Take a fresh look at the redesigned SAP List Viewer in SAP NetWeaver ’04: Write programs to present tabular data in less time and with fewer lines of code

by Falko Schneider

Falko Schneider
SAP NetWeaver Business Intelligence, SAP AG

Falko Schneider is the Director of Research and Development for SAP NetWeaver Business Intelligence. He designed and developed the first version of the SAP List Viewer (ALV), which was first shipped to customers with R/3 4.0. During the development of R/3 4.6, he became the project lead for ALV development and was responsible for the control-based ALV variant, the ALV Grid Control. In 2000, Falko joined the SAP BW team as a development manager for ALV and SAP Query, and also assumed responsibility for Data Warehousing and integrating Business Planning functionality within SAP BW.

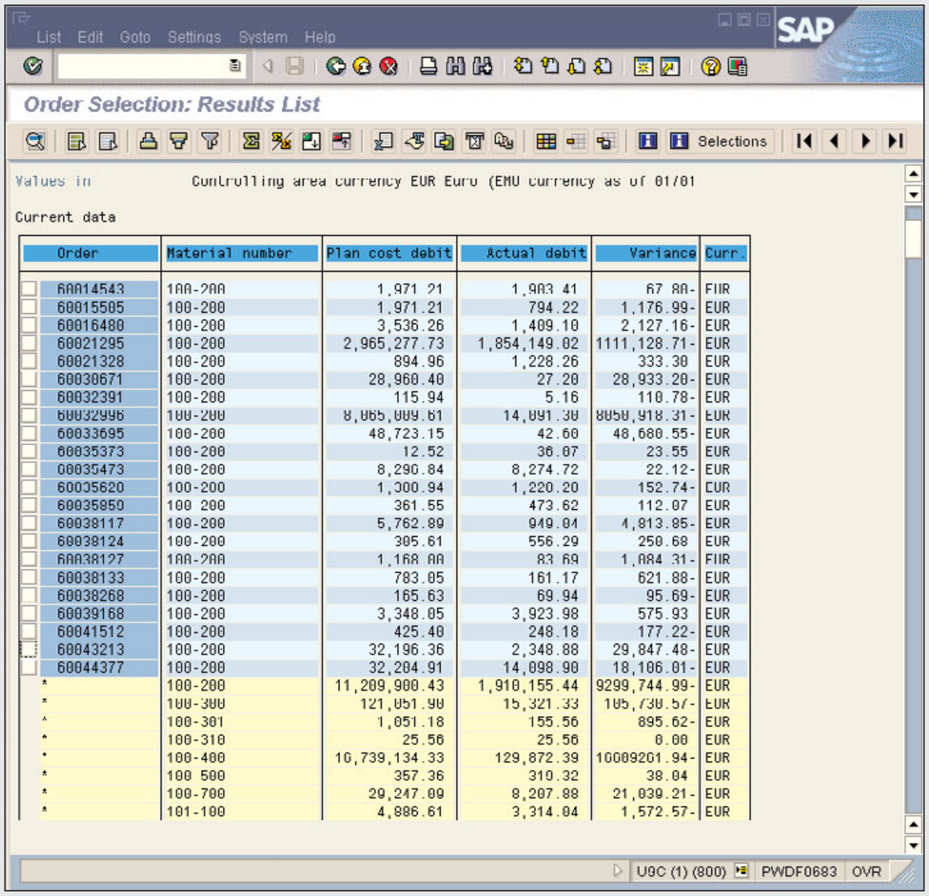
Conceptually, developing business applications can be thought of as a two-dimensional endeavor. You obviously need to implement the expected transactional tasks, such as data entry and data processing. In addition, you are often faced with the need to present tabular data with a common look and feel, as well as allow users to create views that reflect their particular needs. Data entry and data processing tasks are necessarily business-specific and therefore require specialized coding and functionality. However, the requirements for reporting on and interacting with tabular data are typically quite similar: I want to sort the data, I want to filter the data, I want to total a column, I want to use certain criteria to aggregate the data, and so on.

Most developers would agree that using an SAP-provided tool to implement generic reporting and interaction capabilities is a far more attractive option than coding the report layout and generic services such as sorting and totaling from scratch each time:

- Productivity would be significantly increased because you need to write less code and all tabular data automatically appears with a common look and feel within your application.
- Programs would be easier to maintain and support because you can write clearly structured programs that separate the business logic for retrieving the data from the tool-provided logic for the user interface and generic services.
- As an added bonus, you would also be able to reap the benefits of any SAP-internal enhancements such as performance improvements and memory optimization.

The good news is that you now have such a tool in the SAP List Viewer (ALV).[1] SAP NetWeaver ’04 provides a completely redesigned API for the

[1] The name *SAP List Viewer* was introduced in SAP R/3 Release 4.6 to replace the formerly used name *ABAP List Viewer*. The original *ALV* acronym has been retained for continuity.



Order Selection: Results List

```
Values in           Controlling area currency EUR Euro (EMU currency as of 01/01

Current data
```

Order	Material number	Plan cost debit	Actual debit	Variance	Curr.
60014543	100-200	1,971.21	1,903.41	67.80-	EUR
60015505	100-200	1,971.21	794.22	1,176.99-	EUR
60016480	100-200	3,536.26	1,409.10	2,127.16-	EUR
60021295	100-200	2,965,277.73	1,854,149.02	1111,128.71-	EUR
60021328	100-200	894.96	1,228.26	333.30	EUR
60030671	100-200	28,960.40	27.20	28,933.20-	EUR
60032391	100-200	115.94	5.16	110.78-	EUR
60032996	100-200	8,065,009.61	14,091.30	8050,918.31-	EUR
60033695	100-200	48,723.15	42.60	48,680.55-	EUR
60035373	100-200	12.52	36.07	23.55	EUR
60035473	100-200	8,296.84	8,274.72	22.12-	EUR
60035620	100-200	1,300.94	1,220.20	152.74-	EUR
60035850	100 200	361.55	473.62	112.07	EUR
60038117	100-200	5,762.89	949.04	4,813.85-	EUR
60038124	100-200	305.61	556.29	250.68	EUR
60038127	100-200	1,168.00	83.69	1,084.31-	EUR
60038133	100-200	783.05	161.17	621.88-	EUR
60038268	100-200	165.63	69.94	95.69-	EUR
60039168	100-200	3,348.05	3,923.98	575.93	EUR
60041512	100-200	425.40	248.18	177.22-	EUR
60043213	100-200	32,196.36	2,348.88	29,847.48-	EUR
60044377	100-200	32,204.91	14,098.90	18,106.01-	EUR
*	100-200	11,209,900.43	1,910,155.44	9299,744.99-	EUR
*	100-300	121,051.90	15,321.33	105,730.57-	EUR
*	100-301	1,051.18	155.56	895.62-	EUR
*	100-310	25.56	25.56	0.00	EUR
*	100-400	10,739,134.33	129,872.39	10609261.94-	EUR
*	100 500	357.36	319.32	38.04	EUR
*	100-700	29,247.09	8,207.88	21,039.21-	EUR
*	101-100	4,886.61	3,314.04	1,572.57-	EUR

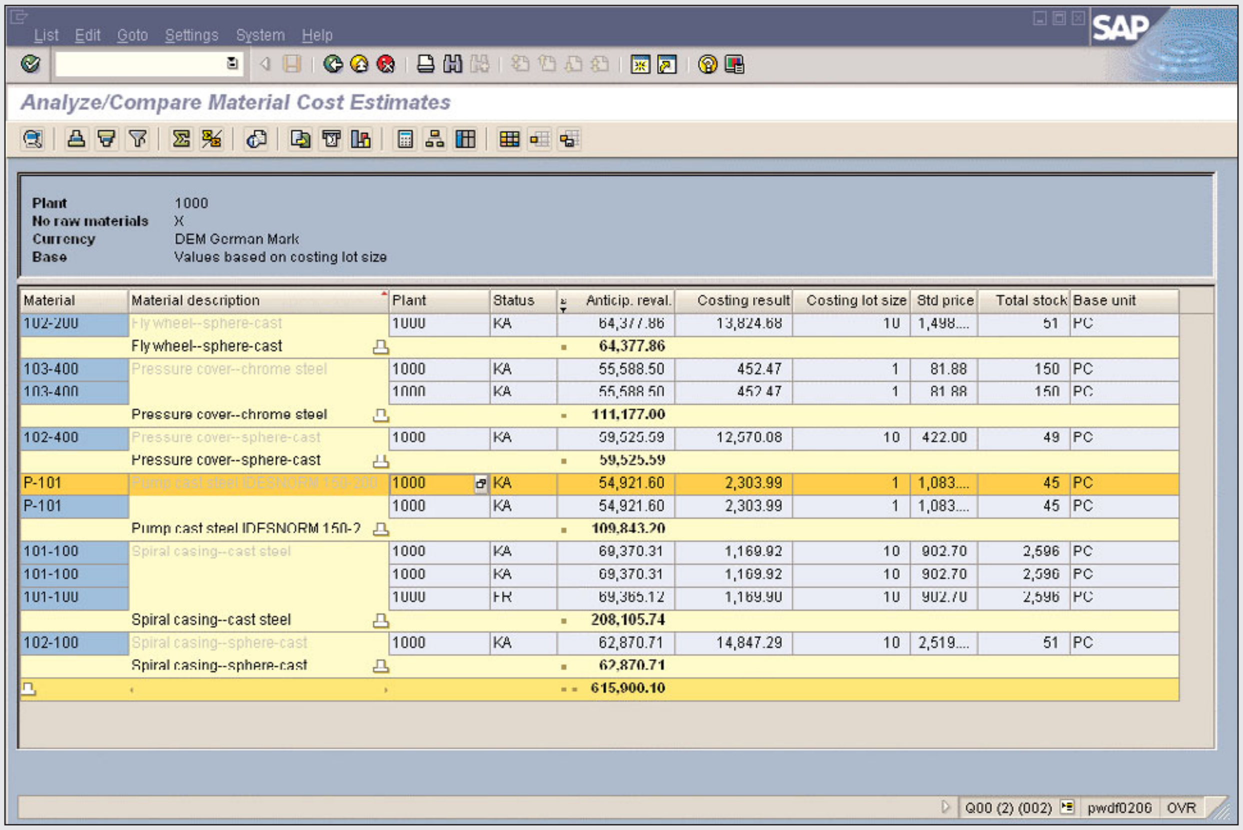
Figure 1 Example report with a simple table using the ABAP list processor (Classic ALV)

SAP List Viewer, which is now fully integrated and officially released for customer use. The goal of this redesign was to offer an object-oriented, robust, and unified object model for presenting tabular data to improve developer productivity. While previous versions of the SAP List Viewer included a separate API for each format variant, the new API provides a single object model for all variants. It is based on the ABAP Objects programming language and uses the SAPGUI technology to display the data.

Whether you are already familiar with the previously available APIs or have just recently discovered the SAP List Viewer, this article shows how the SAP List Viewer can help meet your reporting needs for presenting tabular data in your applications. I start by describing the presentation options that are available based on how the data to be reported is structured. I then introduce the new SAP List Viewer programming model, including an overview of its overall structure and components. Finally, I show how easy it is to use the redesigned API by walking through some simple code examples to get you started.

Options for displaying data in the SAP List Viewer

Before we look at the SAP List Viewer architecture and how to program its API, let's take a brief tour of the display format choices. The SAP List Viewer


List Edit Goto Settings System Help
SAP
Q00 (2) (002) pwdf0206 OVR

Analyze/Compare Material Cost Estimates

Plant	1000
No raw materials	X
Currency	DEM German Mark
Base	Values based on costing lot size

Material	Material description	Plant	Status	Anticip. reval.	Costing result	Costing lot size	Std price	Total stock	Base unit
102-200	Fly wheel--sphere-cast	1000	KA	64,377.86	13,824.68	10	1,498....	51	PC
	Fly wheel--sphere-cast			**64,377.86**					
103-400	Pressure cover--chrome steel	1000	KA	55,588.50	452.47	1	81.88	150	PC
103-400		1000	KA	55,588.50	452.47	1	81.88	150	PC
	Pressure cover--chrome steel			**111,177.00**					
102-400	Pressure cover--sphere-cast	1000	KA	59,525.59	12,570.08	10	422.00	49	PC
	Pressure cover--sphere-cast			**59,525.59**					
P-101	Pump cast steel IDESNORM 150-200	1000	KA	54,921.60	2,303.99	1	1,083....	45	PC
P-101		1000	KA	54,921.60	2,303.99	1	1,083....	45	PC
	Pump cast steel IDESNORM 150-2			**109,843.20**					
101-100	Spiral casing--cast steel	1000	KA	69,370.31	1,169.92	10	902.70	2,596	PC
101-100		1000	KA	69,370.31	1,169.92	10	902.70	2,596	PC
101-100		1000	FR	69,365.12	1,169.90	10	902.70	2,596	PC
	Spiral casing--cast steel			**208,105.74**					
102-100	Spiral casing--sphere-cast	1000	KA	62,870.71	14,847.29	10	2,519....	51	PC
	Spiral casing--sphere-cast			**62,870.71**					
				615,900.10					

Figure 2 Example report with a grid control in full-screen mode

supports a set of predefined presentation options (or *variants*) based on the structure of the underlying data to be displayed and the desired screen appearance. Therefore, your first step is to determine the type of data that you want to display:

- Non-hierarchical data
- Two-level hierarchical data
- Multi-level hierarchical data

You then implement the code to call the API. This code describes how to present the data and also contains any specialized functions that you want users to be able to execute within the report.

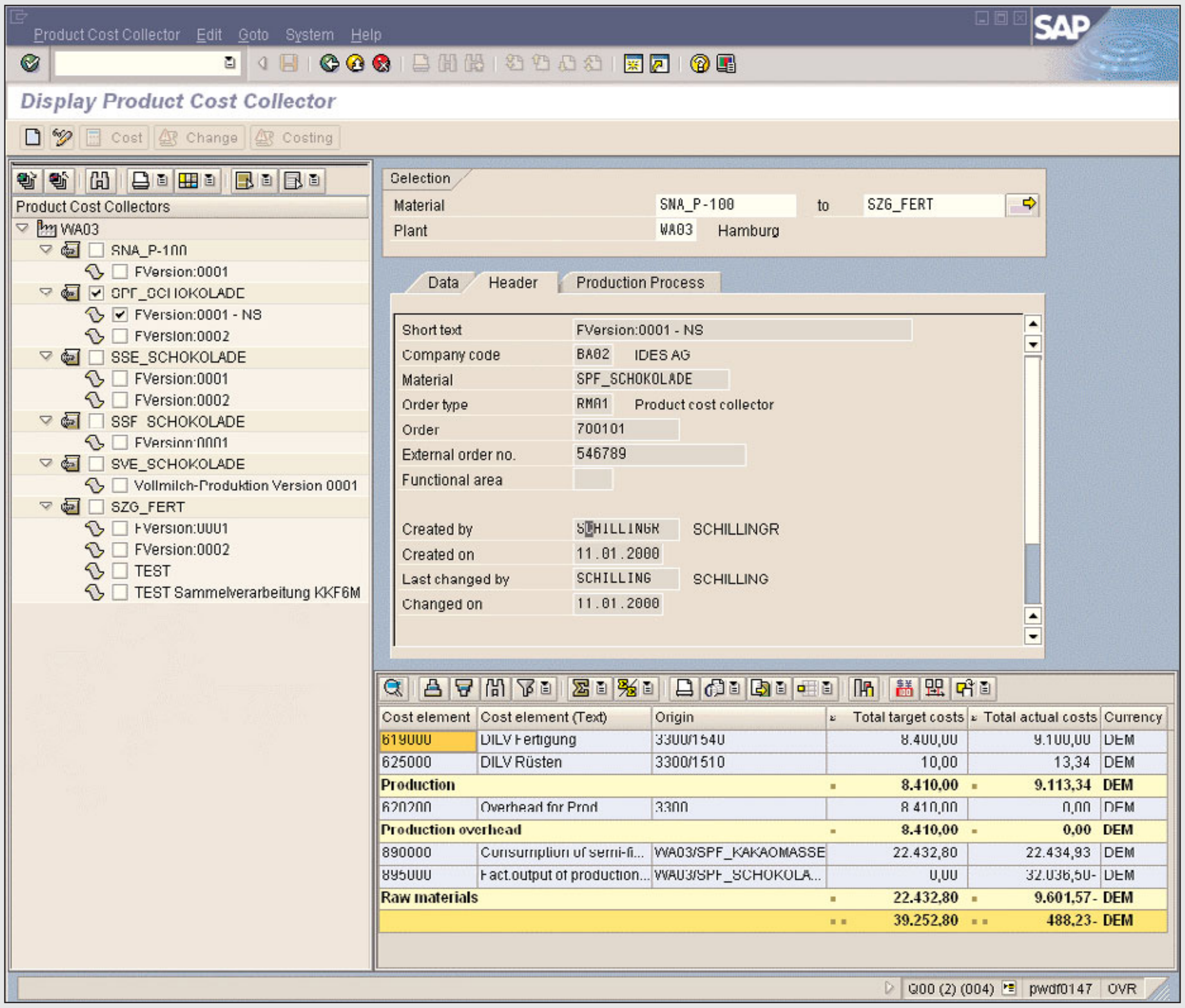
Displaying non-hierarchical data

The simplest SAP List Viewer variant is for displaying a table of data that is not hierarchically structured, which means that the records do not have any parent-child relationships. To implement this variant in your program, you simply select the data to be displayed in an internal table and pass that internal table to the SAP List Viewer API. You have several options for displaying the results:

- A simple table using the ABAP list processor,[2] as you can see in **Figure 1**. This variant of the SAP List Viewer is also referred to as *Classic ALV*.
- A grid control[3] in full-screen mode using the SAP Control Framework, which is shown in **Figure 2**.

[2] The ABAP list processor displays a list defined in an ABAP program using standard ABAP `WRITE` statements.

[3] The grid control is also commonly referred to as the *ALV grid control*. For more information, see the article “A Developer’s Guide to the New ALV Grid Control” by Jens Stumpe (*SAP Professional Journal*, November/December 2000).


Product Cost Collector Edit Goto System Help
SAP
Display Product Cost Collector
Cost Change Costing
Selection
Data Header Production Process

Product Cost Collectors
- WA03
  - SNA_P-100
    - FVersion:0001
  - SPF_SCHOKOLADE
    - FVersion:0001 - NS
    - FVersion:0002
  - SSE_SCHOKOLADE
    - FVersion:0001
    - FVersion:0002
  - SSF_SCHOKOLADE
    - FVersion:0001
  - SVE_SCHOKOLADE
    - Vollmilch-Produktion Version 0001
  - SZG_FERT
    - FVersion:0001
    - FVersion:0002
    - TEST
    - TEST Sammelverarbeitung KKF6M

Material	SNA_P-100	to	SZG_FERT
Plant	WA03	Hamburg	

Short text	FVersion:0001 - NS	
Company code	BA02	IDES AG
Material	SPF_SCHOKOLADE	
Order type	RMA1	Product cost collector
Order	700101	
External order no.	546789	
Functional area		
Created by	SCHILLINGR	SCHILLINGR
Created on	11.01.2000	
Last changed by	SCHILLING	SCHILLING
Changed on	11.01.2000	

Cost element	Cost element (Text)	Origin		Total target costs		Total actual costs	Currency
619000	DILV Fertigung	3300/1540		8.400,00		9.100,00	DEM
625000	DILV Rüsten	3300/1510		10,00		13,34	DEM
Production			■	**8.410,00**	■	**9.113,34**	**DEM**
620200	Overhead for Prod	3300		8.410,00		0,00	DEM
Production overhead			■	**8.410,00**	■	**0,00**	**DEM**
890000	Consumption of semi-fi...	WA03/SPF_KAKAOMASSE		22.432,80		22.434,93	DEM
895000	Fact.output of production...	WA03/SPF_SCHOKOLA...		0,00		32.036,50-	DEM
Raw materials			■	**22.432,80**	■	**9.601,57-**	**DEM**
			■■	**39.252,80**	■■	**488,23-**	**DEM**

Q00 (2) (004) pwdf0147 OVR

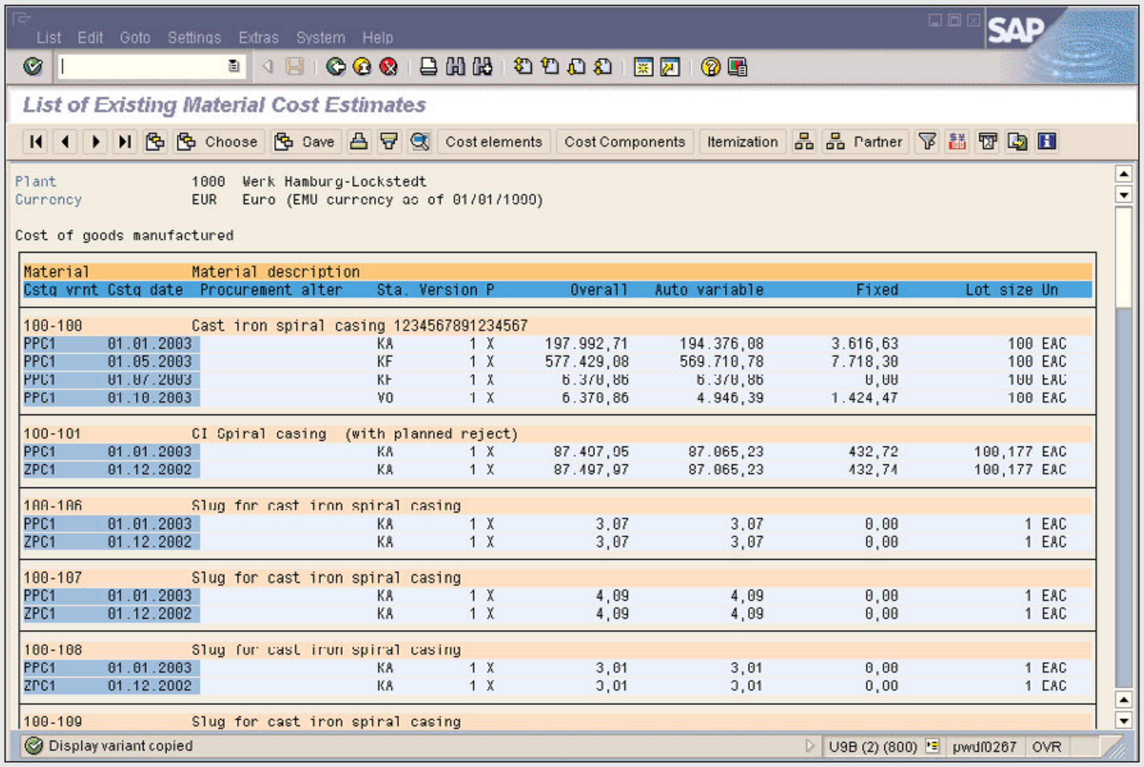
Figure 3 Example transaction with a grid control embedded in a Dynpro container

- A grid control embedded in a Dynpro container using the SAP Control Framework, which appears in **Figure 3**.

Using the SAP Control Framework lends a modern look-and-feel to a report by supporting common controls such as drag-and-drop. But for all new development, SAP strongly recommends using the grid control option (either in full-screen or embedded mode) rather than the ABAP list processor option for another important reason: It offers a huge advantage with respect to performance and memory consumption due to significant improvements made in SAP NetWeaver '04.

Displaying two-level hierarchical data

The SAP List Viewer also supports a variant known as a *hierarchical sequential list* for displaying data that is based on a hierarchy that has *exactly* two levels. For example, you might have an order header with *n* order items that belong to it. To implement this variant, you pass the header and detail data to the SAP List Viewer API in two separate internal tables. You specify the relationship between these data tables by naming the foreign key relationship as a method parameter when calling the API. I explain more about the factory methods that are provided for this purpose later in the article.



```
Plant              1000  Werk Hamburg-Lockstedt
Currency           EUR   Euro (EMU currency as of 01/01/1000)

Cost of goods manufactured

Material           Material description
Cstg vrnt Cstg date  Procurement alter     Sta. Version P      Overall   Auto variable        Fixed      Lot size Un

100-100            Cast iron spiral casing 1234567891234567
PPC1     01.01.2003                       KA      1 X     197.992,71     194.376,08     3.616,63          100 EAC
PPC1     01.05.2003                       KF      1 X     577.429,08     569.710,78     7.718,30          100 EAC
PPC1     01.07.2003                       KF      1 X       6.370,86       6.370,86         0,00          100 EAC
PPC1     01.10.2003                       VO      1 X       6.370,86       4.946,39     1.424,47          100 EAC

100-101            CI Spiral casing  (with planned reject)
PPC1     01.01.2003                       KA      1 X      87.407,05      87.065,23       432,72      100,177 EAC
ZPC1     01.12.2002                       KA      1 X      87.497,97      87.065,23       432,74      100,177 EAC

100-106            Slug for cast iron spiral casing
PPC1     01.01.2003                       KA      1 X           3,07           3,07         0,00            1 EAC
ZPC1     01.12.2002                       KA      1 X           3,07           3,07         0,00            1 EAC

100-107            Slug for cast iron spiral casing
PPC1     01.01.2003                       KA      1 X           4,09           4,09         0,00            1 EAC
ZPC1     01.12.2002                       KA      1 X           4,09           4,09         0,00            1 EAC

100-108            Slug for cast iron spiral casing
PPC1     01.01.2003                       KA      1 X           3,01           3,01         0,00            1 EAC
ZPC1     01.12.2002                       KA      1 X           3,01           3,01         0,00            1 EAC

100-109            Slug for cast iron spiral casing
```

Figure 4 Example report with a hierarchical sequential list

Figure 4 illustrates how two-level hierarchical data is displayed. Note that the detail data (in this example, Costing variant or Costing date) appears directly below its associated header data (in this example, Material and Material description).

Displaying multi-level hierarchical data

The other SAP List Viewer variant for presenting hierarchically structured data supports the display of an arbitrary number of levels. When using this variant,

Why you should consider using the SAP List Viewer

While primarily a development tool, the SAP List Viewer offers significant productivity advantages for developers and users alike.

Starting with the developer's perspective, the SAP List Viewer is a powerful programming interface that you can supply with any type of tabular data for display to users. It also provides generic services that you can integrate directly into an application context. These services cover the complete spectrum, from actions such as sorting or filtering that can be performed on any type of tabular data to reporting-related

Continues on next page

***Sidebar** continued*

capabilities such as aggregation or graphical display options. You can combine these generic services with application-specific functionality that is performed on the displayed data but controlled by the application. Your job is significantly easier because you simply incorporate the generic services in your applications without writing the code to implement them. The SAP List Viewer also handles report formatting and layout. Consequently, you can focus on developing the business logic to retrieve the data and the application-specific functionality to further process the data.

Using the SAP List Viewer not only speeds up development, but also makes your code more manageable and future-proof. In keeping with good programming practices, you can easily separate the business logic from the report user interface logic in your programs. This structure is a prerequisite for being able to reuse the business logic, as well as for reserving the option to replace the presentation layer with a Web-based UI at some point in the future.

Now let's turn to the user's perspective. Instead of each application (and perhaps report) reflecting a different presentation style as determined by the developer, data displayed with the SAP List Viewer looks and feels the same in all applications that use its API. Data is presented in a state-of-the-art, standardized, easy-to-use user interface that has a consistent appearance and familiar controls.

As a result, the developers who write the programs and the users who work with the results of the programs are all more productive. These factors help to reduce your total cost of ownership for an application in terms of user training time, amount of development effort, and code supportability and maintainability.

you pass the data to the SAP List Viewer API in one internal table and, in a separate internal table, pass the metadata that describes the parent-child relationship of the hierarchy nodes. You have two options for displaying the results:

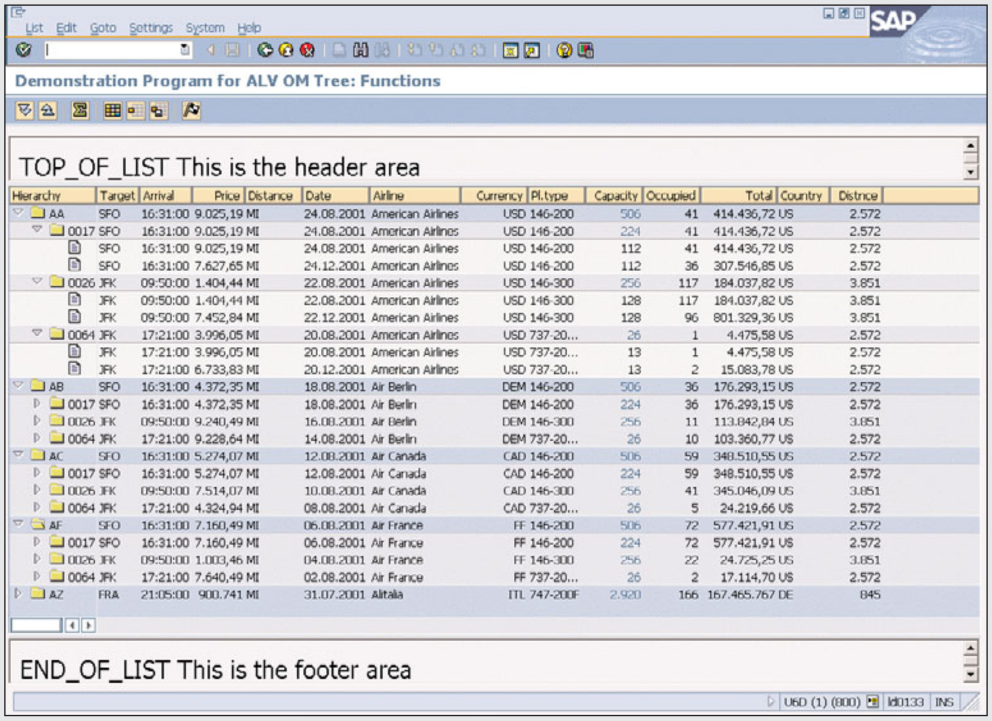
- A tree control[4] in full-screen mode using the SAP Control Framework, as shown in **Figure 5**. This new presentation option was added in SAP NetWeaver '04. In this example, you can see that the multi-level hierarchy consists of the carrier ID on level 1 (AA, AB, AC), the flight number on level 2 (0017, 0026, 0064), and the individual flight bookings on level 3.
- A tree control that is embedded in a Dynpro container, which you can see in **Figure 6**.

[4] The tree control is based on the column tree control, which is a SAPGUI control that represents a tree-table combination. The `CL_GUI_COLUMN_TREE` class is the ABAP object wrapper for the column tree control. The API for this SAP List Viewer variant uses this class and enhances the column tree control with additional services, such as defining which columns should be displayed or totaled or whether to print the tree representation.

In both cases, the columns must display the same attributes/fields for all levels of the hierarchy. For example, in Figure 6 the Quantity, Unit, and Materials attributes (as displayed in columns two, three, and five) are valid for every hierarchy level. For higher levels in the hierarchy, the Value column also shows the total accumulated from any lower-level nodes.

Note!

You cannot use the SAP List Viewer to present different structures on more than two hierarchy levels. The tree control always requires the same structure with respect to the attributes shown in the table. The hierarchical sequential list can accommodate different structures, although only for exactly two hierarchy levels.



Demonstration Program for ALV OM Tree: Functions

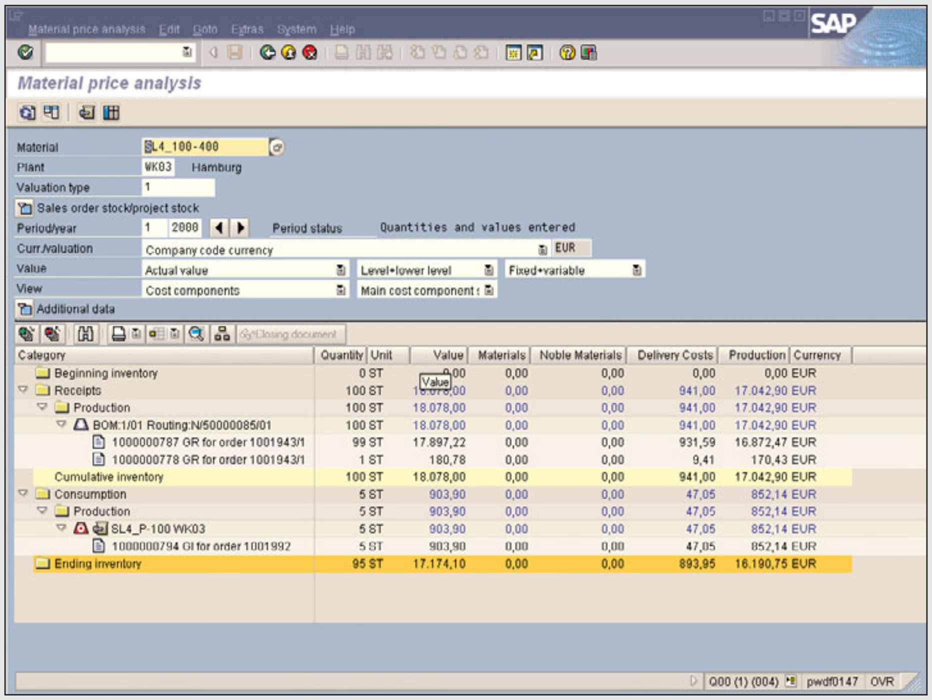
TOP_OF_LIST This is the header area

Hierarchy	Target	Arrival	Price	Distance	Date	Airline	Currency	Pl.type	Capacity	Occupied	Total	Country	Distnce
AA	SFO	16:31:00	9.025,19	MI	24.08.2001	American Airlines	USD	146-200	506	41	414.436,72	US	2.572
0017	SFO	16:31:00	9.025,19	MI	24.08.2001	American Airlines	USD	146-200	224	41	414.436,72	US	2.572
	SFO	16:31:00	9.025,19	MI	24.08.2001	American Airlines	USD	146-200	112	41	414.436,72	US	2.572
	SFO	16:31:00	7.627,65	MI	24.12.2001	American Airlines	USD	146-200	112	36	307.546,85	US	2.572
0026	JFK	09:50:00	1.404,44	MI	22.08.2001	American Airlines	USD	146-300	256	117	184.037,82	US	3.851
	JFK	09:50:00	1.404,44	MI	22.08.2001	American Airlines	USD	146-300	128	117	184.037,82	US	3.851
	JFK	09:50:00	7.452,84	MI	22.12.2001	American Airlines	USD	146-300	128	96	801.329,36	US	3.851
0064	JFK	17:21:00	3.996,05	MI	20.08.2001	American Airlines	USD	737-20...	26	1	4.475,58	US	2.572
	JFK	17:21:00	3.996,05	MI	20.08.2001	American Airlines	USD	737-20...	13	1	4.475,58	US	2.572
	JFK	17:21:00	6.733,83	MI	20.12.2001	American Airlines	USD	737-20...	13	2	15.083,78	US	2.572
AB	SFO	16:31:00	4.372,35	MI	18.08.2001	Air Berlin	DEM	146-200	506	36	176.293,15	US	2.572
0017	SFO	16:31:00	4.372,35	MI	18.08.2001	Air Berlin	DEM	146-200	224	36	176.293,15	US	2.572
0026	JFK	09:50:00	9.240,49	MI	16.08.2001	Air Berlin	DEM	146-300	256	11	113.842,84	US	3.851
0064	JFK	17:21:00	9.228,64	MI	14.08.2001	Air Berlin	DEM	737-20...	26	10	103.360,77	US	2.572
AC	SFO	16:31:00	5.274,07	MI	12.08.2001	Air Canada	CAD	146-200	506	59	348.510,55	US	2.572
0017	SFO	16:31:00	5.274,07	MI	12.08.2001	Air Canada	CAD	146-200	224	59	348.510,55	US	2.572
0026	JFK	09:50:00	7.514,07	MI	10.08.2001	Air Canada	CAD	146-300	256	41	345.046,09	US	3.851
0064	JFK	17:21:00	4.324,94	MI	08.08.2001	Air Canada	CAD	737-20...	26	5	24.219,66	US	2.572
AF	SFO	16:31:00	7.160,49	MI	06.08.2001	Air France	FF	146-200	506	72	577.421,91	US	2.572
0017	SFO	16:31:00	7.160,49	MI	06.08.2001	Air France	FF	146-200	224	72	577.421,91	US	2.572
0026	JFK	09:50:00	1.003,46	MI	04.08.2001	Air France	FF	146-300	256	22	24.725,25	US	3.851
0064	JFK	17:21:00	7.640,49	MI	02.08.2001	Air France	FF	737-20...	26	2	17.114,70	US	2.572
AZ	FRA	21:05:00	900.741	MI	31.07.2001	Alitalia	ITL	747-200F	2.920	166	167.465.767	DE	845

END_OF_LIST This is the footer area

U6D (1) (800) ld0133 INS

Figure 5 Example report with a tree control in full-screen mode



Material price analysis

Material	SL4_100-400	
Plant	WK03	Hamburg
Valuation type	1	

Sales order stock/project stock

Period/year	1	2000	Period status	Quantities and values entered
Curr./valuation	Company code currency	EUR		
Value	Actual value	Level+lower level	Fixed+variable	
View	Cost components	Main cost component s		

Additional data

Category	Quantity	Unit	Value	Materials	Noble Materials	Delivery Costs	Production	Currency
Beginning inventory	0	ST	[illegible]	0,00	0,00	0,00	0,00	EUR
Receipts	100	ST	[illegible]	0,00	0,00	941,00	17.042,90	EUR
Production	100	ST	18.078,00	0,00	0,00	941,00	17.042,90	EUR
BOM:1/01 Routing:N/50000085/01	100	ST	18.078,00	0,00	0,00	941,00	17.042,90	EUR
1000000787 GR for order 1001943/1	99	ST	17.897,22	0,00	0,00	931,59	16.872,47	EUR
1000000778 GR for order 1001943/1	1	ST	180,78	0,00	0,00	9,41	170,43	EUR
Cumulative inventory	100	ST	18.078,00	0,00	0,00	941,00	17.042,90	EUR
Consumption	5	ST	903,90	0,00	0,00	47,05	852,14	EUR
Production	5	ST	903,90	0,00	0,00	47,05	852,14	EUR
SL4_P-100 WK03	5	ST	903,90	0,00	0,00	47,05	852,14	EUR
1000000794 GI for order 1001992	5	ST	903,90	0,00	0,00	47,05	852,14	EUR
Ending inventory	95	ST	17.174,10	0,00	0,00	893,95	16.190,75	EUR

Q00 (1) (004) pwdf0147 OVR

Figure 6 Example transaction with a tree control embedded in a Dynpro container

Feature	No hierarchy			Two-level hierarchy	Multi-level hierarchy	
	Classic ALV	ALV grid control (full-screen)	ALV grid control (Dynpro container)	Hierarchical sequential list	ALV tree control (full-screen)	ALV tree control (Dynpro container)
Sorting						
Sort by up to nine criteria	X	X	X	X	–	–
Build totals	X	X	X	X	X	X
Build subtotals	X	X	X	X	–	–
Expand/compress lines or sub-trees	X	X	X	X	X	X
Build mean value, maximum, minimum	X	X	X	–	X	X
Screening						
Filter	X	X	X	X	–	–
Delete filter	X	X	X	X	–	–
Search	X	X	X	X	X	X
Layout management						
Hide/unhide columns	X	X	X	X	X	X
Change layout	X	X	X	X	X	X
Save layout	X	X	X	X	X	X
Choose layout	X	X	X	X	X	X
Set default layout	X	X	X	X	X	X
Exporting						
Excel inplace	–	X	X	–	–	–
Direct export to Excel	X	X	X	–	–	–
Download as file	X	X	X	X	–	–
Mail	X	X	X	X	–	–
Export to Lotus	X	X	X	–	–	–
Printing						
Print preview	–	X	X	–	X	X
Print	X	X	X	X	X	X
Top of page	X	X	–	X	X	–
End of page	X	X	–	X	X	–
Automatic printing of pages, dates	–	X	X	–	–	–
Column headers in two lines	X	–	-–	X	–	–
Specialty functions						
Application-specific functions	X	X	X	X	X	X
Graphical display	–	X	X	–	–	–
ABC analysis*	–	X	X	–	–	–

* ABC analysis is a process for determining the importance of an object. You perform an ABC analysis to classify objects according to specific criteria or performance measures. For example, an object can be a material, a vendor, or a plant. Each object is assigned an indicator of A (important), B (less important), or C (relatively unimportant).

Figure 7 Standard services that are available with the SAP List Viewer

Figure 7 provides an overview of the generic user interaction features that are available in the SAP List Viewer for each variant. You can optionally enable any of these standard services in your programs quickly and easily, without writing any code. Notice that the non-hierarchical variants support the most

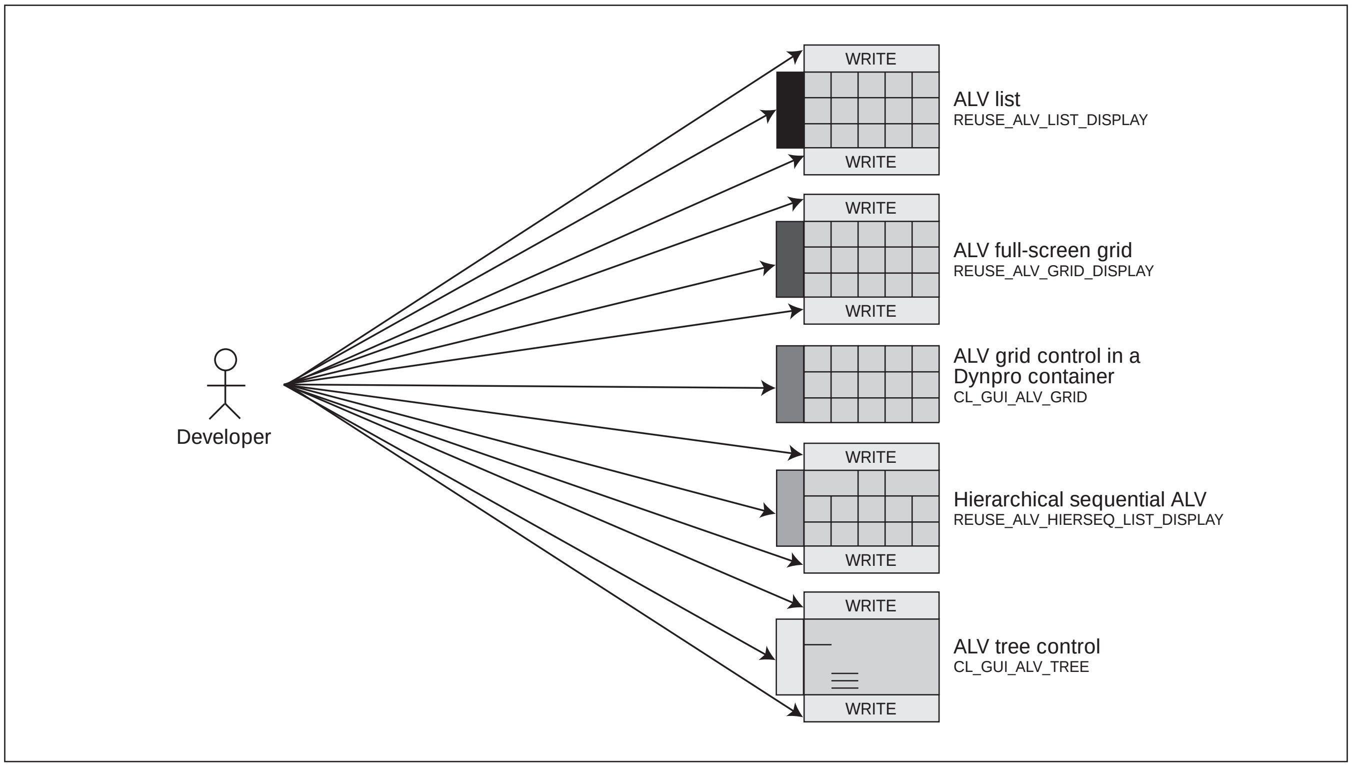


Figure 8 The old SAP List Viewer model

options for presentation control. Further notice that the tree-based variants for displaying a multi-level hierarchy do not offer any export capabilities, for example to a spreadsheet application such as Microsoft Excel.

Understanding the old SAP List Viewer model

In releases before SAP NetWeaver ’04, a separate API existed for each SAP List Viewer variant. However, only the variant for displaying non-hierarchical data using the ALV grid control in a Dynpro container and its corresponding API (`CL_GUI_ALV_GRID`) were officially released for customer use. **Figure 8** illustrates this model, showing the name of each API below its corresponding SAP List Viewer variant.

This model has several drawbacks, not least of which is that you are faced with a set of similar, yet slightly different, APIs. For example, some parameter names are identical for all of the APIs, but the referenced Data Dictionary (DDIC) structures are different. In addition, the interfaces for these APIs contain large structures with many flags for parameterization. Unfortunately, it is relatively easy to parameterize these interfaces inconsistently, simply by setting contradictory flags in the structures that you pass when you call the API. For example, suppose you want to display a field in a table. In the metadata structure for the field (that is, the field catalog), you define that the field has the data type `CHAR`. But in the same field catalog you also accidentally define that you want to calculate a total for this field, which is of course only possible for numerical fields.

Performing runtime checks for correctness or consistency for all parameter combinations is clearly impossible due to the potentially negative effect on performance. As a result, the programming axiom for the SAP List Viewer was that the parameterization must be consistent. In other words, developers are personally responsible for ensuring that they do not call the API with contradictory flags and settings. While perhaps acceptable for SAP internal use, this expectation is unreasonable for production environments and thus the reason why these APIs have never been released for customer use.

Another drawback of the model used prior to SAP NetWeaver ’04 is that SAP List Viewer supported a data- and metadata-driven interface only for the data that is displayed in a table. In other words, the SAP List Viewer simply renders the passed data for display based on the transferred metadata. However, you often need to display additional information in a report, such as at the beginning or end of a table. For example, at the top of a list you might want to display the entered selection criteria for the resulting data set. Or at the end of a list you might want to show the number of records processed or other status information.

Note!

Since the SAP List Viewer has been redesigned and the new object model is now officially released and available with SAP NetWeaver ’04, SAP strongly recommends using the new API for all new development. However, you do not need to migrate any programs that use the existing APIs.

The old model lacked a powerful API that could address these requirements with a similar data- and metadata-driven approach. Instead, to populate these areas you explicitly write the information from the application either by using ABAP `WRITE` statements (in the ABAP list processor environment) or by using the methods of the dynamic documents[5] for displaying this type of information in an HTML control (in the SAP Control Framework environment). Consequently, the content of these areas is not known to the SAP List Viewer, which means that you are responsible for creating top-of-list/end-of-list areas that conform to the UI technology used to display the data and have the same look-and-feel as other reports.

Introducing the new SAP List Viewer model

Now that you understand the shortcomings of the previous SAP List Viewer solution, let’s examine the new object model that is available in SAP NetWeaver ’04 (see **Figure 9**). For comparison purposes, the diagram maps the new API and its classes to the SAP List Viewer variants and their corresponding APIs that you used in the past. Note the addition of a new full-screen tree control, which was previously impossible because no API existed.

With the introduction of this new object model, all of the old APIs become obsolete. The new object model exposes SAP List Viewer functionality through a single, unified API with three top-level classes:

- `CL_SALV_TABLE` for displaying non-hierarchical tabular data
- `CL_SALV_HIERSEQ_TABLE` for displaying hierarchical sequential lists (or two-level hierarchical data)
- `CL_SALV_TREE` for displaying multi-level hierarchical data

Note!

The hierarchical table and tree variants have some intrinsic restrictions, such as the number of columns that can be displayed or the column width. For more information, use the Class Builder (transaction SE24) to refer to the documentation for the associated classes (`CL_SALV_HIERSEQ_TABLE` and `CL_SALV_TREE`).

[5] Dynamic documents describe an ABAP object wrapper for creating HTML screens. You call methods provided by the dynamic documents classes, which create the corresponding HTML internally. For more information, refer to the SAP Help Portal at http://help.sap.com. Navigate to the SAP NetWeaver documentation and select Application Platform → ABAP Technology → UI Technology → Controls and Control Framework for SAP GUI → Dynamic Documents.

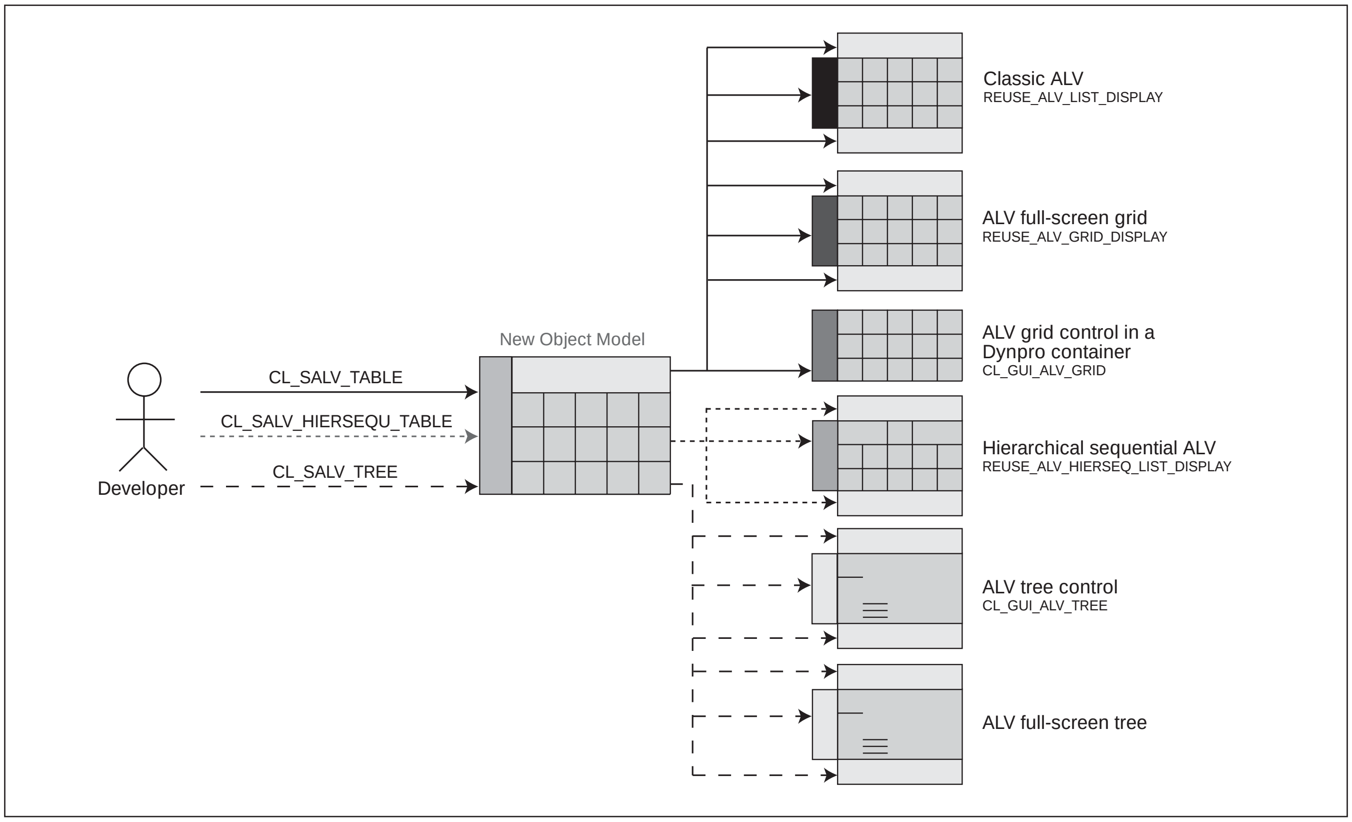


Figure 9 The new SAP List Viewer model in SAP NetWeaver ’04

Beyond the obvious benefit of a single, consistent API, the new SAP List Viewer object model offers many enhancements that are sure to make your life as a developer easier. The table in **Figure 10** summarizes

Aspect	Key features
Simplified and faster programming	• The object-oriented programming model is based on ABAP Objects, which better fits the skillset of developers today.
Error prevention	• The API automatically derives the data type, preventing metadata mismatches. • The API provides methods with built-in error-checking to avoid inconsistent parameterization. • The SAP List Viewer automatically handles back-end/front-end updates triggered by metadata changes, eliminating manual synchronization. • API methods that take a constant as a parameter have a global interface attribute as the default, simplifying identification of valid parameters.
Unified object model	• The API provides metadata-driven control of the top- and end-of-list areas. • Classes define all metadata, which allows for reuse with different variants. • You work with a streamlined object set consisting of a single unified API for all.

Figure 10 Advantages of the new SAP List Viewer model

the key features. We'll examine the aspects of error prevention and the unified object model in more detail in the next sections.

Built-in safeguards help prevent errors

Let's begin our brief tour by reviewing several enhancements that prevent many of the common sources of errors in the old SAP List Viewer model:

- **Automatically derived data type:** The SAP List Viewer now determines the data type of the data that is passed to the new API. This meta-information is derived automatically and is no longer part of the metadata interface that you specify in your code. In the past, developers commonly passed incorrect metadata for data type descriptions. This inconsistent parameterization caused a mismatch between the data and metadata describing the data types of the table fields, resulting in runtime errors.

- **Error-checking for consistent parameterization:** The new API provides `SET` and `GET` methods[6] for all metadata settings, such as "Hide this column" or "Total the values of this column" parameters. You set meta-information with the `SET` method and ask the API to return the current metadata setting with the `GET` method. These methods also provide runtime error-checking in the event of inconsistent parameterization. Suppose you try to parameterize the SAP List Viewer with contradictory metadata, for example by specifying the "Total the values of this column" parameter when the data type of the column is not numeric. An exception is raised during runtime explaining the reason for the error so you can correct the appropriate parameter(s).

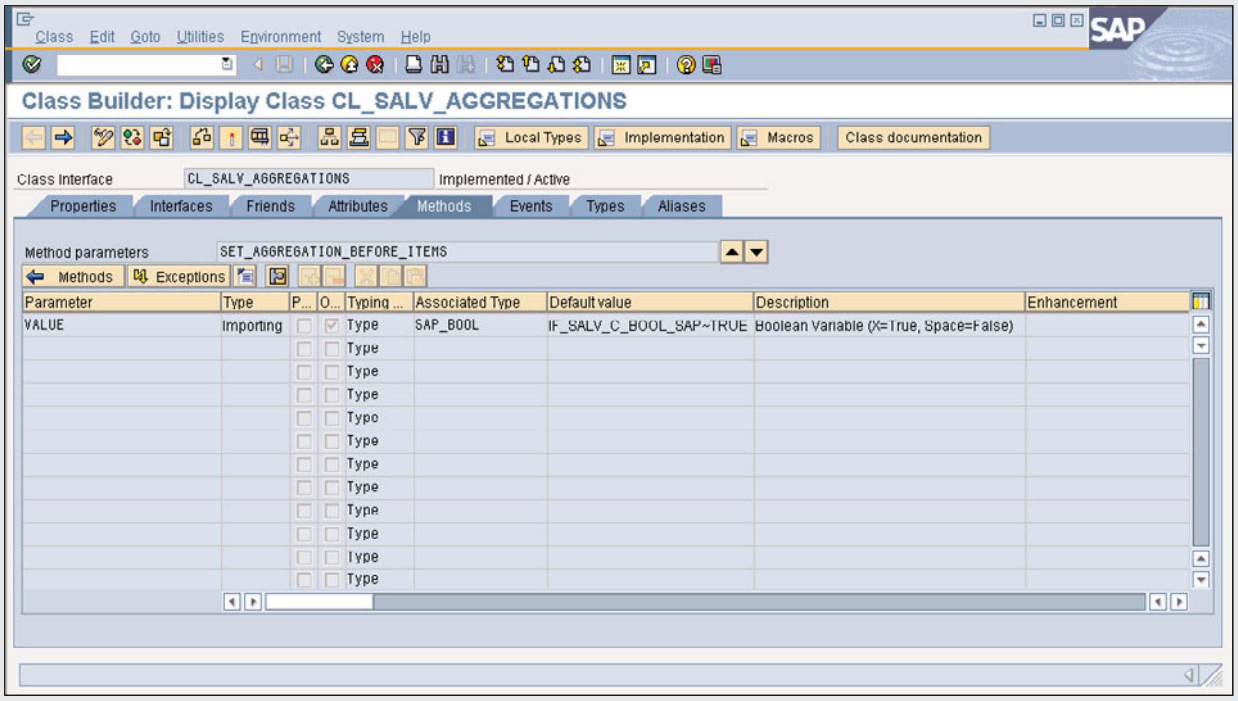
- **Automatic front-end/back-end metadata synchronization:** For SAP Control Framework variants such as the ALV grid control, the SAP List Viewer now completely handles any back-end/front-end updates that are triggered by metadata changes. For example, suppose you define a button in the SAP List Viewer toolbar and implement an event handler to handle the action. When a user clicks on the button, your program checks whether a certain column is visible and, if so, hides the column by calling the `SET` method of the column object for setting the `VISIBLE` property to `FALSE`. The object model registers that the change has been made to the back end and automatically updates the front end when the event handler is finished. With the old APIs, you needed to explicitly call the `REFRESH_TABLE_DISPLAY` method of the API in your event handler to ensure that any metadata changes programmed in the back-end code were also reflected in the front end. Otherwise, an inconsistency would exist between the back end and front end, causing unexpected behavior in the UI or possibly even runtime errors.

- **Simplified identification of valid constant parameters:** Some methods that are provided with the new API take a constant as a parameter. The new object model offers a consistent approach for identifying the possible constant values for these methods so you can easily determine the valid set of parameters. Global interface attributes are used to represent constants for parameterizing such methods. In the parameter view of the Class Builder (transaction SE24) for a method of this type, you can see if a parameter requires a constant as a value (see **Figure 11**). If a constant is expected, the Default value column shows a global interface value as the default, such as `IF_SALV_C_BOOL_SAP~TRUE` in this example. Double-click on this default value to open the attribute view of the corresponding interface definition and view a list of valid constant values. Then simply copy and paste the constant value into your code.

A unified object model adds control and streamlines usage

To conclude our tour of the key enhancements in the new object model, let's turn to its architectural

[6] For more information about the `SET` and `GET` methods that are available within a class, refer to the documentation for the corresponding class in the Class Builder (transaction SE24).


Class Edit Goto Utilities Environment System Help
SAP
Local Types
Implementation
Macros
Class documentation
Properties Interfaces Friends Attributes Methods Events Types Aliases
Methods
Exceptions

Class Builder: Display Class CL_SALV_AGGREGATIONS

Class Interface CL_SALV_AGGREGATIONS Implemented / Active

Method parameters SET_AGGREGATION_BEFORE_ITEMS

Parameter	Type	P...	O...	Typing ...	Associated Type	Default value	Description	Enhancement
VALUE	Importing	☐	✓	Type	SAP_BOOL	IF_SALV_C_BOOL_SAP~TRUE	Boolean Variable (X=True, Space=False)	
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				
		☐	☐	Type				

Figure 11 Identifying a valid set of constant parameters in the Class Builder

advantages. By providing a single API and a unified object model for all variants, the redesigned SAP List Viewer offers a much improved development experience:

- **Metadata-driven top-of-list and end-of-list areas:** The new API offers a fully data- and metadata-driven interface that now also addresses the top-of-list and end-of-list areas of a table. You simply pass the data to display in these areas to the API and describe how to display the data using metadata instead of using `WRITE` statements in your code to present the data.

- **Reusable class-defined metadata:** Metadata is modeled as classes that you can reuse in different SAP List Viewer variants. If you learn how to use the metadata for a particular class and variant, you can leverage that knowledge for programming other variants. Derived classes are provided to define additional variant-specific information, such as the column object for the ALV full-screen tree or the derived events object for tree-specific events.

- **Streamlined object set:** As mentioned earlier, the new object model provides a single API for all variants with three top-level classes. You use just one class (`CL_SALV_TABLE`) for all three representations of non-hierarchical data — that is, Classic ALV, ALV full-screen grid, and ALV grid control embedded in a Dynpro container. In the past, you had to learn how to work with a different API for each representation. In addition, you use the same class (`CL_SALV_TREE`) for both the ALV full-screen tree and the ALV tree embedded in a Dynpro container.

In my experience, the SAP List Viewer is most commonly used to display non-hierarchical data. Therefore, in the rest of this article I focus on this variant to show you how to use the new API. However, you follow the same basic approach with

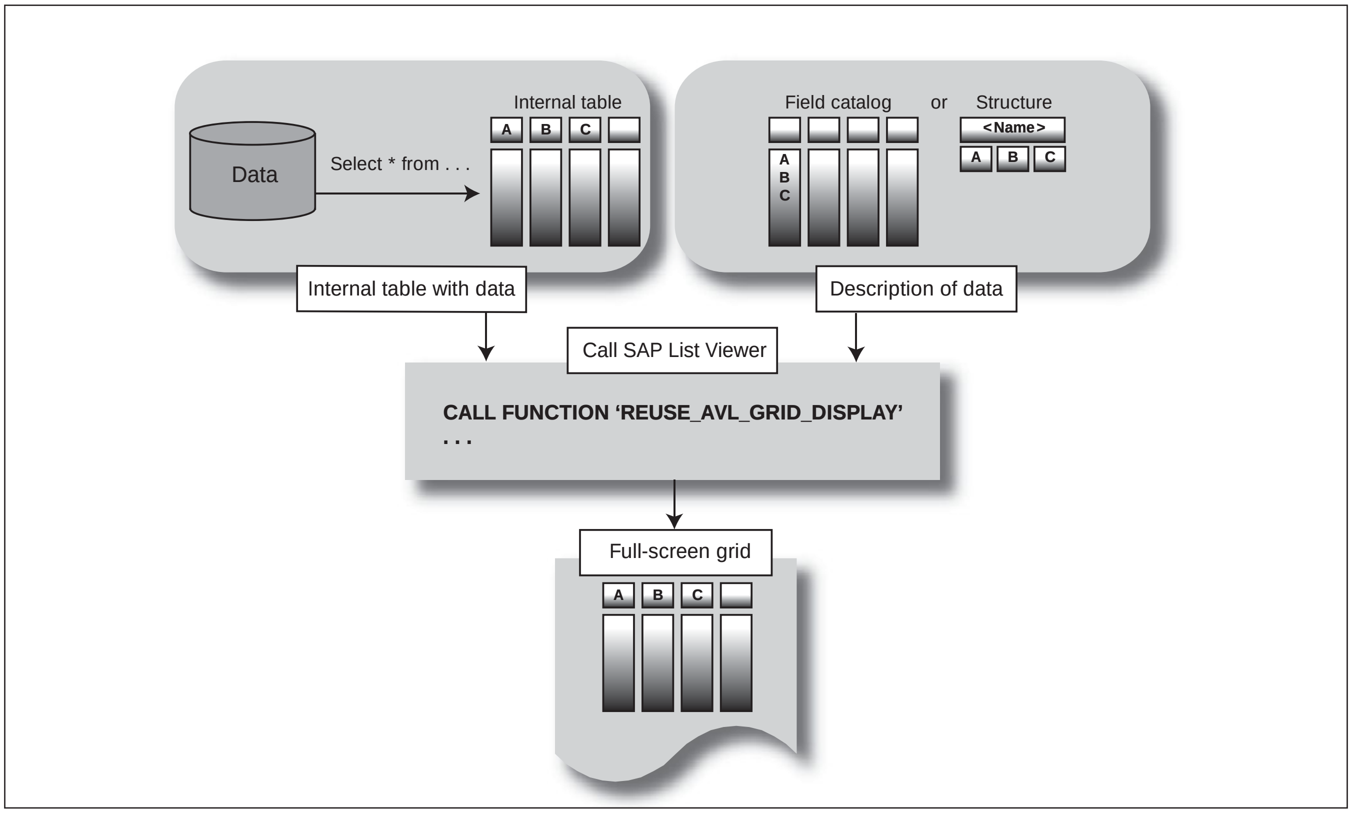


Figure 12 The old programming model for the SAP List Viewer

the other variants. To learn more about the ALV tree (`CL_SALV_TREE`) or the hierarchical sequential list (`CL_SALV_HIERSEQ_TABLE`), refer to the correspon-ding class documentation in the Class Builder (transaction SE24).

Understanding the old programming model

Figure 12 illustrates the main principles of programming the SAP List Viewer with the old APIs. This example shows how to display an ALV full-screen grid, but the principles are basically the same for displaying all non-hierarchical data.

Here is what you would do in your application code when using the old SAP List Viewer APIs. First select the data to be displayed in the SAP List Viewer in an internal table. Then pass the data and its corresponding metadata description (or field catalog, which is in a separate internal table) to the API function module (in this example, `REUSE_ALV_GRID_DISPLAY`). Each record in the field catalog describes one field of the internal data table, which is then represented as a column in the SAP List Viewer. The field catalog contains structural information such as data type, internal length, and number of decimals, as well as settings that control display options such as header text, visibility, and alignment.

If the internal table is based on a DDIC structure, transfer the name of this structure instead of a field catalog. In this case, the SAP List Viewer reads the field information from the DDIC and builds the field catalog automatically using defaults such as "all table columns are visible" for the display option. These defaults are quite useful for rapid prototyping because you do not have to specify or pass any metadata and you see immediate results with very limited coding. Later on, you can specify the necessary metadata to

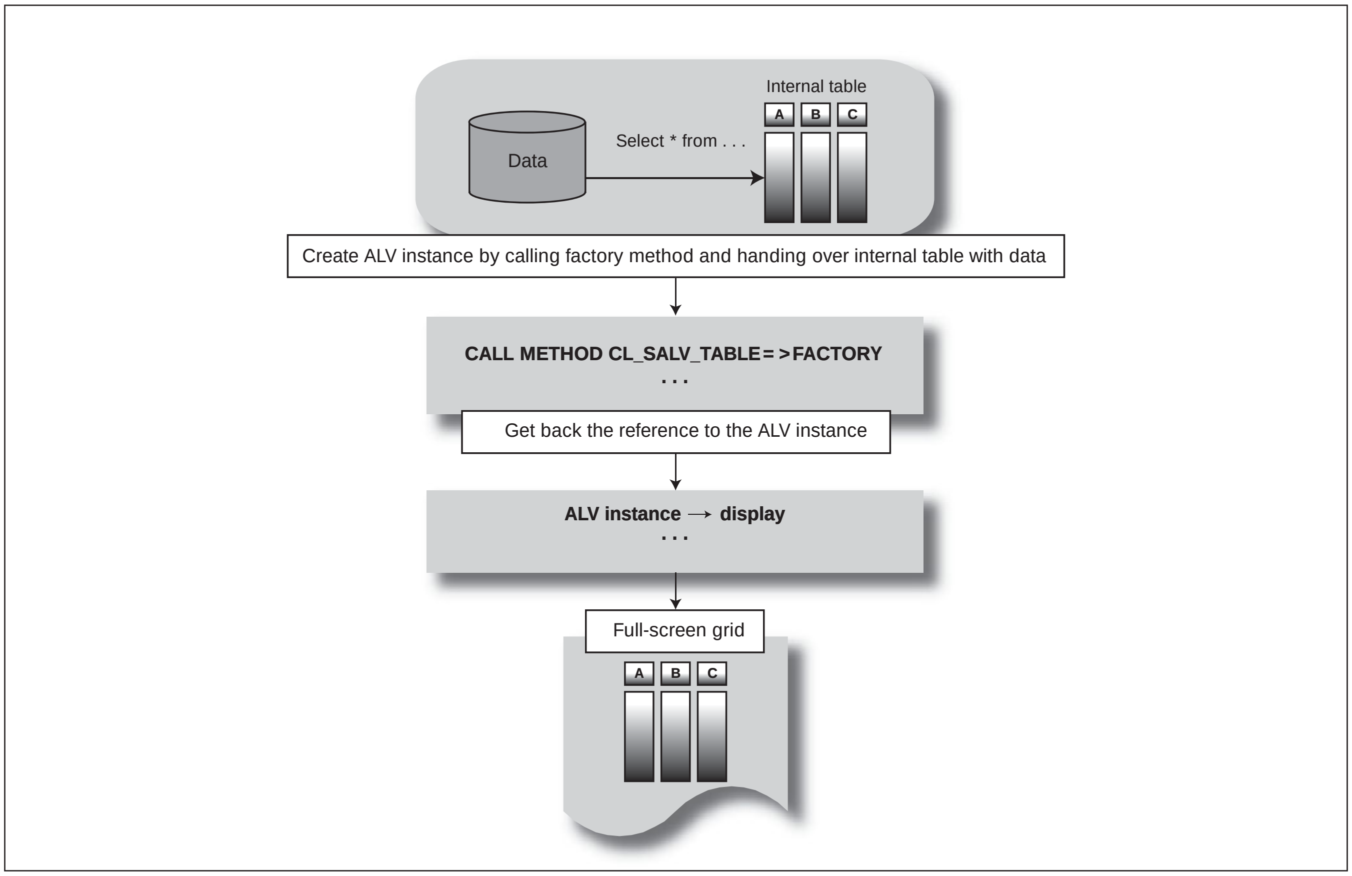

Internal table
A
B
C
Data
Select * from . . .
Create ALV instance by calling factory method and handing over internal table with data
CALL METHOD CL_SALV_TABLE=>FACTORY
. . .
Get back the reference to the ALV instance
ALV instance → display
. . .
Full-screen grid
A
B
C

Figure 13 The new SAP List Viewer programming model in SAP NetWeaver ’04

display the data exactly as you want, for example by identifying which columns to hide when the report is presented to the user. The data is displayed on the screen as a result of the ALV function module call and all generic functions such as sorting, filtering, and so on are represented as buttons in the toolbar.

Introducing the new programming model

Figure 13 describes the main principles of the new object-oriented programming model. For ease of comparison with the old programming model, this example also shows how to display an ALV full-screen grid. As with the old model, the principles are basically the same for displaying all non-hierarchical data.

Main principles of the new object model

In terms of how you write the code, you still select the data in an internal table. But with the new model you call the `FACTORY` method of the class for the desired representation and pass the internal table to it. This example uses the `CL_SALV_TABLE` class in order to display an ALV full-screen grid. The `FACTORY` method returns an instance of this class. To display the table, you call the `DISPLAY` method of this instance. The SAP List Viewer determines all structural information such as the data type, internal length, and so on, preventing the potential data type inconsistencies described earlier. The SAP List Viewer examines the internal data that is passed to the `FACTORY` method and automatically derives the correct metadata. For all display settings, the SAP List Viewer provides default settings that, in most cases, represent the desired

```
report  salv_learn_map_table_1.
*... Select Data
data:
  gt_outtab type table of sflight.

select * from sflight into corresponding fields of table gt_outtab.

*... Create Instance and Display as Fullscreen Grid Control
data:
  gr_table  type ref to cl_salv_table.

call method cl_salv_table=>factory
  importing
    r_salv_table = gr_table
  changing
    t_table      = gt_outtab.

*... Display Table
gr_table->display( ).
```

Figure 14 Example code to display data in an ALV full-screen grid

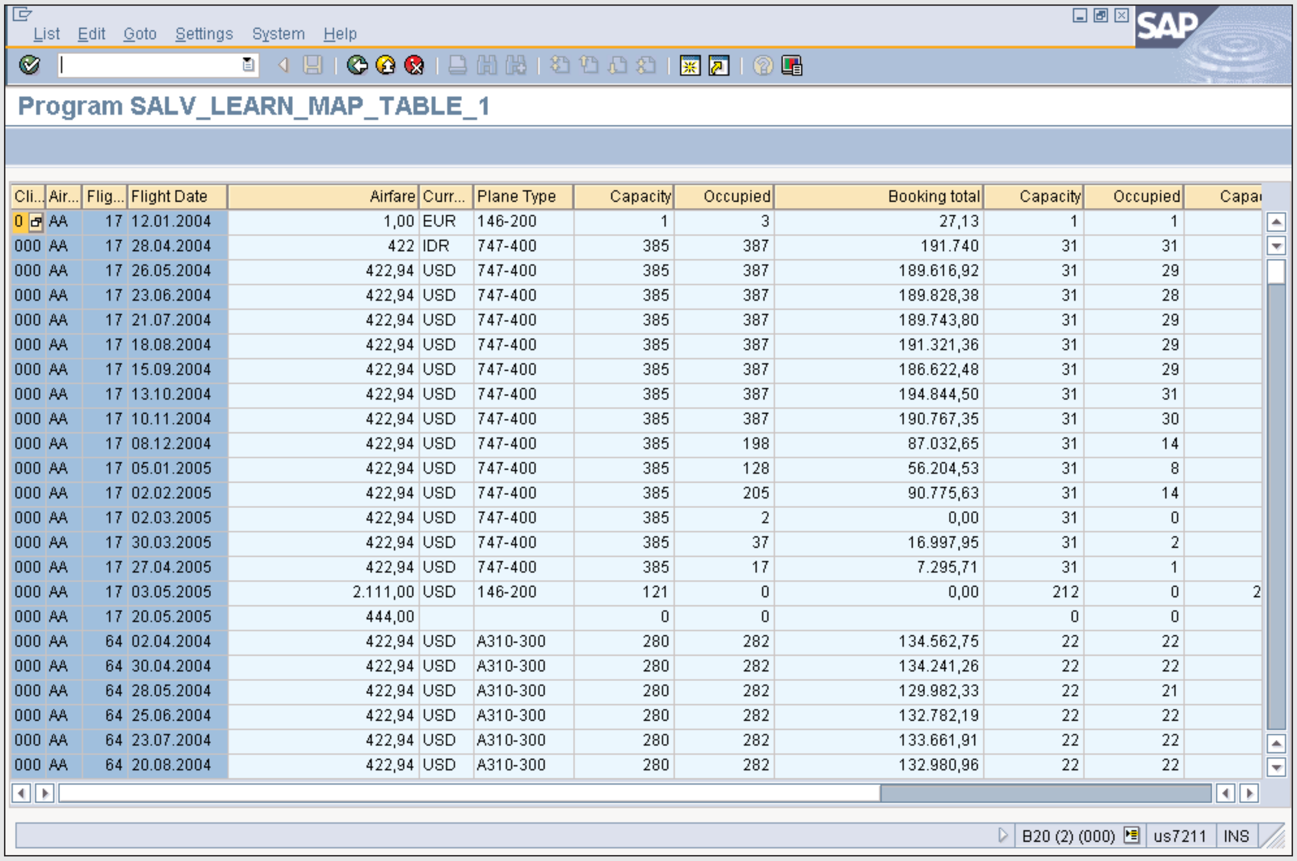
behavior. For example, the alignment is set to left-justified for all character-based or string-based data types; the alignment is set to right-justified for all numeric data types; and the column width is derived from the output-length setting in the DDIC if a field in the internal table references a DDIC data element.

Note!

Metadata is derived from dynamically created internal tables in the same manner. However, with Basis Release 6.40, SAP recommends that you use the ABAP runtime type information (RTTI) classes to create dynamic tables instead of using the `CL_ALV_TABLE_CREATE` class. This technique is now the official way to create dynamic data structures in ABAP.

I want to call your attention to an important change in the default behavior of the new object model in comparison to the old programming model. When you use the new API, by default the presented UI offers no generic services. Of course, you can easily switch services selectively for each service (e.g., sort ascending), for a group of services (e.g., sort and filter functions such as sort ascending, sort descending, filter on, and filter off), or for all services at once. I provide an example of how to do this later in the article (see the "Example #2: Displaying generic services to users" section).

SAP decided to change this default behavior based on past experience with the SAP List Viewer. We have observed that this tool is often used to display very simple lists such as two columns with five data rows. Developers often did not take advantage of the option to switch off any generic services that might not be relevant for a table of this type, such as Excel inplace representation or aggregation functionality. As a result, simple lists were often



Program SALV_LEARN_MAP_TABLE_1

Cli...	Air...	Flig...	Flight Date	Airfare	Curr...	Plane Type	Capacity	Occupied	Booking total	Capacity	Occupied	Capa
0	AA	17	12.01.2004	1,00	EUR	146-200	1	3	27,13	1	1	
000	AA	17	28.04.2004	422	IDR	747-400	385	387	191.740	31	31	
000	AA	17	26.05.2004	422,94	USD	747-400	385	387	189.616,92	31	29	
000	AA	17	23.06.2004	422,94	USD	747-400	385	387	189.828,38	31	28	
000	AA	17	21.07.2004	422,94	USD	747-400	385	387	189.743,80	31	29	
000	AA	17	18.08.2004	422,94	USD	747-400	385	387	191.321,36	31	29	
000	AA	17	15.09.2004	422,94	USD	747-400	385	387	186.622,48	31	29	
000	AA	17	13.10.2004	422,94	USD	747-400	385	387	194.844,50	31	31	
000	AA	17	10.11.2004	422,94	USD	747-400	385	387	190.767,35	31	30	
000	AA	17	08.12.2004	422,94	USD	747-400	385	198	87.032,65	31	14	
000	AA	17	05.01.2005	422,94	USD	747-400	385	128	56.204,53	31	8	
000	AA	17	02.02.2005	422,94	USD	747-400	385	205	90.775,63	31	14	
000	AA	17	02.03.2005	422,94	USD	747-400	385	2	0,00	31	0	
000	AA	17	30.03.2005	422,94	USD	747-400	385	37	16.997,95	31	2	
000	AA	17	27.04.2005	422,94	USD	747-400	385	17	7.295,71	31	1	
000	AA	17	03.05.2005	2.111,00	USD	146-200	121	0	0,00	212	0	2
000	AA	17	20.05.2005	444,00			0	0		0	0	
000	AA	64	02.04.2004	422,94	USD	A310-300	280	282	134.562,75	22	22	
000	AA	64	30.04.2004	422,94	USD	A310-300	280	282	134.241,26	22	22	
000	AA	64	28.05.2004	422,94	USD	A310-300	280	282	129.982,33	22	21	
000	AA	64	25.06.2004	422,94	USD	A310-300	280	282	132.782,19	22	22	
000	AA	64	23.07.2004	422,94	USD	A310-300	280	282	133.661,91	22	22	
000	AA	64	20.08.2004	422,94	USD	A310-300	280	282	132.980,96	22	22	

Figure 15 Example report using an ALV full-screen grid

overloaded with functionality that is not useful, or worse, confusing in that context.

You now have a basic understanding of how the new SAP List Viewer works and why. Let’s move on to the fun part of the article, which is how to write the necessary code.

Creating simple SAP List Viewer reports

In the first example (**Figure 14**), I show you how to implement the simplest form of an SAP List Viewer call. First, select the data from the database as shown in the “Select Data” section in Figure 14. Then pass the data to be displayed to the `FACTORY` method of the object model, as shown in the “Create Instance and Display as Fullscreen Grid Control” section. Specify the class associated with the desired variant; all the examples in this article use the `CL_SALV_TABLE` class for displaying non-hierarchical data. This method returns an instance of the object. Finally, call the `DISPLAY` method of the instance to display the data (see the “Display Table” section). If you call the `FACTORY` method without any additional parameters as shown in this example, the SAP List Viewer presents the data in the default format as an ALV full-screen grid. See **Figure 15** for the resulting report.

```
report  salv_learn_map_table_1_1.
*... Select Data
data:
  gt_outtab type table of sflight.

select * from sflight into corresponding fields of table gt_outtab.

*... Create Instance and Display as Classic ALV
data:
  gr_table  type ref to cl_salv_table.

call method cl_salv_table=>factory
  exporting
    list_display = abap_true
  importing
    r_salv_table = gr_table
  changing
    t_table      = gt_outtab.

*... Display Table
gr_table->display( ).
```

Figure 16 Example code to display data in a Classic ALV

The code for displaying non-hierarchical tabular data using the other SAP List Viewer variants is very similar. For example, **Figure 16** shows how to implement a Classic ALV. As you can see, the step to select the data from the database is exactly the same. However, when you pass the data to be displayed to the `FACTORY` method, include the `LIST_DISPLAY` parameter with the value set to `TRUE` (see the "Create Instance and Display as Classic List" section). The method again returns an instance of the object. As in the previous example, call the `DISPLAY` method of the instance to display the data.

To round out our set of simple examples, the third variant for displaying non-hierarchical tabular data is an ALV grid control in a Dynpro container. The example in **Figure 17** illustrates how to implement this variant. Once again, start by selecting the data from the database, as in the previous examples. For this variant, you then create an instance of the container in which the ALV grid control is to be displayed, as shown in the "Create Instance of Container" section. Next call the `FACTORY` method as described in the "Create Instance and Display as Grid within a Dynpro" section. In addition to passing the data table, notice that here you include two additional parameters: `R_CONTAINER`, which contains the reference to the container, and `CONTAINER_NAME`. The method returns an instance of the object. And finally, call the `DISPLAY` method of the instance to display the data.

You have now seen how easy it is to produce simple reports using the SAP List Viewer. However,

```
report  salv_learn_map_table_1_2.
*... Select Data
data:
  gt_outtab type table of sflight.

select * from sflight into corresponding fields of table gt_outtab.

*... Create Instance of Container
data:
  gr_container  type ref to cl_gui_custom_container.

create object gr_container
  exporting
    container_name = 'CONTAINER'.

*... Create Instance and Display as Grid within a Dynpro
data:
  gr_table  type ref to cl_salv_table.

call method cl_salv_table=>factory
  exporting
     r_container    = gr_container
     container_name = 'CONTAINER'
  importing
    r_salv_table = gr_table
  changing
    t_table      = gt_outtab.

*... Display Table
gr_table->display( ).
```

Figure 17 Example code to display data in an ALV grid control in a Dynpro container

if you want to change the metadata defaults before the report appears on the screen, you need to call the appropriate `GET` and `SET` methods for the metadata objects in your code. The SAP List Viewer API instantiates all of these methods as a result of the factory method. In the following section, I provide an overview of these metadata objects and show how easily you can use them to program the desired table formatting and behavior.

Using metadata objects for table formatting and behavior

Report metadata is modeled as classes that you can reuse in different SAP List Viewer variants. To get a better idea of what characteristics are available, let’s examine the metadata structure for the

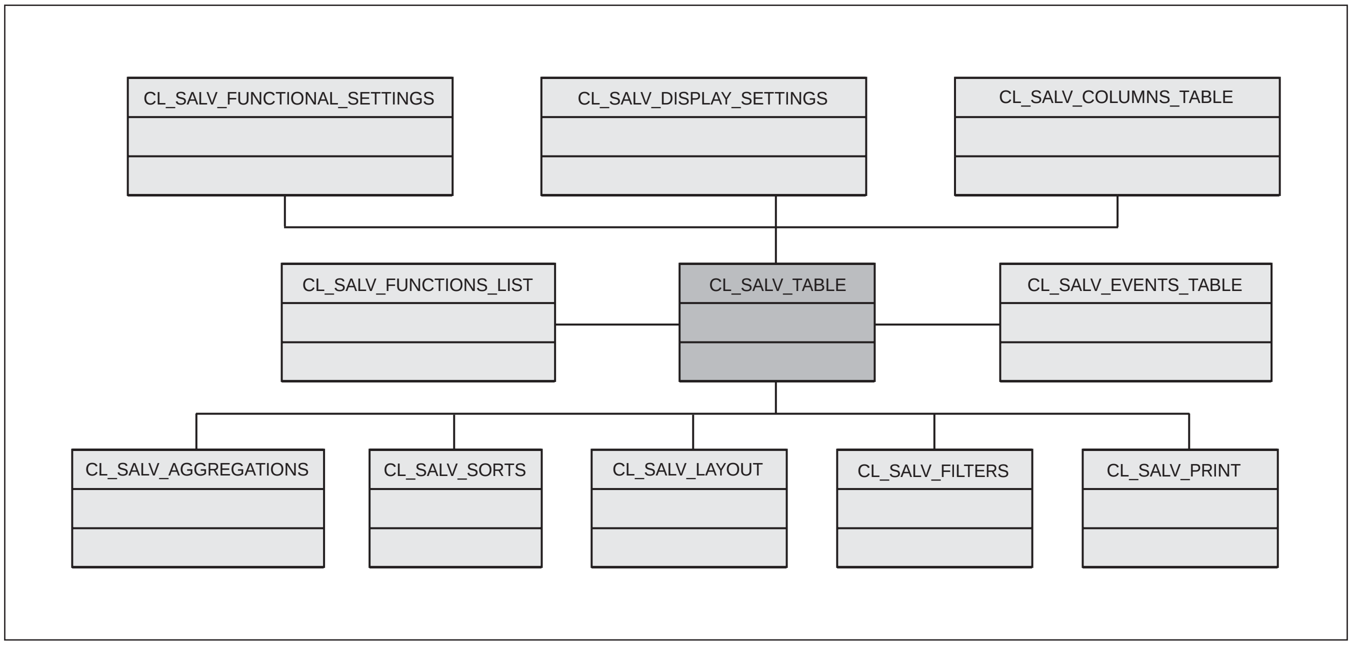


Figure 18 Main metadata classes of the CL_SALV_TABLE class

`CL_SALV_TABLE` class. **Figure 18** contains a UML diagram of the classes that define the different aspects of metadata information offered by the SAP List Viewer. You access these metadata objects from the `CL_SALV_TABLE` ALV instance that you obtain by calling the corresponding `GET` method. The other two top-level classes provide similar metadata classes.

Caution!

When applying aggregation to a table, be careful to set the internal column size appropriately. The internal size of the column must be sufficient to accommodate the individual values as well as the result of the aggregation. A column that is too small for the aggregated result causes a field overflow, preventing the display of the result. For example, never use `INT2` as the data type for a column if you want users to be able to aggregate its values. Unless the aggregated value is unrealistically between -327678 and 32767, the result will not fit into an `INT2` variable.

You access all of these main metadata classes directly from the `CL_SALV_TABLE` object. These classes contain further derived subclasses (see **Figure 19**) that define the metadata for every concrete object, such as a specific column or sort criterion. Notice the deliberate similarity between the names of the main classes and their subclasses. The naming convention is that the class that defines the general metadata (which is also valid for all subclasses) uses the plural form of the object portion of its name, such as `CL_SALV_COLUMNS_TABLE`. The class that defines the metadata for a specific object uses the singular form, such as `CL_SALV_COLUMN_TABLE`.

These classes and the methods they provide for manipulating the properties of tables and their component objects are well-documented in the system documentation.[7] However, to give you an overview as a jumpstart, **Figure 20** provides a brief description of the purpose of the metadata objects and the type of information available in each class.

[7] For more information, refer to the SAP Help Portal at http://help.sap.com. Navigate to the SAP NetWeaver documentation and select Application Platform → ABAP Technology → UI Technology → Controls and Control Framework for SAP GUI → SAP List Viewer (BC-SRV-ALV) → ALV Object Model.

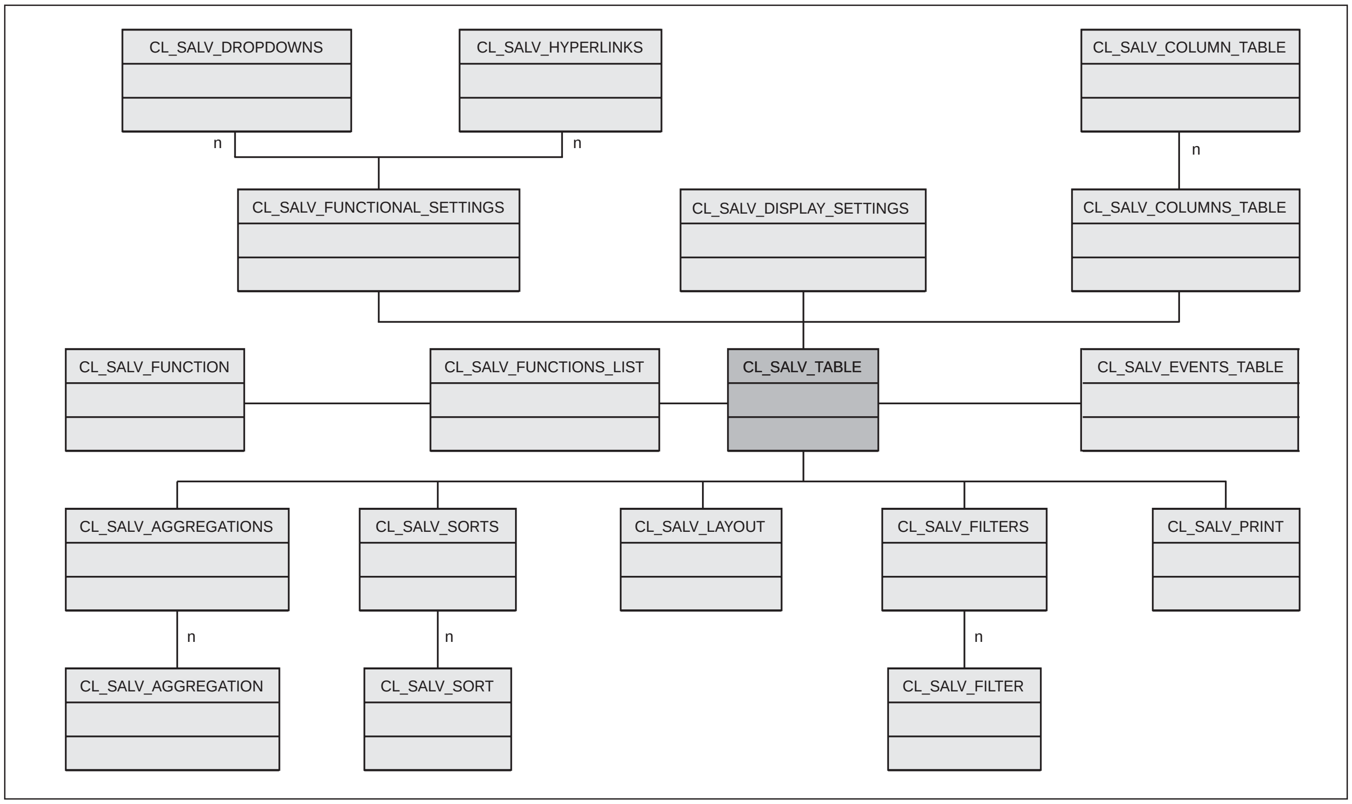


Figure 19 Main metadata classes and their derived subclasses

Class and purpose	Examples of common actions
CL_SALV_AGGREGATIONS	
Define aggregation options for a table. This object stores a list of aggregation options plus the collective aggregation properties for the table. A separate CL_SALV_AGGREGATION object for each aggregation option stores its properties.	• Set collective properties such as whether to display the aggregated result before or after the aggregated line items. • Set individual properties such as the type of aggregation (total, minimum, maximum, or average) to apply to a column.
CL_SALV_COLUMNS_TABLE	
Manage the columns in a table. This object stores a list of columns and the collective column properties for the table. A separate CL_SALV_COLUMN_TABLE object for each column stores its properties, which you use SET and GET methods to change.	• Obtain a list of columns or change column order. • Change individual column properties, such as alignment, visibility, width, and title. • Change collective column properties, such as optimizing width or displaying/hiding headers.
CL_SALV_DISPLAY_SETTINGS	
Control the visual appearance of the overall table.	• Produce a striped effect by shading alternate table rows light and dark.

Figure 20 Overview of the main metadata classes

Continues on next page

Figure 20 *continued*

Class and purpose	Examples of common actions
CL_SALV_DROPDOWNS	
Define a drop-down list for a cell. This object stores a list of drop-downs that contain value lists. Each drop-down is identified by a key that you can link to a column.	• Define a drop-down list for a column.
CL_SALV_EVENTS_TABLE	
Implement application-specific functionality in a program.	• Add an application-specific button to the ALV toolbar. • Execute application-specific logic when a user double-clicks on a table cell.
CL_SALV_FILTERS	
Define filters to apply to a table. This object stores the filter criteria. A separate CL_SALV_FILTER object for each criterion stores its defined selection options.	• Define filters and selection options.
CL_SALV_FUNCTIONAL_SETTINGS	
Manage user interaction by defining the functional settings that control what happens when users click on different areas of a table.	• Define which function to execute if users double-click on a cell. • Define that single-clicking on a column header sorts the table by that column.
CL_SALV_FUNCTIONS_LIST	
Control which generic functions are available to users in the toolbar.	• Expose some, all, or no generic functions, such as sort, filter, and so on.
CL_SALV_HYPERLINKS	
Define a hyperlink for a cell. This object stores a list of hyperlinks. Each hyperlink is identified by a key that you can link to a column.	• Define a hyperlink for a column.
CL_SALV_LAYOUT	
Control whether users are authorized to change and save report settings for future reuse.	• Enable or disable layout control.
CL_SALV_PRINT	
Specify the information to include when users print the report.	• Add a cover page that shows the total number of records, number of subtotals calculated, and number of records filtered.
CL_SALV_SORTS	
Define sort criteria to apply to a table. This object contains collective sort properties for the table, including whether a sort order is defined and, if so, the list and order of sort criteria. A separate CL_SALV_SORT object for each criterion stores its properties.	• Define sort criteria and properties, such as sort sequence (ascending or descending).

Considerations for controlling functional settings

In designing the new object model, we decided to distinguish clearly between the settings that influence the visual design and display behavior of a report (that is, the `CL_SALV_DISPLAY_SETTINGS` class) and the settings that influence the interaction design of the report (that is, the `CL_SALV_FUNCTIONAL_SETTINGS` class). The `CL_SALV_FUNCTIONAL_SETTINGS` class includes the properties that influence the interaction model and control, such as which function is executed if the user double-clicks on a cell or whether single-clicking on a column header sorts the records in the displayed table by that column.

As the default, single-clicking on the column header always highlights the whole column. From a user perspective, the SAP List Viewer follows the object-action[*] interaction paradigm. First you select or highlight an object, and then you choose the action you want to execute on the object. Thus, the easiest way to total the values of a column is to click on the column to highlight it and then click on the toolbar button that provides the total function.

Suppose you decide to change the interaction behavior of the SAP List Viewer so that single-clicking on the column header no longer highlights the column, but instead sorts the table immediately. Be aware that subsequently all column-related interactions such as sort, total, filter, hide, and so on can only be executed in the SAP List Viewer settings dialog. Pressing a button for a column-related interaction without first selecting anything displays this dialog. The default column selection option is no longer available, as you have now reserved it for the sort functionality.

[*] The object-action model specifies that, when using a GUI, the user first selects an object and then selects the action to be performed on the object.

Some examples to get you started

Now that you have a basic understanding of the available classes and methods, I want to walk through some examples (including the necessary code) that illustrate how to perform some of the most common tasks:

- How to hide a table column from display
- How to control which generic functions are visible to users
- How to add application-specific functions
- How to control whether users can change and save the report layout in runtime

In the following sections, I describe how to achieve the desired result and show the corresponding code. Feel free to try these examples in your own system environment, especially if you are new to the SAP List Viewer.

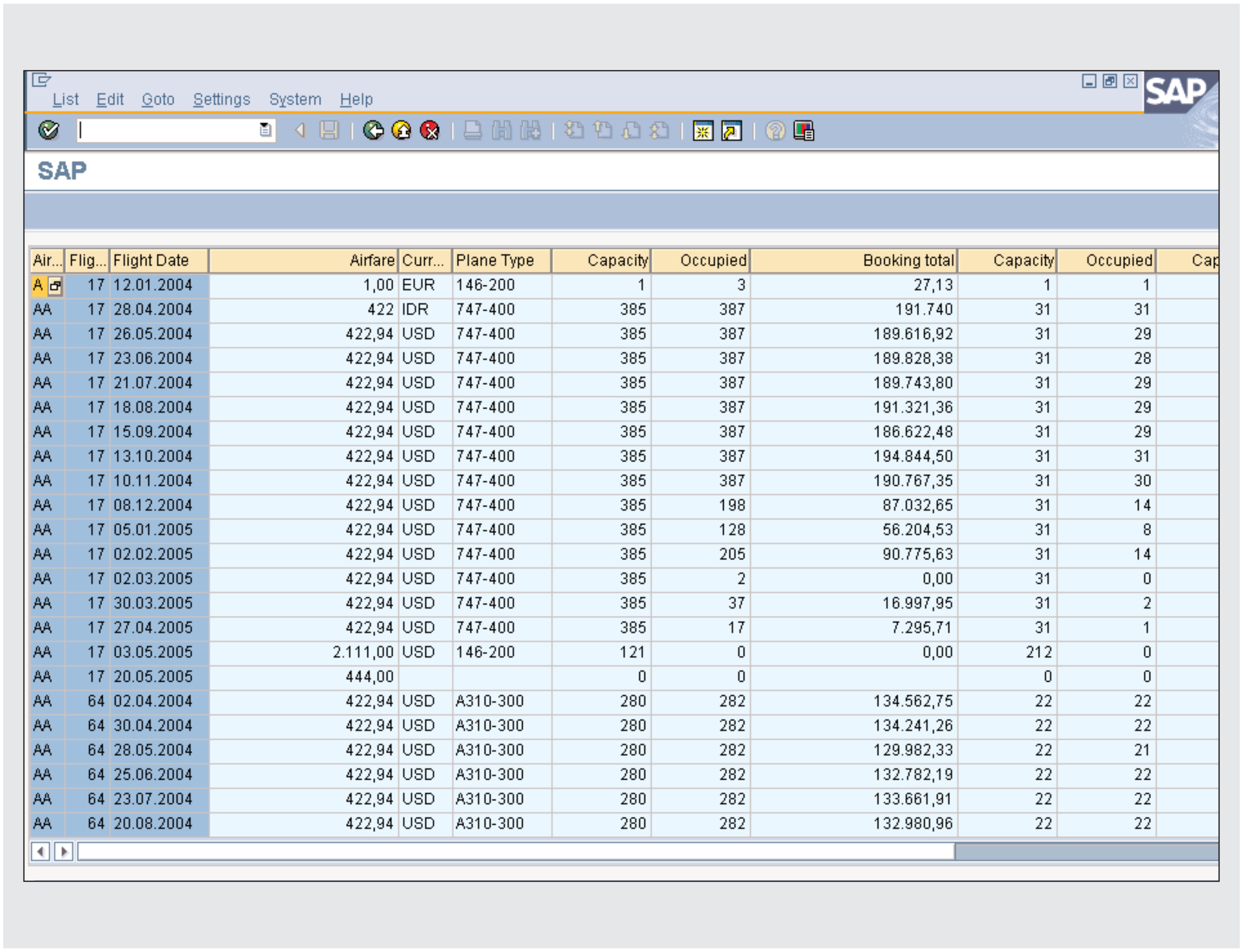
Example #1: Hiding columns

The SAP List Viewer automatically creates a default set of columns based on the table of data that is passed to the API. However, the set of provided columns may not always correspond to the intended design of the report. Therefore, the SAP List Viewer provides a way for you to control the table layout by defining which columns to display and in what order.

```
report  salv_learn_map_table_2.

*... Select Data
data:
  gt_outtab type table of sflight.

select * from sflight into corresponding fields of table gt_outtab.

*... Create Instance
data:
  gr_table  type ref to cl_salv_table.

call method cl_salv_table=>factory
  importing
    r_salv_table = gr_table
  changing
    t_table      = gt_outtab.

*... Disable Column
data:
  lr_columns type ref to cl_salv_columns_table,
  lr_column  type ref to cl_salv_column.

lr_columns = gr_table->get_columns( ).
lr_column = lr_columns->get_column( 'MANDT' ).
lr_column->set_visible( abap_false ).

*... Display Table
gr_table->display( ).
```

Figure 21 Example code to hide a column

In this first example, I show you how to remove one or more columns from the set of columns to be displayed. Building on the simple examples that I provided earlier, the "Disable Column" section of the example code in **Figure 21** shows you how to proceed if you want to hide a column. First call the `GET_COLUMNS` method to retrieve the `CL_SALV_COLUMNS_TABLE` object from the object model. This object contains a list of available columns that were passed in the internal table and also provides the `GET_COLUMN` method for retrieving a column. Next retrieve the column that you want to hide and set its `VISIBLE` property to `FALSE`. In this example, the `MANDT` column in the internal table that was passed holds client information, which you typically do not want to show



Air...	Flig...	Flight Date	Airfare	Curr...	Plane Type	Capacity	Occupied	Booking total	Capacity	Occupied	Cap
A	17	12.01.2004	1,00	EUR	146-200	1	3	27,13	1	1	
AA	17	28.04.2004	422	IDR	747-400	385	387	191.740	31	31	
AA	17	26.05.2004	422,94	USD	747-400	385	387	189.616,92	31	29	
AA	17	23.06.2004	422,94	USD	747-400	385	387	189.828,38	31	28	
AA	17	21.07.2004	422,94	USD	747-400	385	387	189.743,80	31	29	
AA	17	18.08.2004	422,94	USD	747-400	385	387	191.321,36	31	29	
AA	17	15.09.2004	422,94	USD	747-400	385	387	186.622,48	31	29	
AA	17	13.10.2004	422,94	USD	747-400	385	387	194.844,50	31	31	
AA	17	10.11.2004	422,94	USD	747-400	385	387	190.767,35	31	30	
AA	17	08.12.2004	422,94	USD	747-400	385	198	87.032,65	31	14	
AA	17	05.01.2005	422,94	USD	747-400	385	128	56.204,53	31	8	
AA	17	02.02.2005	422,94	USD	747-400	385	205	90.775,63	31	14	
AA	17	02.03.2005	422,94	USD	747-400	385	2	0,00	31	0	
AA	17	30.03.2005	422,94	USD	747-400	385	37	16.997,95	31	2	
AA	17	27.04.2005	422,94	USD	747-400	385	17	7.295,71	31	1	
AA	17	03.05.2005	2.111,00	USD	146-200	121	0	0,00	212	0	
AA	17	20.05.2005	444,00			0	0		0	0	
AA	64	02.04.2004	422,94	USD	A310-300	280	282	134.562,75	22	22	
AA	64	30.04.2004	422,94	USD	A310-300	280	282	134.241,26	22	22	
AA	64	28.05.2004	422,94	USD	A310-300	280	282	129.982,33	22	21	
AA	64	25.06.2004	422,94	USD	A310-300	280	282	132.782,19	22	22	
AA	64	23.07.2004	422,94	USD	A310-300	280	282	133.661,91	22	22	
AA	64	20.08.2004	422,94	USD	A310-300	280	282	132.980,96	22	22	

Figure 22 Example report with MANDT column hidden

in a report. **Figure 22** shows the result, with the client information column no longer visible.

Example #2: Displaying generic services to users

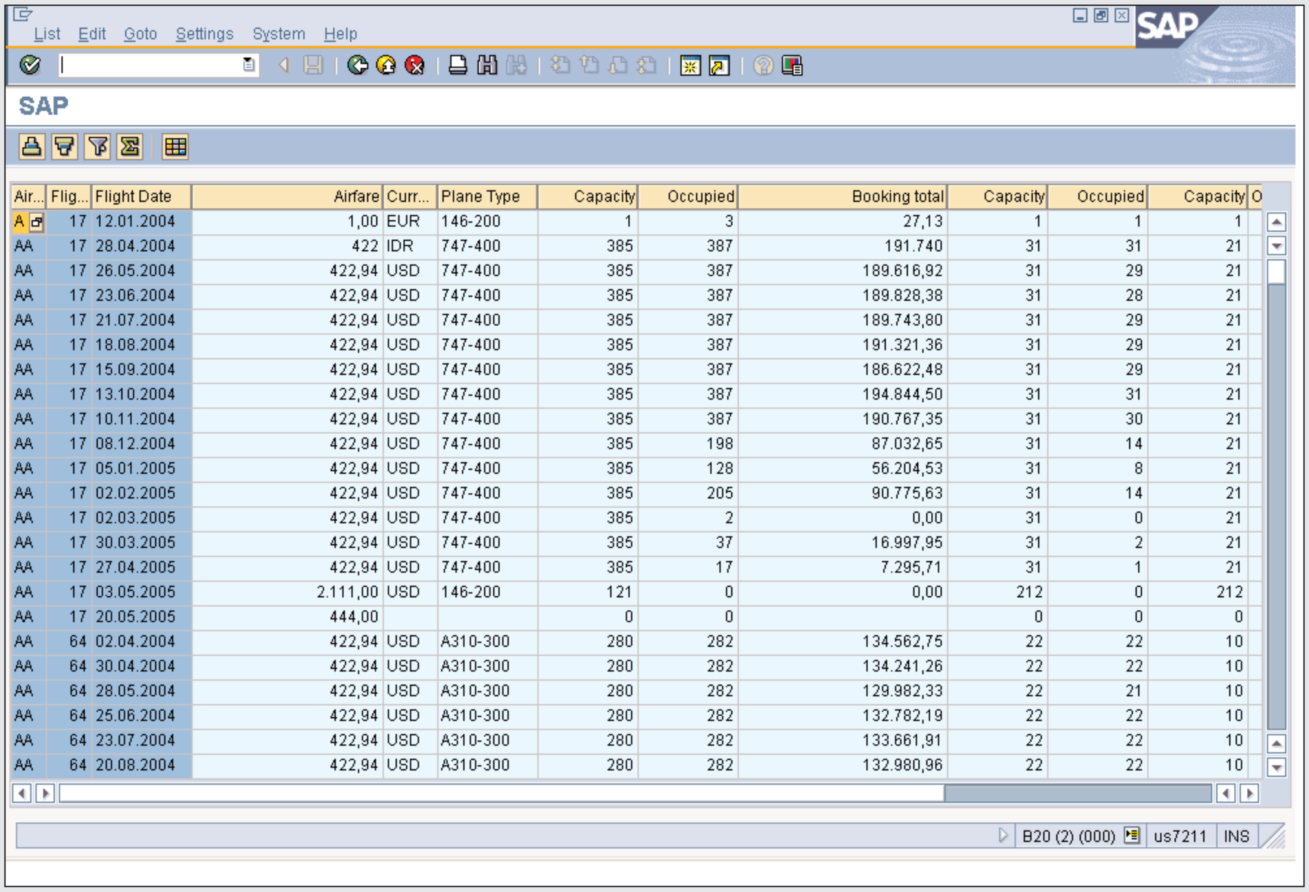
The SAP List Viewer provides a generic set of built-in services such as sort, filter, aggregation, and so on. By default, none of these generic functions are displayed to users. Instead, you decide which services suit your report and then activate the desired functions. If you need all of these generic services, use the `SET_ALL` method to activate/deactivate all functions in a single step. Alternatively, you can use the `SET_DEFAULT` method to activate/deactivate a predefined subset of functions that are commonly used in all reports (sort, filter, aggregation, sub-aggregation, column choice, and search, for example). Other methods, such as `SET_GROUP_VIEW`, allow you to activate a group of related functions.

In this second example, I describe how to use these valuable methods. I start by showing you how to activate a predefined set of functions in a single step using the `SET_DEFAULT` method. Look at the

```
report salv_learn_map_table_3 .

*... Select Data
data:
  gt_outtab type table of sflight.

select * from sflight into corresponding fields of table gt_outtab.

*... Create Instance
data:
  gr_table  type ref to cl_salv_table.

call method cl_salv_table=>factory
  importing
    r_salv_table = gr_table
  changing
    t_table      = gt_outtab.

*... Disable Column
data:
  lr_columns type ref to cl_salv_columns_table,
  lr_column  type ref to cl_salv_column.

lr_columns = gr_table->get_columns( ).
lr_column = lr_columns->get_column( 'MANDT' ).
lr_column->set_visible( abap_false ).

*... Enable Generic ALV Functions
data: gr_functions type ref to cl_salv_functions_list.
gr_functions = gr_table->get_functions( ).
gr_functions->set_default( ).

*... Display Table
gr_table->display( ).
```

Figure 23 Example code to activate the default set of services

"Enable Generic ALV Functions" section of the code in the example shown in **Figure 23**. First call the `GET_FUNCTIONS` method to retrieve the `CL_SALV_FUNCTIONS_LIST` object from the object model. Then call the `SET_DEFAULT` method of this object to activate the predefined set of SAP List Viewer functions. As you can see in **Figure 24**, the buttons for sort ascending, sort descending, total, filter, and column choice now appear in the toolbar that is located above the list.



Air...	Flig...	Flight Date	Airfare	Curr...	Plane Type	Capacity	Occupied	Booking total	Capacity	Occupied	Capacity	O
A	17	12.01.2004	1,00	EUR	146-200	1	3	27,13	1	1	1	
AA	17	28.04.2004	422	IDR	747-400	385	387	191.740	31	31	21	
AA	17	26.05.2004	422,94	USD	747-400	385	387	189.616,92	31	29	21	
AA	17	23.06.2004	422,94	USD	747-400	385	387	189.828,38	31	28	21	
AA	17	21.07.2004	422,94	USD	747-400	385	387	189.743,80	31	29	21	
AA	17	18.08.2004	422,94	USD	747-400	385	387	191.321,36	31	29	21	
AA	17	15.09.2004	422,94	USD	747-400	385	387	186.622,48	31	29	21	
AA	17	13.10.2004	422,94	USD	747-400	385	387	194.844,50	31	31	21	
AA	17	10.11.2004	422,94	USD	747-400	385	387	190.767,35	31	30	21	
AA	17	08.12.2004	422,94	USD	747-400	385	198	87.032,65	31	14	21	
AA	17	05.01.2005	422,94	USD	747-400	385	128	56.204,53	31	8	21	
AA	17	02.02.2005	422,94	USD	747-400	385	205	90.775,63	31	14	21	
AA	17	02.03.2005	422,94	USD	747-400	385	2	0,00	31	0	21	
AA	17	30.03.2005	422,94	USD	747-400	385	37	16.997,95	31	2	21	
AA	17	27.04.2005	422,94	USD	747-400	385	17	7.295,71	31	1	21	
AA	17	03.05.2005	2.111,00	USD	146-200	121	0	0,00	212	0	212	
AA	17	20.05.2005	444,00			0	0		0	0	0	
AA	64	02.04.2004	422,94	USD	A310-300	280	282	134.562,75	22	22	10	
AA	64	30.04.2004	422,94	USD	A310-300	280	282	134.241,26	22	22	10	
AA	64	28.05.2004	422,94	USD	A310-300	280	282	129.982,33	22	21	10	
AA	64	25.06.2004	422,94	USD	A310-300	280	282	132.782,19	22	22	10	
AA	64	23.07.2004	422,94	USD	A310-300	280	282	133.661,91	22	22	10	
AA	64	20.08.2004	422,94	USD	A310-300	280	282	132.980,96	22	22	10	

B20 (2) (000) us7211 INS

Figure 24 Example report with the default services visible in the toolbar

```
*... Only Activate Sort Functionality (Group of Functionality/Button for
*Sort Ascending and Button for Sort Descending)
data: gr_functions type ref to cl_salv_functions_list.
gr_functions = gr_table->get_functions( ).
gr_functions->set_group_sort( ).

*... Display Table
gr_table->display( ).
```

Figure 25 Example code to activate a group of related services

In some cases, you may want to activate a group of related services, as shown in the example in **Figure 25**. As in the previous example, start by retrieving the `CL_SALV_FUNCTIONS_LIST` object

```
*... Only Activate Search Functionality (One Function/Button for Search)
data: gr_functions type ref to cl_salv_functions_list.
gr_functions = gr_table->get_functions( ).
gr_functions->set_find( ).

*... Display Table
gr_table->display( ).
```

Figure 26 Example code to activate a single service

from the object model. Then call the SET_GROUP_SORT method to activate all functions related to sorting and make these buttons visible in the toolbar.

Finally, the example code in **Figure 26** shows how to activate a single function. As in the previous examples, first retrieve the CL_SALV_FUNCTIONS_LIST object from the object model. Then call the SET_FIND method to activate the search function and make this particular button visible in the toolbar.

Example #3: Adding application-specific events to a report

The SAP List Viewer is quite often seamlessly embedded into an application, where it provides reporting functionality with its generic functions, but also offers application-specific functionality. For example, you might want users to be able to double-click on a cell of a table to perform some operation or add an application-specific button to the SAP List Viewer toolbar. Suppose you want to show detailed information for a purchase order in the table. In order to implement this feature, you add an application-specific button that implements the functionality to show the detail information of the purchase order. When the button is clicked, the code determines which purchase order was selected and then calls a dialog box to display its details.

In order to react to user interactions, the application must be notified when an interaction occurs. You provide this notification by “firing” events. If an application is registered to a certain event, when that event is raised the application gets control and can react accordingly. You use the CL_SALV_EVENTS_TABLE object to register an application to an event such as a link-click or a double-click.

In this third example (see **Figure 27**), I show you how to program the registration and reaction to the

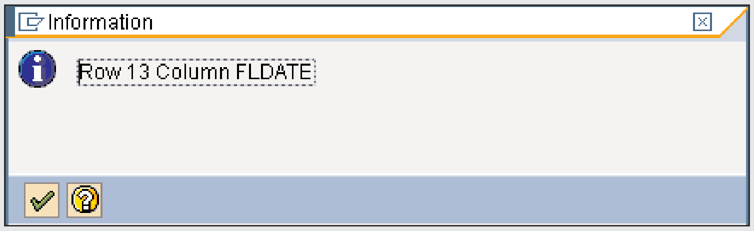
```
report salv_learn_map_table_4 .

*---------------------------------------------------------------------*
*       CLASS lcl_handle_events DEFINITION
*---------------------------------------------------------------------*
```

Figure 27 Example code to implement an event handler

Continues on next page

Figure 27 *continued*

```
class lcl_handle_events definition.
  public section.
    methods:
      on_double_click for event double_click of cl_salv_events_table
        importing row column.
endclass.                    "lcl_handle_events DEFINITION

*---------------------------------------------------------------------*
*       CLASS lcl_handle_events IMPLEMENTATION
*---------------------------------------------------------------------*
class lcl_handle_events implementation.
  method on_double_click.
    message i000(0k) with 'Row' row 'Column' column. "#EC NOTEXT
  endmethod.                    "on_double_click
endclass.                    "lcl_handle_events IMPLEMENTATION

start-of-selection.
*... Select Data
  data:
    gt_outtab type table of sflight.

  select * from sflight into corresponding fields of table gt_outtab.

*... Create Instance
  data:
    gr_table  type ref to cl_salv_table.

  call method cl_salv_table=>factory
    importing
      r_salv_table = gr_table
    changing
      t_table      = gt_outtab.

*... Register to Double-Click Event
  data:
    gr_handle_events type ref to lcl_handle_events.
  data:
    lr_events type ref to cl_salv_events_table.

  lr_events = gr_table->get_event( ).
  create object gr_handle_events.
  set handler gr_handle_events->on_double_click for lr_events.

*... Display Table
  gr_table->display( ).
```



Row 13 Column FLDATE

Figure 28 Dialog box showing the selected row and column

double-click event. Start by looking at the "Register to Double-Click Event" section at the end of the code. First call the `GET_EVENT` method to retrieve the `CL_SALV_EVENTS_TABLE` object from the object model. Then register the desired event, in this case double-click. Now you need to define and implement the event handling, which in this example is displaying the row and column on which the event occurred. A local event handler executes this task, as shown in the "CLASS lcl_handle_events DEFINITION" section at the beginning of the code. Here you define the method `ON_DOUBLE_CLICK` for handling the double-click event that is raised by the `CL_SALV_EVENTS_TABLE` class. Then implement the `ON_DOUBLE_CLICK` method to handle the event, as shown in the "CLASS lcl_handle_events IMPLEMENTATION" section.

Now when a user double-clicks on a cell in the report in the SAP List Viewer, a dialog box appears as shown in **Figure 28** with information on the row and column on which the user double-clicked. The code that produces this dialog box is in the "CLASS lcl_handle_events IMPLEMENTATION" section in Figure 27.

I specifically chose this example because it makes clear that receiving the cell coordinates where a user interaction happened gives the application all the relevant information needed to act further on the identified object. For example, you might now want to display detailed information for this flight date, such as what destinations and airlines are available.

Example #4: Enabling user control of report layouts

The SAP List Viewer gives you the flexibility of allowing users to create and save report layouts during runtime as needed. If layout handling is active, users can save the current report settings (for example, the set of displayed columns, the sort and filter criteria used, and any aggregations executed) to a persistent view with a unique name. You can allow users to save persistent views only for themselves (user-dependent) or for all users (user-independent).

Always allow users to save their own persistent views, which are only visible to them and therefore do not hinder other users. However, only authorize specific users such as administrators to save user-independent persistent views that are visible to all users, since only they know which layouts are meaningful in this business context. The best approach is to programmatically allow all users to save both user-dependent and user-independent layouts. Then use the `S_ALV_LAYOUT` authorization object in the SAP List Viewer to grant certain users the permission to save user-independent views.

The example in **Figure 29** shows how to use the `CL_SALV_LAYOUT` class of the object model to activate layout control. Start by looking at the "Enable ALV Layout Control" section in the middle. First call the `GET_LAYOUT` method to retrieve the `CL_SALV_LAYOUT` object from the object model. This object provides methods for setting the following values:

```
report salv_learn_map_table_5 .

*... Select Data
data:
  gt_outtab type table of sflight.

select * from sflight into corresponding fields of table gt_outtab.

*... Create Instance
data:
  gr_table  type ref to cl_salv_table.

call method cl_salv_table=>factory
  importing
    r_salv_table = gr_table
  changing
    t_table      = gt_outtab.

*... Enable ALV Layout Control
data:
  gr_layout type ref to cl_salv_layout.

gr_layout = gr_table->get_layout( ).

*... Permit/Activate Saving of Layouts - can only be done if a unique key exists
*with which the layouts for this report can be saved
data:
  ls_key type salv_s_layout_key.

ls_key-report = sy-repid.

gr_layout->set_key( ls_key ).

*... Permit Saving of All Types of Layouts
gr_layout->set_save_restriction( if_salv_c_layout=>restrict_none ).

*... Display Table
gr_table->display( ).
```

Figure 29 Example code to activate layout control

Terminology change for user-saved views in SAP R/3 4.6

In SAP R/3 Release 4.0 and 4.5, the term *display variants* was used to refer to user-saved views. In Release 4.6, we decided to change this rather technical term to a more transparent term, *layout*. The name of the class (`CL_SALV_LAYOUT`) in the new object model reflects this name change.

If you are familiar with the function modules of the previous API, note that their parameters still reflect the naming conventions used prior to Release 4.6. For example, the `REUSE_ALV_LIST_DISPLAY` function module is used for displaying tables as standard SAP R/3 lists. Its layout parameter (`IS_LAYOUT`) has nothing to do with the layouts in the new object model that can be saved with a unique name, but instead comprises settings that you make in the new object model with the `CL_SALV_DISPLAY_SETTINGS` and `CL_SALV_FUNCTIONAL_SETTINGS` classes.

- Unique key with which the layouts are to be saved
- Save restriction, which defines whether a user can save user-dependent views, user-independent views, or both types
- Specific layout to be loaded when the SAP List Viewer is first called, which is separate from default layout handling
- Default layout handling, which controls whether layouts can be specified as the default (default layouts are automatically loaded when the SAP List Viewer is first called if no specific layout is defined)

Next look at the code in the “Permit/Activate Saving of Layouts” section. Here you call the `SET_KEY` method and pass a unique key (such as the program name) to activate layout handling. To set the save restriction, call the `SET_SAVE_RESTRICTION` method as shown in the “Permit Saving of All Types of Layouts” section. The parameter for this method defines whether a user can save user-dependent views, user-independent views, or both types. The most commonly used parameter is `NONE`, since you typically implement the usage restriction using the `S_ALV_LAYOUT` authority object.

I hope these examples of how to use the most important metadata objects of the SAP List Viewer API have increased your understanding of the programming paradigm for the new object model. It should now be clear just how quickly and easily you can program the presentation of your data using this new API. In my opinion, the most helpful enhancement is the automatic creation of column objects based on the data table that you pass to the object model. The column object of the SAP List Viewer is very important because it describes the metadata of this data table. In the past, the column metadata was often not defined correctly and caused runtime problems. The SAP List Viewer has now assumed this responsibility, which greatly reduces these types of errors.

Useful tips and techniques

As you have seen, the new object-oriented programming model generally makes life a lot easier for you when programming the SAP List Viewer. I want to leave you with a few tips that I have discovered along the way to help you get the most out of working with the new API:

- Demo programs, including sample code that focuses on specific aspects of programming the SAP List Viewer, such as event or layout handling, are available with SAP NetWeaver ’04. Enter SALV_DEMO_* in the ABAP Editor (transaction SE38) and press F4.

- A new form object is provided for using the metadata-driven interface to design the top-of-list and end-of-list areas of a report. You can find demo programs with sample code by entering SALV_FORM_DEMO* in the ABAP Editor (transaction SE38) and pressing F4. Use these examples to become familiar with the programming model of the form object.

- Take special care when working with an ALV grid control or ALV tree control that is embedded in a Dynpro container. Although the ALV control is embedded in the Process Before Output (PBO) and Process After Input (PAI) processing of the Dynpro container, the SAP List Viewer instance does not know anything about the current PBO or PAI state of the Dynpro container. Therefore, if any user interaction occurs on the Dynpro container outside of the ALV control, the PAI of the Dynpro will be processed. If you now need the selection state of the ALV control in order to process an event based on selected line items, first call the `GET_METADATA` method of the object model in order to manually synchronize the front end and back end. Automatic synchronization would cause roundtrips that are to be avoided for performance reasons. In contrast, when events are triggered within the SAP List Viewer, this method is called internally. The SAP List Viewer knows that synchronization is required because the application has registered this event and needs a consistent state at all times.

- If you want to change the structure and content of the displayed table in the program, do not call the `FACTORY` method again. Instead, call the `SET_DATA` method in order to pass a new data table to the object model and notify the SAP List Viewer to create new metadata because the existing metadata is no longer valid. The SAP List Viewer instance is then reused to display the newly structured data. For internal reasons of front-end/back-end inconsistencies, do not call the `SET_DATA` method in an event handler to avoid unexpected behavior or even runtime errors.

- You can display icons as well as text in a table. However, note that you cannot use icons with the checkbox, hyperlink, or text cell types.[8]

Looking beyond SAP NetWeaver ’04

The new SAP List Viewer object model defines a metadata view of tabular data that is the basis for data display using the SAPGUI technology. The concept of providing an API for displaying tabular data that offers a rich set of generic functions also applies to the new SAP UI technology, Web Dynpro.[9] SAP NetWeaver Release 2004s provides a Web Dynpro List Viewer component that has a very similar metadata model to the object model described in this article. Therefore, the value of your investment in learning how to use the SAP List Viewer will not diminish, even if you switch from the SAPGUI-based environment to the new Web Dynpro UI technology at some point in the future.

Conclusion

The SAP List Viewer is a powerful and flexible tool that enables you to embed reporting and interaction on tabular data directly into your applications. The new object-oriented programming model is easy to learn because of its logical structure. Its ability to be used with all SAP List Viewer variants increases your productivity by offering a generic set of built-in services (sort, filter, etc.) that you simply incorporate into your programs, no code required. You do not have to handle data presentation because the SAP List Viewer provides a common look and feel for all

[8] For more information about cell types, refer to the SAP List Viewer documentation on the SAP Help Portal at http://help.sap.com. Navigate to the SAP NetWeaver documentation and select Application Platform → ABAP Technology → UI Technology → Controls and Control Framework for SAP GUI → SAP List Viewer (BC-SRV-ALV) → ALV Object Model.

[9] The Web Dynpro programming model allows you to develop and model browser-based business applications. It provides a standards-based, device-independent runtime environment, bridging the gap between different platforms such as J2EE, ABAP, and Microsoft .NET, as well as between different Internet browsers and mobile platforms.

reports. Plus, the object model now automatically derives the data type for data tables that are passed to it, freeing you from the often error-prone task of defining metadata for data types.

Not surprisingly, SAP uses the SAP List Viewer extensively for displaying tabular data. In mySAP ERP 2004, you will find approximately 8,000 ALV-based lists in the system. For example, look at actual cost line items for orders (transaction KOB1), cost centers (transaction KSB1), and system monitors such as process overview (transaction SM50). Therefore, the SAP List Viewer is equally valuable from the perspective of user productivity. All lists based on the SAP List Viewer API look and feel the same, which provides the consistency that users expect in order to be able to efficiently perform their business tasks.

So why do I recommend that you take another look at the new and improved SAP List Viewer? Because the bottom line is that it has saved me significant time and effort with my SAP NetWeaver ’04 reporting needs and I am confident that you will experience similar rewards.

Falko Schneider is the Director of Research and Development for SAP NetWeaver Business Intelligence (BI) and is responsible for Data Warehousing, Business Planning, SAP List Viewer, and SAP Query development. He began working for SAP AG in 1994 as an application developer for Controlling Product Costing (CO-PC) where he was responsible for reporting. He designed and developed the first version of the SAP List Viewer, which was used internally at SAP and first shipped to customers with R/3 Release 4.0. During the development of R/3 Release 4.6, he assumed the role of project lead for SAP List Viewer development and was responsible for the control-based SAP List Viewer variant, the ALV Grid Control. In 2000, Falko joined the SAP Business Information Warehouse (BW) team as a development manager for SAP List Viewer and SAP Query development. He also assumed responsibility for Data Warehousing and integrating Business Planning functionality within SAP BW. Besides heading up this part of the development organization for SAP NetWeaver BI, Falko continues to work as a project manager for SAP List Viewer development. He can be reached at falko.schneider@sap.com.