Advanced techniques for enhancing your SAP Enterprise Portal iViews with Visual Composer — a purely model-driven, code-free development approach

by Karl Kessler

Karl Kessler
Product Manager, SAP NetWeaver, SAP AG

Karl Kessler studied computer science at the Technical University in Munich. He joined SAP in 1992. In 1994, Karl became a member of the ABAP Workbench product management group, and in 1996, he became product manager for business programming languages and frameworks. In 2003, Karl assumed responsibility for product management of the SAP NetWeaver platform, including SAP Web Application Server, SAP NetWeaver Developer Studio, SAP Enterprise Portal, Web services, J2EE, and ABAP. You can reach him at karl.kessler@sap.com.

My first article[1] in this two-part article series introduced the key components of Visual Composer and demonstrated how to build a simple flight lookup iView based on the SAP R/3 IDES training system. This article goes beyond the basics to explore some advanced techniques that you can use to make your iViews more usable, as well as look and feel more professional. In this discussion, we'll extend the example FlightListiView created in the previous article by adding drill-down functionality and charts, integrating external Web content using HTML views, and leveraging portal eventing. This article also explains how to provide default values for BAPI input fields and how to use operators to filter, sort, and group BAPI result data.

Before learning about these new features, let's review some important points from the first article:[2]

- Visual Composer is a WYSIWYG tool that technically savvy non-programmers can use to develop and maintain their own iViews on SAP NetWeaver and SAP Enterprise Portal (SAP EP) 6.0 SP2 platforms. Strategically, Visual Composer "fits" in between portal templates (used to generate iViews for existing, Web-based content) and more sophisticated programming tools, such as Web Dynpro for Java and Web Dynpro for ABAP (used to develop rich Web-based user interfaces). Visual Composer iViews can access both SAP and non-SAP systems, including SAP R/3, CRM, and BW systems, non-SAP systems accessible via RFC, and SQL databases through stored procedures accessible via JDBC (support for Web services is planned for an upcoming release). Rather than connect directly, Visual Composer uses the portal to connect to these back-end systems.

[1] See "Get started creating SAP Enterprise Portal iViews with Visual Composer — a purely model-driven, code-free development approach" on page 3 of this issue of *SAP Professional Journal*.

[2] You can refer to Figures 1-3 in the previous article (see pages 4-7) for visual reinforcement of the review provided here, or you can skip this review if you prefer.

- Visual Composer is divided into two parts: a design-time component and a runtime component.
  - The design-time component is a server-based tool that must be installed on a Microsoft Internet Information Server (IIS) and have access to a SQL Server database for storing design metadata. You can access Visual Composer from any Web browser on which the Microsoft XML Core Services (MSXML) plug-in and the Adobe Scalable Vector Graphics (SVG) plug-in are installed.[3] The URL to access Visual Composer is http://<myIIS>/vcserver (where *<myIIS>* is the IP address or host name of the IIS hosting the design-time component and *vcserver* is the name of the default directory for the application). Accessing Visual Composer using this approach greatly simplifies setup and maintenance and makes the tool available virtually anywhere, anytime. You don't need to install client-side components to develop content.

Note!

There are plans to port the design-time component to run on SAP Web Application Server, just like SAP EP does.[4]

  - The runtime component is built into SAP NetWeaver and SAP EP 6.0 SP2, so Visual Composer can be deployed and run on these servers without any modifications. This component provides the libraries needed for the portal to render the Visual Composer iViews at runtime and supports the design-time component in retrieving metadata (i.e., BAPI parameter details) from back-end systems.

- The heart of the Visual Composer programming model is the object hierarchy. The *model* is the root object that manages all of the objects associated with a particular development effort. A model in Visual Composer is equivalent to a project in many other development tools. Next is a *module*, which serves as a container (similar to a folder in a folder hierarchy), primarily used to group development objects. Next is a *page*, which groups one or more iViews together. Each page provides layout and eventing functionality for the iViews it contains. Each *iView* consists of one or more data sources and controls. The data source objects (e.g., BAPIs, function modules, InfoCubes) are "imported" by connecting to the selected back-end system. Visual Composer imports the data source metadata (i.e., input and output parameters, etc.) and renders a visual representation of the data source that you can drag and drop, connect to controls, etc.
- The Visual Composer modeling environment is comprised of a central workspace and sets of tabs, tools, task panels, and toolbars that you use to design and develop your portal content.

With these key points in mind, let's add some more sophisticated features to the example FlightListiView we created in the previous article. We'll start with adding drill-down functionality and charts.

Adding drill-down functionality to an iView

Drilling down to details is a pretty common scheme in Web applications and portal content. In the previous installment of this article series, we created a simple iView that displays flights based on search criteria. Now we want to expand the capability of our iView to display the details of a selected flight.

Our approach, in a nutshell, will be to extend our example FlightListiView so that the system calls an additional BAPI (BAPI_SFLIGHT_GETDETAIL) to retrieve flight details for a particular flight (recall from the previous article that the example iView uses BAPI_SFLIGHT_GETLIST to retrieve the list of

[3] These plug-ins are easy and fast to install. The MSXML 4.0 plug-in (downloadable from http://msdn.microsoft.com/XMLDownloads/XML/default.aspx) parses your models and the Adobe SVG 3.0 plug-in (downloadable from www.adobe.com/svg/viewer/install/main.html) renders the workspace, tools, task panels, menus, and toolbars.

[4] The design-time component currently runs on Microsoft IIS and SQL Server for historical reasons (i.e., it was developed to run on Unix).

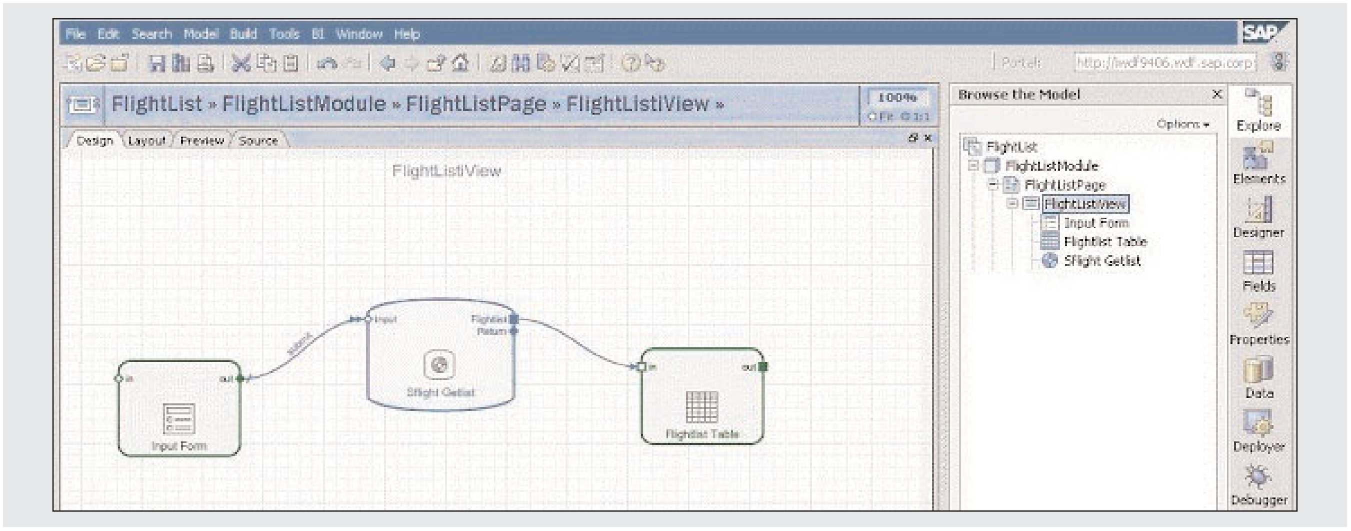


Figure 1 The example FlightList model

flights).[5] Let's begin by adding the necessary control that will allow us to drill down to the details of a selected flight.

To add drill-down functionality to our FlightListiView, we will perform the following steps:

1. Display the FlightList model in the Design view.
2. Add the new BAPI to the FlightList model.
3. Link the new BAPI to the Flightlist table.
4. Map screen fields to BAPI parameters to establish the data flow.
5. Create an output form view to display the BAPI results.
6. Refine the new user interface control.
7. Test and deploy the revised FlightListiView.

Let's walk through each step.

Step 1: Display the FlightList model in the Design view

Launch Visual Composer by navigating to http://<myIIS>/vcserver. After the application starts, open the model we created in the previous article by clicking on the FlightList link under the "Open an existing model" heading in the Get Started task panel.[6]

Click on the Design tab if it does not appear by default. Your FlightList model should resemble the one shown in **Figure 1**, just as we left it at the end of the previous article.

Next we need to log on to the portal (if you haven't already), so we can access the BAPI we need for the drill-down functionality we want to add.

Step 2: Add the new BAPI to the FlightList model

First, let's log on to the portal. To quickly determine whether you are logged on to your portal, check the traffic light icon at the upper-right corner of the Visual Composer window, to the right of the Portal field. If the traffic light is green, you are already logged on. If it is red, simply click on the traffic light and enter your portal user ID and password to log on.[7]

[5] Like BAPI_SFLIGHT_GETLIST, BAPI_SFLIGHT_GETDETAIL is available from the SAP R/3 IDES training system.

[6] If the FlightList model is not listed in the Get Started task panel, or if the Get Started task panel doesn't appear, use the Open command on the File menu or the Open button on the toolbar, navigate to the folder in which the model is stored, and open it from there.

[7] Remember, you must have content administration rights in order to perform the steps in this article series.

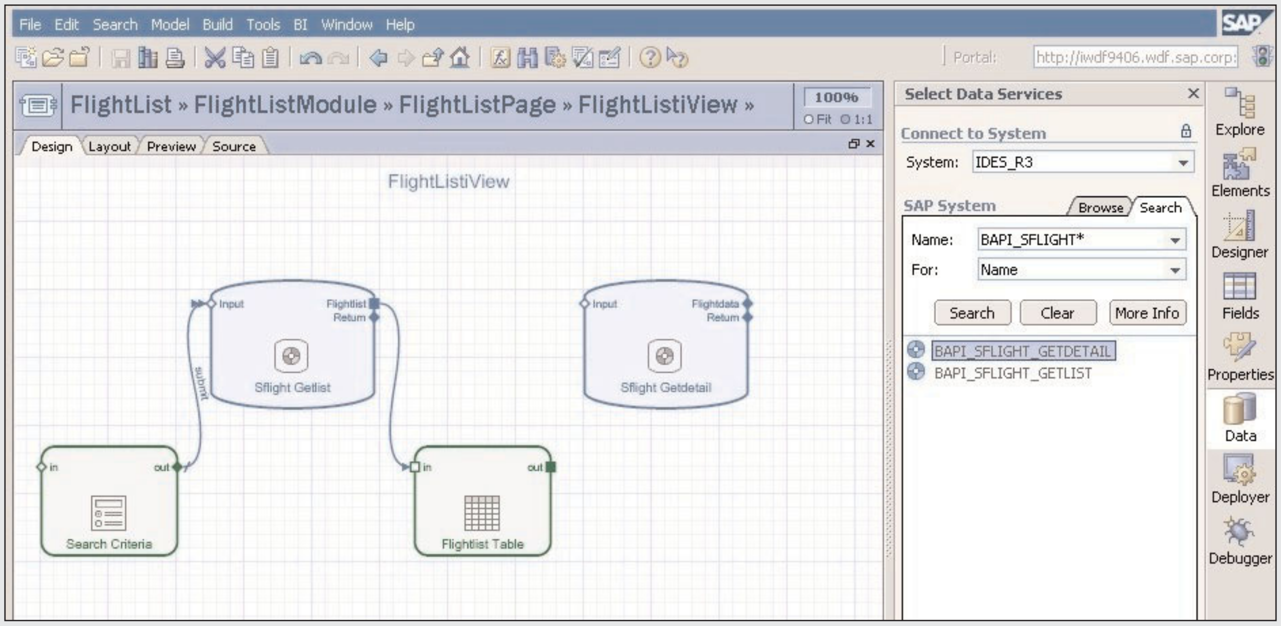


Figure 2 Adding BAPI_SFLIGHT_GETDETAIL to our FlightList model

Once you're logged on, you need to select the BAPI that retrieves the details for the drill-down functionality you want to add to the FlightList model — in this case, BAPI_SFLIGHT_GETDETAIL. Although you know the BAPI you need in this case, an easy way to search for a BAPI, as you did in the previous article, is to click on the Data tool on the task panel toolbar, select the back-end system, select the Search tab, enter the search string "BAPI_SFLIGHT_*", and click on Search. Now drag and drop BAPI_SFLIGHT_GETDETAIL from the task panel onto the workspace. The BAPI element appears, as shown in **Figure 2**. Remember that, if there are no existing BAPIs to meet your needs, you can ask your ABAP developer to create one that does.

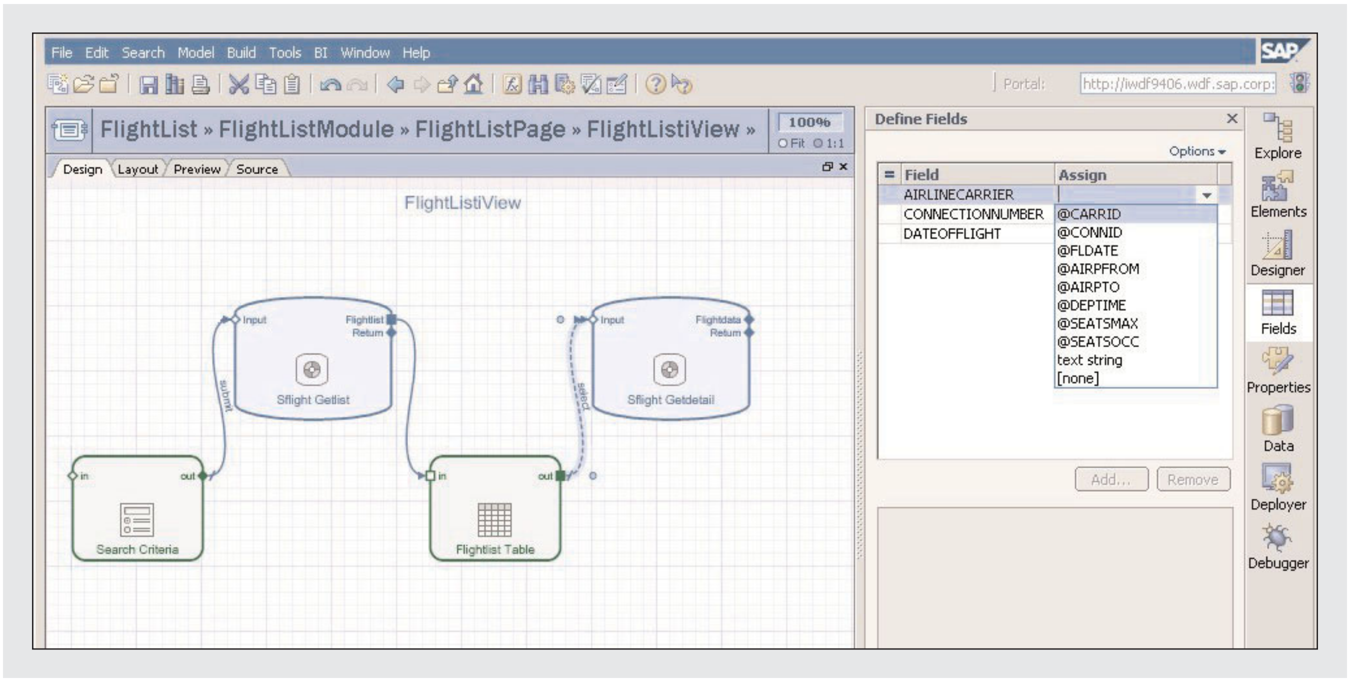
Tip!

When development is taking place on SAP systems, ask your ABAP developer to strictly adhere to BAPI development standards — they govern the naming, function, and parameters of function modules. This will help ensure consistency and accuracy in the development process, minimizing potential errors.

As we did in the previous article, next we need to connect the BAPI to a control. In this case, we need to link this BAPI to the output parameter (i.e., the Flightlist table) that displays the flight we are searching for.

Step 3: Link the new BAPI to the Flightlist table

The next step is to tell Visual Composer to call BAPI_SFLIGHT_GETDETAIL when a user selects a particular flight in the Flightlist table. This is extremely easy to do. All you have to do is connect the output parameter of the Flightlist table to the input parameter of the BAPI. Click on the out port of the Flightlist table, and then drag and drop it onto the input port of the BAPI_SFLIGHT_GETDETAIL object, linking the two parameters together.



Field	Assign
AIRLINECARRIER	@CARRID
CONNECTIONNUMBER	@CONNID
DATEOFFLIGHT	@FLDATE
	@AIRPFROM
	@AIRPTO
	@DEPTIME
	@SEATSMAX
	@SEATSOCC
	text string
	[none]

Figure 3 Linking the input parameter of the BAPI to the output parameter of the Flightlist table

As shown in **Figure 3**, the connection is automatically labeled with the word “select,” indicating that the BAPI (BAPI_SFLIGHT_GETDETAIL in this case) will be called when the user “selects” a flight in the Flightlist table.

Now that we’ve told Visual Composer to invoke the BAPI when a user selects an entry in the table view (i.e., we have established the flow of control), we need to tell it how to populate the BAPI’s input parameters. (See the sidebar on the following page for an important consideration to keep in mind about BAPI behavior.)

Step 4: Map screen fields to BAPI parameters to establish the data flow

To map a field to a BAPI input parameter, click on the connection between the BAPI_SFLIGHT_GETDETAIL object and the Flightlist table, click on the Fields tool on the task panel toolbar, and then select the BAPI input parameter you want to map (see Figure 3). You must map one parameter at a time to the field. The BAPI input parameters are listed in the Field column in the Define Fields task panel, and the related table control fields are displayed in a corresponding drop-down list next to a selected parameter. These fields all start with an “at” sign (@) because they are variables assigned at runtime, depending on which flight the user selects. In this case, all three of the input parameters are required by the BAPI and must be filled in. Select the AIRLINECARRIER input parameter, and then choose the corresponding table control field (@CARRID). Repeat this procedure for the other two required BAPI input parameters.

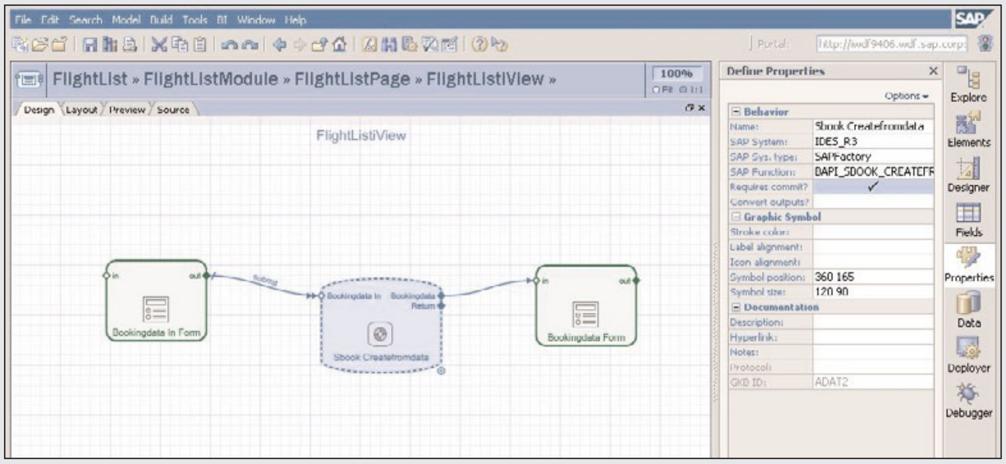
The next step is to add a form view to our layout to display the flight details returned by BAPI_SFLIGHT_GETDETAIL.

Step 5: Create an output form view to display the BAPI results

To create our output form view, click on the Flightdata Return output port, drag it to the right,

Handling BAPI updates

You should be aware that BAPIs might post new records or changes to the database that must be committed. As you can see in the Define Properties task panel in the screenshot below, BAPI_SBOOK_CREATEFROMDATA requires a COMMIT. Checking the "Requires commit?" property for a function module instructs Visual Composer to issue a manual COMMIT after the function returns. This also holds true for testing the BAPI, as discussed in the first installment of this article series.



Define Properties

Options

Behavior	
Name:	Sbook Createfromdata
SAP System:	IDES_R3
SAP Sys. type:	SAPFactory
SAP Function:	BAPI_SBOOK_CREATEFR
Requires commit?	✓
Convert outputs?	
Graphic Symbol	
Stroke color:	
Label alignment:	
Icon alignment:	
Symbol position:	360 165
Symbol size:	120 90
Documentation	
Description:	
Hyperlink:	
Notes:	
Protocol:	
GKD ID:	ADAT2

and then select Add Form View from the context menu (see **Figure 4**).

The Flightdata form appears when you release the mouse button. Next, as we did in the previous article for our other user interfaces, we will use the Layout tab to refine the usability of our new control.

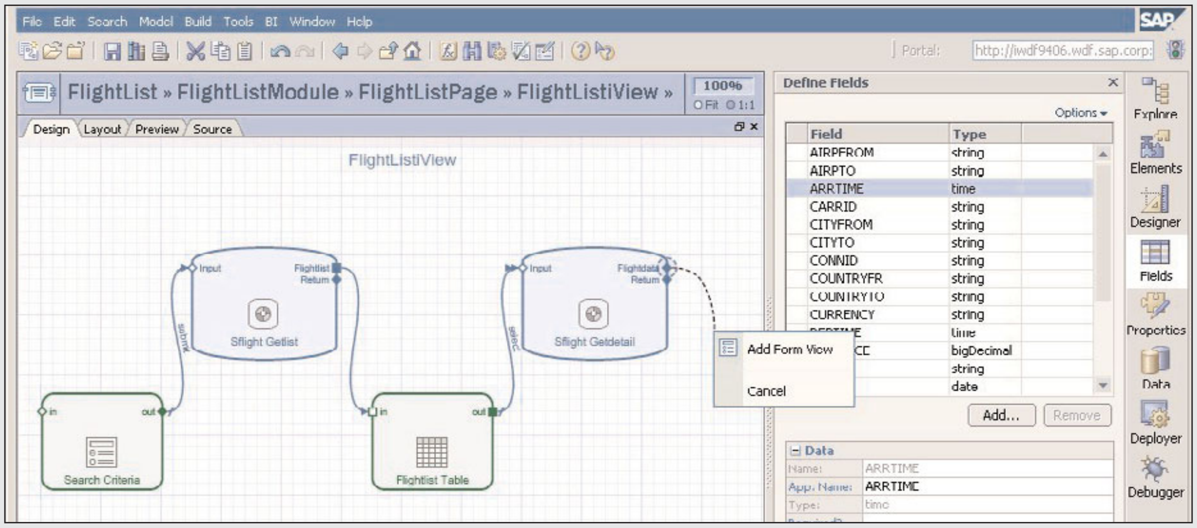
Step 6: Refine the new user interface control

Click on the Layout tab and scroll to the bottom of the view. You'll see an empty Flightdata form control, which we'll use as the output screen for BAPI_SFLIGHT_GETDETAIL. By default, this new control doesn't include any fields, so our first task is to add some.

Click on the Fields tool on the task panel toolbar (if the Define Fields task panel isn't already displayed). The long list of output parameters from BAPI_SFLIGHT_GETDETAIL appears. As shown in **Figure 5**, I've selected 11 fields from the list for demonstration purposes.

Note!

At this point, you'll probably want to rename the default field labels to something more descriptive. To avoid getting off track, I'll leave them as is. Refer to the previous article for details on how to rename labels.

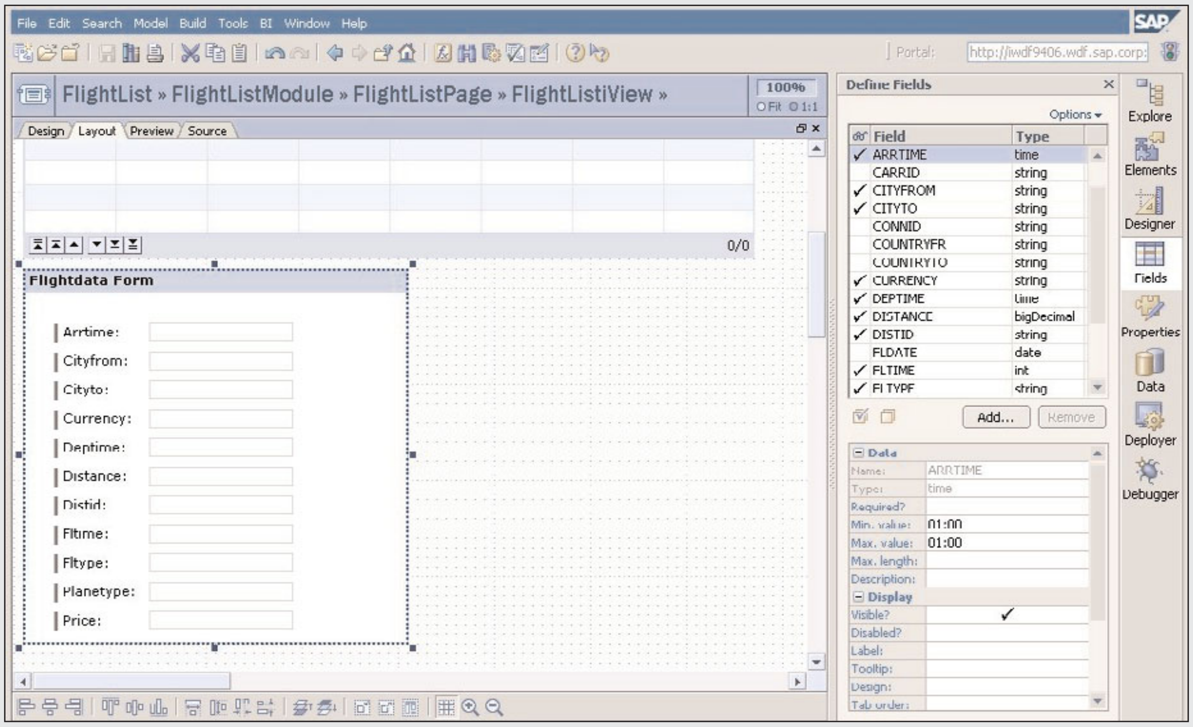


Field	Type
AIRPFROM	string
AIRPTO	string
ARRTIME	time
CARRID	string
CITYFROM	string
CITYTO	string
CONNID	string
COUNTRYFR	string
COUNTRYTO	string
CURRENCY	string
DEPTIME	time
CE	bigDecimal
	string
	date

Data

Name:	ARRTIME
App. Name:	ARRTIME
Type:	time

Figure 4 Creating an output form view for the flight detail results


File Edit Search Model Build Tools BI Window Help
Portal: http://iwdf9406.wdf.sap.corp:
FlightList » FlightListModule » FlightListPage » FlightListView »
100%
Design Layout Preview Source
0/0
Define Fields
Options
Add...
Remove
Explore
Elements
Designer
Fields
Properties
Data
Deployer
Debugger

Flightdata Form

Arrtime:	
Cityfrom:	
Cityto:	
Currency:	
Deptime:	
Distance:	
Distid:	
Fltime:	
Fltype:	
Planetype:	
Price:	

	Field	Type
✓	ARRTIME	time
	CARRID	string
✓	CITYFROM	string
✓	CITYTO	string
	CONNID	string
	COUNTRYFR	string
	COUNTRYTO	string
✓	CURRENCY	string
✓	DEPTIME	time
✓	DISTANCE	bigDecimal
✓	DISTID	string
	FLDATE	date
✓	FLTIME	int
✓	FLTYPE	string

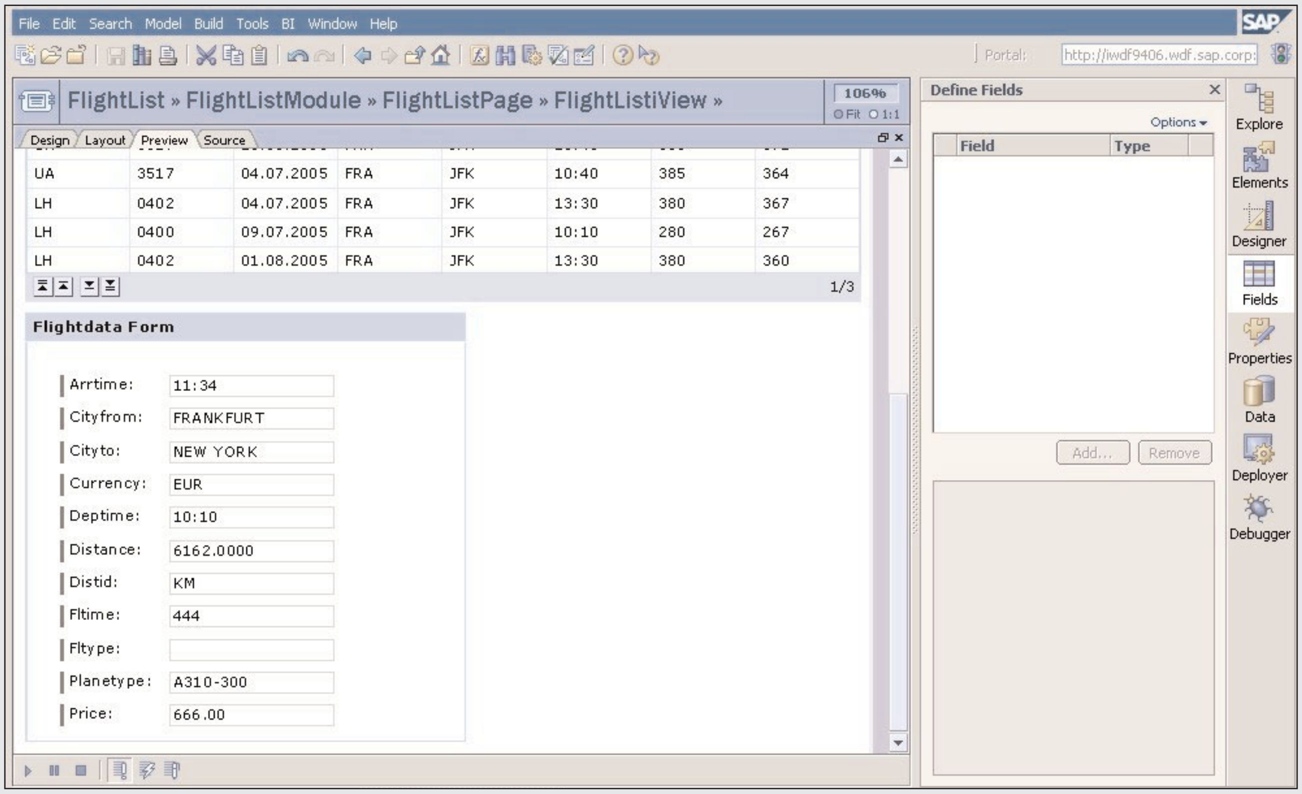
Data

Name:	ARRTIME
Type:	time
Required?	
Min. value:	01:00
Max. value:	01:00
Max. length:	
Description:	

Display

Visible?	✓
Disabled?	
Label:	
Tooltip:	
Design:	
Tab order:	

Figure 5 Selecting the BAPI output fields to appear on the output screen


File Edit Search Model Build Tools BI Window Help
Portal: http://iwdf9406.wdf.sap.corp:
FlightList » FlightListModule » FlightListPage » FlightListiView »
106%
Fit 1:1
Design Layout Preview Source
Define Fields
Options
Field Type
Add... Remove
Explore
Elements
Designer
Fields
Properties
Data
Deployer
Debugger

UA	3517	04.07.2005	FRA	JFK	10:40	385	364
LH	0402	04.07.2005	FRA	JFK	13:30	380	367
LH	0400	09.07.2005	FRA	JFK	10:10	280	267
LH	0402	01.08.2005	FRA	JFK	13:30	380	360

1/3

Flightdata Form

Arrtime:	11:34
Cityfrom:	FRANKFURT
Cityto:	NEW YORK
Currency:	EUR
Deptime:	10:10
Distance:	6162.0000
Distid:	KM
Fltime:	444
Fltype:	
Planetype:	A310-300
Price:	666.00

Figure 6 Previewing the new drill-down functionality

With the data retrieved, the flow of control established, and the layout defined, we're now ready to run the revised iView for the first time.

Step 7: Test and deploy the revised FlightListiView

Before we deploy the revised FlightListiView, click on the Preview tab to test the added functionality. Enter some flight criteria to perform a search, and then select a flight from the results table. **Figure 6** shows the details of the flight for which I searched (i.e., a flight leaving the city of Frankfurt and arriving in the city of New York).

The deployment procedure is the same as described in the previous article. Click on the Deployer tool on the task panel toolbar, check the syntax for errors, and then follow the prompts to build and send the business package to your portal.

We've now successfully added drill-down functionality to the FlightListiView. See how easy this was? Let's try adding another functionality. In the next section I'll show you how to add charts to an iView.

Adding charts to an iView

As you might imagine, some content is more easily digested in graphical form, such as sales data. You can easily detect trends, averages, etc. — things that are not easy to spot quickly when you are looking at raw data. Being able to understand data at a glance is especially important when building “cockpits” with SAP EP, where the user needs to interpret multiple iViews quickly.

Visual Composer offers all standard types of charts, such as pie charts, bar charts, and line charts, similar to those available from any Microsoft Office application. Each chart type has its specific characteristics with respect to its properties and the way data is supplied. For example, in a bar chart, you must define the range of possible input values (the x-axis) and supply these input values with potentially multiple data series that represent the bars in numerical form. This data is typically stored in tables that need to be associated with the chart in the Visual Composer modeling environment.

The good news is that adding charts to iViews using Visual Composer is very easy. The basic idea is to add chart elements to the iView layout and then connect their input parameters to the data sources that supply the chart with data. Visual Composer automatically renders the graphical representation of the data selected for charting. (You can customize the way Visual Composer renders the data, but that goes beyond the scope of this article. However, if you have worked with charts before, you can probably figure out how to make chart-related enhancements.)

To demonstrate how to add a chart, let’s change our iView to display a bar chart showing the number of occupied seats vs. overall seat capacity for each flight in the Flightlist table. Our chart will let portal users quickly see which flights have seats remaining and which flights are fully booked.

To add a bar chart to the FlightListiView, follow these steps:

1. Add a Chart View element to the FlightList model.
2. Link the chart to the Flightlist table.
3. Define a custom field to supply data for the chart’s x-axis.
4. Select the fields to use in the chart.
5. Refine the layout of the chart.
6. Test and deploy the revised FlightListiView.

We’ll take a look at each of these steps in turn.

Step 1: Add a Chart View element to the FlightList model

Switch to the Design tab and click on the Elements tool on the task panel toolbar to display the available elements for the FlightList model. Drag-and-drop the Chart View element onto the workspace. A new chart view object is added to the layout (see **Figure 7** on the following page).

We next need to select a data source for the chart.

Step 2: Link the chart to the Flightlist table

In this example, we want to set the Flightlist table as the data source for the chart by connecting the output port of the Flightlist table to the input port of the chart view. Because this is an association,[8] the connection is specifically marked with a crosshatched line (see Figure 7).

Now that the chart knows where to retrieve data from, we need to define a single-value key field that will serve as the chart’s x-axis.

Step 3: Define a custom field to supply data for the chart’s x-axis

Chart views require a single-value key field. Our Flightlist table, however, consists of three fields: carrier ID, connection ID, and flight date. Therefore, in order to be able to use the table as the source for our chart, we need to add a field to the table that concatenates the three fields into a single field. We’ll do this by adding a custom “computed” field to the Flightlist table called FlightID that includes a formula for “adding” together the three fields. For example, adding together carrier ID LH, connection ID 0400, and flight date 11/03/2005 would result in the single-value field LH 0400 11/03/2005. Custom fields are computed when the table view is filled, which occurs

[8] Association means that nothing is called or executed to produce the chart. It is merely a different way to represent the data.

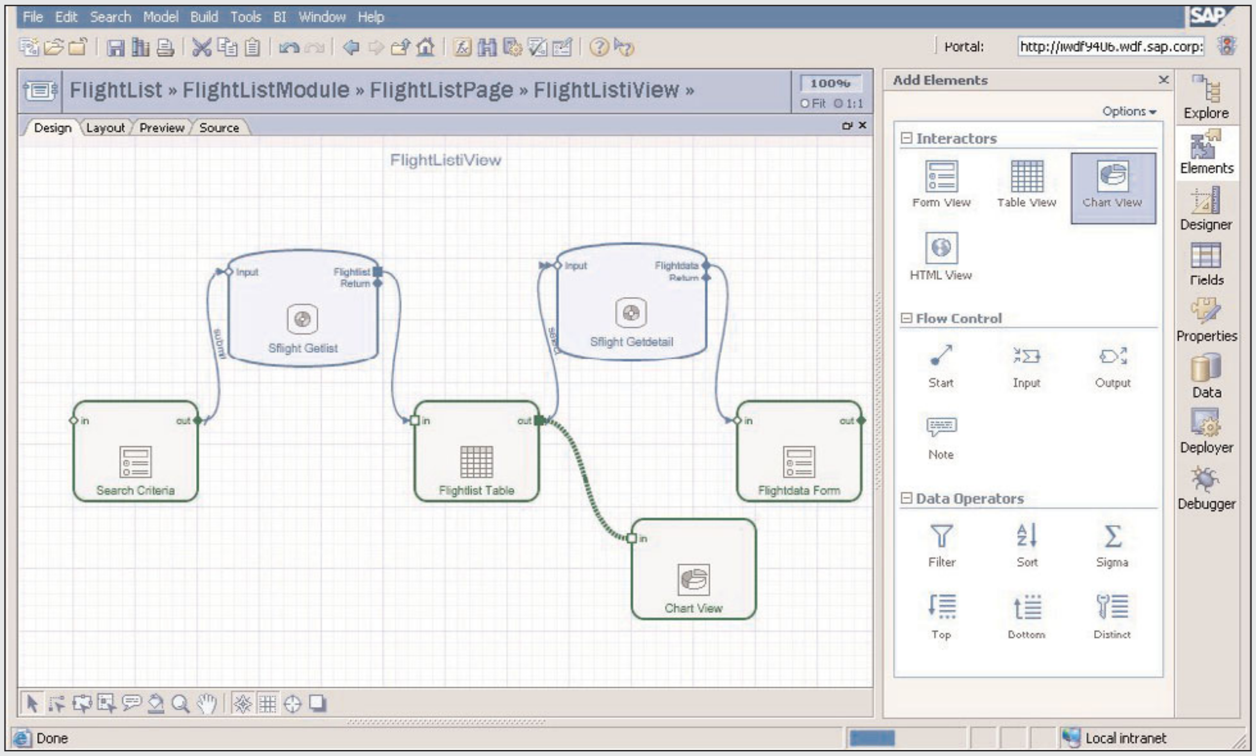


Figure 7 Adding the Chart View element

after the search is executed and the data with which the custom fields are computed is retrieved.

As shown in **Figure 8**, we can add this field without leaving the Design view. Simply click on the Flightlist table and select the Fields tool on the task panel toolbar. Next, click on the Add button in the Define Fields task panel. The Add New Field dialog appears, in which you enter the name of the field you want to add. In our case, enter “FlightID” as the name of the new field, and set the field type to “Text Field.” The custom field is then added to the field list for the table control.

Because this field is only required internally, you can hide it by selecting the field in the Define Fields task panel, and deselecting the Visible property for the field in the properties area in the lower part of the panel.

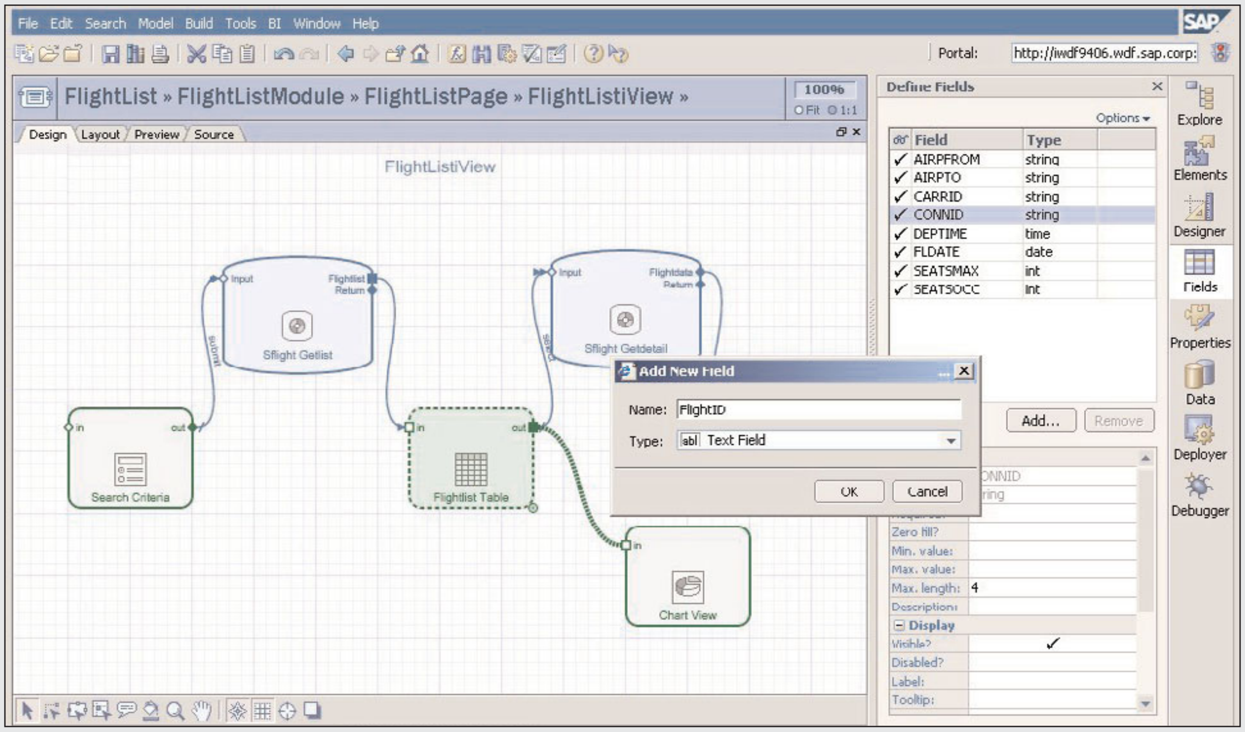
Finally, you need to define the formula for the field that will concatenate the three fields into one. Double-click on the FlightID field to open the Expression Editor window (see **Figure 9**). In the Computation column for the FlightID field, enter the following formula:

```
@carrid+@connid+DSTR(@fldate,
'XML_DATE')
```

where the plus sign (+) is the Visual Composer operator for concatenating string (i.e., text type) fields.

As you can see from the formula entered for the FlightID field, references to other field names are preceded by the @ (e.g., @carrid refers to the CARRID column). To concatenate two or more text fields together, insert a plus sign (+) between each field, as we have done with our formula.

The `DSTR` function formats a date expression as a string using a specific format, such as the one we have



Field	Type
✓ AIRPFROM	string
✓ AIRPTO	string
✓ CARRID	string
✓ CONNID	string
✓ DEPTIME	time
✓ FLDATE	date
✓ SEATSMAX	int
✓ SEATSOCC	int

Add New Field

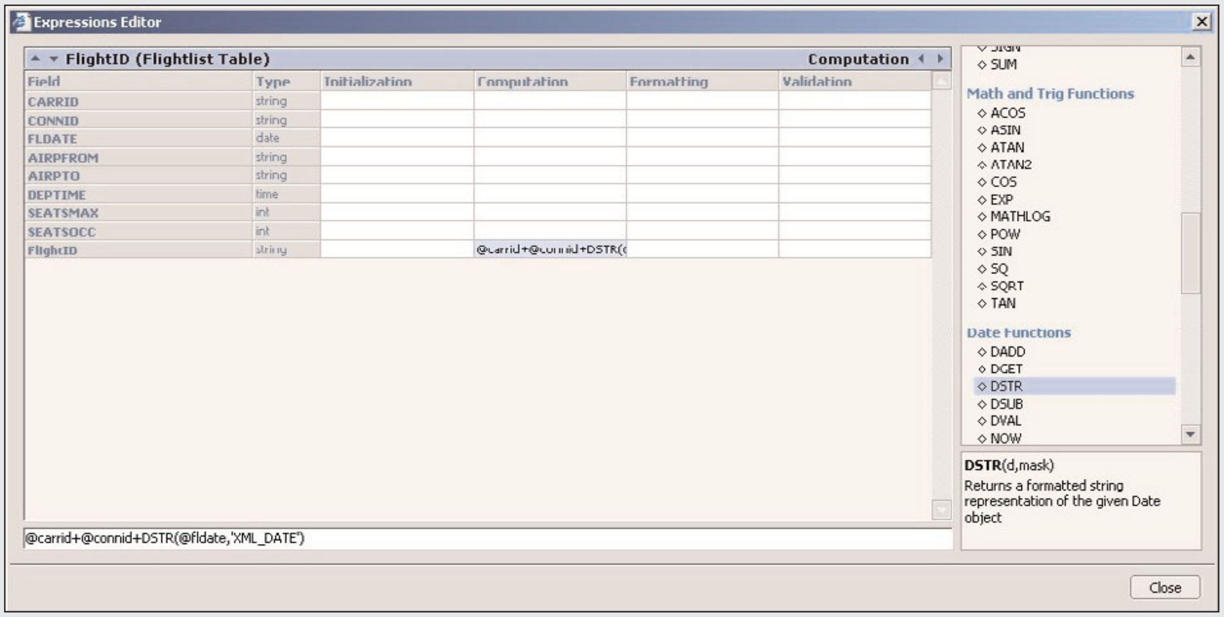
Name: FlightID

Type: Text Field

OK Cancel

Zero fill?	
Min. value:	
Max. value:	
Max. length:	4
Description:	
Display	
Visible?	✓
Disabled?	
Label:	
Tooltip:	

Figure 8 Defining a single-value key field to supply data for the chart's x-axis



Field	Type	Initialization	Computation	Formatting	Validation
CARRID	string				
CONNID	string				
FLDATE	date				
AIRPFROM	string				
AIRPTO	string				
DEPTIME	time				
SEATSMAX	int				
SEATSOCC	int				
FlightID	string		@carrid+@connid+DSTR((		

@carrid+@connid+DSTR(@fldate,'XML_DATE')

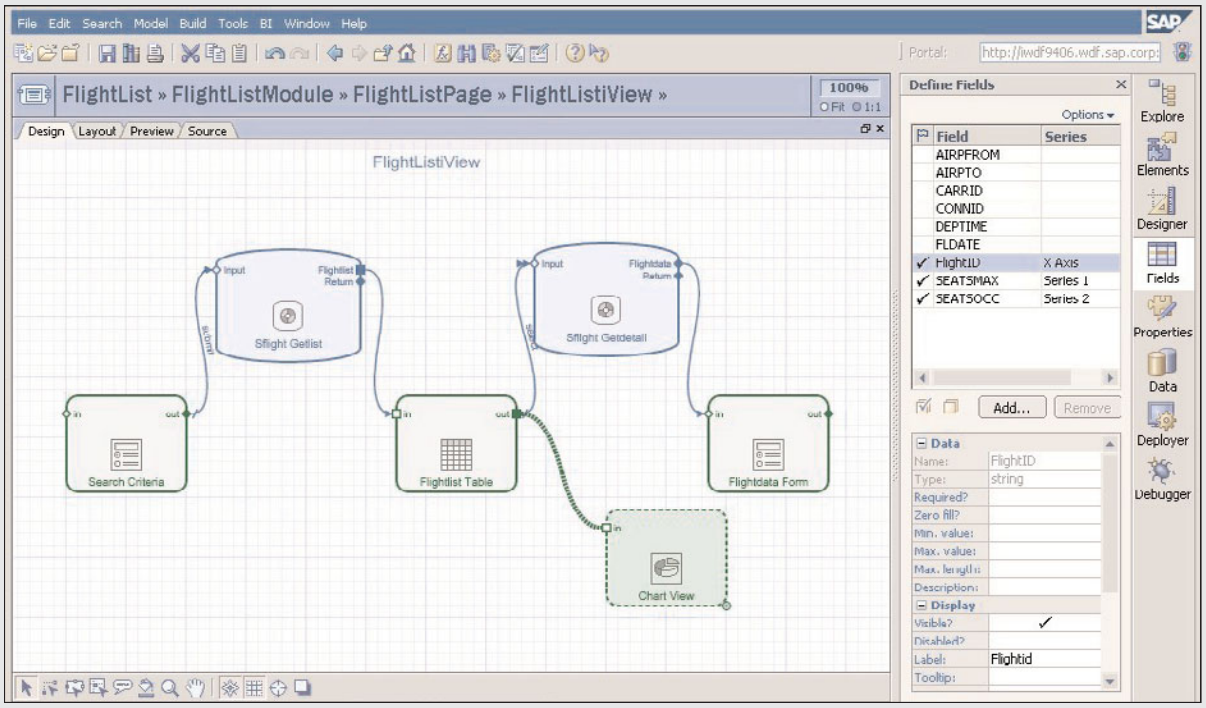
◇ SUM

Math and Trig Functions
◇ ACOS
◇ ASIN
◇ ATAN
◇ ATAN2
◇ COS
◇ EXP
◇ MATHLOG
◇ POW
◇ SIN
◇ SQ
◇ SQRT
◇ TAN

Date Functions
◇ DADD
◇ DGET
◇ DSTR
◇ DSUB
◇ DVAL
◇ NOW

DSTR(d,mask)
Returns a formatted string representation of the given Date object

Figure 9 Using the Expression Editor to build a formula to concatenate three fields into one field



	Field	Series
	AIRPFROM	
	AIRPTO	
	CARRID	
	CONNID	
	DEPTIME	
	FLDATE	
✓	FlightID	X Axis
✓	SEATSMAX	Series 1
✓	SEATSOCC	Series 2

Data	
Name:	FlightID
Type:	string
Required?	
Zero fill?	
Min. value:	
Max. value:	
Max. length:	
Description:	
Display	
Visible?	✓
Disabled?	
Label:	Flightid
Tooltip:	

Figure 10 Specifying the x-axis and the data series for the chart

used (`XML_DATE`) to specify a date in XML syntax. We need the `DSTR` function to convert date format into string format. The formatting rules are not important here, since we will not display the column.

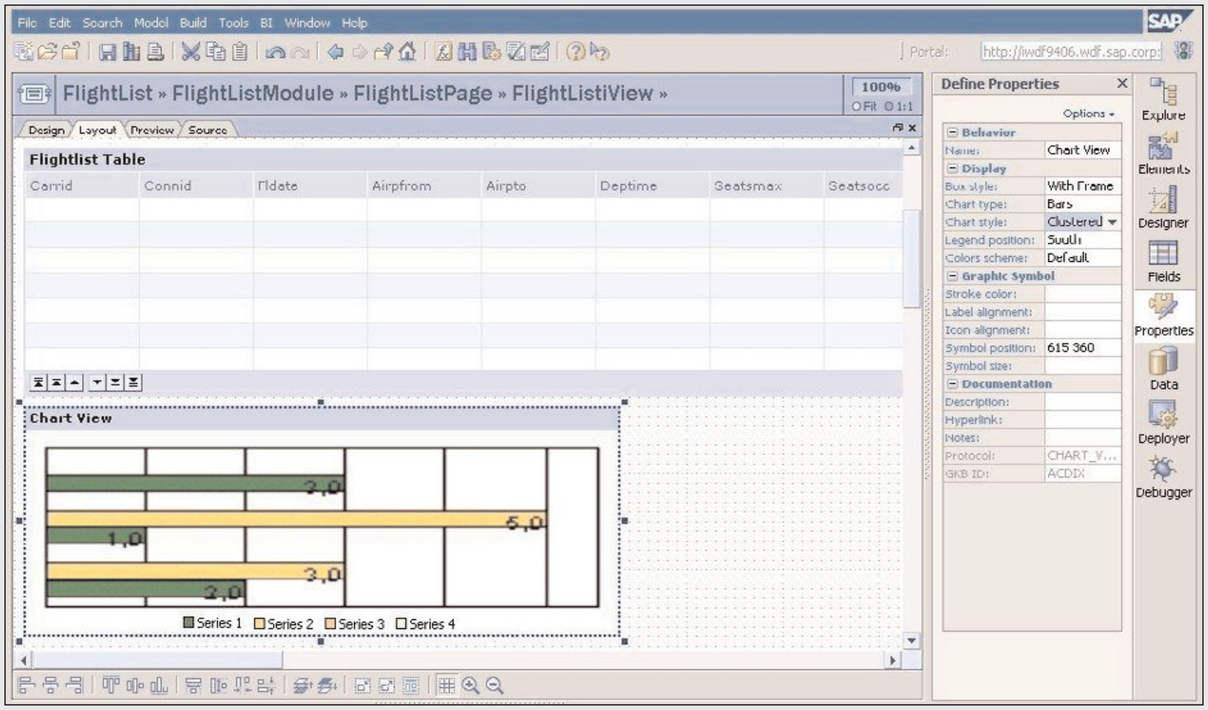
> ***Note!***
>
> The Expression Editor provides a variety of functions. A brief description of a selected function appears at the lower right of the Expression Editor. For a more detailed explanation of a function, you can use the Visual Composer online help (available from the menu bar or from the SAP Help Portal at http://help.sap.com).

Once you've entered the formula, close the Expression Editor. Next we need to specify the fields we want to use in the chart and the data we want the chart to display.

Step 4: Select the fields to use in the chart

In the workspace, click on the chart view, if it's not already highlighted, and make sure the Fields tool is selected on the task panel toolbar.

Now we need to identify which fields we want to use in the chart by checking the box to the left of the field in the Define Fields task panel. First, let's select the FlightID field as the field that will provide values to the x-axis. Click on the Series column, click on the drop-down arrow that appears, and then select the X Axis option, as shown in **Figure 10**.


File Edit Search Model Build Tools BI Window Help
Portal: http://iwdf9406.wdf.sap.corp:
FlightList » FlightListModule » FlightListPage » FlightListiView »
100%
Design Layout Preview Source
Flightlist Table
Carrid Connid Fldate Airpfrom Airpto Deptime Seatsmax Seatsocc
Chart View
2,0
5,0
1,0
3,0
2,0
Series 1 Series 2 Series 3 Series 4
Explore
Elements
Designer
Fields
Properties
Data
Deployer
Debugger

Define Properties

Options

Behavior	
Name:	Chart View
Display	
Box style:	With Frame
Chart type:	Bars
Chart style:	Clustered
Legend position:	South
Colors scheme:	Default
Graphic Symbol	
Stroke color:	
Label alignment:	
Icon alignment:	
Symbol position:	615 360
Symbol size:	
Documentation	
Description:	
Hyperlink:	
Notes:	
Protocol:	CHART_V...
GKB ID:	ACDIX

Figure 11 Modifying the chart layout and its display properties

Next, select the numerical fields that will provide the data series to be displayed graphically as bars in the bar chart. In our case, these are the SEATSOCC and SEATSMAX fields, which show the number of occupied seats and the overall seat capacity, respectively. Specify them as Series 1 and Series 2, as shown in Figure 10, the same way you selected the X Axis option. (The list of fields available for the chart view element matches that of the Flightlist table, because we linked the output port of the table to the input port of the chart in Step 2.)

With the chart input and data defined, we can now examine the chart layout and make any necessary modifications.

Step 5: Refine the layout of the chart

As with screens, you can modify the layout of a chart. Switch to the Layout tab, and then scroll down to the bottom of the view to locate the new chart object (see **Figure 11**). Here you can resize the chart, and select any of the display properties you want to apply from the Define Properties task panel. As you can see in Figure 11, I've chosen the Clustered chart style.

Note!

Keep the number of data series per chart small, so that the information being conveyed is easily and readily understood.

Note!

You can reverse the orientation of the chart if you prefer vertical columns. Click on the drop-down arrow in the Chart style field in the Define Fields task panel and select a chart type with vertical columns.

In the Layout view, the chart is displayed with fictitious data. To see the chart with live data, switch to the Preview tab, which we'll do next.

Step 6: Test and deploy the revised FlightListiView

Before we deploy the revised FlightListiView, click on the Preview tab to test the added chart with live data in the FlightListiView. Since we associated the chart with the Flightlist table, once the table is filled, the chart is drawn. The availability (or capacity) is displayed as Series 1; the number of booked seats is displayed as Series 2 (see **Figure 12**).

The deployment procedure is the same procedure described previously. Simply click on the Deployer tool on the task panel toolbar, check the syntax for errors, and then follow the prompts.

At this point you know how to add drill-down functionality and charts to an iView. Now that you've gained some experience adding functionality enhancements, let's venture into adding Web functionality to an iView. In the next section, we'll look at how to add HTML views to an iView to integrate external content.

Adding an HTML view to an iView

With the ever-increasing number of business applications on the Internet (e.g., online order status, shipment tracking, flight status), it won't be long before you'll want to leverage these resources from your iViews. For example, we might want to enhance our FlightListiView so that users can double-click on a flight to check its status. Alternatively, we might want to let users view Web content that is associated with the selected flights.

One way to do this is to add an HTML view, which is a control that contains Web content accessed via a URL (e.g., airport information, weather information, related hotel sites, etc.). An HTML view is a powerful way to integrate your iViews with external content. You can design your iView to dynamically insert data into the URL, so you can pass a selected flight number, order number, ZIP code, tracking number, and so on as part of the URL. As a simple example, let's modify our FlightListiView to display airport information from www.worldairportguide.com.

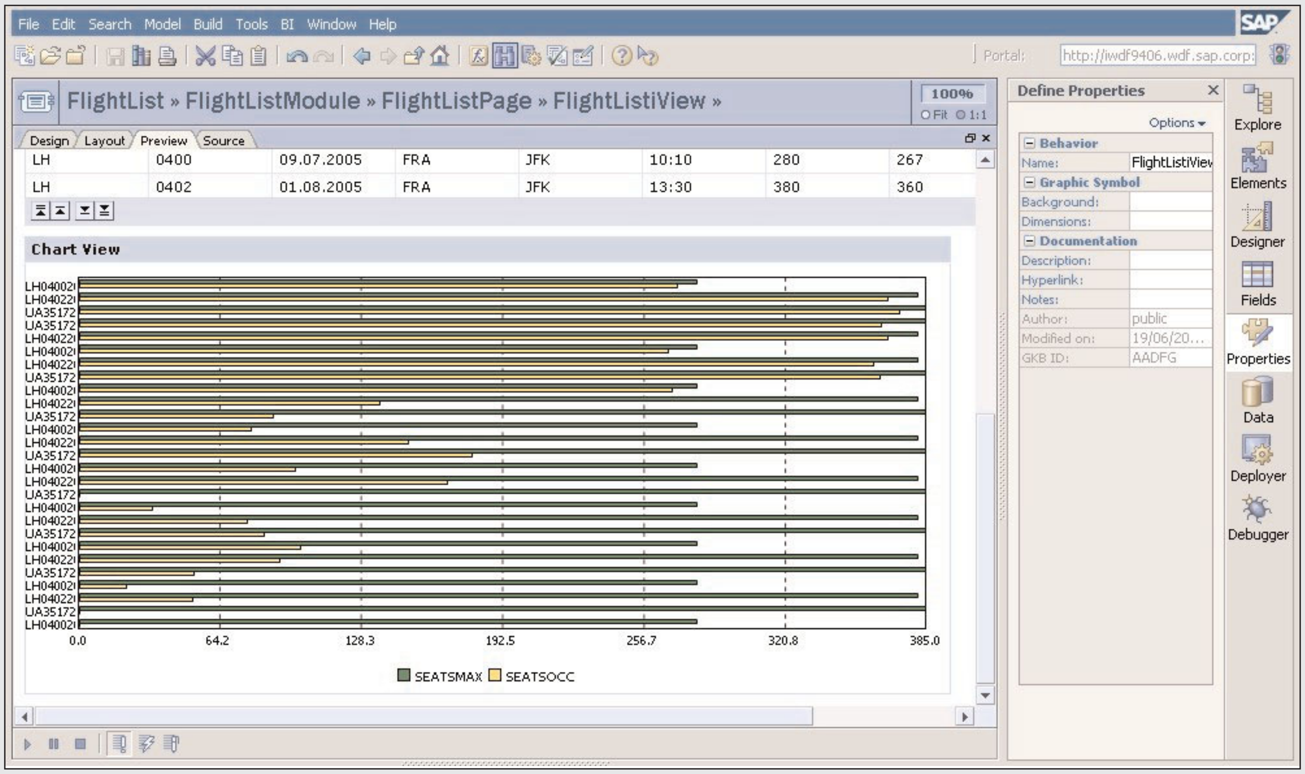
The procedure for adding an HTML View element to our example iView is similar to the one for adding charts:

1. Add an HTML View element to the FlightList model.
2. Link the HTML view to the Flightlist table.
3. Define a custom field to supply the HTML view with a URL.
4. Map the custom field to the HTML view.
5. Refine the layout of the HTML view.
6. Test and deploy the revised FlightListiView.

Let's quickly go through the steps.

Step 1: Add an HTML View element to the FlightList model

Switch to the Design tab and click on the Elements tool on the task panel toolbar. Drag the HTML View element from the Add Elements task panel, and drop it onto the workspace. A new HTML view object is added to the layout.



LH	0400	09.07.2005	FRA	JFK	10:10	280	267
LH	0402	01.08.2005	FRA	JFK	13:30	380	360

Name:	FlightListiView
Background:	
Dimensions:	
Description:	
Hyperlink:	
Notes:	
Author:	public
Modified on:	19/06/20...
GKB ID:	AADFG

Figure 12 Previewing the chart in the FlightListiView

Step 2: Link the HTML view to the Flightlist table

Next, connect the HTML view to the Flightlist table (see **Figure 13** on the following page). As with the chart element, we set the Flightlist table as the data source for the HTML view by connecting the output port of the Flightlist table to the input port of the HTML view.

Step 3: Define a custom field to supply the HTML view with a URL

Now you need to add a custom field to the Flightlist table that computes the URL that will feed into the HTML view. As we did when we added the chart, click on the Fields tool on the task panel toolbar, click on Add to add a new field, and then double-click on the field in the field list to open the Expression Editor (refer back to Figure 9). Now enter the following expression:

```
www.worldairportguide.com/airports/
'+@airpto+'/'+@airpto+'.asp
```

This formula will generate a URL for the currently selected flight in the Flightlist table based on the airport code.

Step 4: Map the custom field to the HTML view

Finally, so the HTML view will know to draw its URL from this field, you need to modify the URL property for this object. Click on the HTML view

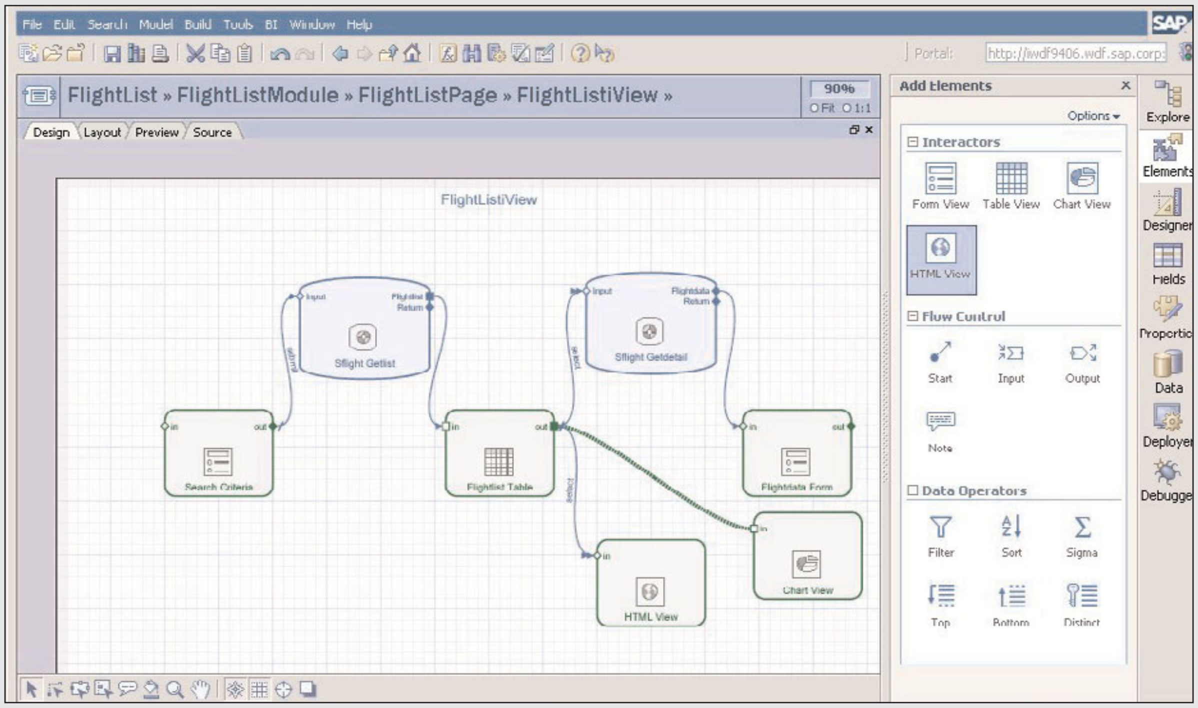
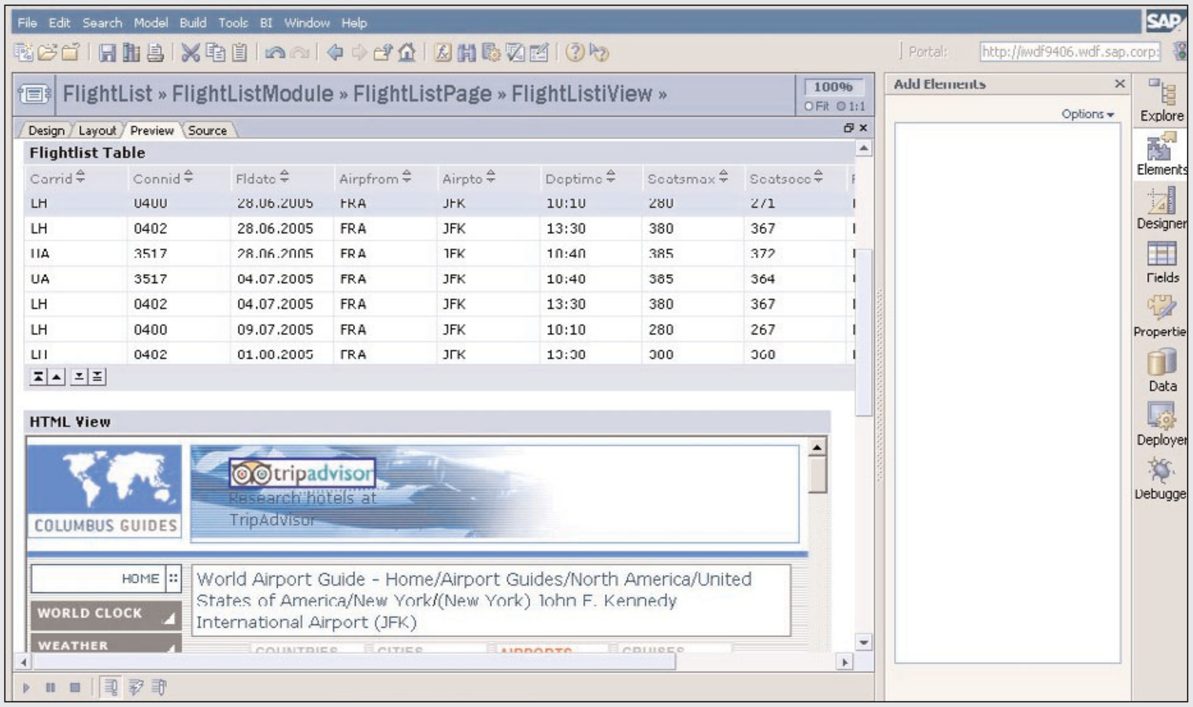


Figure 13 The HTML view linked to the Flightlist table


File Edit Search Model Build Tools BI Window Help
Portal: http://iwdf9406.wdf.sap.corp
FlightList » FlightListModule » FlightListPage » FlightListiView »
100%
Design Layout Preview Source
Add Elements
Options
Explore
Elements
Designer
Fields
Properties
Data
Deployer
Debugger

Flightlist Table

Carrid	Connid	Fldate	Airpfrom	Airpto	Deptime	Seatsmax	Seatsocc
LH	0400	28.06.2005	FRA	JFK	10:10	280	271
LH	0402	28.06.2005	FRA	JFK	13:30	380	367
UA	3517	28.06.2005	FRA	JFK	10:40	385	372
UA	3517	04.07.2005	FRA	JFK	10:40	385	364
LH	0402	04.07.2005	FRA	JFK	13:30	380	367
LH	0400	09.07.2005	FRA	JFK	10:10	280	267
LH	0402	01.08.2005	FRA	JFK	13:30	300	360

HTML View

COLUMBUS GUIDES
tripadvisor
Research hotels at TripAdvisor
HOME
WORLD CLOCK
WEATHER
COUNTRIES
CITIES
AIRPORTS
CRUISES

World Airport Guide - Home/Airport Guides/North America/United States of America/New York/(New York) John F. Kennedy International Airport (JFK)

Figure 14 Previewing the HTML view in the FlightListiView

control in the workspace, and click on the Properties tool in the task panel toolbar to open the Define Properties task panel. Map the URL property to the custom field in the table control.

Step 5: Refine the layout of the HTML view

As you did with the chart, you can use the Layout view to improve the appearance of the HTML view.

Step 6: Test and deploy the revised FlightListiView

Before we deploy the revised FlightListiView, click on the Preview tab to test the added HTML view in the FlightListiView (see **Figure 14**). Note that the HTML view is refreshed with the appropriate information when you select a line in the table.

The deployment procedure is the same procedure described previously. Click on the Deployer tool on the task panel toolbar, check the syntax for errors, and follow the prompts.

Up to this point, you have learned how to add functionality to iViews. In the next section, I will show you how to improve the usability of your iViews through a technique called "eventing."

Integrating iViews with portal eventing

If we continue adding functionality to our example iView, at some point it will become large, complex, and difficult to maintain. To avoid this scenario, we can divide the views and functions of the iView into individual iViews that display on the same page. Since the views and functions are interrelated, however, we need a way for them to "talk" to one another, so that when a user performs an action in one iView, the iView can notify other iViews that the action occurred and they can react as needed. This is called "portal eventing."

To illustrate why portal eventing is useful, imagine a portal page containing five iViews that report various regional sales statistics. One way to let the user view the different regions is to place a Sales Region drop-down box in each of the five iViews. A better and more efficient way is to place a sixth iView on the page with a single region drop-down. Therefore, when a user selects North America, this iView can use portal eventing to notify the other five iViews, to update themselves with data for North America.

Fortunately, the portal runtime offers a standard feature called the Portal Client-Side Eventing Framework for exactly this purpose. Using this event framework often involves coding in Java. Visual Composer, however, offers a very easy way to do this that doesn't involve any coding.

Note!

In this context, the iView in which the event occurs is called the master iView. The iView or iViews that receive and respond to the event are called dependent iViews.

The procedure to connect two different iViews is pretty straightforward. All you have to do is define suitable output and input ports for the connected iViews to make the communication happen.

To implement portal eventing in our FlightList model, we will separate the flight list controls (i.e., the Flightlist table and the Search Criteria form) from the flight detail controls (i.e., the BAPI_SFLIGHT_GETDETAIL object and the Flightdata form) — all of which are currently on the FlightListiView. We need to create a new iView and then move the BAPI object and Flightdata form to the new iView. We also need to create a connection between the two iViews to establish communication, so the FlightListiView can notify the new iView to display the details of a selected flight.

1. Remove the BAPI object and the Flightdata form from the FlightListiView.

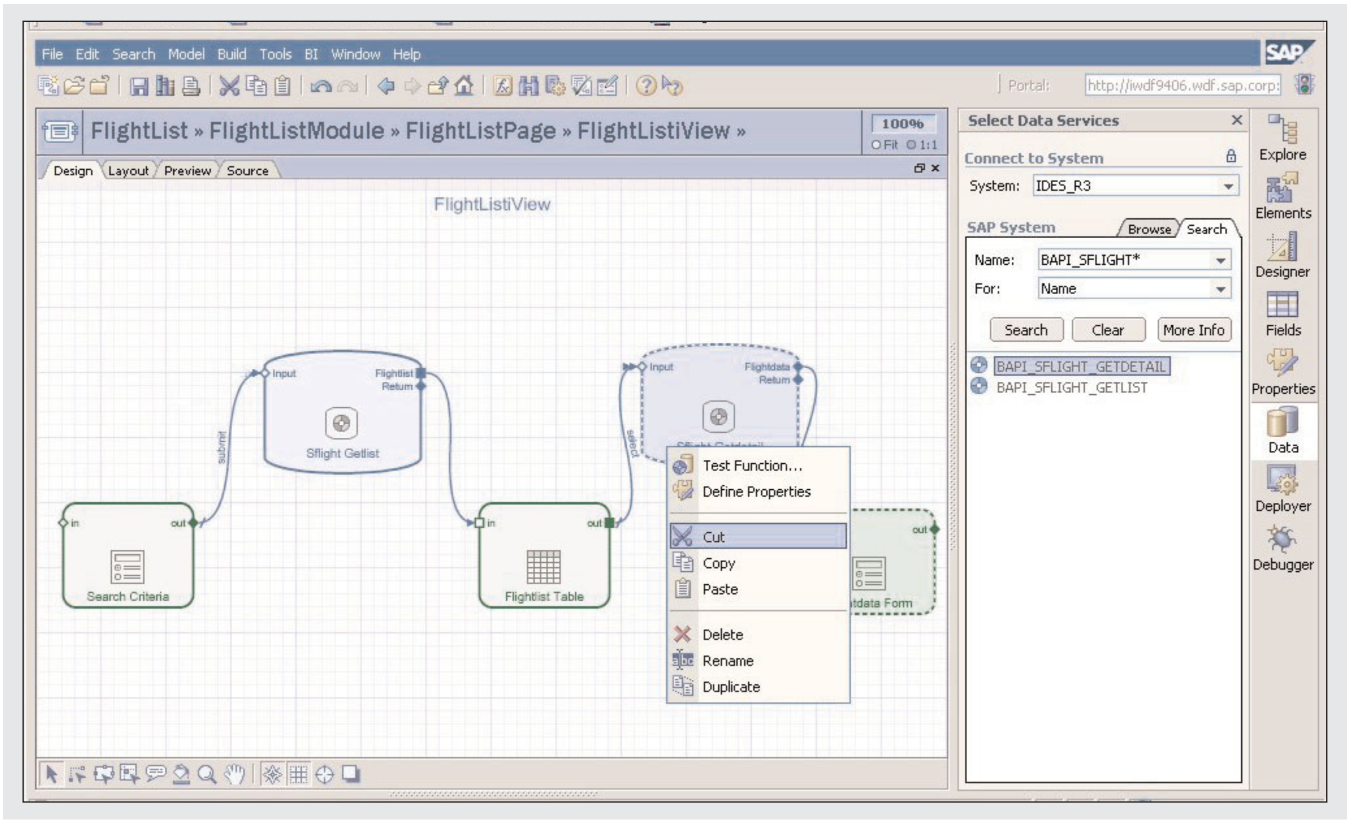


Figure 15 Removing the BAPI object and the Flightdata form from the FlightListiView

2. Add a new iView to the FlightListPage for displaying flight details.

3. Add an output port to the Flightlist table to connect to the new iView.

4. Make the Flightlist table fields available to the output port.

5. Add the BAPI object and the Flightdata form to the new iView.

6. Add an input port to the new iView to connect to the Flightlist table.

7. Define the receiving fields for the input port and map them to the BAPI object.

8. Deploy the iViews and test them at the page level.

The following sections detail each of these steps.

Step 1: Remove the BAPI object and the Flightdata form from the FlightListiView

Switch to the Design tab and navigate to the FlightListiView. We need to remove the BAPI_SFLIGHT_GETDETAIL object and the Flightdata form from the FlightListiView so we can use them in the new iView that we are going to create. To select both objects, press and hold the Shift key, click on each object to select it, right-click on the selection, and then select Cut from the context menu, as shown in **Figure 15**.

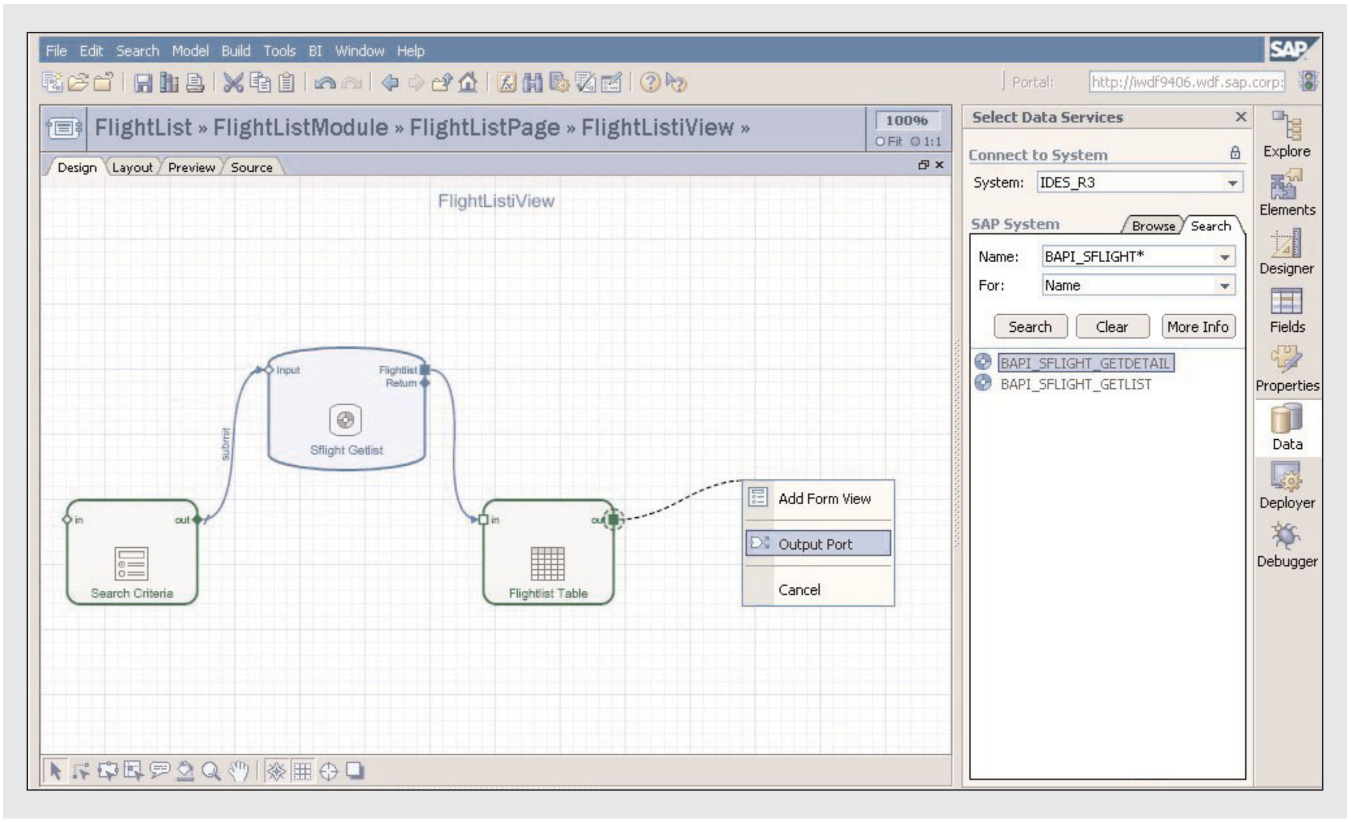


Figure 16 Adding an output port to the Flightlist table

Step 2: Add a new iView to the FlightListPage for displaying flight details

To create the new iView, navigate up the hierarchy to the page level. Remember from the previous article that only the iView element is available at the page level. Select the Elements tool on the task panel toolbar, drag and drop the iView element onto the workspace, and then rename the element "FlightDetailiView." We'll return to this iView shortly, but first we need to add an output port to the Flightlist table.

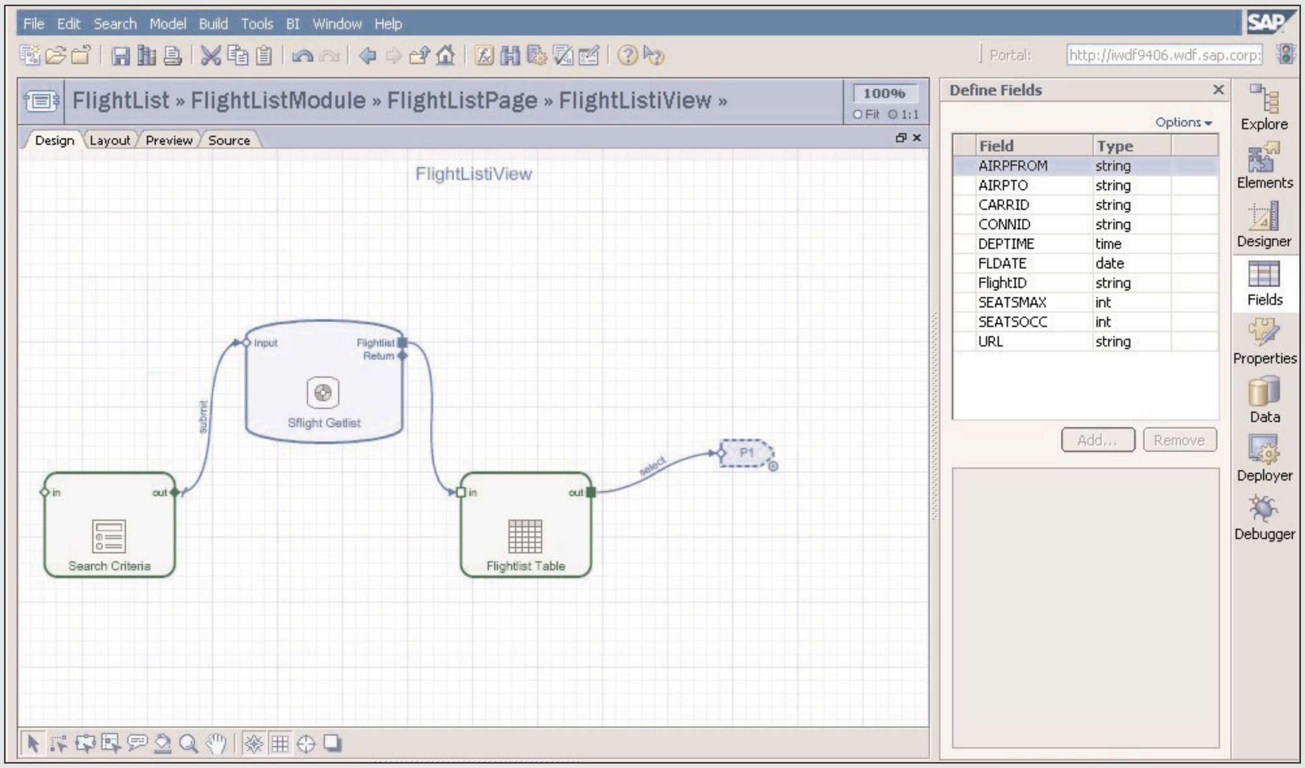
Step 3: Add an output port to the Flightlist table to connect to the new iView

To add an output port to the table of flights, drill down into the FlightListiView. Right-click on the Flightlist table, and then select Output Port on the context menu (see **Figure 16**).

To rename the output port, right-click on the port, select the Rename option on the context menu, and then enter "P1" as the name of the port.

Caution!

Remember the name you assign the port. For portal eventing to work, the name of the output port must match the name of the input port exactly.



Field	Type
AIRPFROM	string
AIRPTO	string
CARRID	string
CONNID	string
DEPTIME	time
FLDATE	date
FlightID	string
SEATSMAX	int
SEATSOCC	int
URL	string

Figure 17 The FlightList table with the P1 out port selected and available fields listed

Step 4: Make the Flightlist table fields available to the output port

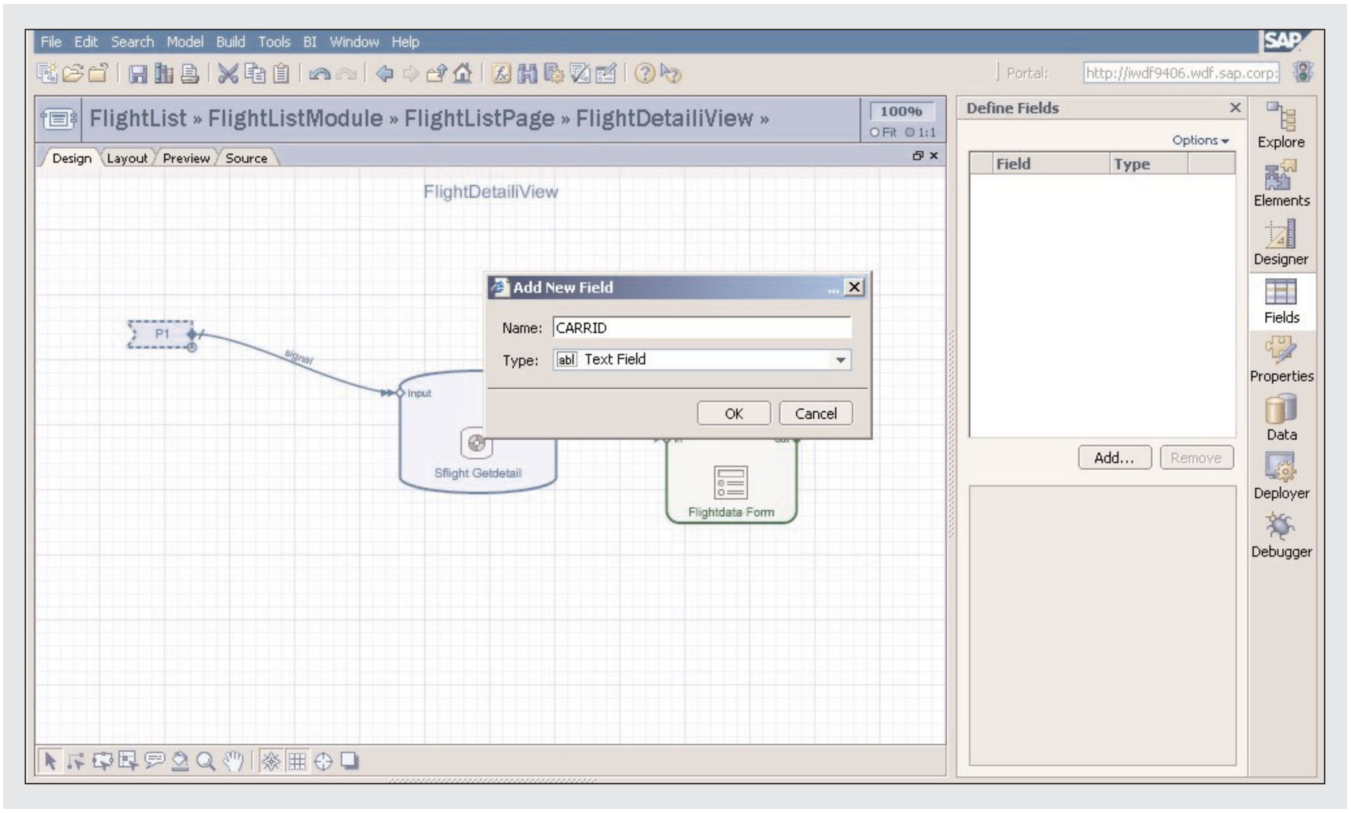
Before we add the input port to the FlightDetailiView, we need to make sure the fields in the Flightlist table are available to its P1 output port. We need to refer to these “sending” fields when we define the receiving fields for the input port, so data can be retrieved when a flight is selected. To make the fields available to the output port, make sure the P1 port object is selected, and select the Fields tool on the task panel toolbar, (see **Figure 17**). The Define Fields task panel displays the fields that are visible to the P1 port. Keep in mind that for a proper field transport, the names of the output port’s sending fields and the names of the input port’s receiving fields must be identical. Now we need to return to the FlightDetailiView.

Step 5: Add the BAPI object and the Flightdata form to the FlightDetailiView

Navigate to the page level and double-click on the FlightDetailiView. Right-click anywhere in the white space of the workspace, and select Paste from the context menu. The objects we cut previously from the FlightListiView — the BAPI object and the Flightdata form — are pasted into the FlightDetailiView.

Step 6: Add an input port to the new iView to connect to the Flightlist table

We now need to feed the BAPI object in the FlightDetailiView with data from the Flightlist table


File Edit Search Model Build Tools BI Window Help
Portal: http://iwdf9406.wdf.sap.corp:
FlightList » FlightListModule » FlightListPage » FlightDetailiView »
100%
Design Layout Preview Source
FlightDetailiView
P1
signal
Input
Sflight Getdetail
Flightdata Form
Define Fields
Options
Field Type
Add... Remove
Explore
Elements
Designer
Fields
Properties
Data
Deployer
Debugger

Add New Field

Name: CARRID

Type: Text Field

OK Cancel

Figure 18 Adding a receiving field to the input port

via its P1 output port. Click on a connector and drag it to the input parameter on the left side of the BAPI object. When you release the mouse button, the context menu appears. Select Input Port from the menu and then rename the port "P1."

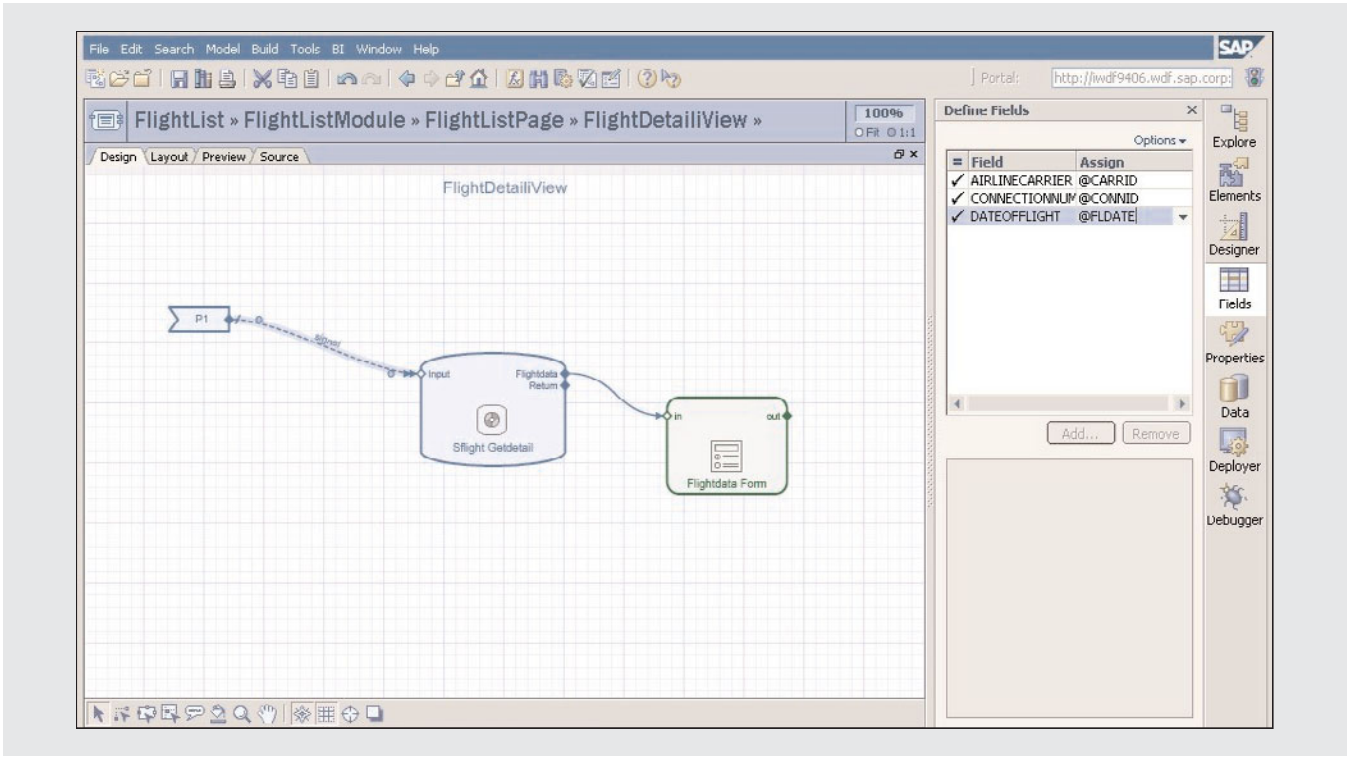
Step 7: Define the receiving fields for the input port and map them to the BAPI object

Now, we need to define the receiving fields for the input port. We need to add the CARRID, CONNID, and FLDATE fields. Make sure the P1 input port is selected, and then select the Fields tool on the task panel toolbar. Click on the Add button in the Define Fields task panel, and then in the Add New Field dialog, enter "CARRID" and make sure the type is "Text Field" (see **Figure 18**). Add the CONNID and FLDATE fields as Text Field types in the same way. Remember that the field names must match the fields shown in Figure 17 exactly.

Finally, we need to map the input port fields to the input parameters of BAPI_SFLIGHT_GETDETAIL. For each input parameter of the BAPI_SFLIGHT_GETDETAIL object, map the corresponding input port field (see **Figure 19** on the following page).

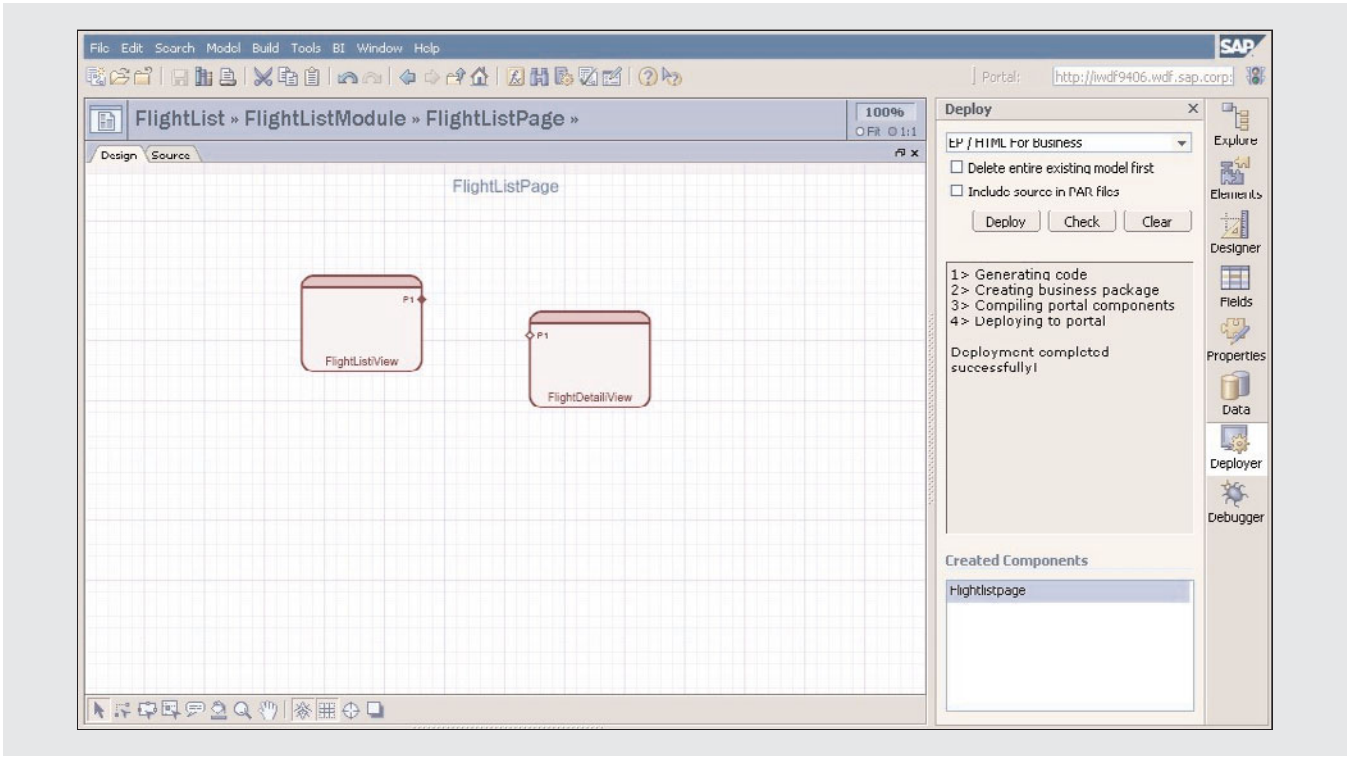
Step 8: Deploy the iViews and test them at the page level

Unlike the previous examples, where we tested each iView individually by previewing them, we need to test the iViews at the page level (see **Figure 20** on the following page) because you can only preview one iView at a time. To test both iViews — FlightListiView and FlightDetailiView — we need to deploy the page to the portal server. You can deploy the complete page containing both iViews in one step.



Field	Assign
✓ AIRLINECARRIER	@CARRID
✓ CONNECTIONNUM	@CONNID
✓ DATEOFFLIGHT	@FLDATE

Figure 19 Three fields mapped to the BAPI

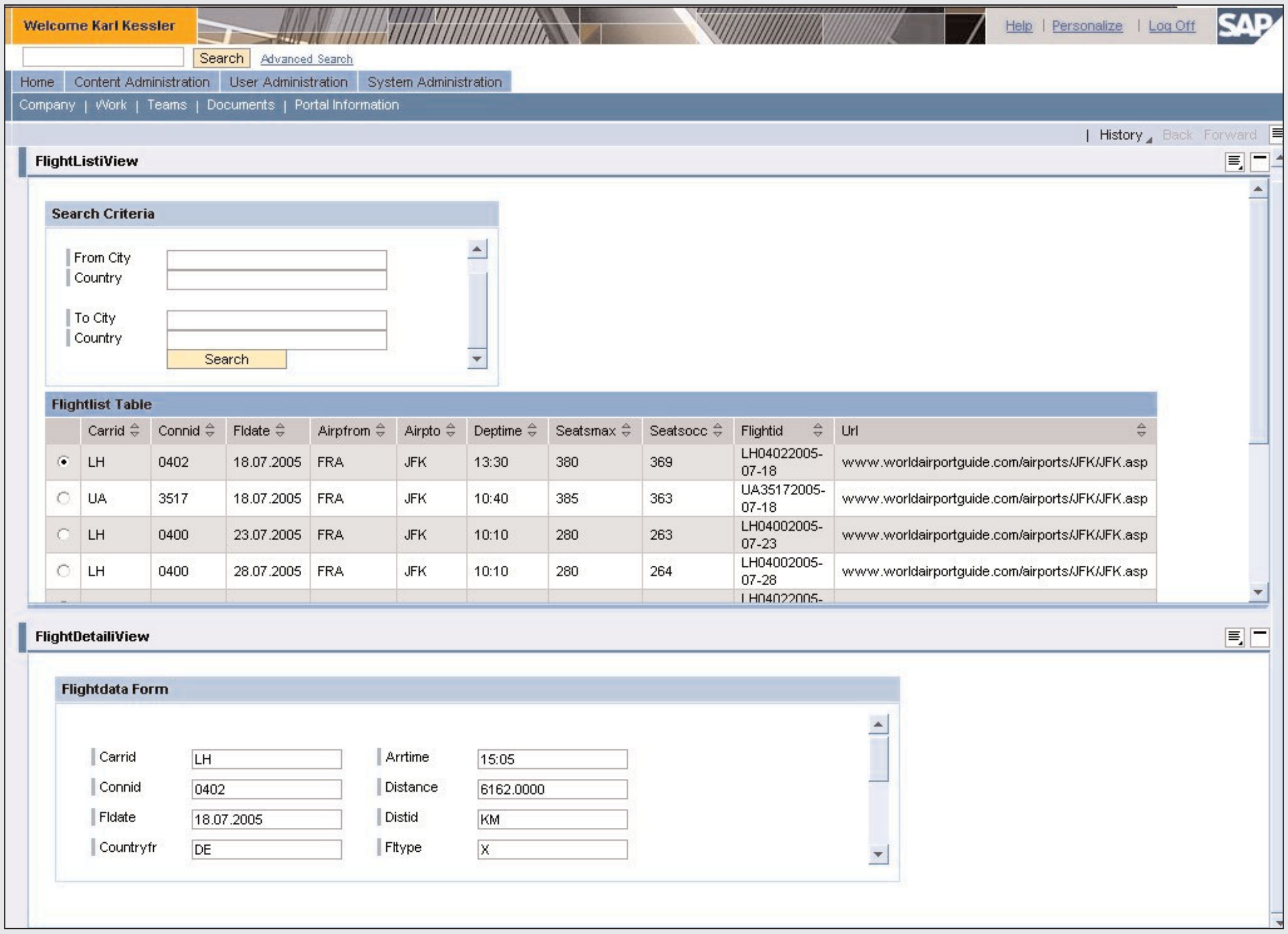


```
1> Generating code
2> Creating business package
3> Compiling portal components
4> Deploying to portal

Deployment completed
successfully!
```

Flightlistpage

Figure 20 Deploying the new page that contains the two iViews



From City	
Country	
To City	
Country	

	Carrid	Connid	Fldate	Airpfrom	Airpto	Deptime	Seatsmax	Seatsocc	Flightid	Url
●	LH	0402	18.07.2005	FRA	JFK	13:30	380	369	LH04022005-07-18	www.worldairportguide.com/airports/JFK/JFK.asp
○	UA	3517	18.07.2005	FRA	JFK	10:40	385	363	UA35172005-07-18	www.worldairportguide.com/airports/JFK/JFK.asp
○	LH	0400	23.07.2005	FRA	JFK	10:10	280	263	LH04002005-07-23	www.worldairportguide.com/airports/JFK/JFK.asp
○	LH	0400	28.07.2005	FRA	JFK	10:10	280	264	LH04002005-07-28	www.worldairportguide.com/airports/JFK/JFK.asp
									LH04022005-	

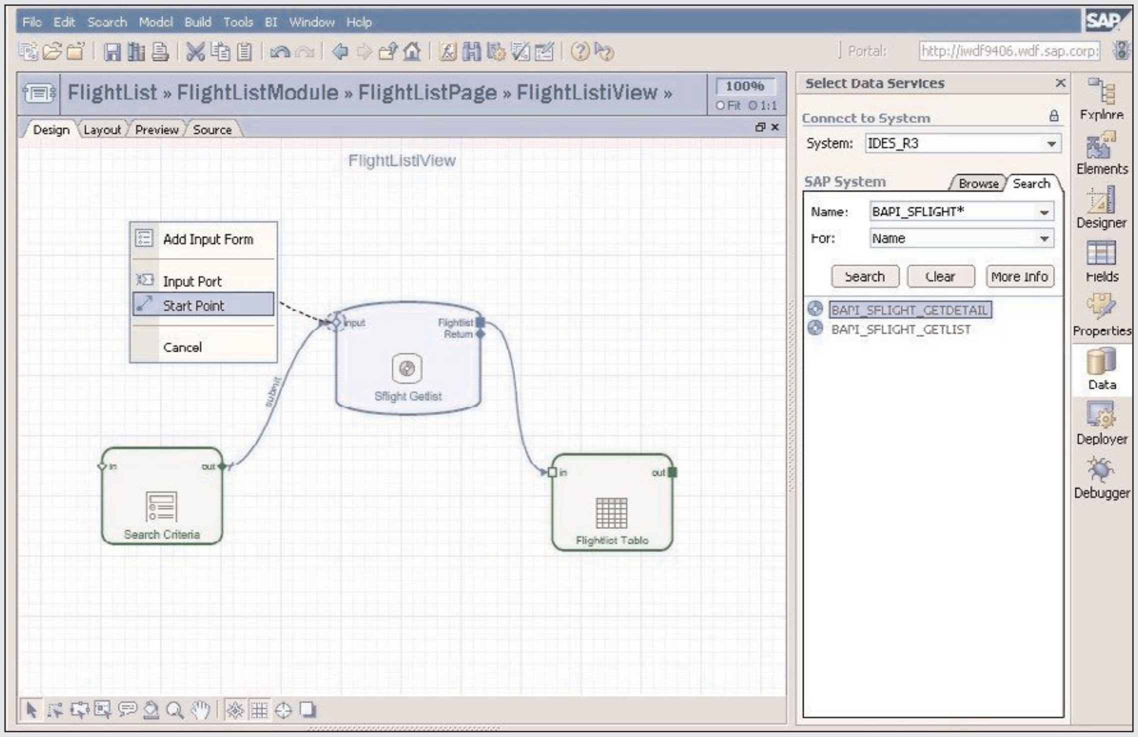
Carrid	LH	Arrtime	15:05
Connid	0402	Distance	6162.0000
Fldate	18.07.2005	Distid	KM
Countryfr	DE	Fltype	X

Figure 21 FlightDetailiView displaying search results

Be sure to check the iViews before deploying the page. Note that after successful deployment, an item — the FlightListPage component — is created. You can use this component to launch the complete portal page, including both iViews. Deploying a page is analogous to deploying an iView. Click on the Deployer tool on the task panel toolbar, check the syntax for errors, and follow the prompts.

Figure 21 shows the results. At the top of the screen, the iView displays the flights based on the entered search criteria. Whenever a flight is selected from the FlightListiView, the FlightDetailiView automatically updates itself to display the related detail information.

What we have just seen is an easy way to connect individual iViews, thereby reducing the size and complexity associated with a single, large iView. As you continue to build iViews, you will inevitably come up with other ways to take advantage of this technique.



System: IDES_R3

Name: BAPI_SFLIGHT*

For: Name

BAPI_SFLIGHT_GETDETAIL

BAPI_SFLIGHT_GETLIST

Figure 22 Adding a start point to supply input defaults for BAPI_SFLIGHT_GETLIST

> ***Note!***
>
> Although Figure 21 looks very similar to placing both views within the same iView, separating them means that the user can customize each iView individually (i.e., each has its own title bar). For example, you could collapse the FlightDetailiView if you were not interested in the particular flight details and just display the flight list instead.

Other ways to improve iView usability

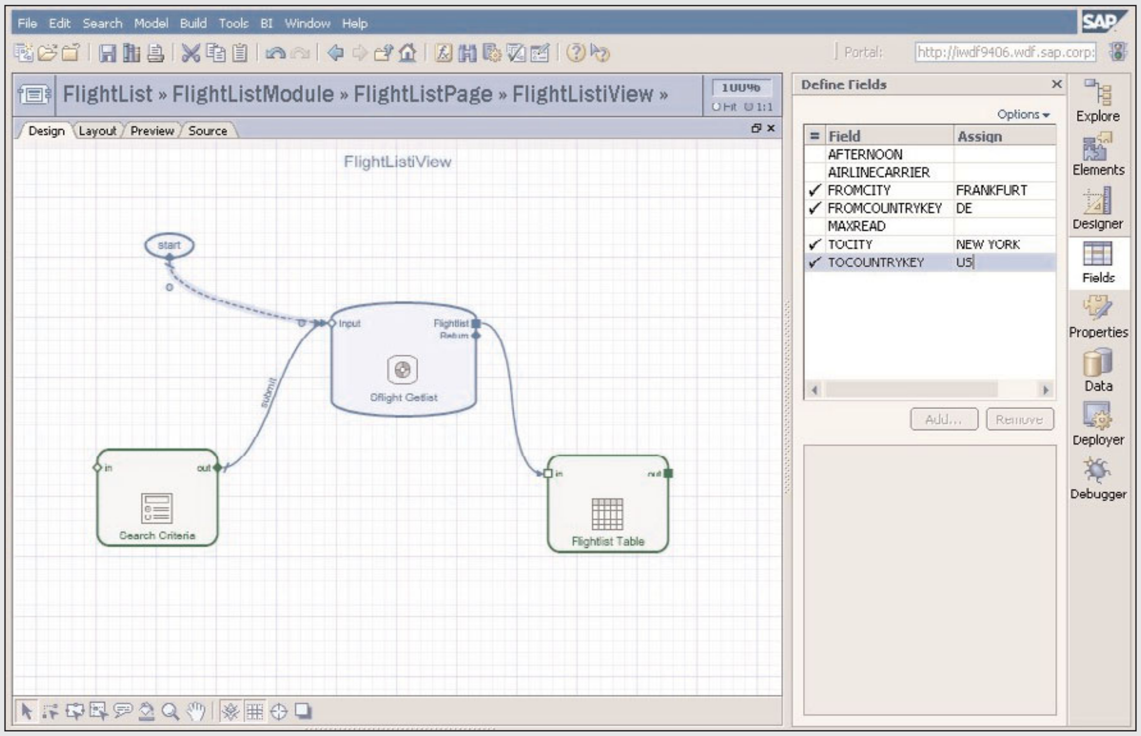
As long as we're in the neighborhood, there are a few other quick techniques I want to bring to your attention to improve the usability of your iViews.

Adding default field values for input forms

Typing values into input forms takes a lot of time, and can seriously degrade user efficiency. So anything you can do to hasten data entry will significantly increase productivity.

One easy thing you can do is supply default values for your input forms. Visual Composer lets you do this by defining a "start point." Start points are bound to the input port of particular function modules and call the BAPI directly without user interaction. You can combine a start point with an input form. In this case, the BAPI is called with the initial values, but the user can supply different values later on once the initial call is executed.

As you can see in **Figure 22**, I've added a start point for BAPI_SFLIGHT_GETLIST. For our flight example, we can supply default values for the



	Field	Assign
	AFTERNOON	
	AIRLINECARRIER	
✓	FROMCITY	FRANKFURT
✓	FROMCOUNTRYKEY	DE
	MAXREAD	
✓	TOCITY	NEW YORK
✓	TOCOUNTRYKEY	US

Figure 23 Defining the default values for the start point

mandatory input fields FROMCITY, FROMCOUNTRYKEY, TOCITY, and TOCOUNTRYKEY. These default values become the start point — that is, the values for these fields are automatically filled in when the iView is launched. To add a start point, right-click on the input port and then select Start Point on the context menu.

To specify the default values for each field, you map the input parameters of the BAPI to the values you would like to pass. As shown in **Figure 23**, the default value for the FROMCITY field is "FRANKFURT" and the default value for the TOCITY field is "NEW YORK."

Filtering, sorting, and aggregating data

Up to this point, you've seen how to display BAPI data in a table view (i.e., a grid format), and how to propose default values for input fields. Another technique to add to your toolbox for building professional-looking iViews is the ability to filter, presort, and group table results before they're displayed.

Visual Composer offers some powerful operators with which to easily do this:

- **Filter**, to narrow the range of rows displayed in the results table by specifying criteria that the resulting data must match
- **Sort**, to sort the results table by one or more fields in ascending or descending order
- **Sigma**, to aggregate data from a record set to a single record (e.g., using the SUM function to calculate totals, using the AVG function to calculate averages, using the COUNT function to determine the number of records)
- **Top**, to return a specified number of topmost records (e.g., 10 topmost sales regions)

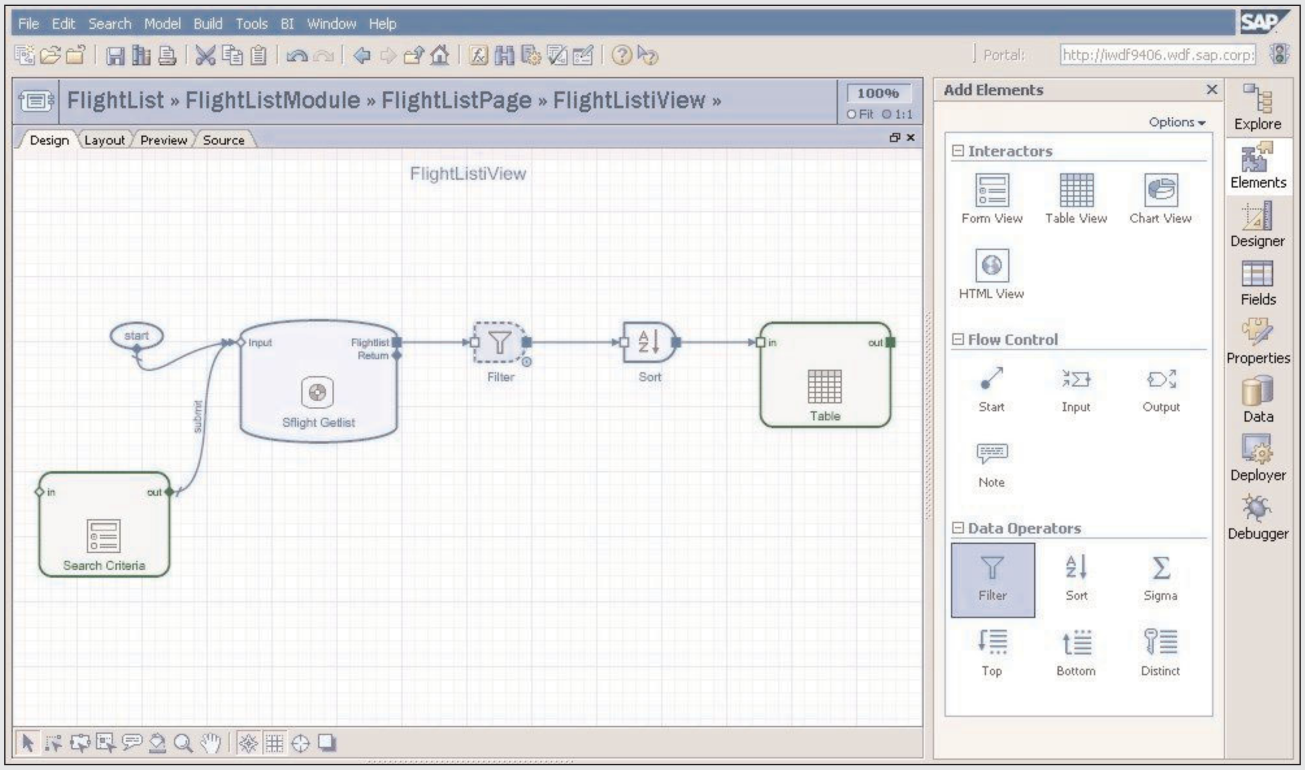


Figure 24 Filtering and sorting search results

- **Bottom**, to return a specified number of bottom-most records (e.g., 10 bottommost cost centers)
- **Distinct**, to eliminate duplicates

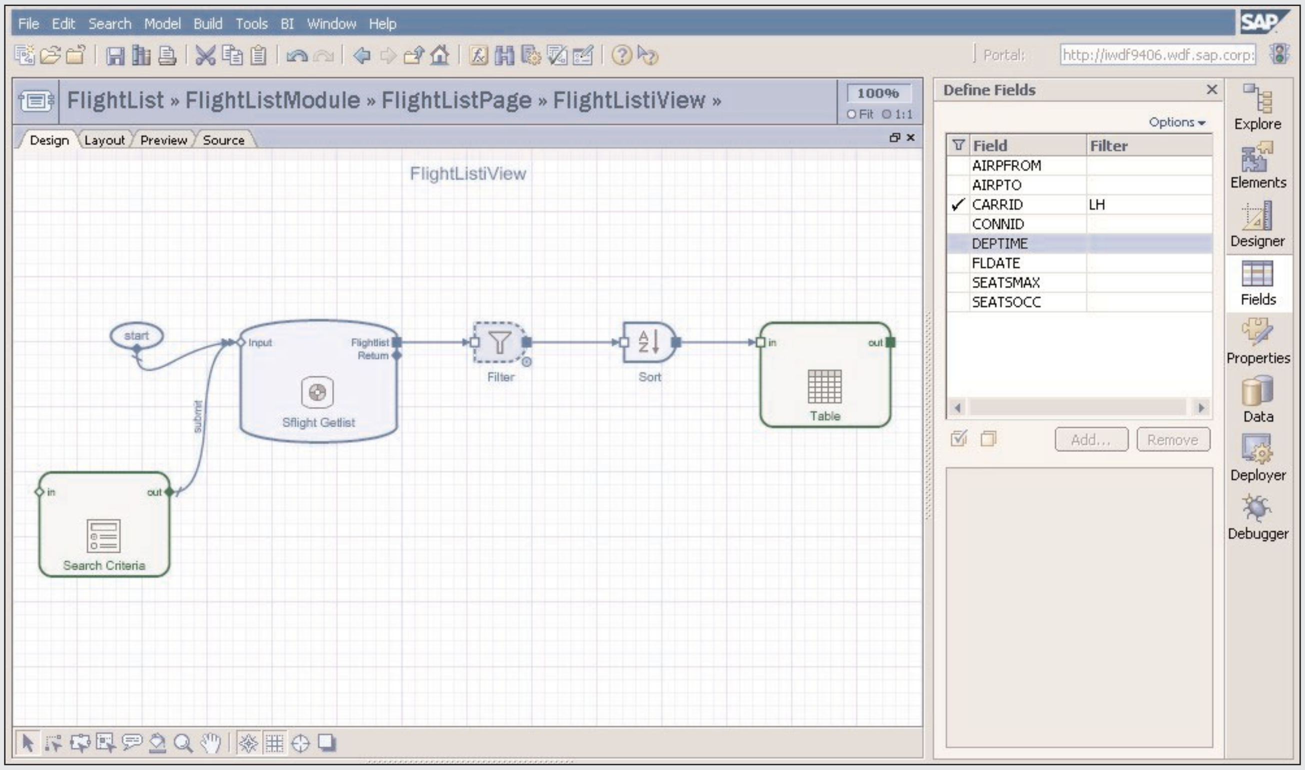
These operators are applied directly to the output port of a BAPI and can be applied alone or in combination. You simply drag the operator from the Data Operators section of the Add Elements task panel (refer back to Figure 8) onto the workspace, connect it to the output port of the BAPI element to which you want to apply the operator, and then fine-tune the properties of the operator in the Define Properties task panel. You can also "daisy chain" multiple operators together.

To demonstrate how this works, I've connected the Filter and Sort operators to the output port of BAPI_SFLIGHT_GETLIST in the FlightListiView (see **Figure 24**); therefore, before the data is actually displayed in the table control, it will be filtered and sorted.

Note!

Some BAPIs won't offer the specific selection criteria you need. For example, you may want to only retrieve orders that occurred within the last 10 days, but the BAPI might not let you select by date. In these cases, use a filter instead.

To filter or sort data, you need to specify the filter or sort criteria by selecting the operator and clicking on the Fields tool on the task panel toolbar. As shown in **Figure 25**, I've specified that the data be filtered to only pass through flight records with the CARRID value "LH." I also defined the sort criteria to list the flight dates in descending order, which is not shown here.



	Field	Filter
	AIRPFROM	
	AIRPTO	
✓	CARRID	LH
	CONNID	
	DEPTIME	
	FLDATE	
	SEATSMAX	
	SEATSOCC	

Figure 25 Specifying the filter criteria

Conclusion

In this second installment of a two-part article series on Visual Composer, we have explored some advanced techniques you can use to really bring your iViews to life: drill-down functionality, charts, HTML views, portal eventing, input field defaults, and BAPI output operators.

I concluded the first article by pointing out that Visual Composer can also be used to develop iViews that leverage data from SAP BW and non-SAP systems, not just OLTP systems like SAP R/3 and SAP CRM, with Web services support planned for an upcoming release. This alone is proof that we've only touched the tip of the proverbial iceberg as to what you can build with Visual Composer. For this reason, I encourage you to begin exploring Visual Composer yourself, and go beyond the techniques I've presented here. This article series has hopefully given you the inspiration and foundational knowledge you need to get started.

Tip!

For performance reasons, if you can narrow the results by hard-coding a BAPI input value, do this instead of applying a filter. The filter is applied after the data is selected and transmitted to the portal server over the network, so it is much slower than not selecting that data in the first place.

Looking ahead, Visual Composer will be extended to facilitate seamless integration between model-driven and code-based applications. For example, complex custom controls might be developed with SAP NetWeaver Developer Studio in Java and then imported into Visual Composer as new compound elements visible in the task panel. In addition, the design-time component will be ported to run on SAP Web AS instead of requiring Microsoft IIS and SQL Server. Finally, the output options of the tool will be expanded as well. Currently (as of SAP EP 6.0), Visual Composer generates HTMLB for the iView user interface. Additional output options — including Web Dynpro and Macromedia's Flex technology — are planned. Rest assured that these extensions will not negate any of your prior Visual Composer efforts; however, existing Visual Composer iViews will continue to work with little or no modification.