of 1

readCustomerBasicData

Figure 6 Service retrieval via logical destination

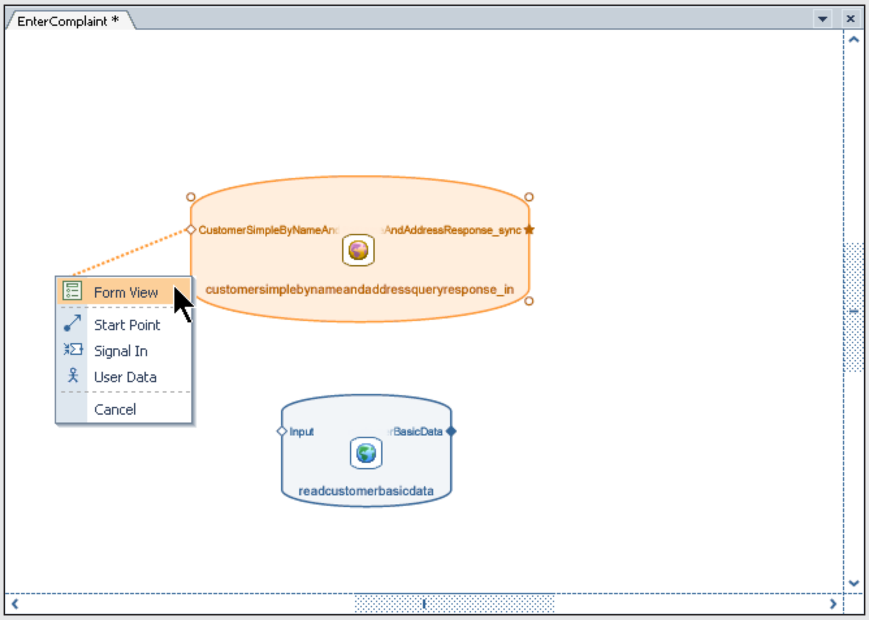
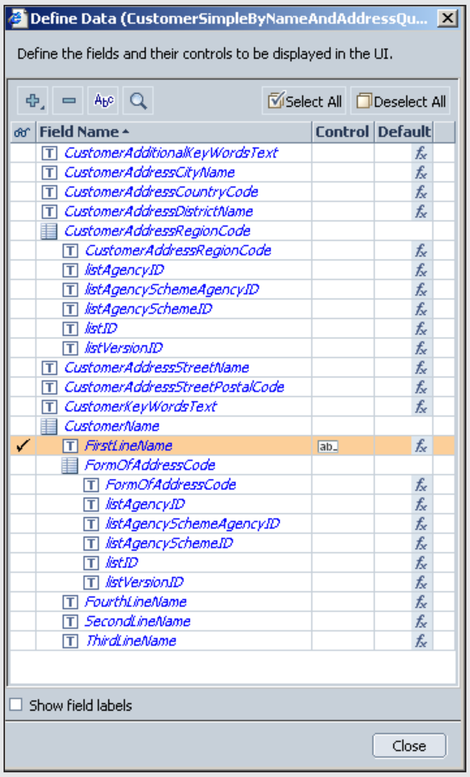


Figure 7 Creating an input form

of the service for the input form. Since you need only a fraction of those parameters for the service call, I recommend that you first deselect all the parameters (by clicking on the button Deselect All) and then choose only those fields that are relevant to this UI. In this case, you only need FirstLineName (see **Figure 8**).

3. Next, drag a line from your service's output plug and release the mouse button somewhere in the workspace. Once again, a context menu pops up listing your options. Choose the Table View entry from the context menu to display the results in a table instead of an array. From the dialog that automatically pops up and shows you the output parameters, choose only FirstLineName and ID. You need the ID later to retrieve the detailed data for that customer.

The label "submit" represents an event that Visual Composer creates for you. Visual Composer identifies the form as an input form. To transmit something from a form to a service, you need a button associated with the event. Visual Composer generates both the button and the event


Define Data (CustomerSimpleByNameAndAddressQu...

Define the fields and their controls to be displayed in the UI.

Select All   Deselect All

	Field Name	Control	Default
	CustomerAdditionalKeyWordsText		fx
	CustomerAddressCityName		fx
	CustomerAddressCountryCode		fx
	CustomerAddressDistrictName		fx
	CustomerAddressRegionCode		
	CustomerAddressRegionCode		fx
	listAgencyID		fx
	listAgencySchemeAgencyID		fx
	listAgencySchemeID		fx
	listID		fx
	listVersionID		fx
	CustomerAddressStreetName		fx
	CustomerAddressStreetPostalCode		fx
	CustomerKeyWordsText		fx
	CustomerName		
✓	FirstLineName	ab	fx
	FormOfAddressCode		
	FormOfAddressCode		fx
	listAgencyID		fx
	listAgencySchemeAgencyID		fx
	listAgencySchemeID		fx
	listID		fx
	listVersionID		fx
	FourthLineName		fx
	SecondLineName		fx
	ThirdLineName		fx

☐ Show field labels

Close

Figure 8 Picking fields for the input form

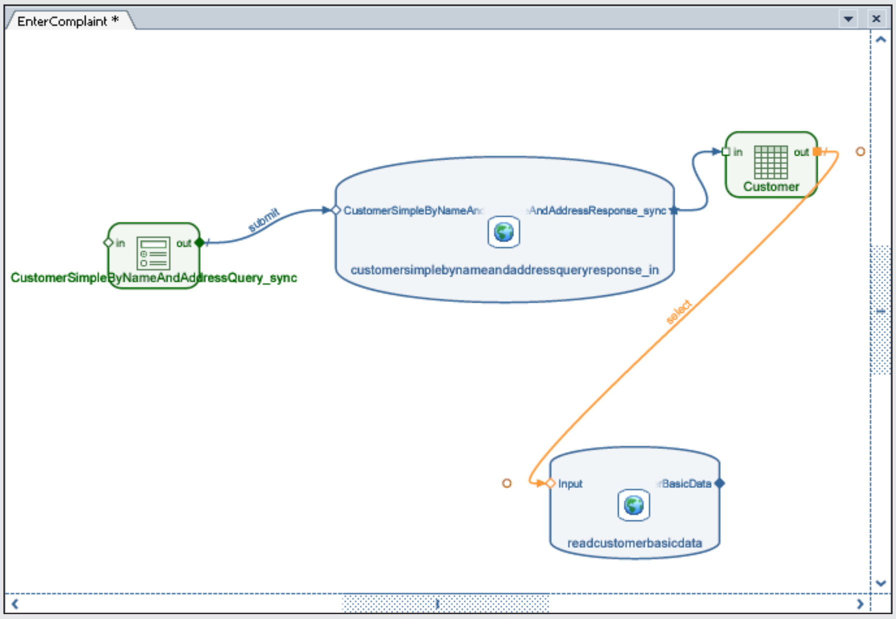


Figure 9 Connecting the output table with the input for the next service

automatically for you. When you switch to the Layout board, you can see the button and its associated event.

4. Now comes the tricky part. To ensure that the selected customer is transmitted to the next service call, you connect the table's output plug to the service's input plug, as shown in **Figure 9**.

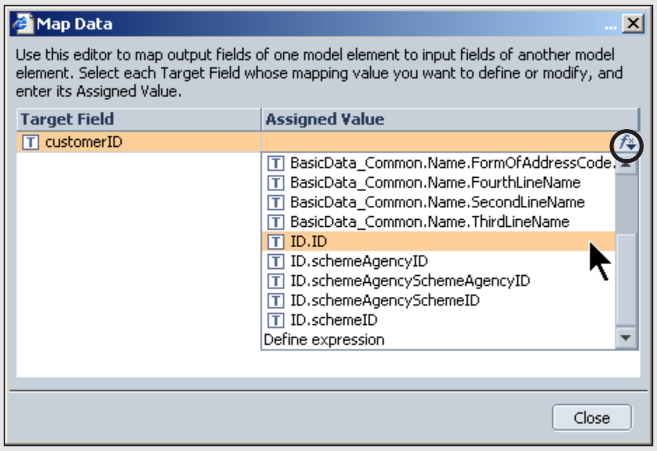
   Note that the label that appears closest to the just-created connection is named "select" and indicates that this connection will fire every time you select a customer from the table. Visual Composer saves you a lot of effort; imagine trying to program all this manually.

5. So far, you have a connection between the table and the service, but the service expects the ID as input so it can retrieve the appropriate customer's detailed data. You have to tell Visual Composer which fields to transfer as input parameters to the service call. To achieve this, right-click on the connection between the Customer output table and the input plug of the readCustomerBasicData service, and select Map Data from the context menu. The dialog that pops up gives you a list of the input parameters required to call the service. In this case, the service contains only one input field, customerID (see **Figure 10** on the next page). Instead of working with hundreds of input parameters (as you would with a complex enterprise service), you have a simple interface with an easy-to-understand input parameter.

6. Now, you need to tell Visual Composer which of the fields in the table to input to the service readCustomerBasicData. Click on the fx button

Tip!

You can always change your field selection for either the input or output forms or tables by right-clicking on the appropriate view (Form View, Table View) and choosing Define Data from the context menu.


Map Data

Use this editor to map output fields of one model element to input fields of another model element. Select each Target Field whose mapping value you want to define or modify, and enter its Assigned Value.

Target Field	Assigned Value
customerID	

BasicData_Common.Name.FormOfAddressCode
BasicData_Common.Name.FourthLineName
BasicData_Common.Name.SecondLineName
BasicData_Common.Name.ThirdLineName
ID.ID
ID.schemeAgencyID
ID.schemeAgencySchemeAgencyID
ID.schemeAgencySchemeID
ID.schemeID
Define expression

Figure 10 Mapping the input parameter of a service

next to the input field in the Assigned Value column (as indicated by the circle) to get a list of the available fields from which you can choose. In this case, ID.ID is the one you want. Click on it and the text "ID.ID" appears at the top next to the customerID field.

7. The final step is to create the form on which you display the customer's detailed data. Since you already know how to create forms for the output parameters of a service, this is easy. Drag and drop a line from the readCustomerBasicData service's output plug to somewhere in the workspace, and choose Form View from the context menu that pops up. Because your service was tailored for this UI, display all the fields that the service returns, and then confirm the Define Data dialog by clicking on Close.

So far, you have worked only on the Design board. Now, you need to refine the UI that Visual Composer created. For this, click on the Layout board icon in the workspace's toolbox. As a result, you see the UI with which the end user will work during runtime, the one that was automatically derived from your modeling efforts. It already contains the Submit button mentioned earlier. Now you can rearrange the UI controls via drag and drop, as shown in **Figure 11**.

The Layout board is where you adjust all the visual parts of your UI. You can make text fields read-only, assign a tool tip to a field, or add an action to a button. To configure UI controls, click on the control you want and click on the Configure button on the task panel toolbar. By doing this, you can switch to configuration mode.

As an exercise, click on the Submit button of your UI to see the details of its configuration. The Action parameter is one configuration parameter that button can have. This parameter reflects the event triggered when you click on the button. If you modify the configuration parameters a little for each UI control, you can see what consequences your changes will have. You can find more information on the SAP Help Portal, which documents each of the UI controls along with its configuration parameters. You can also follow the menu path on page 27 for Visual Composer, then continue from Modeling Composite Views with Visual Composer → Visual Composer Reference Guide → Task Panels → Configure <*element*> Task Panel. Here you can find all the configuration parameters for each UI component.

On the Design board the connection between the input form and the first service call is labeled submit. This is the same value as the button's Action

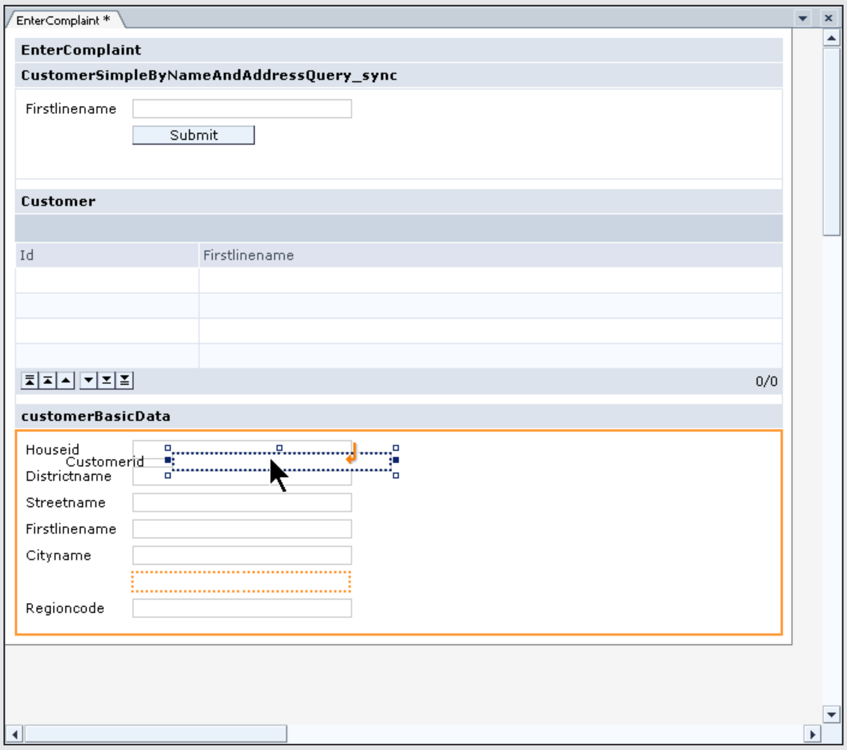


Figure 11 Moving the fields in your UI to create the sequence you want

parameter. Visual Composer has automatically assigned the value to the button.

So far, you've modeled views based on the input and output parameters of the associated services. This approach makes it easy and convenient to build a UI. Now, let's look at adding data that has no service.

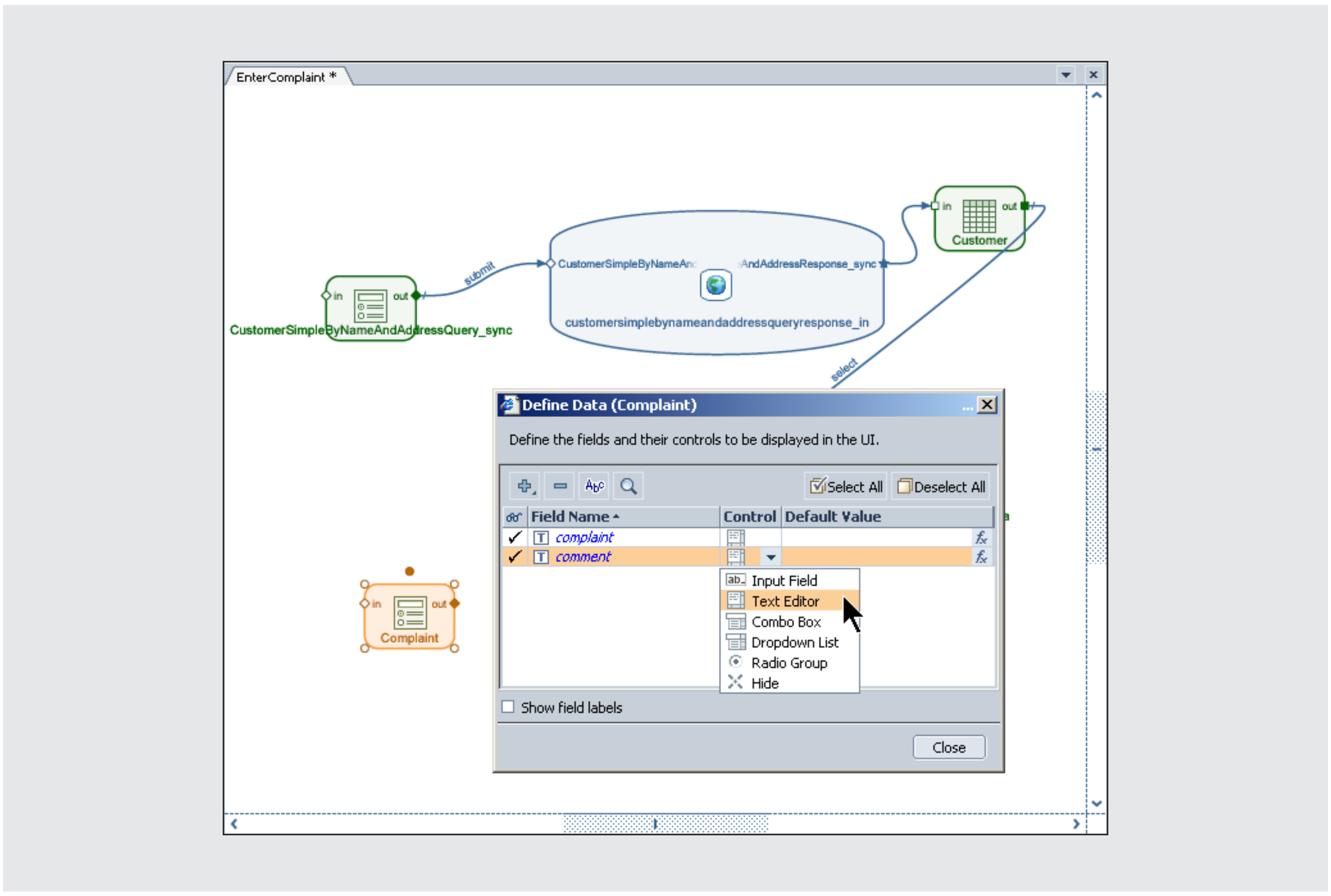
Adding data that doesn't have a service to the UI

What can you do if you need to add data that doesn't have an appropriate service to your UI? For example, on the call center agent's screen, you need one field to enter data about the complaint itself and another field for comments that the agent would like to add to the complaint. If automation can't help, you will have to add the comment field manually.

To do this, return to the Design board, click on the Compose button in the task panel toolbar on the right (**Figure 2**), and drag a Form View into the workspace. If you wish, you can rename the Form View via its context menu and the menu item Rename. I have named it Complaint for the example in **Figure 12** on the next page. Right-click on the form and choose Define Data from the context menu; the dialog for data definition pops up. Previously, you used it to select those fields that you wanted to see in your UI from the interface to which the UI was connected.

Because this new form is not connected, it is initially empty: no fields, no buttons, and no actions. You have to add them step by step:

1. Click on the plus icon, choose the appropriate data type from its context menu (e.g., STRING) and adjust the name of the field according to its use (e.g., Complaint). To the right of the field's name you see a drop-down list of the possible


EnterComplaint *
CustomerSimpleByNameAndAddressQuery_sync
submit
CustomerSimpleByNameAn AndAddressResponse_sync
customersimplebynameandaddressqueryresponse_in
Customer
select
Complaint
in
out

Define Data (Complaint)

Define the fields and their controls to be displayed in the UI.

Select All | Deselect All

	Field Name	Control	Default Value
✓	complaint		
✓	comment		

Input Field
Text Editor
Combo Box
Dropdown List
Radio Group
Hide
Show field labels
Close

Figure 12 Adding fields to a form and assigning UI controls

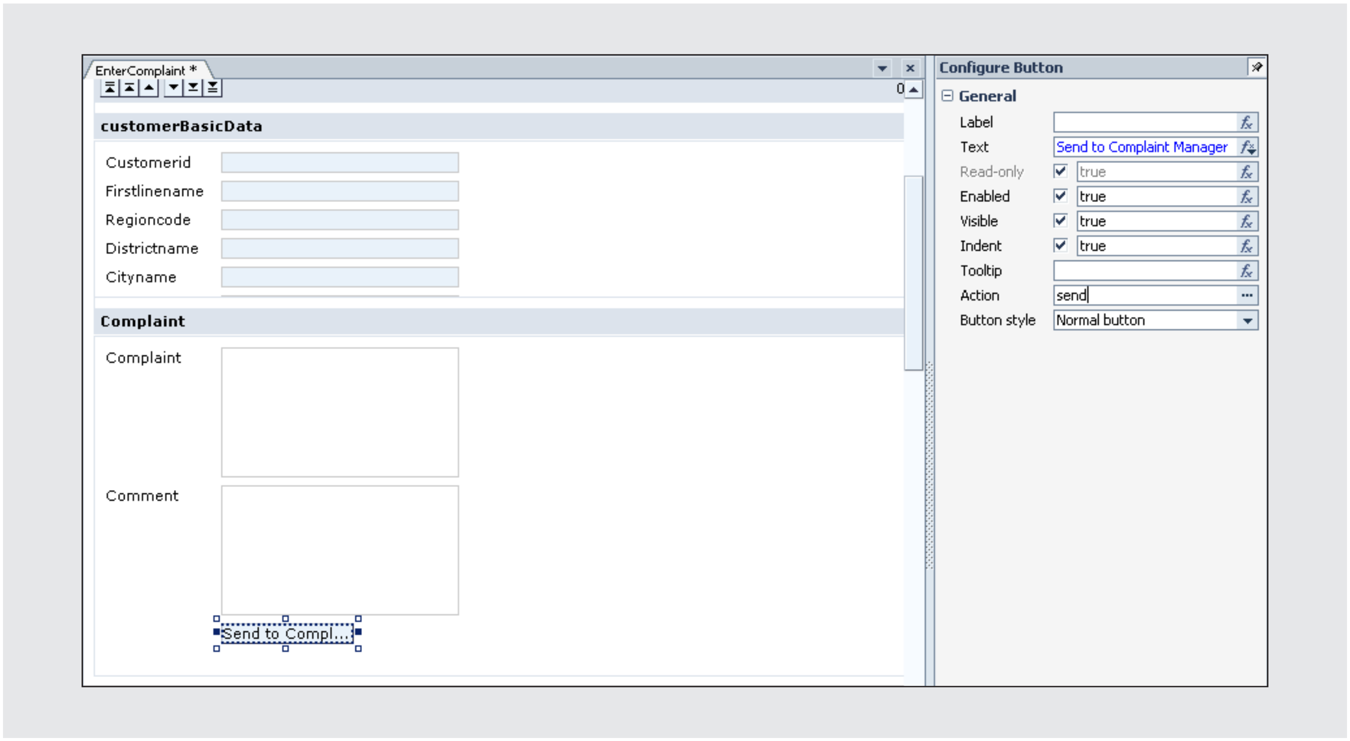
UI controls you can assign to this field. By default, it is set to Input Field for STRING fields. Because a complaint description can be quite lengthy, select Text Editor as a more appropriate control for long texts.

2. Repeat this step for the Comment field and the dialog looks like the one in **Figure 12**.

3. To add a button to this UI, switch to the Layout board. Now, simply drag and drop the button underneath the two new text editor fields, Complaint and Comment, as shown in **Figure 13**.

4. Then, you switch back to configuration mode, add an appropriate text to the button (e.g., Send to Complaint Manager), and add a new event. As events are just names, you can enter any name you want. In this case, consider naming it "send" to indicate that you want to hand over the data to the next process role when this button is pushed.

5. Now that the action is in place, when the button is pushed and the event fired, you finish this screen and hand over the data to the next role in the process — in this case, the complaint manager. From a technical point of view, SAP GP is responsible for transferring data from step to step. Therefore, you need to signal SAP GP that the UI is complete and the data is available. For this mechanism to work, the UI must fulfill the contract agreed upon by the original developers of Visual Composer and the SAP GP framework on how to communicate between the two; for example, the data that needs to be passed between them. It is the UI's task to notify SAP GP that the data is available and SAP GP can proceed to the



Label:
Text: Send to Complaint Manager
Read-only: ✓ true
Enabled: ✓ true
Visible: ✓ true
Indent: ✓ true
Tooltip:
Action: send
Button style: Normal button

Figure 13 Configuring a button for the UI

next step. Visual Composer implements this contract by providing an end point, a signal to SAP GP that the user has finished the UI. You can reach an end point from a view of the UI via an event (action) and, pass on the appropriate data at the same time. Therefore, let's look into the details of modeling an end point.

6. The button that triggers the event for leaving the screen (UI) is on your newly created Complaint form. On the Design board, drag a line from the output plug of this Complaint form and choose End Point from the context menu that pops up when you release the mouse button, as shown in **Figure 14** on the next page.

   Behind the scenes, Visual Composer automatically adds all of the fields from the form to the end point. In this case, that means it includes the two fields, Complaint and Comment. But you also need the ID of the customer in the end point. Since Visual Composer can't add the Comment field automatically because it's not part of the Complaint UI, you have to add it manually. You do this exactly the same way you add fields to a form: Right-click on the end point and choose Define Data. Next, you add the field customerID of type STRING to the end point.

   But how do you fill these fields once you leave the UI? The comment and complaint fields are filled from the form that's connected to the end point. It's easy to verify that these fields are already mapped by right-clicking on the connection between the form and the end point and choosing Map Data from the context menu, as shown in **Figure 15** on the next page.

   The customerID field, however, has to be established in the end point's Define Data dialog. You can choose the correct field from the list of all available fields by using the fx button to the right of the appropriate Default Value field. **Figure 16** on page 41 shows the correct selection for the example.

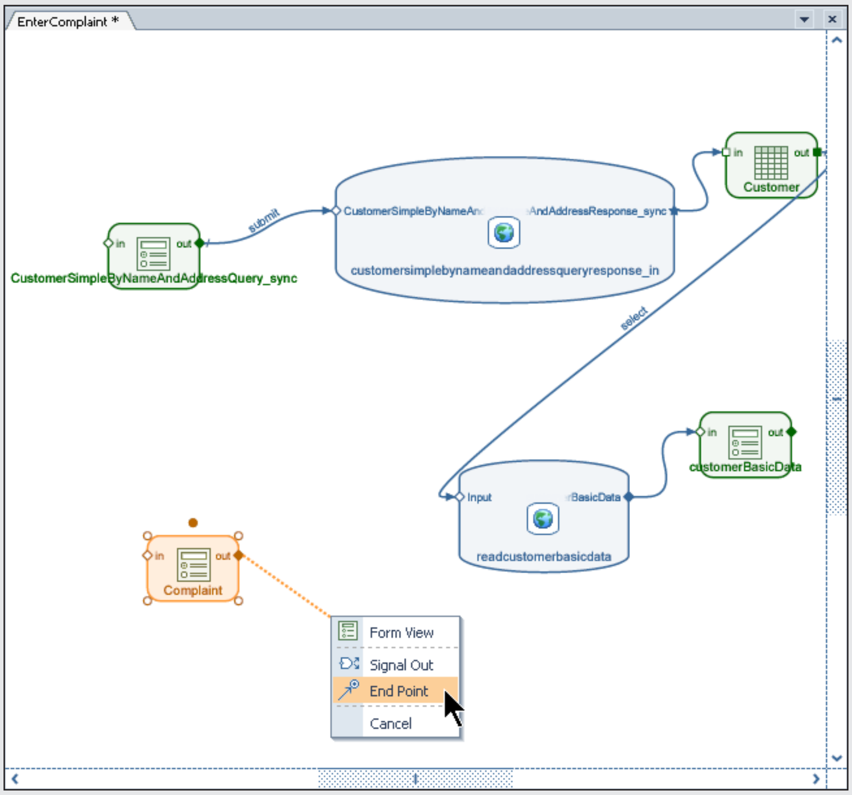
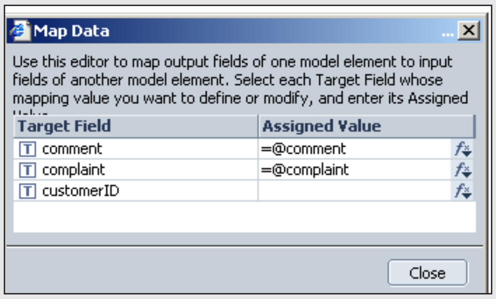


Figure 14 Creating an end point

7. Finally, you have to assign an event to the connection between the form and the end point to tell Visual Composer what should activate this connection; in this case, the connection executes when you click on the button. To assign the event, click on the connection, switch to configuration mode by clicking on the Configure button on the task panel toolbar (**Figure 2**), and choose the send action from the drop-down list of the Select mode field, as shown in **Figure 17**. This is the action you assigned to the button in Step 4 (**Figure 13**). Your UI model is now ready to be tested.

Testing the UI

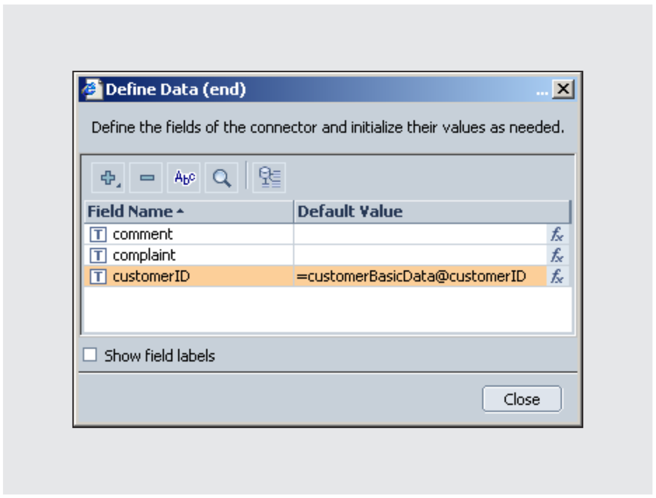
Another benefit of working with Visual Composer is its short model-test cycle. Especially with UI modeling tools, it's essential to try out the latest changes immediately. Visual Composer supports this


Map Data

Use this editor to map output fields of one model element to input fields of another model element. Select each Target Field whose mapping value you want to define or modify, and enter its Assigned Value.

Target Field	Assigned Value
comment	=@comment
complaint	=@complaint
customerID	

Figure 15 Mapping between form and end point



Define Data (end)

Define the fields of the connector and initialize their values as needed.

Field Name	Default Value
comment	
complaint	
customerID	=customerBasicData@customerID

Show field labels

Figure 16 Creating the connector's customerID field

need with one-click deployment. Select Deploy from the task panel toolbar (**Figure 2**) and click on the Deploy button in the context-sensitive task panel. Visual Composer then translates and deploys the model. It provides a link that guides you to the UI runtime. Follow the link and the final result should look like the screen shown in **Figure 18** on the next page but without the data.

Enter some test data (e.g., "Henderson"), and test to see whether the service calls to the enterprise service and the dummy service work. This will show you the UI screenshot with working services. Note that the customerBasicData part of the screen doesn't change because your dummy service always returns the same static data that you originally selected from

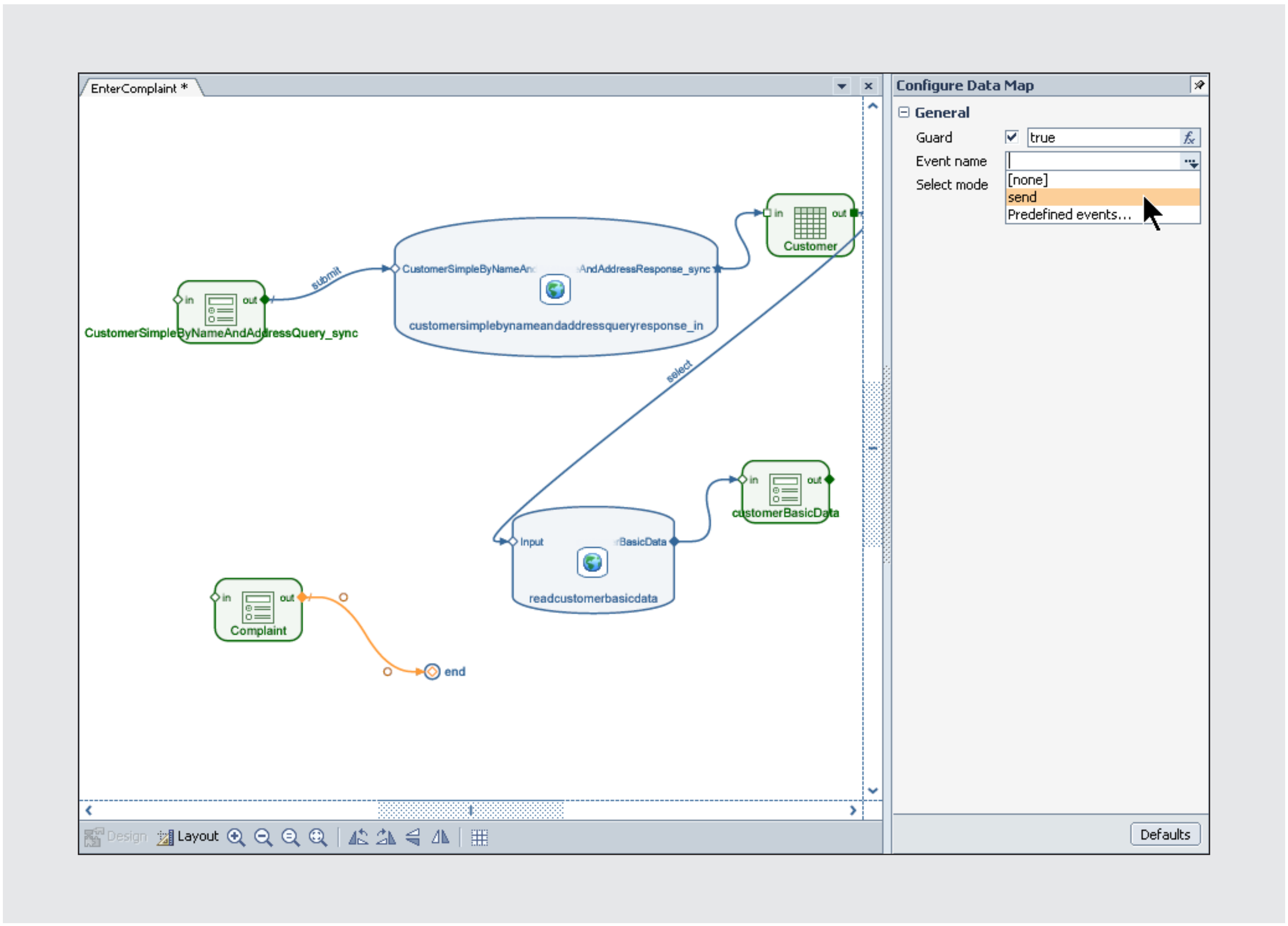
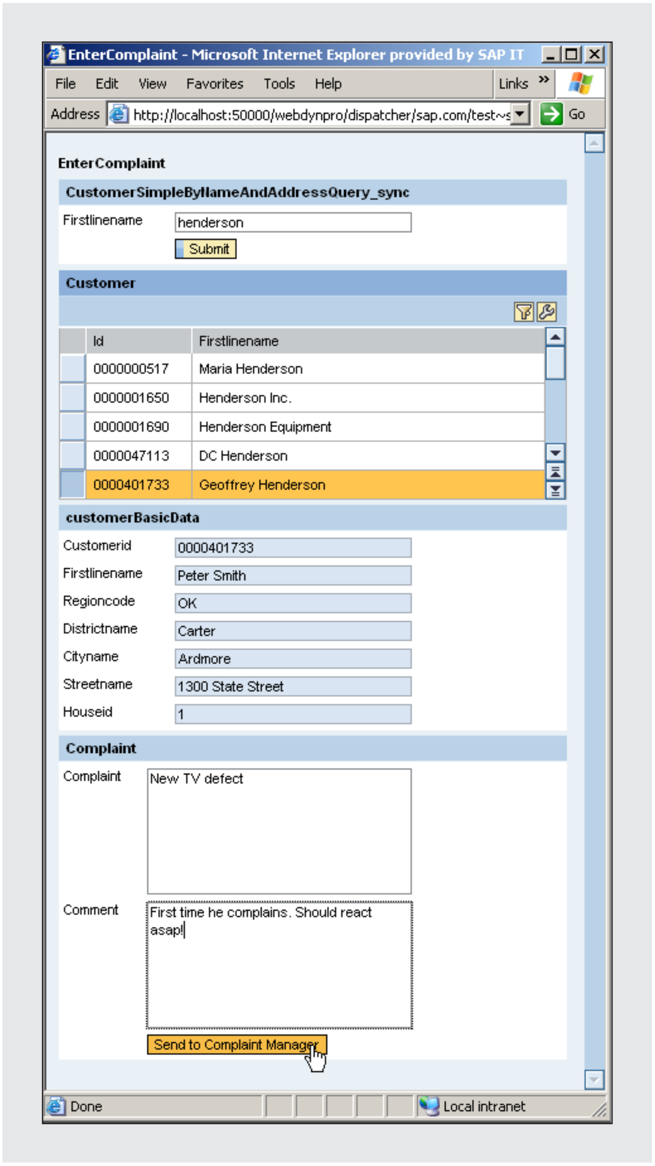


Figure 17 Assigning an event to a connection

the Customer table independent of the customer. To close this UI, click on the Send to Complaint Manager button.

With that you have finished your first UI. As you now have a good understanding of how UI modeling with Visual Composer works, you can concentrate on the differences, the tips, and the tricks for the remaining UIs.

Next, let's focus on the complaint manager's UI.


EnterComplaint - Microsoft Internet Explorer provided by SAP IT
File Edit View Favorites Tools Help
Links
Address
Go
Submit
Send to Complaint Manager
Done
Local intranet

http://localhost:50000/webdynpro/dispatcher/sap.com/test~s

EnterComplaint

CustomerSimpleByNameAndAddressQuery_sync

Firstlinename: henderson

Customer

Id	Firstlinename
0000000517	Maria Henderson
0000001650	Henderson Inc.
0000001690	Henderson Equipment
0000047113	DC Henderson
0000401733	Geoffrey Henderson

customerBasicData

Customerid	0000401733
Firstlinename	Peter Smith
Regioncode	OK
Districtname	Carter
Cityname	Ardmore
Streetname	1300 State Street
Houseid	1

Complaint

Complaint: New TV defect

Comment: First time he complains. Should react asap!

Figure 18 Running services on the call center agent's UI

Creating the complaint manager's UI

To figure out what's needed on this screen, let the business story be your guide. The complaint manager wants to see the details of the complaint immediately, as well as the details about the customer who complained. So, where does the data come from? In the exercise — the initial call center agent's UI — you modeled an end point as a contract between the UI and SAP GP. What does SAP GP do with the data? It saves the data in the SAP GP context, which is a container for process data that is entirely maintained by SAP GP. From there, SAP GP can pass on the data to other steps in the process.

For this technique to work, the receiving UI must fulfill a contract similar to that for the output parameter. To receive data from SAP GP, the Visual Composer UI must provide an appropriate start point. Similar to the end point, you have to model the data fields that the UI expects to receive from SAP GP. This is how it looks:

1. Create a new Visual Composer model named ApproveComplaint.

2. Drag the start point onto the workspace.

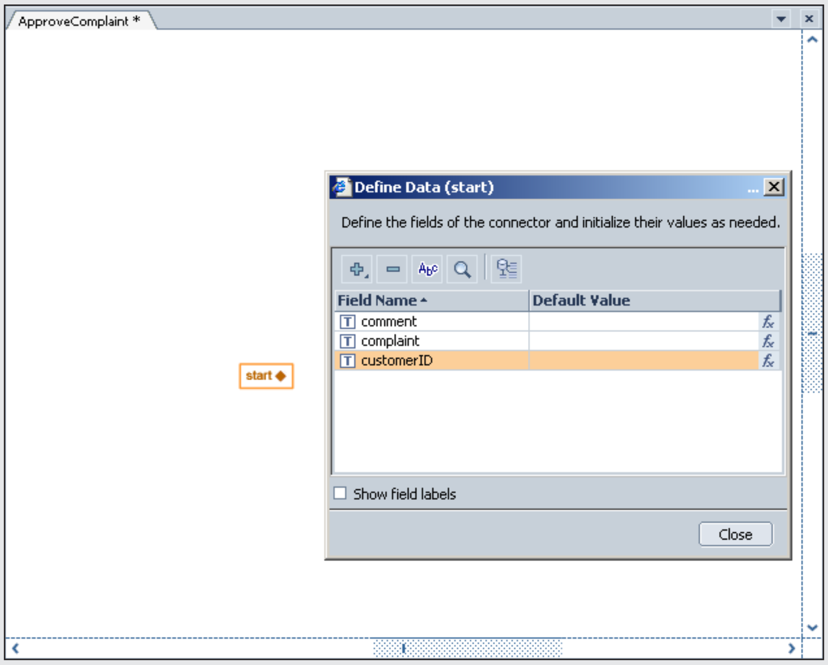
3. Right-click on the start point, and choose Define Data from the context menu.

4. Add the fields you expect to receive from SAP GP to the start point: comment, complaint, and customerID, which are all of type STRING (as shown in **Figure 19**).

That's all you have to do to fulfill the contract between SAP GP and Visual Composer for any incoming data.

Advanced features

The ApproveComplaint screen contains advanced features that deal with issues that you typically face when developing composite applications. It's worthwhile to explore them further. You can connect a start point to either a UI or a service call depending



Define Data (start)

Define the fields of the connector and initialize their values as needed.

Field Name	Default Value
comment	
complaint	
customerID	

☐ Show field labels

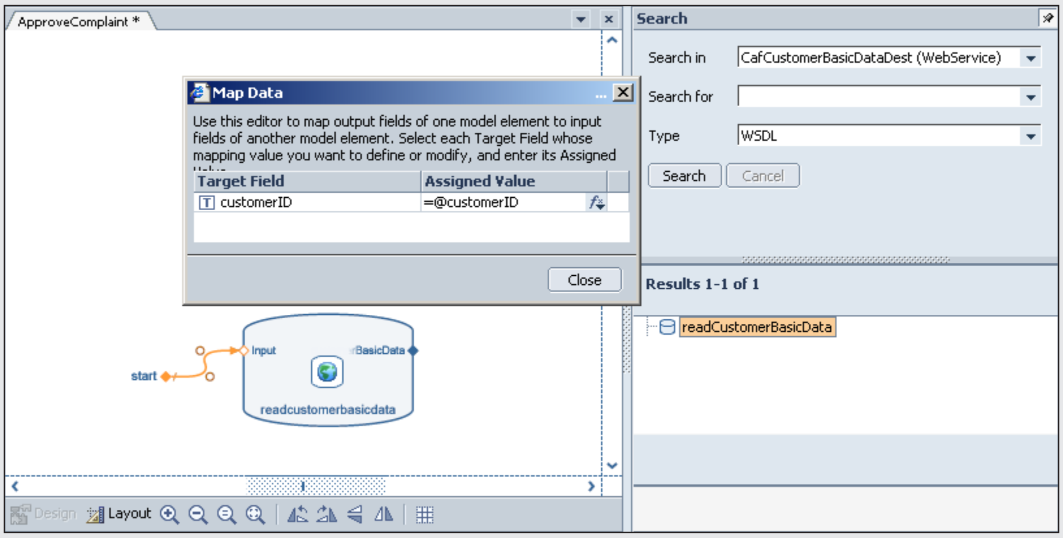
Figure 19 Defining the start point for the complaint manager's UI

on your particular needs. If you just want to display the form, connect the start point to it. However, sometimes you need to call a service *before* you display the UI. A typical use case would be to prefill the drop-down fields or any other UI fields. In this case you have to connect the start point with the service call for retrieving data, so that it executes first. The returned data is then used to fill the fields of the form and display the information. In this case, you get the customerID from the SAP GP context.

To display the customer's details to the complaint manager, you call the appropriate service to prefill the form. In this example, you should display the details of the customer referenced by the CustomerID to the complaint manager. To accomplish this, you need to make another call to the readCustomerBasicData service. Therefore, search for the service again, drag it from the result list onto the workspace, and connect the start point with the service's input plug. Since the service requires an input parameter, you have to fill it from the start point. Right-click on the newly created connection and choose Map Data from the context

Tip!

During the process-modeling phase of your composite development project, you have to define which data moves from one step to the next. Because this happens on the field level, based on the field names, it is of the highest importance that you use exactly the same names for those fields that should contain the same domain-specific content. This dramatically lowers your mapping efforts from step to step at a later stage. I will explain this point in detail in the third article of this series. This practice is important for service interface definitions as well.



Map Data

Use this editor to map output fields of one model element to input fields of another model element. Select each Target Field whose mapping value you want to define or modify, and enter its Assigned

Target Field	Assigned Value
customerID	=@customerID

Close

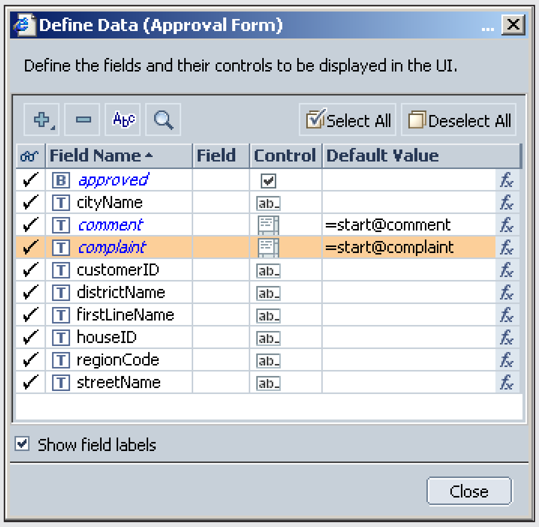
Figure 20 Filling the service's input parameter from the start point

menu. Assign the customerID of the start point to the identically named input parameter of the service, as shown in **Figure 20**.

Next, drag a connection from the service's output plug, release the mouse button, and choose Form View from the context menu. Visual Composer takes all of the service call's returned values to your UI automatically. Fortunately, the service is tailored to the UI's needs (i.e., it contains exactly those fields that you need). However, you also want to display the data coming from the SAP GP context (the complaint and the comment fields) plus one additional Boolean field indicating that the complaint manager has either accepted or rejected the complaint. To achieve this, right-click on the form, select Define Data from the context menu, and add three new fields to the form: the complaint and comment fields of type STRING and the approved field of type Boolean, as shown in **Figure 21**.

You must prefill the fields with default values. In this case, the data is coming from the start point. Thus, you ensure that the data entered by the call center agent displays accurately on the complaint manager's display screen.

Next, you define the end point with the fields that you want to return to SAP GP: the approved and comment fields. You have two ways to create the end point:



Define Data (Approval Form)

Define the fields and their controls to be displayed in the UI.

	Field Name	Field	Control	Default Value
✓	*approved*		☑	
✓	cityName		ab	
✓	*comment*			=start@comment
✓	*complaint*			=start@complaint
✓	customerID		ab	
✓	districtName		ab	
✓	firstLineName		ab	
✓	houseID		ab	
✓	regionCode		ab	
✓	streetName		ab	

☑ Show field labels

Figure 21 Selecting fields on the approval form

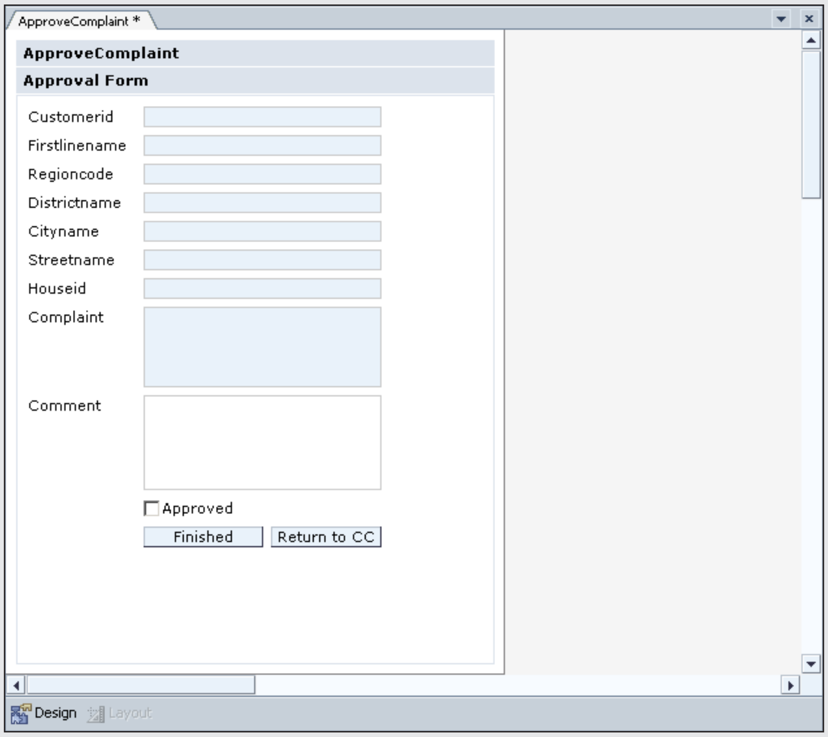


Figure 22 Adding buttons for the complaint manager's decision

1. Drag the appropriate connector from the task panel to the workspace and add the fields manually. Then, connect the form's output plug to the end point and map the fields from the form to the end point. If the names of the form fields and the names of the end point fields are identical (including uppercase and lowercase letters), then Visual Composer can map them automatically.

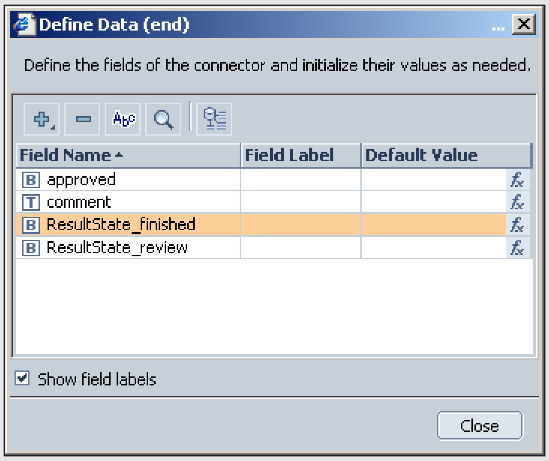
2. Derive the fields for the end point from the form. Drag a line from the form's output plug and select End Point from the context menu. As Visual Composer automatically takes all form fields to the end point, you have to remove some superfluous fields here so you can return only the approved and comment fields.

Next, switch to the Layout board and add two buttons representing the complaint manager's decision (as shown in **Figure 22**):

- Finished button, associated with the new action Finish

- Return to CC button, associated with the new action Review

Now, you are faced with a serious problem: This screen can fire one of two events, either Finish or Review. Depending on the fired action, the process continues either with the call center agent, who has to review the complaint (e.g., important information is missing), or with the persistency of the complaint. But how can you tell SAP GP which button the complaint manager pushed? To do this, you have to introduce the notion of result states, a useful feature you should know if you model your processes with an SAP GP framework. UIs such as those from Visual Composer can notify SAP GP which button the complaint manager pressed by setting specific Boolean values as output parameters. The developers of SAP GP and Visual Composer agreed upon the



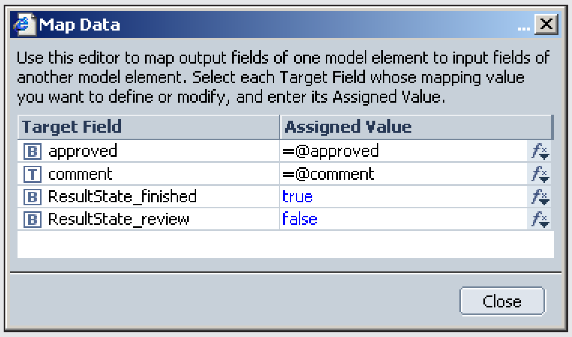
Define Data (end)

Define the fields of the connector and initialize their values as needed.

Field Name	Field Label	Default Value
approved		
comment		
ResultState_finished		
ResultState_review		

☑ Show field labels

Figure 23 Defining the result states in the end point



Map Data

Use this editor to map output fields of one model element to input fields of another model element. Select each Target Field whose mapping value you want to define or modify, and enter its Assigned Value.

Target Field	Assigned Value
approved	=@approved
comment	=@comment
ResultState_finished	true
ResultState_review	false

Figure 24 Setting result state variables to reflect the action taken

following rule to make result states work for Visual Composer:

Provide as many Boolean output variables as you have result states in your UI. The names of those variables have to follow a certain naming convention. They have to begin with ResultState_ (recognize the underscore at the end) followed by the result state's name, which can be any string without spaces. Examples of valid result state names are ResultState_finished and ResultState_review. To indicate which of these result states actually fired, you have to set exactly one of those result states to true.

This rule applies to your UI as well as Visual Composer and defines the two result states just mentioned for the end point — ResultState_finished and ResultState_review. The data definition for your end point should look like the screen shown in **Figure 23**.

Now that the result states are in place, you need to know how to model the firing of the different events in Visual Composer and make sure that only one result state is set to true. For each result state, you need a connection from the form's output plug to the end point, you must assign the events to them, and you have to set their result states correctly. In detail, you must follow these steps:

1. Drag a second connection from the form to the end point.
2. For the first connection, assign the event "finish."
3. For the second connection, assign the event "review."
4. Open the mapping dialog for the "finish" connection and set the result state variables to fixed values according to the fired action. In other words, ResultState_finished should be set to true, and ResultState_review should be set to false, as shown in **Figure 24**.
5. Open the mapping dialog for the second connection (the one labeled as "review") and adopt these values as well. This time the values should be the exact opposite of what they were in the previous step. ResultState_finished should be set to false, and ResultState_review should be set to true.

That was the final step for creating the approval screen. The final result looks like **Figure 22**, and the data and screen flow are shown in **Figure 25**.

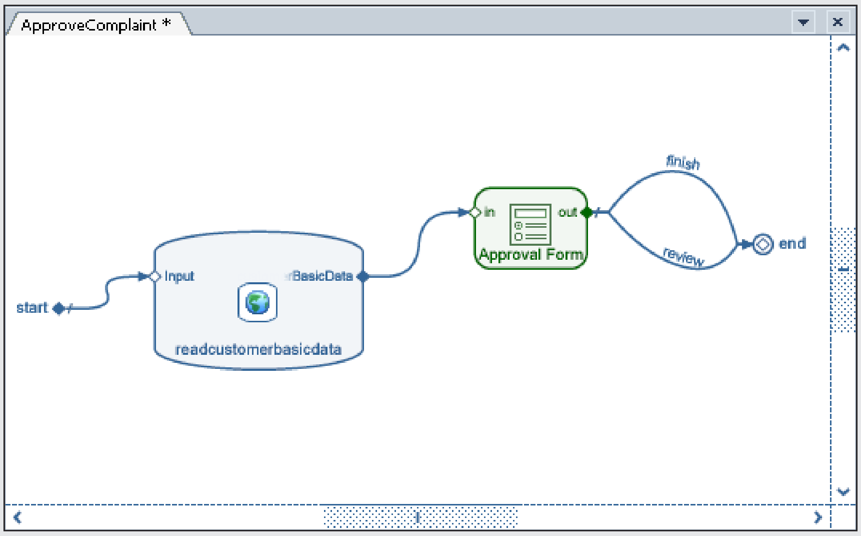


Figure 25 Showing data and screen flow for the approval screen

The review screen for the call center agent is almost identical to the approval screen with the exception that it doesn't have result states. The call center agent just reviews the complaint manager's comment and reacts accordingly, adding a note in the comment field, and submitting it.

Due to the similarities between the complaint manager's approval screen and the call center agent's review screen, you should use the approval UI as a blueprint and create a copy of it. Just open the approval screen and choose Model → Save As from the main menu. Enter ReviewComplaint as the new model's name and save it. Now, you can adapt the copy as follows:

1. The review screen looks like **Figure 22** except for the screen title which has changed to ReviewComplaint due to the copy operation. Remove one button and change the text of the remaining button to "Submit." Assign an identically named action to the button so that it executes the submit event.
2. The data and screen flow should look like **Figure 25** with only a single connection between the approval form and the end point. Therefore, remove the "finished" line and assign the "submit" action to the remaining connection.
3. Verify that the start point contains the fields comment, complaint, and customerID.
4. Verify that the start point's customerID field has filled the input parameter of the readCustomer BasicData service.
5. The end point contains only the comment field. Remove all other fields from the end point.

Now, the UIs of your application are ready for the process layer.

Conclusion

The goal of Step 3 in developing a composite application is to create the UIs that the application needs to accomplish its goal. In the article example, you learned how to create three UIs with Visual Composer. The first UI enables a call center agent to document a customer's complaint. This UI includes incorporating services to access a customer's ID number with only its name and to retrieve the rest of the customer's detailed information. The second UI notifies the complaint manager of the problem so he or she could ensure a timely solution. The third UI provides the approval screen. Then, the article discussed some advanced features available in Visual Composer and SAP GP.

In the next issue, I'll talk about modeling the process and discuss a service for persisting your data in a database using SAP CAF.