Rich Heilman, Thomas Jung

Next Generation ABAP™ Development

www.sap-press.com
Contents at a Glance

1 Workbench Tools and Package Hierarchy ............ 23
2 Data Dictionary Objects .................................... 43
3 Data Persistence Layer ..................................... 71
4 Consuming a Web Service .................................. 119
5 Shared Memory Objects .................................. 143
6 Model Class ............................................. 163
7 ABAP and SAP NetWeaver Master Data Management .............................................. 193
8 ABAP Unit ............................................. 221
9 Exposing a Model as a Web Service .................... 235
10 Exposing a Model as a Web Service Using SAP NetWeaver Process Integration ............ 249
11 Classic Dynpro UI/ALV Object Model ................. 273
12 Web Dynpro ABAP .................................... 303
13 Business Server Pages .................................. 345
14 Adobe Forms .......................................... 375
15 SAP NetWeaver Portal ................................. 407
16 RSS Feed Using an ICF Service Node ................. 449
17 Closing ............................................... 465
A Code Samples .......................................... 469
B The Authors .......................................... 473
# Contents

Introduction ....................................................................................................................... 15

1 Workbench Tools and Package Hierarchy ................................................................. 23
  1.1 Log on and Explore ................................................................................................. 24
      1.1.1 Workbench Object Browser ........................................................................... 24
      1.1.2 Object Browser List ....................................................................................... 25
      1.1.3 Workbench Settings ....................................................................................... 28
      1.1.4 New ABAP Editor ......................................................................................... 28
      1.1.5 Additional New Workbench Tools ................................................................. 32
      1.1.6 Debugger ........................................................................................................ 36
  1.2 Package Hierarchy for the Project ......................................................................... 39
      1.2.1 Project Requirements ..................................................................................... 39
      1.2.2 Package Hierarchy ......................................................................................... 40

2 Data Dictionary Objects ............................................................................................... 43
  2.1 Designing Data Relationships ............................................................................... 43
      2.1.1 Table Relationship Graphic .......................................................................... 44
      2.1.2 SAP Data Modeler ......................................................................................... 46
  2.2 Data Dictionary Fixed Value Domains ................................................................... 47
      2.2.1 Single Value Domains .................................................................................... 48
      2.2.2 Interval Value Domains .................................................................................. 48
  2.3 Data Dictionary Text Tables ................................................................................... 49
      2.3.1 Data Elements and Domains ......................................................................... 50
      2.3.2 Transparent Table Creation and Relationships ........................................... 51
      2.3.3 Maintenance View .......................................................................................... 56
      2.3.4 Generated Table Maintenance ....................................................................... 58
  2.4 Data Dictionary Data Tables ................................................................................... 61
      2.4.1 Enhancements ................................................................................................ 61
      2.4.2 Indexes .......................................................................................................... 62
      2.4.3 Lock Objects .................................................................................................. 64
  2.5 Search Helps .......................................................................................................... 66

3 Data Persistence Layer ............................................................................................... 71
  3.1 Persistent Objects ................................................................................................... 72
      3.1.1 Creating the Persistent Object Class ............................................................. 73
      3.1.2 Base Classes and Agent Classes ................................................................. 75
Contents

3.1.3 Persistent Data Mapper .............................................. 77
3.1.4 Coding with a Persistent Object ................................. 79

3.2 Exception Classes .......................................................... 82
3.2.1 Advantages of Exception Classes ................................. 83
3.2.2 Creating an Exception Class ......................................... 84

3.3 Business Object Classes ........................................................... 89
3.3.1 What Is a Business Object Class? ................................. 90
3.3.2 Business Object Class Structure ................................... 92
3.3.3 Multiple Object Selection ........................................... 95
3.3.4 Select-Options as a Query Criteria ............................... 97
3.3.5 Complex Business Objects ........................................... 100
3.3.6 Modification Operations ............................................. 103
3.3.7 ZIP Compression ......................................................... 104

3.4 Data Load Programs ........................................................ 107
3.4.1 Test Data Generator .................................................... 107
3.4.2 Backup and Recovery Program ................................. 112

4 Consuming a Web Service ................................................... 119
4.1 Proxy Generation ............................................................. 121
4.1.1 Accessing the WSDL Document .................................. 121
4.1.2 Creating the Client Proxy ............................................ 122
4.1.3 Logical Ports ............................................................... 125
4.1.4 Using the Client Proxy Object ..................................... 127
4.1.5 Implementing into the Business Object Layer .............. 128

4.2 Logical Ports ............................................................................ 130
4.2.1 Runtime ...................................................................... 130
4.2.2 Call Parameters ........................................................... 132
4.2.3 Operations .................................................................. 136
4.2.4 Errors .......................................................................... 136
4.2.5 PI Receiver .................................................................. 137
4.2.6 Application-Specific Global Settings ............................ 138
4.2.7 Application-Specific Operations .................................. 138

4.3 Extended Protocols ................................................................ 140

5 Shared Memory Objects ...................................................... 143
5.1 Getting Started ................................................................. 144
5.1.1 Area Root Class Creation ............................................. 144
5.1.2 Defining the Shared Memory Area .............................. 150
5.1.3 Testing the Shared Memory Object ............................ 152
5.1.4 Shared Memory Monitor ............................................. 154
5.2 Automatic Preloading .............................................................. 155
  5.2.1 Adding the Interface ................................................... 155
  5.2.2 Modifying the Read Program ...................................... 157
5.3 Implementing into the Business Object Layer .................... 157
  5.3.1 Developing a Test Program ......................................... 158
  5.3.2 Modifying the Business Object Class ........................... 159
  5.3.3 Testing the Changes .................................................... 162

6 Model Class .......................................................................... 163
  6.1 Class Overview ............................................................... 163
    6.1.1 What Is Model View Controller? ................................. 164
    6.1.2 Creating the Model Class ............................................ 164
  6.2 Transactional Methods ............................................................ 168
    6.2.1 Read Method .............................................................. 169
    6.2.2 Record Locking Methods ............................................ 171
    6.2.3 Save Method .............................................................. 172
    6.2.4 Getters ........................................................................ 173
  6.3 Utility Methods ....................................................................... 175
  6.4 Emailing .................................................................................. 177
    6.4.1 Email Setup ................................................................. 178
    6.4.2 Running an Email Test ................................................. 182
    6.4.3 Email Method ............................................................. 183
  6.5 Regular Expressions ................................................................. 189

7 ABAP and SAP NetWeaver Master Data Management ....... 193
  7.1 What Is Master Data Management? ......................................... 194
    7.1.1 Technical Architecture .............................................. 194
    7.1.2 Clients ........................................................................ 196
    7.1.3 Basic MDM Administration ....................................... 198
    7.1.4 Modeling in SAP MDM ............................................... 200
  7.2 Configuring the SAP MDM ABAP API .............................. 201
    7.2.1 Installation of the MDM API Add-on .......................... 202
    7.2.2 Configuring the MDM API Connection ........................ 204
    7.2.3 Authentication with the MDM API ............................. 205
  7.3 Coding with the MDM ABAP API ............................................ 206
    7.3.1 Class-Based API .......................................................... 207
    7.3.2 Function Module Based API ........................................ 209
    7.3.3 Non-Unicode ABAP Systems ....................................... 211
    7.3.4 Simple Read ABAP Systems ....................................... 212
Contents

7.3.5 Full Read ................................................................. 215
7.3.6 Integrating the MDM Repository into Your Business
Object Class ......................................................... 217

8 ABAP Unit ................................................................. 221
8.1 Overview of ABAP Unit Tests ...................................... 221
  8.1.1 Test Classes .................................................. 222
  8.1.2 Test Attributes ............................................... 222
  8.1.3 Test Methods ............................................... 225
  8.1.4 Assertions .................................................. 225
8.2 Creating ABAP Unit Tests ........................................... 227
  8.2.1 Creating the ABAP Unit Test Class ....................... 227
  8.2.2 Fixture Implementation .................................... 229
  8.2.3 Test Method Implementations ............................. 229
8.3 Executing the ABAP Unit Test ..................................... 232

9 Exposing a Model as a Web Service ............................. 235
9.1 Web Service Definition ........................................... 235
  9.1.1 Service Definition Wizard ................................. 236
  9.1.2 The Service Definition .................................... 240
  9.1.3 Releasing the Web Service ............................... 241
9.2 Testing the Web Service ........................................... 243
  9.2.1 Web Service Administration .............................. 243
  9.2.2 Using the Web Service Homepage ....................... 244

10 Exposing a Model as a Web Service Using
SAP NetWeaver Process Integration ............................ 249
10.1 Modeling a Service in SAP NetWeaver Process Integration .... 250
  10.1.1 Integration Builder ....................................... 251
  10.1.2 Simple Data Types ....................................... 253
  10.1.3 Complex Data Types ..................................... 259
  10.1.4 Message Types .......................................... 261
  10.1.5 Message Interface ....................................... 263
10.2 Implementing the Service as a Server Proxy .................... 264
10.3 Creating a Service Definition ................................... 270
11 Classic Dynpro UI/ALV Object Model ........................................ 273

11.1 ALV Object Model Overview ........................................... 273

11.1.1 ALV Tool Overview .................................................. 274

11.1.2 Display Types ....................................................... 276

11.2 Getting Started ........................................................... 276

11.2.1 Package Selection .................................................. 277

11.2.2 Basic Program Coding ......................................... 277

11.3 Modifying the ALV Output ............................................. 281

11.3.1 ALV Functions ...................................................... 282

11.3.2 Modifying Column Attributes .................................. 283

11.3.3 Modifying Display Settings ..................................... 285

11.3.4 Saving Layout Variants ......................................... 286

11.4 Making the ALV Interactive .......................................... 287

11.4.1 Adding Buttons ..................................................... 287

11.4.2 Defining Selections ............................................... 289

11.4.3 Event Handling ..................................................... 290

11.5 Object Orientation with Classic Dynpro ............................ 294

11.5.1 Restructuring the Dialog Program ............................. 294

11.5.2 Creating the Controller Class .................................. 296

11.5.3 Enjoy Control Events ........................................... 299

11.5.4 Dynpro Events .................................................... 300

12 Web Dynpro ABAP .............................................................. 303

12.1 Overview of the Components ........................................ 304

12.1.1 Course Frame Component ...................................... 305

12.1.2 Faculty Detail Component ...................................... 305

12.1.3 Faculty List Component ........................................ 307

12.1.4 Course Details Component .................................... 308

12.2 Component Usage ..................................................... 309

12.3 General UI Features .................................................... 311

12.3.1 Value Help .......................................................... 311

12.3.2 Required Fields .................................................... 314

12.3.3 Change/Display Mode ........................................... 317

12.4 ALV Component ........................................................ 320

12.4.1 ALV Component Usage .......................................... 320

12.4.2 Context Mapping .................................................. 321

12.4.3 ALV Implementation ............................................ 322

12.5 Dialog Popup Window ................................................ 325

12.5.1 Windows from the Same Component ....................... 325

12.5.2 Windows from an External Component Usage .............. 327
Contents

12.6 Table Popins ................................................................. 332
  12.6.1 What Is a Table Popin? ........................................... 332
  12.6.2 Designing the Table Popin ....................................... 333
  12.6.3 Context Design ....................................................... 334
  12.6.4 Logic to Populate the Context for a Popin ................. 335

12.7 File Upload/Download .................................................. 336
  12.7.1 File Downloads ....................................................... 336
  12.7.2 File Uploads .......................................................... 339

12.8 Web Dynpro Debugger ................................................. 341

13 Business Server Pages .................................................... 345

  13.1 Internet Facing BSP Application ................................. 346
    13.1.1 Stateless versus Stateful ...................................... 346
    13.1.2 Application Layout ............................................. 348
    13.1.3 Custom Style Sheets ........................................... 349
    13.1.4 Course Overview Page ......................................... 353
    13.1.5 AJAX ................................................................. 356

  13.2 BSP Extensions .......................................................... 363
    13.2.1 Upload of Open Source Solution ............................ 363
    13.2.2 Creating the BSP Extension .................................. 365
    13.2.3 Creating the BSP Extension Element ....................... 366
    13.2.4 Design Time Validation ....................................... 369
    13.2.5 Runtime Validation ............................................. 370
    13.2.6 Element Rendering ............................................. 371
    13.2.7 Testing the New Extension ................................... 373

14 Adobe Forms .............................................................. 375

  14.1 Infrastructure and Setup .......................................... 376
    14.1.1 Adobe Document Services Infrastructure ................. 376
    14.1.2 Exposing the Service from the J2EE Engine ............. 377
    14.1.3 Configuring the Service Interface from ABAP .......... 378

  14.2 Function Module Based Forms .................................... 381
    14.2.1 Creating the Interface ....................................... 381
    14.2.2 Form Interface to Context Mapping ....................... 383
    14.2.3 Form Layout Editor ........................................... 387
    14.2.4 Coding Against the Form Function Module .............. 389

  14.3 Web Dynpro Based Forms .......................................... 395
    14.3.1 Web Dynpro View Creation .................................. 395
    14.3.2 Form Design from Web Dynpro ............................... 397
Contents

14.3.3 Table Output in Forms ................................................ 398
14.3.4 Making the Form Interactive ....................................... 401
14.4 Offline Forms ................................................................. 402

15 SAP NetWeaver Portal ......................................................... 407
  15.1 Creating a System Configuration ...................................... 407
     15.1.1 User Mapping ..................................................... 414
     15.1.2 System Test ......................................................... 415
  15.2 Creating Portal Content ................................................... 416
     15.2.1 iView Creation .................................................... 416
     15.2.2 Role Assignment ................................................. 420
     15.2.3 Running Examples ............................................... 421
  15.3 Portal Eventing ............................................................. 424
     15.3.1 Throwing a Portal Event ...................................... 425
     15.3.2 Catching a Portal Event ....................................... 427
  15.4 SAP NetWeaver Visual Composer ...................................... 429
     15.4.1 Consuming a Web Service ................................... 431
     15.4.2 Consuming an RFC ............................................. 432
     15.4.3 Building the User Interface .................................. 436
     15.4.4 Building Value Help ............................................ 437
  15.5 SAP NetWeaver Business Client ....................................... 443

16 RSS Feed Using an ICF Service Node ................................. 449
  16.1 What Is an ICF Service Node? ....................................... 450
  16.2 Handler RSS Feed ........................................................ 451
      16.2.1 HTTP Handler Class Test Implementation ............ 452
      16.2.2 ICF Node Creation and Handler Association .......... 453
      16.2.3 RSS Handler Implementation ................................ 454
      16.2.4 RSS Handler Application Logic ............................ 455
      16.2.5 Handler for Attachment Downloads ...................... 461

17 Closing .............................................................................. 465

Appendix .................................................................................... 467
  A Code Samples ....................................................................... 469
  B The Authors ........................................................................... 473

Index .......................................................................................... 475
Introduction

This book represents 20 years of collective experience from real world ABAP Development projects. When setting out to write this book, our challenge was to share what it was like to be part of a cutting edge ABAP development project with our readers.

Ideally, our goal was for each reader to be able to sit down and observe an entire project from start to finish, and therefore learn the integral techniques of modern ABAP development. They would be able to see the latest ABAP technologies in action, in addition to examining the design and development processes used to maximize these technologies. Unfortunately, few developers ever get the opportunity to observe a project in this way. Too often they have to learn on the job, while dealing with unrealistic deadlines. Therefore, our objective was to allow you, the reader, to see and participate in the evolution of such a project in small incremental steps.

For that reason, this book is not your typical programming guide. Instead of focusing on just the technological aspects of developing in ABAP, we will study a fictional project so you can see how a project is developed. Each chapter will represent a phase or layer of this project’s development, as well as one or two new key ABAP technologies. If you’re interested in delving into these new technologies straightaway, you’re welcome to skip to these respective chapters; however, we, the authors, encourage you to read this book in chronological order so that you’ll have an opportunity to see the entire narrative of this project play out.

Fictional Project

Throughout the book, we’ll be using a single fictional project for our practice scenario. This project takes place at a university, which is a long time SAP customer who runs their business systems on SAP R/3. For over four years, they’ve been running SAP R/3 4.6C and have used the Finance and Human Resources modules of SAP R/3 extensively, in addition to custom developing many modules of their own.
Introduction

This university is in the middle of a typical upgrade cycle. They have begun the process of updating their SAP R/3 4.6C system to SAP ERP 6.0 (formerly named mySAP ERP 2005). SAP ERP 6.0 will run on top of SAP NetWeaver — specifically SAP NetWeaver 7.0 (formerly named SAP NetWeaver 2004s). They are also in the process of implementing the SAP NetWeaver Portal, as well as considering using SAP NetWeaver Process Integration (SAP NetWeaver PI — formerly known as SAP NetWeaver Exchange Infrastructure or SAP XI) and SAP NetWeaver Master Data Management (SAP NetWeaver MDM) in the near future.

This university has a small but strong IT team. Our story will focus on Russel, the lead developer of the IT team at this university. Russel has many years of experience in ABAP development to support the university’s systems. Like many developers, he reads about the latest ABAP development technologies and techniques, but is somewhat constrained by the release level of the university’s R/3 system. Consequently, he feels that his development skills are not up to date; for example, he has done very little ABAP Object-Oriented (ABAP OO) programming and has virtually no web-based development. Still, Russel is quite excited about the future upgrade to SAP ERP 6.0. He sees this as an opportunity to update his skills and learn about the newest ABAP development techniques.

Little does Russel know that he is about to get a crash course in ABAP development on SAP NetWeaver 7.0. In addition to the upgrade activities, the university is just beginning to offer a new distance learning curriculum. Like new offerings at many universities, this distance learning curriculum will offer online versions of many courses for people looking to complete their degrees, or take part in continuing education without disrupting their current career.

In support of this new curriculum, the university realizes that it will need significant new custom development. Their ERP system will house this development and ABAP will be the language in which the system is developed. This project will enable Russel to build the data access, business logic, and user interface aspects of this new system. This project will also be the first time that Russel will build something that entails enterprise service-oriented architecture (enterprise SOA).

Please note again that the context for the project that we’re going to study throughout this book is fictional. It does not feature an actual university or SAP customer. The characters that we will meet, like Russel, are not real people; however Russel’s experiences and reactions to events are based on our
Introduction

(i.e., the authors) experiences, and hopefully will touch a familiar chord with many of you.

Structure of the Book
The structure of this book reflects the workflow of the development project. The first half of the book focuses on creating the data and application logic layers and then service-enabling them. The second half of the book focuses on creating the user interface layers.

- Chapter 1: Workbench Tools and Package Hierarchy
  Before we begin our project, we will review some of the changes and enhancements to the ABAP Workbench. In this chapter, we will look at the new ABAP Editor, the Refactoring Assistant, the new development tools perspectives in transaction SE80, and the new debugger. Lastly, we will create the packages and package hierarchies for the project.

- Chapter 2: Data Dictionary Objects
  In this chapter, we will model the data relationships and build the corresponding Data Dictionary objects. We’ll study the tools for generating table maintenance, creating lock objects, and utilizing foreign keys. We’ll also explore the new technology of strings and binary strings within transparent tables.

- Chapter 3: Data Persistence Layer
  In this chapter, we’ll build the logic that controls the persistence of application data. We’ll start by generating persistent object classes for the underlying data dictionary tables created in Chapter 2. Then, we’ll build a set of business object classes to hide the inner technical details of the persistent objects. In addition to the new technology of Persistent Objects, we’ll show you how to use ZIP compression on large strings.

- Chapter 4: Consuming a Web Service
  Not all project data will originate from one centralized system. For example, in the sample application, some data will be stored in a legacy system and accessed remotely via Web Services. In this chapter, we will examine the process for generating a Web Service proxy object and integrating this proxy into the data persistence layer.

- Chapter 5: Shared Memory Objects
  After some analysis, it will become apparent that the sample application has some static data that will be accessed repeatedly. In this chapter, we
will describe how you can provide the best performance by structuring the data access for this type of data into an ABAP Shared Memory Object.

- **Chapter 6: Model Class**
  In this chapter, we will begin to implement the core application logic, which is implemented as a Model Class. This same class will later be used as the business logic layer of all the UI technology examples. This chapter focuses primarily on object-oriented design patterns while introducing techniques for sending email and manipulating XML.

- **Chapter 7: ABAP and SAP NetWeaver Master Data Management**
  This is the first of our “What-If” chapters. Here, we look at an alternative approach to the project where our master data is modeled and stored in SAP NetWeaver Master Data Management, instead of the local Data Dictionary. This chapter will focus on how we would alter the data persistence layer to read this data via the SAP NetWeaver MDM ABAP application programming interface instead of directly from the local database.

- **Chapter 8: ABAP Unit**
  Before building any additional objects on top of the existing application logic, this is a good point in the project to unit test what has been completed. In this chapter, we'll look at the built-in unit test tool, ABAP Unit, and examine how unit test classes can be integrated directly into the model class.

- **Chapter 9: Exposing a Model as a Web Service**
  Not all the logic from the sample model class will be exposed via a user interface. Instead, some of the data was designed to be exposed as a Web Service so that it can be accessible to external systems as well. In this chapter, we'll examine the Inside-Out approach for generating Web Services.

- **Chapter 10: Exposing a Model as a Web Service Using SAP NetWeaver Process Integration**
  This is the second of the two “What-If” chapters. In the previous chapter, we looked at the Inside-Out approach of generating Web Services via remote enabled function modules. In this chapter, we'll look at the world of Enterprise Service Modeling. We'll show you how the same logic could be modeled in SAP NetWeaver Process Integration and then implemented as a server proxy in ABAP using the Outside-In approach.

- **Chapter 11: Classic Dynpro UI/ALV Object Model**
  In this chapter, we turn our attention to user interface logic. In the sample application requirements, there are a group of internal users who are full time SAP GUI users and who need powerful reporting tools. Therefore,
we’ll learn how to build a classic Dynpro screen on top of the Model View Controller, which uses the ALV Object Model for its reporting output.

- **Chapter 12: Web Dynpro ABAP**
  Since most of the sample application’s users are not SAP GUI users, we’ll look at how you can build a Web Dynpro user interface for these users. This chapter will focus on real world Web Dynpro applications that contain multiple component usages, ALV integration, and table popins.

- **Chapter 13: Business Server Pages**
  The next user interface use case is for an Internet-facing application. This user interface needs to be highly customized and stateless for scalability. Therefore, in this chapter, we will use Business Server Pages in order to show the flexibility they provide for highly customized style sheets and AJAX integration.

- **Chapter 14: Adobe Forms**
  Adobe Forms technology offers an interesting paper-like alternative user interface. In this chapter, we’ll look at each of the major types of Adobe Forms — print forms, online interactive forms, and offline interactive forms.

- **Chapter 15: SAP NetWeaver Portal**
  Although we have focused on ABAP as the primary development environment until now, it is also important to see how some of the SAP NetWeaver Portal technologies can be used with the best aspects of ABAP. In this chapter, we’ll explore how to wrap each of our user interface examples in iViews within the SAP NetWeaver Portal and how portal eventing can be used for cross iView communication. We’ll also look at how we can use SAP NetWeaver Visual Composer to build code-free applications that consume ABAP services.

- **Chapter 16: RSS Feed Using an ICF Service Node**
  In this chapter, we’ll examine how Internet Communication Framework Service Nodes can be combined with XML processing in ABAP to produce interesting Web 2.0 type projects. As the final example of the book, we’ll implement an RSS Feed using these technologies.

- **Chapter 17: Closing**
  In the final chapter, we will look back on the completed project and review the most important points of what has been discussed.
Prerequisites

Whether you are relatively new to ABAP development or an experienced veteran, there is something in this text for everyone. We do, however, assume that the reader is already familiar with the ABAP Workbench and has some development experience in ABAP as of the 4.x release level. We will primarily focus on new techniques and tools that were introduced in the 6.x and higher releases.

The state of the ABAP development environment described in this book is SAP NetWeaver 7.0 SPS10. As SAP ERP 6.0 has been announced to be the primary release of ERP for customers through 2010, SAP expects this to become the "go-to" ERP release for many years to come. Therefore, capabilities of ABAP in SAP NetWeaver 7.0 will likely become the base-line technology level for most customer development as well.

If you don’t already have access to a SAP NetWeaver 7.0 system, you can always download the free trial edition from the SAP Developer Network (https://www.sdn.sap.com/irj/sdn/downloads). This trial software has a full ABAP development environment, enabling you to recreate nearly all the examples contained within this book.

To help you follow along with the project as it unfolds in this book, we have also provided you with the source code for all examples in the book, as well as many supporting objects that are not discussed in detail on the accompanying CD. This should help to facilitate your skipping certain chapters if you want, without having to forego the prerequisite objects.

The source code on the CD is available in several different formats:

- First, there is a transport file. This is the simplest way to import all the development objects that are discussed in this book in their correct packages.
- Not all developers have the necessary security to import a transport file. For this reason, we have also included many of the development objects in SAPlink format (the open source XML based mechanism for exchanging ABAP development objects) and plain text files.

For complete instructions on how to work with each of these import formats, see the ReadMe.pdf file in the root directory of the CD or Appendix A.

In case you were wondering, please note that we won’t forget about older releases just because our primary focus is on SAP NetWeaver 7.0. The technologies that we’ll discuss were primarily released since SAP R/3 4.6C. As we
introduce each technology, we will try to indicate in which release it was first introduced, and, what differences, if any, there are between the releases.

As of SAP NetWeaver 7.0, SAP’s ABAP foundation developers have not stopped innovating around the ABAP environment. As you read this book, dedicated teams are currently working on additional features and powerful new functions for the ABAP development environment. We will point out these anticipated features throughout the book; however they will simply be identified with the notation “Future Functionality.”

With the direction of SAP ERP 6.0, SAP NetWeaver 7.0 will be an established release for many years to come. Therefore, some of this future functionality might find its way into SAP NetWeaver 7.0 via backports of the functionality delivered with support packages. Other new features may be too extensive to deliver in this way, and therefore be postponed until the next major release of SAP NetWeaver, or some other, as yet undetermined, delivery mechanism.

Acknowledgments

I would like to thank my wife, Shonna, for her love and support during the entire project, and for understanding how important this project was to me. Additionally, I would like to thank my kids, Kearston and Gavin, for their unconditional love and understanding while Daddy was working. Thanks for putting up with the laptop on the table during dinner and the shortened play time. Without my family’s support, the past few months would have been much more difficult.

Thanks to my parents, for giving me the necessary foundation that enabled me to be successful in life. Thanks to my sister, Angie, for her inspiration and support, which gave me the “can-do” attitude that this project required.

I would also like to thank our editor, Stefan Proksch, for enabling me to share my knowledge with the rest of the world. Thanks to the SDN community for supplying great content that allowed me to learn directly from the experts. Last but certainly not least, I would like to thank my co-author, Thomas Jung, for the opportunity to work together on this project.

York (PA), May 2007

Rich Heilman
Introduction

I also must start off by thanking my wife, Shari. Without her support, I certainly couldn’t have completed the work required to create this book. What is even more amazing to me is that this time, she knew exactly what she was getting herself into and yet, she still agreed to let me work on the project. As with all accomplishments in my life, they simply would not have been possible without her love and support!

To my children, Megan and Madison, I owe my thanks as well. To them, it probably seemed like Daddy was hardly around for the last few months, since most nights and weekends he disappeared into his office.

The main character in the book, Russel, is named after my father. It is ironic that when we are teenagers, we want nothing to do with our parents, but as we grow older, we realize that the greatest compliment is hearing someone say how much we are like our parents. Mom and Dad gave me so much while I was growing up. I only wish they could be here today so that I could thank them.

To my friend, Brian McKellar, thanks for getting me started writing on SDN and giving me the opportunity to learn from you during our first book project together. I do and will carry those lessons with me in everything I do.

To the old gang at Kimball, there certainly is a little bit of each of you in this book, as I learned so much from everyone that I have had the opportunity to work with over the years.

To my new colleagues in SAP NetWeaver Product Management, for a virtual team who only sees one another a few times a year, we are an amazingly close-knit group. Everyone is absolutely wonderful to work with and so willing to share his or her knowledge.

To my friend and coworker, Peter McNulty, even before I decided to come to SAP, I figured if there were people this good in Product Management, then it was certainly an organization of which I wanted to be a part. Peter is always available to pitch ideas off of, and our discussions have influenced more than a few sections of this book.

To our editor, Stefan, thanks so much for giving us the opportunity to create this book. Your guidance and support have been instrumental in its completion.

Lastly, to my co-author, Rich, it has been a real pleasure working with you on this project, and to think it all started because you accepted an invitation to co-present with me at SDN Day at the SAP TechEd in Las Vegas in 2006.

Jasper (IN), May 2007

Thomas Jung
Russel has finished the programming for the database access layer. He now wants to optimize the read access to some of the data that is read frequently, but not updated often. For this, he will turn to the new shared memory objects technology that was introduced in SAP NetWeaver 2004.

5 Shared Memory Objects

Shared memory objects are ABAP Object Instances, which can be stored in the shared memory area on an application server. Instead of going to the database to retrieve the required data, the data is accessed through the shared memory, thereby providing faster data retrieval.

This shared memory area can be accessed by all of the ABAP programs running on that application server. Before the upgrade to SAP ERP 6.0, Russel used the EXPORT/IMPORT statements with the SHARED BUFFER or SHARED MEMORY extensions to access a similar memory buffer area. So what are the advantages of using this new functionality?

- First, it is read access to shared memory without the need to copy it into user session memory. Technically, an application does a remote attach to the memory segment within shared memory and directly interacts with it.
- Secondly, the new shared memory technique is implemented through ABAP Objects; therefore, you are provided with robust tools to interact with shared memory through code. Ultimately, you aren’t just buffering raw sets of data; you’re also providing a shared mechanism to access the business logic wrapped around this data.
- There are also dedicated tools for the monitoring and administration of these shared areas and the objects within them. Transaction SHMM, for example, provides tools to monitor the size and number of objects within a shared area, as well as enabling administrators to force objects out of memory if necessary.
5.1 Getting Started

Russel has spent a considerable amount of time developing the database access layer for this project and wants to ensure that performance is at an optimal level. He decides to leverage the shared memory objects functionality to increase performance when accessing some of the data in the database.

To use this feature of the ABAP runtime environment, Russel will have to create several new types of objects. Shared memory objects are implemented in two parts — the shared object root and area classes.

- The root class is the definition of the object that will be stored in shared memory. An instance (or multiple instances) of this class will reside in shared memory. Therefore this class’s attributes should represent the data that you want cached and the methods of the class are the way that you access this data.

- The shared memory area class, on the other hand, will be a generated class. It abstracts a section of shared memory that is set aside for one or more instances of a particular root class. The methods of this area class provide the tools to attach to the shared memory area in order to read or manipulate it. The sole purpose of the area class is to return instances of the root class.

5.1.1 Area Root Class Creation

Russel decides that the ZCS_COURSE table would be a good candidate to create a shared memory object. Shared memory objects should primarily be used for objects that are read often, but updated infrequently. This is due to the locking mechanism that is used by shared objects. Although having multiple read locks across separate user sessions is possible and is the norm, any form of change lock is exclusive (i.e., it doesn’t even allow parallel read locks on the same area instance).

This does make ZCS_COURSE a good fit. New courses are rarely created or changed during the school year. All updates are done all at once, before planning for the next semester begins. Technically, this means that this table will have frequent read accesses by students and teachers concurrently, but the data will rarely change.

Russel’s first step in implementing a shared memory object to represent ZCS_COURSE is to create the area root class. This class implements the setter and getter methods, which are used to access the data to be stored in the shared
memory area. It could also include business logic that further manipulates the data during access operations. For instance, it might include calculations, the results of which could also be stored in shared memory. This is where the value of the shared memory object can extend well beyond the scope of just the buffering of data stored within the database.

Russel creates the class \texttt{ZCL\_CS\_COURSE\_SHMO\_ROOT} and assigns it to the \texttt{ZCS\_DDIC} package using transaction code SE80 (see Figure 5.1).

Russel then sets the Shared Memory-Enabled checkbox on the Properties tab (see Figure 5.2). This tells the system that the class is eligible to be used as a root class for a shared memory object.

The idea of using shared memory objects is to store data in memory, which can be used at runtime. Therefore, Russel needs to add an attribute to this class that will hold the data retrieved from the \texttt{ZCS\_COURSE} database table.

Although it is technically possible to create public attributes of the root class that can be accessed directly from an instance of the class, Russel wants to follow good object-oriented designs and encapsulate all of his attribute accesses within methods. This gives him more control in case he wants to embed other operations within an access to this attribute. Therefore he defines the attribute as a Private Instance attribute (see Figure 5.3).
The class now requires methods that can be used to populate or read this attribute. To start, Russel needs a **set** method, which will be used to fill the `COURSE_LIST` attribute with all records in the database table. This method should be defined as a **Public Instance** method (see Figure 5.4).

---

**Figure 5.2** Root Class Properties

**Figure 5.3** Define Attribute

**Figure 5.4** Define SET_COURSE_LIST Method
In the implementation of the \texttt{SET\_COURSE\_LIST} method, Russel leverages the persistent object for database table \texttt{ZCS\_COURSE} to fill the instance attribute \texttt{COURSE\_LIST}. As shown in Listing 5.1, Russel is simply borrowing some of the persistent object code from the method \texttt{READ\_ALL\_COURSES} of the class \texttt{ZCL\_CS\_COURSE} that he wrote in Chapter 3. He then loops through the objects and populates the returning parameter with the values.

\begin{verbatim}
METHOD set_course_list.
  DATA: l_agent TYPE REF TO zca_cs_course_pers,
        l_pers_obj TYPE REF TO zcl_cs_course_pers,
        l_objects TYPE osreftab.
  FIELD-SYMBOLS: <wa_object> LIKE LINE OF l_objects,
                  <wa_course> LIKE LINE OF course_list.
  DATA: query_manager TYPE REF TO if_os_query_manager,
        query TYPE REF TO if_os_query.
  TRY.
    l_agent = zca_cs_course_pers->agent.
    query_manager = cl_os_system->get_query_manager( ).
    query = query_manager->create_query( ).
    l_objects =
      l_agent->if_os_ca_persistency~get_persistent_by_query( 
        i_query = query ).
    IF LINES( l_objects ) = 0.
      RAISE EXCEPTION TYPE zcx_cs_course
        EXPORTING
          textid = zcx_cs_course->bad_query.
    ENDIF.
    LOOP AT l_objects ASSIGNING <wa_object>.
      l_pers_obj ?= <wa_object>.
      APPEND INITIAL LINE TO course_list
        ASSIGNING <wa_course>.
      <wa_course>-syllabi = l_pers_obj->get_syllabi( ).
      <wa_course>-cost = l_pers_obj->get_cost( ).
      <wa_course>-course_id = l_pers_obj->get_course_id( ).
      <wa_course>-course_schedule =
        l_pers_obj->get_course_schedule( ).
      <wa_course>-course_sdesc =
        l_pers_obj->get_course_sdesc( ).
      <wa_course>-course_year =
        l_pers_obj->get_course_year( ).
      <wa_course>-credit_hrs = l_pers_obj->get_credit_hrs( ).
\end{verbatim}
Shared Memory Objects

Russel also needs to define the GET methods, which will be used to retrieve the data. First, Russel needs a GET method to retrieve all the courses. The signature of this method will contain a RETURNING parameter, which is defined as the table type ZCS_COURSES_TT (see Figure 5.5 and Figure 5.6).

Listing 5.1 SET_COURSE_LIST Method Implementation

```csharp
<wa_course>-currency = l_pers_obj->get_currency( ).
<wa_course>-deletion_flag =
    l_pers_obj->get_deletion_flag( ).
<wa_course>-description =
    l_pers_obj->get_description( ).
<wa_course>-end_time = l_pers_obj->get_end_time( ).
<wa_course>-faculty_id = l_pers_obj->get_faculty_id( ).
<wa_course>-major = l_pers_obj->get_major( ).
<wa_course>-semester = l_pers_obj->get_semester( ).
<wa_course>-start_time = l_pers_obj->get_start_time( ).
<wa_course>-student_limit =
    l_pers_obj->get_student_limit( ).
ENDLOOP.
ENDTRY.
ENDMETHOD.
```

Figure 5.5 Define GET_COURSE_LIST Method

Figure 5.6 Define GET_COURSE_LIST Method Signature
Of course as soon as Russel uses a returning parameter, he negates one of the advantages of the shared memory object, namely, the copy free read. Imagine if you had a very large table that could either be exported to shared memory or placed in a shared memory object. In this example, you want to sort the internal table and then read a subset of the records.

With an internal table that was simply exported to shared memory, the entire table would have to be imported before any operations could be performed on it. This entails making a copy of the entire internal table and placing it into the internal session of the running application.

With a shared memory object, however, all of this logic could be placed within the shared object root class and only the resulting few records would be returned. This prevents you from having to copy anything, but the result set, out of shared memory and into the internal session.

In the shared object root class that Russel is building, he needs to support both kinds of accesses. He will eventually build a method that returns a single record, but some applications also need access to the entire course listing. For these applications, it doesn’t make sense to keep a constant read attachment to the shared object instance, therefore, he decides to return a copy of the entire internal table attribute. Returning parameters are always marked as Pass by Value (see third column in Figure 5.6), making this copy operation happen automatically.

The GET_COURSE_LIST has a very simple implementation. Russel only needs to pass the instance attribute COURSE_LIST to the RETURNING parameter RE_COURSE_LIST (see Listing 5.2).

```
METHOD get_course_list.
  re_course_list = course_list.
ENDMETHOD.
```

Listing 5.2 GET_COURSE_LIST Method Implementation

Additionally Russel needs a GET method, which will be used to get a single course record. By clicking on the Parameters button, the signature of the method is displayed. The signature of this method contains an IMPORTING parameter for the COURSE_ID, which will be used to select the specific course. The second parameter is a RETURNING parameter, which will be used to return the course data. This RETURNING parameter is typed like ZCS_COURSE (see Figure 5.7 and Figure 5.8).
Again, the implementation for the `GET_COURSE` method is fairly simple. A simple read statement will read the `COURSE_LIST` attribute and return the corresponding row based on the `IMPORTING` parameter `COURSE_ID` (see Listing 5.3).

```abap
METHOD get_course.
  READ TABLE course_list INTO course
    WITH KEY course_id = course_id.
ENDMETHOD.
```

**Listing 5.3** `GET_COURSE` Method Implementation

### 5.1.2 Defining the Shared Memory Area

Russel now needs to create the shared memory area. The transaction code `SHMA` allows you to create the area and define its properties. When the shared memory area is created, a global class with the same name as the area is created automatically. Therefore, we recommend that you use the standard naming convention for classes, `CL_*` or `ZCL_*`, to name the memory area. This shared memory area class inherits from the class `CL_SHM_AREA`, which is a sub-class of `CL_ABAP_MEMORY_AREA`, giving it all the necessary methods for accessing area root class.

Russel uses transaction `SHMA` to create the shared memory area. The subsequent screen allows him to specify the properties of the area as well as the
root class that this area will be defined for (see Figure 5.9). For now Russel will leave the default properties that were suggested, no limits on the area size, lifetime, or number of versions. Later you will see how he can use some of these properties to set up automatic initialization of his shared object on the first read request.

![Figure 5.9 Area Properties](image)

Now that the area class has been generated, Russel can look at the public methods that he will use to access the shared memory object (see Figure 5.10). The ATTACH methods will return area handles, which are instances of the area class.

![Figure 5.10 Methods of the Area Class](image)
For example, the \texttt{ATTACH\_FOR\_READ} method will return an area handle, which can then be used to read the shared memory area. Similarly, the \texttt{ATTACH\_FOR\_WRITE} method will return an area handle, which will allow you to write to the shared memory area. The \texttt{DETACH\_AREA} method removes the binding between the area class and the area handle.

### 5.1.3 Testing the Shared Memory Object

Russel wants to see the shared memory object in action before trying to use it directly in the rest of the course system. He decides to develop several short test programs to get a feel for how it all works. The first program will be a test write program, which will create the area instance of the area root class and place it into the shared memory area (see Listing 5.4).

```
REPORT zcs_course_shmo_write.

DATA: course_handle TYPE REF TO zcl_cs_course_shmo,
     course_root TYPE REF TO zcl_cs_course_shmo_root.

TRY.
  course_handle = zcl_cs_course_shmo=>attach_for_write( ).
  CREATE OBJECT course_root AREA HANDLE course_handle.
  course_handle->set_root( course_root ).
  course_root->set_course_list( ).
  course_handle->detach_commit( ).
  CATCH cx_shm_attach_error.
  ...
ENDTRY.

Listing 5.4 Write Test Program
```

Notice that fairly normal conventions are used for creating the \texttt{COURSE\_ROOT} instance. Russel still uses the \texttt{CREATE OBJECT} syntax, but now with the new addition \texttt{AREA HANDLE}. These extra statements direct the ABAP runtime to instantiate the root class within shared memory instead of the internal session memory.

Russel writes a second program to test the reading of the data from the shared memory object (see Listing 5.5). This test program will allow Russel to ensure that the \texttt{GET\_COURSE\_LIST} method and the \texttt{GET\_COURSE} method work properly. Before Russel runs this program, he must run the write program to load the memory area. Otherwise, he'll get an ABAP short dump when trying to access an unloaded memory area.
REPORT  zcs_course_shmo_read.

DATA: course_handle TYPE REF TO zcl_cs_course_shmo.
DATA: gt_courses TYPE zcs_courses_tt.
DATA: gs_courses TYPE zcs_course.

PARAMETERS: p_rad1 RADIOBUTTON GROUP grp1 DEFAULT 'X'.
PARAMETERS: p_rad2 RADIOBUTTON GROUP grp1.
PARAMETERS: p_csid TYPE zcs_course-course_id.

AT SELECTION-SCREEN.
  IF p_rad2 = 'X'
    AND p_csid IS INITIAL.
    MESSAGE e001(00) WITH 'Enter a course id'.
  ENDIF.
ENDAT.

START-OF-SELECTION.

TRY.
  course_handle = zcl_cs_course_shmo=>attach_for_read( ).
  CATCH cx_shm_attach_error.
ENDTRY.

CASE p_rad1.
  WHEN 'X'.
    gt_courses = course_handle->root->get_course_list( ).
  WHEN OTHERS.
    gs_courses = course_handle->root->get_course( p_csid ).
    APPEND gs_courses TO gt_courses.
ENDCASE.

course_handle->detach( ).

LOOP AT gt_courses INTO gs_courses.
  WRITE:/ gs_courses-course_id,
    gs_courses-course_sdesc+0(20),
    gs_courses-faculty_id,
    gs_courses-semester,
    gs_courses-course_year,
    gs_courses-major,
    gs_courses-credit_hrs,
    gs_courses-student_limit,
    gs_courses-deletion_flag,
    gs_courses-start_time,
    gs_courses-end_time,
Shared Memory Objects

Listing 5.5 Read Test Program

5.1.4 Shared Memory Monitor

The shared memory monitor provides an interface in which you can monitor the area instances in the shared objects memory. The monitor allows you to view areas, area instances, versions, and locks. Drill-down functionality allows you to drill into these overviews via double-clicking on them.

Russel goes to transaction SHMM to check that the data has been written to the shared memory area by his test applications. He can see that there is one instance of the area class stored in the shared memory area ZCL_CS_COURSE_SHMO (see Figure 5.11). He can also see memory usage, number of instances, number of versions, and the status breakdown of the versions.

Figure 5.11 Shared Memory Monitor — Areas

If the COURSE_LIST attribute of the area root class was defined as Public, Russel could also see the data that is currently stored in the shared memory object. Private attributes, however, are not visible. This is also where he can delete shared memory areas.

For developers, the ability to delete a shared memory area within this transaction is probably one of the shared memory monitor’s most useful functions. If you make any changes to the coding of the root class and reactivate it, the class will be given a new generation timestamp. The generation timestamp of the root class definition in the database is checked by the area class, whenever an access is made.

Therefore, if you make any changes to the root class after it has been stored within a shared area, this will cause an invalid version exception to be thrown every time you try to access the area. After each change to the root
class, you will have to delete any and all versions of the shared memory area before you can test your changes.

5.2 Automatic Preloading

Russel has reviewed what he has learned so far about shared memory objects and realizes that there are some weaknesses in his test applications. For example, it could be problematic if the shared memory area was read before it had been instantiated via a write operation. In other words, reading an unloaded shared memory area will only result in a short dump.

This can occur after the application server has been shut down and restarted. The shared memory areas are all cleared at this time. For the best reliability of his applications, Russel needs to find a way to preload the memory area at the time of the first read. Fortunately, the shared memory object implementation that SAP supplies has just the optional functionality he needs.

5.2.1 Adding the Interface

In order to take advantage of this functionality, he first must add the interface, IF_SHM_BUILD_INSTANCE, to the area root class ZCL_CS_COURSE_SHMOROOT. Once the interface is added, the BUILD method appears in the Methods tab (see Figure 5.12). This static method is automatically fired if any of the ATTACH methods of the area class are called and the shared memory area has not been loaded.

Now Russel can copy and paste the code, which he wrote in the write test program ZCS_COURSE_SHMWRITE into the BUILD method (see Listing 5.6). This not only sets up the technical initialization of the root instance within
the area, but also provides an opportunity to preload all the data from the database via the call to the `SET_COURSE_LIST` method.

METHOD if_shm_build_instance~build.

DATA: course_handle TYPE REF TO zcl_cs_course_shmo,
    course_root TYPE REF TO zcl_cs_course_shmo_root,
    excep TYPE REF TO cx_root.

TRY.
  course_handle = zcl_cs_course_shmo=>attach_for_write( ).
  CATCH cx_shm_error INTO excep.
  RAISE EXCEPTION TYPE cx_shm_build_failed
         EXPORTING previous = excep.
ENDTRY.

TRY.
  CREATE OBJECT course_root AREA HANDLE course_handle.
  course_root->set_root( course_root ).
  course_root->set_course_list( ).
  course_handle->detach_commit( ).
  CATCH cx_shm_error INTO excep.
  RAISE EXCEPTION TYPE cx_shm_build_failed
         EXPORTING previous = excep.
ENDTRY.

IF invocation_mode = cl_shm_area=>invocation_mode_auto_build.
  CALL FUNCTION 'DB_COMMIT'.
ENDIF.

ENDMETHOD.

Listing 5.6 Build Method Implementation

Simply adding the `BUILD` method is not enough to have it triggered by the area class. Russel must return to transaction SHMM and adjust the properties on his area. He needs to set the flag for Automatic Area Structuring and the Autostart value for Area Structure.

Also he has to define the Constructor Class. This is the class where he implemented the `BUILD` method. Notice that no assumption is made that the `BUILD` method will be part of the root class. That is a common approach, but the `BUILD` method can actually belong to any global class.
5.2.2 Modifying the Read Program

Finally Russel needs to modify the read test program ZCS_COURSE_SHMO_READ. Although the static BUILD method of the root class will be called automatically now, it does so asynchronously.

Instead of simply calling the method ATTACH_FOR_READ, Russel needs to take into account the asynchronous BUILD method and modify the program so that it waits for the shared memory area to be loaded by the BUILD method. Then, he needs to call the ATTACH_FOR_READ method again. The BUILD method is actually fired in a different work process, which accounts for needing the WAIT statement (see Listing 5.7).

```
START-OF-SELECTION.
  TRY.
    course_handle = zcl_cs_course_shmo=>attach_for_read( ).
  CATCH cx_shm_no_active_version.
    WAIT UP TO 1 SECONDS.
    course_handle = zcl_cs_course_shmo=>attach_for_read( ).
  ENDTRY.
Listing 5.7 Read Program Modification
```

Russel can now use transaction SHMM to delete any shared memory areas that may still exist. Since Russel has modified the area root class ZCL_CS_COURSE_SHMO_ROOT, he must delete any existing shared memory areas for this root. If this isn’t done, Russel will get an ABAP runtime exception stating an inconsistency is present.

Russel can now run the read program directly instead of having to first run the write program. The output proves that the preloading of the shared memory object is working correctly.

5.3 Implementing into the Business Object Layer

Russel has finally completed the programming required for the shared memory object and has tested that it works correctly. The next step is to implement this shared memory object in the business object layer of the course system.

The main goal is to swap out the persistent object code and replace it with the shared memory object code. When the exchange is complete, the changes should have no affect on any developments that use the business
object. This allows us to hide any complexities of using the shared memory object from the application logic. Activities like having to wait for the asynchronous BUILD method to complete will all be handled within the business object class now.

### Example Source Code

It is important for you to understand that normally Russel (i.e., the developer) would be directly modifying the business object class ZCL_CS_COURSE to implement the shared memory object. In order to illustrate how the business objects change as we delve further into the development of the examples that accompany this book, we will show you how to implement the shared memory object in a copy of the ZCL_CS_COURSE business object class.

For a complete example of all of the changes that you need to make to the business object class ZCL_CS_COURSE, see the class ZCL_CS_COURSE_SHM_ACCESS.

#### 5.3.1 Developing a Test Program

Russel wants to develop a simple program to test data retrieval using the business object class. This simple report program will retrieve all of the courses and write the data out to a standard list display (see Listing 5.8). Later he will use this same program to test the implementation of the shared memory object for the course database.

```plaintext
REPORT zcs_course_obj_read.

DATA: gt_courses TYPE STANDARD TABLE OF zcs_course_att.
FIELD-SYMBOLS: <gs_courses> LIKE LINE OF gt_courses.
DATA: gt_courses_obj TYPE zcs_courses_tbl.
FIELD-SYMBOLS: <gs_courses_obj> LIKE LINE OF gt_courses_obj.

START-OF-SELECTION.

gt_courses_obj = zcl_cs_course=>read_all_courses( ).
LOOP AT gt_courses_obj
  ASSIGNING <gs_courses_obj>.
  APPEND INITIAL LINE TO gt_courses
    ASSIGNING <gs_courses>.
  MOVE-CORRESPONDING 
    <gs_courses_obj>-course->course
    TO <gs_courses>.
ENDLOOP.

LOOP AT gt_courses ASSIGNING <gs_courses>.
```

158 | www.sap-press.com
Implementing into the Business Object Layer

5.3

WRITE:/ <gs_courses>-course_id,
<gs_courses>-course_sdesc+0(20),
<gs_courses>-faculty_id,
<gs_courses>-semester,
<gs_courses>-course_year,
<gs_courses>-major,
<gs_courses>-credit_hrs,
<gs_courses>-student_limit,
<gs_courses>-deletion_flag,
<gs_courses>-start_time,
<gs_courses>-end_time,
<gs_courses>-course_schedule,
<gs_courses>-cost,
<gs_courses>-currency.

ENDLOOP.

Listing 5.8 Course Object Test Program

5.3.2 Modifying the Business Object Class

Russel has proven that the current business object class ZCL_CS_COURSE works well using the persistent object for the course database. To keep things simple, we’ll focus now on only those changes required for the READ_ALL_COURSES method. Listing 5.9 shows that the code for the persistent object has been removed, and the new code to retrieve the data from the shared memory object has been inserted.

METHOD read_all_courses.

DATA: course_handle TYPE REF TO zcl_cs_course_shmo.
DATA: lt_courses TYPE zcs_courses_tt.
FIELD-SYMBOLS: <ls_courses> LIKE LINE OF lt_courses,
<wa_course> LIKE LINE OF r_courses.
TRY.
  course_handle = zcl_cs_course_shmo=>attach_for_read( ).
  CATCH cx_shm_no_active_version.
    WAIT UP TO 1 SECONDS.
    course_handle = zcl_cs_course_shmo=>attach_for_read( ).
  ENDTRY.
  lt_courses = course_handle->root->get_course_list( ).
  course_handle->detach( ).
 LOOP AT lt_courses ASSIGNING <ls_courses>.
    APPEND INITIAL LINE TO r_courses ASSIGNING <wa_course>.
    <wa_course>-course_id = <ls_courses>-course_id.

www.sap-press.com
Shared Memory Objects

CREATE OBJECT <wa_course>-course
EXPORTING
  _course = <ls_courses>.
ENDLOOP.
ENDMETHOD.

Listing 5.9  READ_ALL_COURSES Method

Also notice that the variable being passed to the CREATE OBJECT statement has changed. Instead of passing the persistent object, Russel is now passing a flat structure, which contains the course data. This means that the signature of the CONSTRUCTOR method of the business object class must also be modified. The _COURSE parameter must be typed like ZCS_COURSE (see Figure 5.13). Because the CONSTRUCTOR is private and only called via static factory methods, this sort of change has no effect on the applications that are using the business object class.

Figure 5.13 Constructor Signature

The CONSTRUCTOR implementation has changed a bit as well. Since Russel is now passing a flat structure to the _COURSE parameter, the CONSTRUCTOR must do something with this data. Russel has added a new private instance attribute called SHMA_DATA. This attribute will hold the data that is passed from the _COURSE parameter (see Listing 5.10).

METHOD constructor.
  me->shma_data = _course.
  me->course_preq_pers =
    me->load_course_preqs( _course-course_id ).
  me->map_shared_to_struc( ).
  me->load_supporting_details( ).
ENDMETHOD.

Listing 5.10  Constructor Modifications

Russel has also added a new method called MAP_SHARED_TO_STRUCT that replaces the mapping from the persistent object, and will be used to map the data from the SHMA_DATA attribute to the COURSE attribute of the business object (see Listing 5.11).
METHOD map_shared_to_struc.

  course-course_id = shma_data-course_id.
  course-course_sdesc = shma_data-course_sdesc.
  course-faculty_id = shma_data-faculty_id.
  course-semester = shma_data-semester.
  course-course_year = shma_data-course_year.
  course-major = shma_data-major.
  course-credit_hrs = shma_data-credit_hrs.
  course-student_limit = shma_data-student_limit.
  course-deletion_flag = shma_data-deletion_flag.
  course-start_time = shma_data-start_time.
  course-end_time = shma_data-end_time.
  course-course_schedule = shma_data-course_schedule.
  course-cost = shma_data-cost.
  course-currency = shma_data-currency.
  course-description = shma_data-description.

* Load faculty using business object class
TRY.
  course-faculty =
    zcl_cs_faculty=>read_faculty( course-faculty_id ).
CATCH zcx_cs_faculty.
ENDTRY.

DATA l_syllabi TYPE xstring.
l_syllabi = shma_data-syllabi.
DATA izip TYPE REF TO cl_abap_gzip.
IF l_syllabi IS NOT INITIAL.
  CREATE OBJECT izip.
  izip->decompress_text( EXPORTING gzip_in = l_syllabi
                        IMPORTING text_out = course-syllabi ).
ENDIF.

FIELD-SYMBOLS: <wa_pers> LIKE LINE OF course_preq_pers,
                 <wa_preq> TYPE zcs_course_preq_att.
LOOP AT course_preq_pers ASSIGNING <wa_pers>.
  APPEND INITIAL LINE TO course-preq ASSIGNING <wa_preq>.
  <wa_preq>-preq_id = <wa_pers>-course_preq->get_preq_id( ).
ENDLOOP.

ENDMETHOD.

Listing 5.11 MAP_SHARED_TO_STRUC Method Implementation
5.3.3 Testing the Changes

Now that Russel has completed the changes required to the business object class, he can use the test program, which he created earlier to see whether the data is being retrieved correctly. Again Russel would have directly modified the business object class ZCL_CS_COURSE, so there would be no changes required to the test program to make it work. For our purposes, we have implemented the changes in a copy of the business object class. Therefore, the test program ZCS_COURSE_OBJ_READ must be slightly modified to use the new business object class ZCL_COURSE_SHM_ACCESS.

Listing 5.12 shows that Russel is simply swapping out the table type used to receive the objects from the business object class, and the static call to method READ_ALL_COURSES.

REPORT zcs_course_obj_read.
DATA: gt_courses TYPE STANDARD TABLE OF zcs_course_att.
FIELD-SYMBOLS: <gs_courses> LIKE LINE OF gt_courses.
*DATA: gt_courses_obj TYPE zcs_courses_tbl.
DATA: gt_courses_obj TYPE zcs_courses_tbl_sma.
FIELD-SYMBOLS: <gs_courses_obj> LIKE LINE OF gt_courses_obj.

START-OF-SELECTION.

* gt_courses_obj = zcl_cs_course=>read_all_courses( ).
  gt_courses_obj =
    zcl_cs_course_shm_access=>read_all_courses().

Listing 5.12 Test Program Modifications
Index

A

ABAP Debugger 292, 341, 342
ABAP Dialog Screen Painter 33
ABAP Editor 28
ABAP exception 454
ABAP JavaScript (AJS) 357
ABAP kernel 34, 203
ABAP language syntax 30
ABAP Objects 143
ABAP Trial Version 469
ABAP Unit 18, 26, 221
ABAP Workbench 17
ABAP+J2EE instance 376
abstract class 76
abstract provider 207
abstraction 71, 169
Action 310
addition
  FOR TESTING 222
Adobe 375
Adobe Document Services (ADS) 376–380, 390, 392, 404, 405
user 379
Adobe Flex 304
Adobe Forms 19, 379, 381, 390, 391, 395, 398, 422, 444
Adobe LifeCycle Designer 375, 381, 387, 398
Table Assistant 398
Adobe Reader 377, 393, 394
ADS → see Adobe Document Services (ADS)
group class 75
AJAX 19, 346, 356, 358, 361, 362, 443
Handler 359
AJS → see ABAP JavaScript (AJS)
allow.ip 206
ALV → see SAP List Viewer (ALV)
append 62
application component 252
archive 200
ASCII 26
assigned role 421
assigning 278
assistance class 166, 304, 310, 322
asynchronous 263
Asynchronous JavaScript and XML (AJAX) → see AJAX
attribute
code 201
authentication 134
authorization group 58
B

BAPI → see Business Application Programming Interface (BAPI)
base class 75, 76
Basic Authentication 135, 379
Basic Authorization 238
Basis Support Package 30
BCS → see Business Communication Services (BCS)
binary string 37, 187, 393
binary table 115
binding 319
bookmark 30
Browser Selection 28
BSP → see Business Server Pages (BSP)
buffer memory 56
buffering 54, 56
settings 52
Business Application Programming Interface (BAPI) 433
Business Communication Services (BCS) 178, 182, 185–188
business object 101, 102, 128, 347
business object class 89, 95, 162, 168–172, 279
Business Server Pages (BSP) 27, 33, 164–166, 303, 345, 403, 413, 416, 424–427, 444, 449
BSP_UPDATE_MIMEREPOS 364
COMPILE_TIME_IS_VALID 369
controller 348, 353, 354
data binding 369
DO_AT_BEGINNING 371
element 366
element handler class 366
extension 27, 365, 425, 427, 428
extension element 165, 363
HTMLB Event Manager 373
model class 353
page attribute 429
PAGE->TO_STRING 360
portalEvent 427
RUNTIME.IS_VALID 370
server caching 361
server event 428
stateful 346
stateless 346
validator 369, 370
view 349

C

cardinality 321, 397
Cascading Style Sheets (CSS) 350, 352, 355, 360
casting 177, 215
central lock process 65
central master data hub 194
certificate 377
channel 457, 458
Chart View 436, 442
class
assistance class 166
CL_ABAP_GZIP 105
CL_ABAP_RANDOM 111
CL_ABAP_REGEX 190
CL_ABAP_ZIP 106
CL_BSP_CONTROLLER 354
CL_BSP_MIMES 386
CL_BSP_MODEL2 165
CL_CAM_ADDRESS_BCS 188
CL_GUI_ALV_GRID 282
CL_MDM_GENERIC_API 203, 207
CL_MIME_REPOSITORY_API 386
CL_SALV_COLUMN_TABLE 284
CL_SALV_DISPLAY_SETTINGS 285
CL_SALV_EVENTS_TABLE 291
CL_SALV_FUNCTIONS 282
CL_SALV_HIERSEQ_TABLE 274
CL_SALV_LAYOUT 286
CL_SALV_SELECTIONS 289
CL_SALV_TABLE 274, 279
CL_SALV_TREE 274
CL_SALV_WD_CONFIG_TABLE 323

CL_SAPUSER_BCS 188
CL_WD_COMPONENT_ASSISTANCE 166
CL_WD_DYNAMIC_TOOL 316
definition 184
Class Builder
Types tab 184
Class Type 73
class-based exception 89, 209
classic Dynpro 295, 422, 447
client dependent 53
client key 64
client proxy 121–124, 265
code
collapse 30
coloring 30
completion 31
page 180
sample 469
template 30
code-free environment 430
Collective Search Help 68
commit work 36
complex data type 61
complex type 259, 261
component 304, 309, 310, 331, 342
cell 320, 329
INTERFACECONTROLLER 320
INTERFACECONTROLLER_USAGE 321
reuse 307, 310, 327, 331
SALV_WD_TABLE 307, 320
structure 38
usage 307, 309, 320, 327–329
compression 105, 106
connection test 415
connector category 413
connector property 410
console hierarchy 200
constructor 94
container control 276
content Administration 416
context 38, 316, 335, 336, 342
attribute 317, 318
element 337
mapping 321, 383
control flush 36
controller 300, 395
controller class 296, 299
controller context 395
cookies 138
CRUD methods (Create, Read, Update, Delete) 90, 168
CSS → see Cascading Style Sheets (CSS)
customer modification 63

data base layout 44
data binding 317, 340
data browser 46, 60
data buffering 52
Data Dictionary 17, 46, 212
  domain 47
  object 43, 44
  Table Type 98
  Type 313
data element 47, 50
data model 46, 196, 200
data persistence layer 72
data relationship 45
Data Service 434
data table 61
data type 252, 253
  xsd:decimal 258
  xsd:string 255, 257, 258
database lock 347
deadlock 65
debugger 36
Debugger XML Viewer 37
Default Port 126
deletion operation 104
Delivery and Maintenance 57, 58
delivery class 51
dequeue 172
desktop 36
detail option
  maxLength 255
development class 40
development coordination 41
dialog 27
  popup 331
  popup window 324
  screen program 295
Dictionary Search Help 314
Direct Type Entry 184
Display Worklist 24
distributed development 71

DIV tag 358
Document Object Model (DOM) 356
domain 48, 50
Drop Down List Box 48
Dynpro 19, 66, 67, 90
Dynpro Event 300

element set 336
elementary search help 66, 68
e-mail 177, 185
endpoint type 236, 270
Enhancement Category 61
Enhancement Framework 26, 61, 63
Enhancement Info System 25
Enjoy Control Event 299
dequeue 171
Enterprise Service 26, 37, 91
Enterprise Service Browser 26, 250
Enterprise Service Modeling 18
Enterprise Service Repository 26, 250, 264
enterprise SOA 16, 91, 194
Entry Point 420
ERP system 196
event
  CLEANUP 209
  namespace 426
  OK_CODE 300
  ONCLICK 324
  onSubmit 402
  PAI-based 301
  parameter 427
  receiver 425
  registration 300
  subscriber 425
event handler 291, 299, 301, 310, 315, 324, 337
  WDDoBeforeAction 315, 316
event handler method 310
exception 170, 454
exception class 82, 83, 84, 111, 269
  CX_DYNAMIC_CHECK 86
  CX_HTTP_EXT_EXCEPTION 455
  CX_MDM_MAIN_EXCEPTION 209
  CX_NO_CHECK 86
  CX_STATIC_CHECK 86
  CX_SY_CONVERSION_CODEPAGE 211
Index

document

exception handling 80
execution duration 223
execution space 36
extended code completion 31
Extended Protocol 137, 140, 141
Extensible Markup Language (XML) 18, 19, 34, 37, 112–124, 131, 141, 249, 252, 266, 267, 356, 376, 382, 395, 404, 405, 427, 449–460
document 404
XML-based interface 382
Extensible Stylesheet Language Transformations (XSLT) 34, 90, 112, 121, 122, 456
editor 34
extension framework 366
external breakpoint 38

F

factory method 169
factory object 207
fault message 253
FCKeditor 363
field
  Boolean 213
  OK_CODE 278
  OPTION 99
  SIGN 99
field catalog 277
field symbol 217, 296
file system 27
File Transfer Protocol (FTP) 119
filter 99
filter object 96
firewall 134
fixed value domain 48
fixture 225, 229
flow content 399
foreign key 57
  relationship 44, 54
form context 384, 395
form field 165
form interface 381
Form View 436
friends 75, 78
Front-End Editor 29
FTP → see File Transfer Protocol (FTP)
Fully Buffered 52

G

Generic API layer 202
generically typed 215
getter method 174
graphic content 384
graphic node 384
graphic reference 384
GUI client 196
GUI status 287, 298

H

header field
  ~path_info 454
header line 278
hierarchical-sequential list 275
Hypertext Markup Language (HTML) 27, 32, 34, 186, 294, 303, 349, 355, 424
  email 186
  page 104
tags 27
viewer 294, 297
viewer control 294
Hypertext Transfer Protocol (HTTP) 346, 449, 451
destination 133
element 351
HTTPS 135, 412
request 346, 450
response 347
return code 454
server cache 362
ICF → see Internet Communication Framework (ICF)
ICF Service Node → see Internet Communication Framework (ICF)
ICM → see Internet Communication Manager (ICM)
Identity Management 421
index 62
inheritance 75, 83, 167
Inside-Out 18, 249
Integration Builder 249–256
interface 40, 331
IF_HTTP_EXTENSION 452
IF_HTTP_HEADER_FIELDS 454
IF_HTTP_HEADER_FIELDS_SAP 454
IF_HTTP_STATUS 455
IF_MDM_ACCESSOR 207
IF_MDM_ADMIN 207
IF_MDM_API_CONFIG 208
IF_MDM_CORE_SERVICES 208
IF_MDM_META 208
IF_WSPROTOCOL 140
SAI_TOOLS 123
internal table 279
Internet address type 181
Internet Communication Framework (ICF) 449–454
handler 450
ICF Service Node 19
Internet Communication Manager (ICM) 34, 178, 361, 452
Internet Mail Gateway 178
Internet Transaction Server (ITS)
Integrated ITS 413, 424
interval value domain 48
iView 19, 413, 415, 416, 418, 420, 421, 422, 424, 433
iView Wizard 416
iXML 456, 458
iXML library 456
Java Swing Library 164
JavaScript 32, 34, 303, 358, 368, 371, 372, 428
JPEG image 26

K
Key field 54
Knowledge Management 408

L
language key 56
language translation 48
layout 342
L-Frame 444
link 313
list 27
load balancing 409
local class 222
local event handler class 290
local path prefix 133
Lock Object 64
locking 171
logging 136
Logical Object Name 204, 205, 207, 210, 211
logical port 125, 126, 130, 133, 140
login group 56
Logon method 412
Logon procedure 134
Logon ticket 135
Lowercase flag 51

M
Maintenance view 56, 58, 59
managed object 75, 76
mapping 132
master data 198
harmonization of 194
Master Data Management → see SAP NetWeaver Master Data Management
MDM → see SAP NetWeaver Master Data Management
MDM4A → see SAP NetWeaver Master Data Management
mds.ini 206
message 170
## Index

<table>
<thead>
<tr>
<th>area</th>
<th>314</th>
<th>class</th>
<th>83, 84</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault</td>
<td>269</td>
<td>ID</td>
<td>138</td>
</tr>
<tr>
<td>interface</td>
<td>263, 270</td>
<td>manager</td>
<td>84</td>
</tr>
<tr>
<td>server</td>
<td>409</td>
<td>text</td>
<td>88</td>
</tr>
<tr>
<td>type</td>
<td>253, 261</td>
<td>variable</td>
<td>65</td>
</tr>
</tbody>
</table>

### Method

<table>
<thead>
<tr>
<th>ASSERT</th>
<th>225, 226, 233</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSERT_EQUALS</td>
<td>227, 229</td>
</tr>
<tr>
<td>ASSERT_NOT_INITIAL</td>
<td>230</td>
</tr>
<tr>
<td>ASSERT_SUBRC</td>
<td>226</td>
</tr>
<tr>
<td>asynchronous</td>
<td>157</td>
</tr>
<tr>
<td>ATTACH</td>
<td>151, 155</td>
</tr>
<tr>
<td>ATTACH_FILE_TO_RESPONSE</td>
<td>338</td>
</tr>
<tr>
<td>BIND_ELEMENT</td>
<td>340</td>
</tr>
<tr>
<td>BIND_TABLE</td>
<td>336</td>
</tr>
<tr>
<td>BUILD</td>
<td>155</td>
</tr>
<tr>
<td>chaining</td>
<td>175</td>
</tr>
<tr>
<td>CL_BSP_RUNTIME=&gt;CONSTRUCT_BSP_ URL</td>
<td>367</td>
</tr>
<tr>
<td>CLASS_CONSTRUCTOR</td>
<td>76</td>
</tr>
<tr>
<td>COMMIT</td>
<td>104</td>
</tr>
<tr>
<td>CONSTRUCTOR</td>
<td>87</td>
</tr>
<tr>
<td>CREATE_COMPONENT</td>
<td>329</td>
</tr>
<tr>
<td>CREATE_PERSISTENT</td>
<td>79, 82</td>
</tr>
<tr>
<td>CREATE_WINDOW_FOR_CMP_USAGE</td>
<td>330</td>
</tr>
<tr>
<td>DELETE</td>
<td>104</td>
</tr>
<tr>
<td>DELETE_PERSISTENT</td>
<td>82</td>
</tr>
<tr>
<td>DISPATCH_INPUT</td>
<td>165, 348</td>
</tr>
<tr>
<td>DO_AT_BEGINNING</td>
<td>366</td>
</tr>
<tr>
<td>DO_REQUEST</td>
<td>359, 362</td>
</tr>
<tr>
<td>factory</td>
<td>169</td>
</tr>
<tr>
<td>GET_FORM_FIELD</td>
<td>359</td>
</tr>
<tr>
<td>GET_PERSISTENT</td>
<td>80, 82</td>
</tr>
<tr>
<td>GET_PERSISTENT_BY_QUERY</td>
<td>81</td>
</tr>
<tr>
<td>getResponseHeader</td>
<td>358</td>
</tr>
<tr>
<td>getter</td>
<td>174</td>
</tr>
<tr>
<td>HANDLE_REQUEST</td>
<td>452, 462</td>
</tr>
<tr>
<td>HAS_ACTIVE_COMPONENT</td>
<td>329</td>
</tr>
<tr>
<td>ON_USER_COMMAND</td>
<td>291, 292</td>
</tr>
<tr>
<td>QUERY</td>
<td>213</td>
</tr>
<tr>
<td>RETRIEVE_SIMPLE</td>
<td>214</td>
</tr>
<tr>
<td>SET</td>
<td>103</td>
</tr>
<tr>
<td>SET_SAVE_RESTRICTION</td>
<td>287</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SET_SCREEN_POPUP</th>
<th>292</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_SCREEN_STATUS</td>
<td>289</td>
</tr>
<tr>
<td>SETUP</td>
<td>225, 228, 229</td>
</tr>
<tr>
<td>simple</td>
<td>212</td>
</tr>
<tr>
<td>SPLICE</td>
<td>106</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>225, 228</td>
</tr>
<tr>
<td>test</td>
<td>225</td>
</tr>
<tr>
<td>TO_BOOLEAN</td>
<td>369</td>
</tr>
<tr>
<td>TO_ENUM</td>
<td>369</td>
</tr>
<tr>
<td>transactional</td>
<td>168</td>
</tr>
<tr>
<td>WDDOINIT</td>
<td>322</td>
</tr>
<tr>
<td>Microsoft .NET</td>
<td>195</td>
</tr>
<tr>
<td>Microsoft Excel</td>
<td>36</td>
</tr>
<tr>
<td>Microsoft Foundation Classes</td>
<td>164</td>
</tr>
<tr>
<td>MIME</td>
<td>see Multipurpose Internet Mail Extensions (MIME)</td>
</tr>
<tr>
<td>modal dialog</td>
<td>312</td>
</tr>
<tr>
<td>model</td>
<td>163, 164, 390, 405</td>
</tr>
<tr>
<td>attribute</td>
<td>168</td>
</tr>
<tr>
<td>binding</td>
<td>165, 166</td>
</tr>
<tr>
<td>class</td>
<td>18, 40, 90, 183, 268, 304, 395, 462</td>
</tr>
<tr>
<td>Model View Controller (MVC)</td>
<td>163–166, 295, 303, 348, 353, 430</td>
</tr>
<tr>
<td>Mount MDM Server</td>
<td>198</td>
</tr>
<tr>
<td>Multipurpose Internet Mail Extensions (MIME)</td>
<td>26, 336, 364, 463</td>
</tr>
<tr>
<td>repository</td>
<td>26, 364</td>
</tr>
<tr>
<td>type</td>
<td>338, 384, 385, 463</td>
</tr>
<tr>
<td>MVC</td>
<td>see Model View Controller (MVC)</td>
</tr>
</tbody>
</table>

### Name Mapping

<table>
<thead>
<tr>
<th>name mapping</th>
<th>237</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>252, 254, 266, 409, 453</td>
</tr>
<tr>
<td>Navigation stack</td>
<td>24</td>
</tr>
<tr>
<td>.NET Connector</td>
<td>119</td>
</tr>
<tr>
<td>New System Wizard</td>
<td>408</td>
</tr>
<tr>
<td>NULL</td>
<td>212</td>
</tr>
</tbody>
</table>

### Object

<table>
<thead>
<tr>
<th>object</th>
<th>MO_ACCESSOR</th>
<th>208</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLHttpRequest</td>
<td>357</td>
<td></td>
</tr>
<tr>
<td>Object Browser</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>list</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>object navigation tree</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object reference</td>
<td>95, 97</td>
<td></td>
</tr>
<tr>
<td>object-oriented</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>abstraction</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>programming</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>occurrence</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td>offline form</td>
<td>402</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>357</td>
<td></td>
</tr>
<tr>
<td>Open SQL</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>outbound plug</td>
<td>310, 405</td>
<td></td>
</tr>
<tr>
<td>Outside-In</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>overflow leader</td>
<td>400</td>
<td></td>
</tr>
</tbody>
</table>

**P**

- package | 17, 41, 73, 122, 123, 266, 277
- hierarchy | 40, 41, 42
- interface | 42
- page | 420
- PAI → see Process After Input (PAI)
- parameter | 82
- abap/test_generation | 222
- RETURNING | 148, 174
- WDEVENT | 337
- pass by value | 149
- pattern | 257, 258, 280
- Payload | 140
- PBO → see Process Before Output (PBO)
- PBO/PAI → see Process Before Output/Process After Input (PBO/PAI)
- PCD → see Portal Content Directory (PCD)
- PDF → see Portable Document Format (PDF)
- persistent attribute | 94
- persistent class | 77, 78
- Persistent Data Mapper | 77
- persistent object | 72, 73, 75, 77, 79, 91, 102, 157, 163, 186, 217
- persistent object class | 17
- Persistent Object Service | 72
- Personal Value List | 313
- PHP | 350
- PI → see SAP NetWeaver Process Integration
- plug | 310
- polymorphism | 83
- popup signal | 440, 441
- popup window | 312, 325

**Q**

- Query Manager | 81
- query service | 81, 82, 96, 99, 218

**R**

- random number generator | 110
- ranges table type | 98
- Reader Rights | 377
- Really Simple Syndication (RSS) | 455, 457, 460
Advisory Board 457
feed 19, 449, 455
reader 460
refactoring 34
Refactoring Assistant 17, 35
reference variable 218
register 291
Regular Expressions 187, 189, 257, 338
relational database 90
Remote Function Call (RFC) 209, 378, 410, 413, 431, 433, 436, 437
destination 133
enabled function module 235
repository 195, 200, 204
Repository Browser 25
Repository Information System 27
response object 337
RFC → see Remote Function Call (RFC)
RISK_LEVEL 223
role 420
role-based 407, 416
RSS → see Really Simple Syndication (RSS)
Ruby on Rails 164, 357
Runtime Type Services (RTTS) 176, 177

S

SAP Business Workplace 178
SAP Customer Relationship Management (SAP CRM) 408
SAP Data Modeler 46
SAP Developer Network (SDN) 64
SAP ERP 408
SAP ERP 6.0 16
upgrade 39
SAP GUI 40, 347, 410, 413, 424, 433, 447
SAP GUI for Windows 29
SAP Interactive Forms by Adobe → see Adobe Forms
SAP List Viewer (ALV) 273, 307, 320
Object Model 19, 273, 276, 282, 294
REUSE_ALV_GRID_DISPLAY 274
SAP Logon 411
SAP MDM Console 196, 198
SAP MDM Data Manager 196
SAP NetWeaver 196
SAP NetWeaver Application Server 408
Java 376
SAP NetWeaver Business Client 443–447
SAP NetWeaver Business Intelligence 408
SAP NetWeaver Exchange Infrastructure → see SAP NetWeaver Process Integration
SAP NetWeaver Master Data Management 18, 193
ABAP API 201, 202, 203, 205, 209, 211, 212
repository 196, 213
server 198
SAP NetWeaver Portal 19, 40, 407
SAP NetWeaver Process Integration 18, 130, 131, 132, 137, 139, 141, 249
SAP NetWeaver Visual Composer 19, 425, 429–437, 442
SAP R/3 23
SAP Service Marketplace 379
SAP Smart Forms 375, 382, 387, 389, 391, 394
SAP Web Application Server 33
SAPconnect 178, 180
SAPlink 20, 357
SAProuter 408
SAPscript 375, 387, 394
screen control 296, 297
SDN → see SAP Developer Network (SDN)
search help 66, 312
search help exit 67
secondary index 63
Secure Sockets Layer (SSL) 119, 238, 379
Client Certificate 135
selection screen 67
SELECT-OPTION 82, 97, 313
self-service website 40
Server Port 411
server proxy 18, 264, 265, 266, 267, 269, 271
service consumption 431
definition 240, 270
Definition Wizard 270
handler 460, 461
interface 270
Node Hierarchy 24
INDEX

operation 268
port 379
Repository 131
session management 446
shared memory 149, 163, 347
shared memory object 18, 90, 144, 157, 159
area class 144
area handle 152
area structure 156
Automatic Area Structuring 156
IF_SHM_BUILD_INSTANCE 155
root class 144
Simple Mail Transfer Protocol (SMTP) 26, 178
Plug-In 178
port 180
Single Sign-On (SSO) 40, 135, 412
Singleton 334, 335
SLD \rightarrow see System Landscape Directory (SLD)
smart client 304
SMTP \rightarrow see Simple Mail Transfer Protocol (SMTP)
SOAP 120, 121, 131, 139, 140
Action 136
Software Component 41, 252
Software Component Version 254
split screen view 30
splitter 297
SQL \rightarrow see Structured Query Language (SQL)
SSL \rightarrow see Secure Sockets Layer (SSL)
SSO \rightarrow see Single Sign-On (SSO)
State Management 138
state property 315
stateful 138, 348
statement
CALL TRANSFORMATION 34, 112, 113, 116
CLASS DEFINITION 222
COMMIT 82
COMMIT WORK 111
CREATE OBJECT 75, 160
DELETE 103
FIND 189
GET REFERENCE OF 218
INSERT 80
IS SUPPLIED 173
MESSAGE 83
RAISE 89
RAISE EXCEPTION 87, 89
REPLACE 189
SHARED BUFFER 143
SHARED MEMORY 143
TRY...CATCH 80, 170
TYPE REF TO 95
UPDATE 103
WAIT 157
WHERE 81, 99
WRITE 276
Structured Query Language (SQL) 62, 90, 98, 109
statement 72
trace 62
style 351
style sheet 349
subclass 282, 284
subform 388, 398, 399
superclass 167
SVG Viewer 430
taxonomy highlighting 30
taxonomy sensitive formatting 30
system alias 413
system connection 407
system field
SY-SUBRC 83, 111
system icon 384
system ID prefix 409
system landscape 407
System Landscape Directory (SLD) 252

T

table 49, 398
buffering 348
FPCONNECT 378
locking 64
row 399
type 95
view 436
table maintenance 50, 51, 56
Generator 58
Settings 50
System 51
Tag Browser 26
test class generation 227
test repository 26
Index

Time To Live (TTL) 459
toolbar 305
toolbox 33
trace file 136
transaction
LPCONFIG 126, 130
MDMAPIC 204, 208
SAINT 203
SBWP 178
SCC4 222
SCOT 179, 180, 182, 183
SD11 46
SEBO 24
SHMA 150
SHMM 143, 154, 157
SICF 24, 178, 243
SM59 133, 378
SMICM 24
SOST 183
SPAM 203
STRUSTSSO2 412
WSADMIN 243, 244, 271
WSCONFIG 239, 242
WSSPROFILE 139
transaction coordination 103
transaction iView 424
transactional method 168
transformation 112
transient data 72
translatable 86
translation 49, 59
transparent table 51
transport guarantees 271
transport layer 41
Transport Management System 34
Transport Organizer 27
tray 305
trusted connection 206
TTL → see Time To Live (TTL)
two process debugger 36
type
ANY 176
LinkToAction 337
WDY_BOOLEAN 317
type definition 254, 255, 259

U

UDDI → see Universal Description, Discovery and Integration (UDDI)
UI element
ContextualPanel 305
FileDownload 336
FileUpload 339, 340, 403
InputField 312, 315
InteractiveForm 395, 402
LinkToAction 324
Table 313, 332
TablePopin 332, 333
TablePopinToggleCell 333
TableView 323
TabStrip 308, 313
TextView 324
TransparentContainer 333
ViewContainerUIElement 305, 307
UIDPW → see User Name and Password via User Mapping (UIDPW)
UME → see User Management Engine (UME)
unbounded 260
Unicode 180, 181, 211
UTF-8 180
Uniform Resource Locator (URL) 359, 418, 450, 462
parameter 359, 462
Unit test 26
Universal Description, Discovery and Integration (UDDI) 131
UNIX 121
upgrade 63
upload 108
URL → see Uniform Resource Locator (URL)
use access 123
used component 309
user maintenance 414
user management 411
User Management Engine (UME) 421
user mapping 412, 413, 414
User Name and Password via User Mapping (UIDPW) 412
user-defined validation 368
Utility method 175
Index

V

validation check 315
validator 370
value
&NC& 58
IS INITIAL 173
templateSource 395
type ref to data 215, 217, 218
value help 49, 54, 311, 312
Value Help Configuration Wizard 437, 440, 441
value help service 439
value range 48
variant 271
view 309, 310, 326, 342, 354
virtual interface 240
visibility 78
Visual Administrator 377, 380
Visual Composer → see SAP NetWeaver
Visual Composer

W

Web AS Path 413
Web Dynpro 19
    component 166
    component controller 426
    context 395, 397, 402
debugger 38
utility class 459
Web Dynpro Code Wizard 316, 318
web server 26
Inside-Out 18, 249
message 135
operation 261
Outside-In 249
Proxy 93
runtime 137
security profile 138
system configuration 431
wizard 236
Web Services Description Language (WSDL) 120–125, 132, 243, 245, 249, 252, 258, 263, 265, 271, 431
Web Services Homepage 243, 244, 271
Web Services Infrastructure 130
window 309, 325, 341
window controller 327
window manager 326, 327, 330
with message class 84, 86
work area 278
work process 36, 66
Workbench option 29
Workbench setting 28
Worklist 255
Workset 416, 422
WSDL → see Web Services Description Language (WSDL)
WYSIWYG 32, 303

X

X.509 Certificate 119
XI → see SAP NetWeaver Process Integration
XML → see Extensible Markup Language (XML)
XSLT → see Extensible Stylesheet Language Transformations (XSLT)

Z

ZIP 91, 105, 106, 113–116
ZIP compression 17