In this sample, we’ll use Chapter 9 to show you how this book is structured to help you master your ABAP Certification exam. Chapter 9 provides you with a basic understanding of how development objects are organized. The chapter starts with a real-world scenario, covers the test objectives, outlines the key concepts, and ends with sample questions and answers surrounding the content.
Preface

The SAP PRESS Certification Series is designed to provide anyone who is preparing to take an SAP certified exam with all of the review, insight, and practice they need to pass the exam. The series is written in practical, easy-to-follow language that provides targeted content that is focused just on what you need to know to successfully take your exam.

This book is specifically written for those preparing to take the SAP Certified Development Associate—ABAP with SAP NetWeaver 7.40 exam, so if you've purchased this book, you're obviously interested in learning how to successfully pass the certification exam. You've come to the right place. This book will help you become an SAP Certified Development Associate in ABAP. It is your first step to propel your career by setting yourself apart from your peers. The certification exam verifies your knowledge of the fundamentals of release 7.4 of the SAP NetWeaver Application Server ABAP. This includes knowledge obtained from attending SAP training courses in the ABAP curriculum and project experience. To help prepare you to pass the exam, we’ll cover the facts and applications of each topic discussed in this book.

You’ll find all of the practical, real-world information you need to get a clear understanding of the topics that will be covered on the exam, insightful tips about the types of questions you’ll encounter, and strategies to use to answer them correctly. The book is closely aligned with the course syllabus and the exam structure, so all of the information provided is relevant and applicable to what you need to know to prepare for the SAP Certified Development Associate—ABAP with SAP NetWeaver 7.40 exam. We explain the SAP products and features using practical examples and straightforward language, so you can prepare for the exam and improve your skills in your day-to-day work as an ABAP developer.

Each book in the series has been structured and designed to highlight what you really need to know. The chapters begin with a clear list of the learning objectives for the chapter, such as this example:
What You’ll Learn:

- How to prepare for the exam
- Understanding the general exam structure
- Practice questions and preparation

From there, you’ll dive into the chapter and get right into the test objective coverage. So let’s take a look at how the book is structured.

Structure of This Book

Let’s discuss how you can use this book to prepare for the exam. This book is divided into two sections:

- **Part I** contains the general introduction to this book and the certification examinations. This section will provide you with an overview of the certification process and the benefit to you and your customer or company. We’ll discuss both the purpose of this certification examination and provide information on additional certification examinations that are available beyond this one.

- **Part II** is a breakdown of the topics covered in the certification examination for C_TAW12_740, the SAP Certified Development Associate—ABAP with SAP NetWeaver 7.40. Each chapter contains a similar structure to assist with understanding that portion of the certification examination.

Part II is the core of the book and discusses each exam topic and determines the key concepts. These key concepts are then explained along with important information that will provide the context for understanding. Each chapter is broken down into one or more subtopics, according to complexity. Illustrations and diagrams are included throughout to ensure that you understand important concepts.

Throughout the book, we’ve also provided several elements that will help you access useful information:

- **Tips** call out useful information about related ideas and provide practical suggestions for how to use a particular function.

- **Notes** provide other resources to explore or special tools or services from SAP that will help you with the topic under discussion. The following boxes are examples of these elements:

  - **Note**
    
    The content provided in each chapter and subtopic does not exhaustively cover everything that appears on the exam. In other words, the certification guide does not cover every exam question, but rather it acts as a refresher to highlight the major points for each topic. In addition, it points out areas where further review is needed.

  - **Tip**
    
    SAP’s help portal ([http://help.sap.com](http://help.sap.com)) and SAP’s Community Network ([http://scn.sap.com](http://scn.sap.com)) both are useful for more detail on each topic.

Each chapter that covers an exam topic is organized in a similar fashion so you can become familiar with the structure and easily find the information you need. Here’s an example of a typical chapter structure:

- **Introductory bullets**
  The beginning of each chapter discusses the techniques you must master to be considered proficient in the topic for the certification examination.

- **Topic introduction**
  This section provides you with a general idea of the topic at hand to frame future sections. It also includes objectives for the exam topic covered.

- **Real-world scenario**
  This part shows a scenario that provides you with a case where these skills would be beneficial to you or your company.

- **Objectives**
  This section reviews the material the authors feel provides you with the necessary information to successfully pass this portion of the test.

- **Key concept refresher**
  This section outlines the major concepts of the chapter. It identifies the tasks you will need to be able to perform properly to answer the questions on the certification examination.
You should pay particular attention to the points raised in the Key Concept Refresher section and those from the Objectives section.

Main part
The next section of the chapter provides the objectives of this section of the test. This includes identifying major points of this topic that are discussed in the chapter.

Often, we identify a general weighting SAP uses for this topic. To simplify the discussion, we have produced a general group of three categories, or ranges (high, average, and low weighting). You should use this information to assist you in determining which chapters you should spend your time studying to maximize your score.

Those chapters that are designated as low have the fewest questions on the certification examination. Those with a designation of average have an average or medium number of questions on the certification examination. The chapters with a high weighting have more questions than the other chapters.

The chapter with the highest weighting contains significantly more questions on the certification examination than any other. Unsurprisingly, Chapter 9, which discusses general ABAP program design, has the highest weighting of the examination.

Important terminology
Just prior to the practice examination questions, we provide a section to review important terminology. This may be followed by definitions of various terms from the chapter.

Practice questions
The chapter then provides a series of practice questions related to the topic of the chapter. The questions are structured in a similar way to the actual questions on the certification examination.

Practice question answers and explanations
Following the practice exercise are the solutions to the practice exercise questions. As part of the answer, we discuss why an answer is considered correct or incorrect.

Note
You should be aware that the practice exercise questions are for self-evaluation purposes only and do not appear on the actual certification examination. Answering the practice exercise questions correctly is no guarantee that you will pass the certification exam.

Glimpse into the Exam Structure
To understand the structure of this certification success guide, it is important to understand the base structure of the exam. We only touch upon the topic here because Chapter 1 covers the exam structure in detail.

Two basic elements define the exam structure for the associate and various levels of certification available. These elements are as follows:

Competency level
Competency is what you are expected to be able to do at a specific level described. In simple terms, if you have reached a competency, then you can do the tasks described by that competency. SAP Education has developed a set of progressive competency levels. As you become more experienced, your competencies move in sequence from level A to D. The competencies include:
Accomplish defined tasks (level A)

Build proposed solutions (level B)

Conceptualize complex processes (level C)

Design integrated strategies (level D)

For example, a simple definition of the level A competency is the ability to accomplish defined tasks during system implementation. As an associate, you should be able to carry out defined tasks given to you when little to no ambiguity exists. For level D, you may need to devise a roadmap and recommendation to a particular solution or issue.

Exam topic

This element is much more familiar. ABAP is a complex programming language for business applications with many parts. A team of experts devised which topics should be tested in each exam. This allows a target for the test development and conveniently provides you with a list of focus areas to prepare.

Tip

The next chapter dives into great detail for each level of the exam, including a topic and competency breakdown. Please reference Chapter 1 for additional information.

You should understand the concept of competency and exam topics now. It is important to understand that they work together. The competency areas are applied to separate topics in the exam. Likewise, this book is also broken down by these same topics to set the focus and align content for exam preparation.

The exam topics may overlap between the competency areas in terms of a subject such as objects, but they are unique in content according to the competency. For example, activity at level A is considered more of a design element and explores unique construction of objects. Activity at level B examines a more fundamental understanding of classes and methods. Thus, it’s possible to discuss operational solutions for objects in one topic, whereas a separate topic covers more strategic problems or concepts. This will become evident in the chapter content and practice questions.

Practice Questions

We want to give you some background on the test questions before you encounter the first few in the chapters. Just like the exam, each question has a basic structure:

- Question stimulus
  The question stimulus varies with the different questions, but its intention is to present a situation or scenario as context for the question. The stimulus complexity depends on the competency level.

- Actual question
  The question comes next and relates to the stimulus. Read the question carefully and be sure to consider the details from the stimulus because they can impact the question.

- Question hint
  This is not a formal term, but we call it a hint because it will tell you how many answers are correct. If only one is correct, normally it will tell you to choose the correct answer. If more than one is correct, like the actual certification examination, it will not indicate the correct number of answers.

- Answers
  The answers to select from depend on the question type. The following question types are possible:

  - Multiple response
    More than one correct answer is possible.

  - Multiple choice
    Only a single answer is correct.

  - True/false
    Only a single answer is correct. These should be minimal, especially as you experience the more advanced exams.

  - Fill in the blank
    This type is of question is rarely found on the associate examination but is found on the professional-level exam. Although capitalization does not matter, only a limited number of answers are considered valid. You should therefore be careful with typing and spelling.
Sequence/ranking
This type of question will also have a single correct answer. The answers will provide the same options in different order, and you must select the correct sequence.

Caution
Media can be presented in a test question as part of the stimulus. For example, a diagram or a code block can be given, followed by the question based on this enriched content. If the question does not appear to make sense by itself, you should look for an exhibit button to display the media.

Summary
With this certification success guide, you’ll learn how to approach the content and key concepts highlighted for each exam topic. In addition, you’ll have the opportunity to practice with sample test questions in each chapter. After answering the practice questions, you’ll be able to review the explanation of the answer, which dissects the question by explaining why the answers are correct or incorrect. The practice questions give you insight into the types of questions you can expect, what the questions look like, and how the answers can relate to the question. Understanding the composition of the questions and seeing how the questions and answers work together is just as important as understanding the content. This book gives you the tools and understanding you need to be successful. Armed with these skills, you’ll be well on your way to becoming an SAP Certified Development Associate in ABAP.
Basic ABAP Programs
and Interface Creation

Techniques You’ll Master:

- Organize development objects
- Identify different types of programs
- Determine program event blocks
- Use Dynpros, selection screens, and modal dialog boxes
- Use authorization checks
- Explain the different types of modularization units
- Identify the effect of messages on program execution
This chapter provides you with a basic understanding of how development objects are organized. We will discuss how these objects are grouped into change requests (commonly called transports) and moved throughout the system landscape. We will cover the different types of programs, identifying their similarities and differences. We will discuss the majority of the event blocks and the basics of Dynpros (often just referred to as screens), selection screens, and modal dialog boxes (see Chapter 12 and Chapter 13 as well). We will cover the use of authorization checks and the different types of modularization units, again identifying both their similarities and differences. Last, we will cover the use of the MESSAGE statement on program execution.

Each of these topics is covered separately, and they are followed by practice questions and solutions.

**Real-World Scenario**

You have been asked to lead a new project implementing a new group of application programs for use in your company’s SAP system. It is your responsibility to put together a development team and produce the subsystem.

In examining the requirements of the subsystem, you see that there are a number of dialog transactions and several report transactions. Some of the data is sensitive, so both the dialog transactions and report transactions will require separate authorization checks to determine if the user is allowed to view the data. The dialog programs will also need to check for authorization to create or change the data. If the user is not authorized to perform the operation, a message must be displayed.

Once you have your development team in place, you will be required to organize the development so that it all reaches production at the same time for go-live. Owing to the complexity of accessing the data and the numerous types of authorization checks, a decision was made to encapsulate the tables’ access to global reuse components. This will also provide the option of buffering retrieved data during dialog transactions.

**Objectives of this Portion of the Test**

The purpose of this portion of the certification examination is to verify that you have detailed knowledge of the ABAP development environment and the capabilities of different types of programs. This portion of the examination will test your knowledge of a wide range of topics. The points you will need to understand from this section include:

- What types of programs exist
- Event blocks
- Dynpros, selection screens, and modal dialog boxes
- Different methods of executing programs
- Authorization checks
- Modularization units
- The effect of messages in a program

Because this is an ABAP certification examination, assessing your general knowledge of ABAP and its environment is the most important objective of the test. The certification examination will give more weight to the material in this chapter than all of the other topics in the examination. This means there will be a higher percentage of questions related to this chapter than any other chapter.

**Key Concepts Refresher**

Unsurprisingly, you typically need to develop ABAP programs as an ABAP developer. You therefore need to understand and be able to perform the following types of tasks when developing ABAP programs:

- Create various types of programs
- Test various types of development
- Move development objects from your development environment to your test and production systems
- Encapsulate data and processing
- Interact with users
- Verify that the user is authorized to perform a specific function
- Inform users about errors and program status

**Organizing Development**

Development projects start out in a development system. The development objects edited or created in a project must then be transported to subsequent systems when the project is complete.

At the start of a development project, normally, the project manager creates a change request in which he identifies who will be working on the project, in either the Transport Organizer or directly in the ABAP Workbench. Then he creates a task for each project employee or developer within the change request. In some cases, where the development is limited, there may be only a single task in a single change request.

As a development object (for example, a program or data element) is edited or created, you assign this object to the change request. The object is thus entered into your task. All repository objects that you work on during a development project are collected within your task. When you complete your development tasks, you perform syntax checking, activation, and testing before you release the task in the change request. Once the project development is complete and all tasks are released, the change request itself is released, but it cannot be released until all of its tasks are released (see Figure 9.1).

Organizing a development project using a change request offers the following advantages:

- Each developer can track and check his project-specific activities.
- The development objects can be processed by all developers involved in the project. For developers who do not belong to the project team, the repository objects remain locked until the project is completed and the change request is released.
- The joint transport of the development objects processed in the project at the time of project completion is automatically ensured by assigning the objects to the change request. The transport route of the involved packages specifies to which subsequent system they are transported.

**ABAP Programming Overview**

ABAP is a programming language developed by SAP for programming business applications within the SAP environment. As discussed in Chapter 3 and Chapter 8, the three-tier architecture used by SAP provides user interaction through the presentation layer, which controls the business logic running on an application server, which retrieves and stores data in the database server. ABAP is the language that controls the business logic running on the application server. ABAP programs are executed in the ABAP runtime environment. The main role of the ABAP program in the application layer is the processing and formatting of data...
from the database layer and its interaction with the user input from the presentation layer.

ABAP provides integrated database access to multiple database systems through the use of Open SQL. It optimizes database access through the use of SAP buffering and provides internal tables for processing tabular data within the program. It uses the concept of online transaction processing (OLTP) integrated into the ABAP runtime environment, enabling multiple users to access the database at the same time using the SAP LUW. It provides outside access to other programming environments with the Remote Function Call (RFC). Also, ABAP has an integrated interface to XML.

ABAP supports both procedural and object-oriented programming models based on subroutines, function modules, and methods of classes. As appropriate for a language designed around international business, the textual language is handled independently of the programming language. ABAP provides for the separation of language-specific program components from the source code, and they are loaded during program execution based on the user's logon language. ABAP supports code pages, single-byte, double-byte, and, after release 6.10, even Unicode.

Each ABAP program starts with an introductory statement. The functions of an ABAP program are implemented in processing blocks, which are defined using modularization statements. The order of the processing blocks is irrelevant for program execution. Following the introductory statement, every program contains a global declaration section where you implement definitions of data objects (for example, variables, structures, and internal tables) and data declarations that are visible in the entire program. Whereas the order of individual definitions and declarations does not matter generally, you must take into account that an ABAP statement can only refer to existing definitions and declarations. Therefore, they must be defined before they can be declared and declared before they can be used. For example, a definition of a local type must occur prior to its first use in the program, normally either as a data declaration or as part of an interface to a modularization unit.

Because ABAP is a language that has evolved over time, it contains several obsolete statements or forms of statements that were replaced with improved language constructs to ensure downward code compatibility but were not eliminated. ABAP Objects prevents the use of almost all of these obsolete additions. In general, you should no longer use these additions in new programs, but you may encounter them in older programs.

The introductory statement must be the first statement in the program after the include programs are expanded during program generation. It can only appear once in the program.

Introductory program statements include the following:

- Executable programs (REPORT)
- Module pools (PROGRAM)
- Function groups (FUNCTION-POOL)
- Class pools (CLASS-POOL)
- Interface pools (INTERFACE-POOL)
- Subroutine pools (PROGRAM), which are now considered obsolete
- Type groups (TYPE-POOL)

All of these are stand-alone compilation units or program types. Include programs are only used in the context of superior programs. Type groups, also known as type pools (program type T), are stand-alone programs from a logical point of view. However, they do not contain executable code but only contain type definitions and constant definitions. Therefore, type groups have their own introductory program statement, the TYPE-POOL.

When you create a program, the ABAP Workbench automatically generates the introductory statement from the program type. If changes are necessary, only the introductory statement's additions should be edited. Examples of a new report and some possible additions are shown in Listing 9.1 and Listing 9.2.

```abap
REPORT z_new_report.
```

**Listing 9.1 A New Introductory Statement for a Report**
The parameter interface of a procedure consists of formal parameters and specifies the possible exceptions of the procedure. The formal parameters are input, output, and input/output parameters and return values. There are also the obsolete table parameters. Subroutines do not provide input or output parameters directly because both USING and CHANGING are treated as a type of changing parameter; they do not provide for returning values. Function modules do not provide returning values. Formal parameters are either generic or completely typed. You can specify either pass by reference or pass by value for most formal parameters, although pass by value is mandatory for some formal parameters.

The differences and similarities are discussed in detail later in the sections ABAP Subroutine, ABAP Function Module, and ABAP Classes and Methods.

Class-based exceptions can be declared using `RAISING` for all procedures (methods, function modules, and subroutines) and can then be propagated from the procedure. Also, `EXCEPTIONS` can be used in methods and function modules to define non-class-based exceptions, which can then be triggered in the procedure using either `RAISE` or `MESSAGE` ...

Listing 9.3 shows the older exceptions that were `MESSAGE`-based and could only exist in function modules or methods. Listing 9.4 shows an example of a class-based exception. A single procedure can use only one type—not both.

```
MESSAGE e008(38) WITH 'FOOBAR'
RAISING table_cannot_be_loaded.

Listing 9.3 Example of a MESSAGE-Based Exception
```

```
RAISE EXCEPTION TYPE cx_sy_dynamic_osql_semantics
  EXPORTING textid =
    cx_sy_dynamic_osql_semantics=>unknown_table_name
  token = 'FOOBAR'.

Listing 9.4 Example of a Class-Based Exception
```

When you decide whether to use pass by reference or pass by value for a formal parameter, you must compare the relative performance and robustness for each type of transfer:

- A pass by reference parameter provides a pointer to the original data object in memory, and the procedure operates directly on the original object.

---

**Listing 9.2 A Modified Report Introductory Statement**

In particular, the keywords **FUNCTION-POOL**, **CLASS-POOL**, **INTERFACE-POOL**, and **TYPE-POOL** should only be generated by the corresponding tools of the ABAP Workbench. The creation of a function group also automatically generates the main program, including the top include containing the **FUNCTION-POOL** statement. The complete name of the program for a function group in the Repository consists of the prefix `SAPL` and the name of the function pool (function group). All statements following the introductory program statement or those that are included as include programs are treated as a single unit by the ABAP compiler.

Each ABAP program is divided into processing blocks. Each accessible statement of an ABAP program that does not belong in the global declaration section belongs to a processing block.

The possible processing blocks are:

- Procedures
- Methods
- Function modules
- Subroutines
- Dialog modules (exclusively for Dynpros)
- Event blocks

Function modules are always defined globally, whereas methods can be defined either globally or locally within the program, and subroutines should only ever be defined locally (subroutine pools now being considered obsolete). The flow logic of Dynpros calls dialog modules to perform their processing. Event blocks are triggered by events in the ABAP runtime environment.
A pass by value parameter copies the value to a new data object or, in the case of a changing parameter, copies it once when called and once again upon completion.

In ABAP, pass by reference is always quicker because no local object is created and no data transport is necessary when calling the procedure. Simply for performance reasons, pass by reference is usually preferred unless there is an explicit or implicit write to an input parameter within the procedure or unless you want to ensure that an input/output (changing) or output parameter is only returned if the procedure ends without error. In such cases, pass by value is mandatory so that the assigned parameter is not simultaneously modified in the calling code when there is a write access to a formal parameter.

The following shows a very simple example of the behavioral difference of call by reference versus call by value. Notice that the code (shown in Listing 9.5) provides four variables to the formal parameters of the subroutine. Two are passed by value, and two are passed by reference; one of each pair is passed as a USING parameter, and one of each pair is passed as a CHANGING parameter. At the point in the program when the call is made (PERFORM pass_by_example), the values of the variables in the calling routine are all zero (see Figure 9.2). At the point of the BREAK-POINT statement (see Figure 9.3), all of the formal parameters have a value of one. However, if the debugger point of view is changed to the calling routine, the original variables passed by reference have changed, whereas the variables passed by value have not (see Figure 9.4).

### FORM pass_by.

```abap
DATA:
  lv_num1 TYPE I,
  lv_num2 TYPE I,
  lv_num3 TYPE I,
  lv_num4 TYPE I.
PERFORM pass_by_example
  USING lv_num1 lv_num2
  CHANGING lv_num3 lv_num4.
ENDFORM.
```

### FORM pass_by_example USING value(p_by_value1) TYPE I

```abap
  p_by_ref1 TYPE I
  CHANGING value(p_by_value2) TYPE I
  p_by_ref2 TYPE I.
ADD 1 TO: p_by_value1, p_by_ref1, p_by_value2, p_by_ref2.
```

### Listing 9.5 Example Code

You should note the following behavior when using pass by reference:

- In subroutines, write access can occur to an input parameter defined with USING without the occurrence of syntax errors. This is not the case with methods or function modules defined with IMPORTING.
- An output parameter that is passed by reference acts like an input/output (changing) parameter. If there is a read access to an output parameter in the
procedure before the value of the parameter is changed, this value is not initial, unlike with pass by value, but is the same as the current value of the actual parameter from the calling code. This also means that if the called procedure does not change an output parameter, the actual parameter in the calling code remains unchanged.

- If a procedure is stopped because of an error (in other words, stopped for a reason other than that it has reached its last statement or a RETURN), all parameters retain the values they contained when the procedure was stopped.

To summarize, pass by reference is always preferable when performance is an issue, whereas pass by value is more suitable for situations where robustness and data consistency are important.

Only pass by reference can be specified for the obsolete table parameters in either subroutines or function modules. As we mentioned, pass by value is mandatory for the return value of function module methods, the output parameters of events in ABAP Objects, and all formal parameters of function modules that are either RFC-enabled or are update modules.

### ABAP Event Blocks

Event blocks are units of code that are executed in a sequence determined by external events: often the runtime environment for ABAP, and occasionally by a user’s actions. The sequence in which the processing blocks occur in the program is irrelevant. However, to make your programs easier to understand, you should include the event blocks in your program in approximately the same order as that in which the system will call them and prior to subroutines or local class implementations.

Event blocks are started using an event keyword and ended by the next processing block: either another event block or other type of procedural block. Within an event block, you cannot declare local data types or data objects. All declarative statements in event blocks belong to the ABAP program and are visible globally (in all subsequent processing blocks). There are two exceptions: The event blocks AT SELECTION-SCREEN and GET are implemented internally as procedures and can contain local data.

Event blocks can be grouped by type. Table 9.1 shows the event types that exist.

<table>
<thead>
<tr>
<th>Type of Event</th>
<th>Event Occurs in Program Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program constructor event</td>
<td>All program types</td>
</tr>
<tr>
<td>Reporting event</td>
<td>Only in executable programs</td>
</tr>
<tr>
<td>Selection screen event</td>
<td>Only during selection screen processing</td>
</tr>
<tr>
<td>List event</td>
<td>Only during list processing</td>
</tr>
</tbody>
</table>

### Table 9.1 Types of Events

With the exception of AT SELECTION-SCREEN and GET, event blocks can be listed multiple times in a program, but normally you will not see this. (The valid case where this would be used is extremely convoluted, involving sharing code between multiple programs or include programs or when the code is inadvertently included multiple times.) The event block START-OF-SELECTION can also be implicitly listed multiple times. Whenever an event occurs, all associated event blocks are executed in the order of their occurrence. Again, the event blocks are triggered by events in the ABAP runtime environment.

The following list details the event blocks and when or how they are triggered.

- **LOAD-OF-PROGRAM**
  This event keyword is triggered when the program is initially loaded into memory. This program constructor event is triggered for all program types. This can be useful to initialize or instantiate a data object when the program begins only once or to indicate to an external process that the program has started. The event is only triggered once, before any other event of a program.

- **INITIALIZATION**
  This reporting event keyword is triggered by the actual start of the program and before the selection screen processing of any existing standard selection screen (the only event that can occur before this is LOAD-OF-PROGRAM).

  When an executable program defines that a standard selection screen is called again by the ABAP runtime environment after an execution, the INITIALIZATION event is triggered again. An example of this is a report in which after displaying the report, the user clicks on the Back button or presses (F3) and returns to the selection screen to rerun the report. In this case, initializing parameters or selection criteria of the selection screen have no effect because
they are automatically supplied with the preceding user inputs from the selection screen during the selection screen event AT SELECTION-SCREEN OUTPUT.

### START-OF-SELECTION

In an executable program, all statements that are not declarations and are listed before the first explicit processing block—or if the program does not contain any explicit processing blocks—are assigned to an implicit event block START-OF-SELECTION, which is inserted before any START-OF-SELECTION event blocks. This event keyword is triggered when the program has completed all selection screen events.

If the program is linked to a logical database, preparatory tasks can be performed at START-OF-SELECTION before the logical database imports the data. If the program is not linked to a logical database, this event block becomes a type of “main program” from which procedures or screens are called. See Listing 9.6 for an explicit example and Listing 9.7 for an implicit example.

```abap
* PAI
AT SELECTION-SCREEN.
PERFORM pai_of_selection_screen.
START-OF-SELECTION.
PERFORM selection.
END-OF-SELECTION.
PERFORM e05_layout_build USING gs_layout.  *wg. Parameters
* Call ABAP/4 List Viewer
CALL FUNCTION 'REUSE_ALV_HIERSEQ_LIST_DISPLAY'
EXPORTING
  i_callback_program  = g_repid
  i_callback_pf_status_set  = ' ',
  i_callback_user_command  = ' ',
  is_layout  = gs_layout
  it_fieldcat  = gt_fieldcat[
  * IT_EXCLUDING
  it_special_groups  = gt_sp_group[]
  * IT_SORT
  * IS_SEL_HIDE
  * I_SCREEN_START_COLUMN  = 0
  * I_SCREEN_START_LINE  = 0
  * I_SCREEN_END_COLUMN  = 0
  * I_SCREEN_END_LINE  = 0
  * l_default  = g_default
  l_save  = g_save
```

### Listing 9.6 Example of Several Event Blocks

```abap
REPORT Z_HELLO_WORLD_SIMPLE.
WRITE / 'Hello World'.
```

### Listing 9.7 Example of an Implicit START-OF-SELECTION

```abap
GET node.
```

The reporting GET event defines an event block whose result is triggered by the ABAP runtime environment if the logical database to which the program is linked provides data in the work area node. The GET node also controls the behavior of the logical database. The event blocks after GET are implemented internally as procedures. Declarative statements in GET event blocks create local data. GET event blocks follow the START-OF-SELECTION event and are followed by the event block END-OF-SELECTION.

### END-OF-SELECTION

In this report event block, all data read by the logical database can be processed. In an executable program without a logical database, there is no need to implement the event block END-OF-SELECTION.
**Chapter 9 Basic ABAP Programs and Interface Creation**

- AT SELECTION-SCREEN
  This statement defines event blocks that are triggered by the ABAP runtime environment during selection screen processing. The event AT SELECTION-SCREEN is the basic form of a whole series of events that occur while the selection screen is being processed. The standard selection screen in an executable program or in the logical database linked to it is automatically called between the INITIALIZATION and START-OF-SELECTION events. You can define event blocks for these events in your program to change the selection screen or process user input.

  Selection screen events occur immediately before sending a selection screen, during the Process Before Output (PBO) of the selection screen (AT SELECTION-SCREEN OUTPUT), after certain user actions on a displayed selection screen (for example, ON VALUE REQUEST or ON HELP REQUEST), or prior to proceeding to the START-OF-SELECTION event during execution. They assist in selection screen processing in the ABAP program or provide validations for the selection screen.

- TOP-OF-PAGE
  This list creation event is triggered by the ABAP runtime environment during the creation of a list. This occurs when a new page is started—that is, immediately before the first line in a new page is to be output. All list outputs that take place in the event block are placed below the standard page header of the list. You cannot output more lines than are available in the page within the event block, and the NEW-PAGE statement is ignored within this event block.

- END-OF-PAGE
  This list creation event is triggered by the ABAP runtime environment during the creation of a basic list if there is a line reservation in the addition LINE-COUNT of the initiating statement for a page footer. A list output that takes place in the event block is placed in this area. Output statements that exceed the reserved area will be ignored.

- AT LINE-SELECTION
  This display of a screen list event is triggered by the ABAP runtime environment, provided the screen cursor is on a list line and you select a function using the function code Pick (or press F2) or double-click on a list line. This was often used to provide drill-down capability or to provide additional detail. During the line selection process, any fields output or placed on the line with the Hide command are returned to their respective global data objects.

- AT USER-COMMAND
  This display of a screen list event is triggered by the ABAP runtime environment if, during the display of a screen list, a function with a self-defined function code was chosen.

**Basic ABAP Statements**

The ABAP syntax is platform-independent. This means that it always has the same meaning or function, irrespective of the relational database system and operating system for the application and presentation server. Applications implemented in ABAP will also be able to run in future releases owing to the upward compatibility of the language. The only type of upward compatibility issue would be related to changing from a non-Unicode system to a Unicode system. For more information on this topic, please see Chapter 11.

For the ABAP syntax, the following general rules apply:

- ABAP programs consist of individual sentences (statements).
- The first word in a statement is called the ABAP keyword, although the keyword may be optional (COMPUTE, for example).
- Each statement ends with a period.
- Words must always be separated by at least one space.
- Statements can be indented.
- ABAP keywords, additions, and operands can be either upper- or lowercase (the ABAP runtime system does not differentiate).
- Statements can extend beyond one line.
- You can place several statements on a single line (though this is not generally recommended).
- Lines beginning with an asterisk (*) in the first column are recognized as comment lines and ignored.
- Double quote marks (”) indicate that the remainder of the line is a comment.
Chapter 9 Basic ABAP Programs and Interface Creation

Note
Although the ABAP runtime system does not differentiate between upper- and lowercase, it has become customary to write keywords and their additions in uppercase letters and operands in lowercase. This form of representation is also used throughout this book. It can be achieved in your own program by clicking on the Pretty Printer button and setting the appropriate option in the settings (Utilities • Settings • ABAP Editor • Pretty Printer).

You can combine consecutive statements with an identical beginning into a chained statement by doing the following:

- Write the identical beginning part of the statement followed by a colon.
- After the colon, list the end parts of the statements (separated by commas).

Note
This short form merely represents a simplified form of syntax and does not offer an improvement in performance.

The ABAP runtime system processes each of the individual statements. The short form makes the most sense with certain statements (for example, TYPES, DATA, SELECT-OPTIONS, PARAMETERS, CLEAR, MOVE, ADD, or WRITE).

Normally, a program is not made up of a single block, but of several units. Many of these modularization units can be used in more than one program. A good program should have at least the database access encapsulated (known also as a reuse component). This creates a division between the design of the program and the database accesses. It is then possible for you to use the same database accesses for different user dialogs or programs.

As an additional benefit of such an encapsulation, often it is possible to buffer the data within the encapsulation, meaning that it needs to be retrieved from the database only once. As an example, a number of standard function modules retrieve data from various tables and store it within the function group. Later attempts to retrieve the same data retrieve it from the internal tables within the function group rather than retrieve it again from the database tables. A common example of this type of buffering within a database encapsulation is the retrieval of address data. As a result, any address retrieval for a customer, vendor, or partner benefits from this optimization. Figure 9.5 shows an example of this.

Figure 9.5 Example of Database Encapsulation

A modularization unit is a part of a program where a particular function is encapsulated. You place part of the source code in a “module” to improve the transparency of the program and to use the corresponding function in the program several times (or in other programs if the modularization is global) without having to implement the entire source code again on each occasion. The improvement in transparency results from the program becoming more function-oriented. It divides the overall task into subfunctions, which are the responsibility of the corresponding modularization unit.

Modularization also simplifies both program maintenance and future design changes. It simplifies maintenance because you need to make changes to the function or corrections to the modularization unit only once and not throughout the programs. If the requirements change, often it is possible to switch out one modularization call for another. You can also process a call as a unit in the Debugger when executing your program.

It is possible to provide modularization either locally or globally. You have two techniques for local program modularization: subroutines (form routines) and methods within local classes. Local modularization units, by definition, are available only in the program in which they are implemented. No other program must be loaded to the user context at runtime when you use a local modularization unit.

Caution
For historical reasons, it is technically possible to call a subroutine from another program. You should not use this option, however, because this technique contradicts the principle of encapsulation of data and functions.
As with local program modularization, there are two techniques for global modularization: function modules and methods within global classes. Global modularization units can be used by any number of programs at the same time. The globally defined modularization units are stored in the ABAP Repository and only loaded when requested by the calling program.

Ideally, the modularization units that are called do not use the data objects of the calling program directly. Nor should the calling program change the data objects of the modularization unit directly. This principle is known as data encapsulation. This is an important technique in developing transparent, maintainable source code.

Parameters are used to exchange data between the calling code and the modularization unit. The total number of parameters in a modularization unit is called the interface or signature. Parameters are differentiated on whether they are used to pass data into the modularization unit (importing parameters), return data from the modularization unit (exporting parameters), or pass data into and return from the modularization unit (changing parameters).

**ABAP Subroutine**

A subroutine has the following characteristics:

- It is introduced with the `FORM` statement.
- You specify the name of the subroutine and the interface after `FORM`.
- The statements of the subroutine follow the `FORM` statement.
- The `ENDFORM` statement concludes the subroutine.

In the interface definition, you list the formal parameters of the subroutine and type them. A formal parameter is how you address the data object provided in the calling code within the subroutine. You must specify the pass type for each parameter:

- Call by value: `USING VALUE`
- Call by value and result: `CHANGING VALUE`
- Call by reference: `CHANGING` or `USING` (although `CHANGING` is recommended if the contents are changed)

**Listing 9.8 Call to a Subroutine with Both USING and CHANGING**

```
FORM create_request USING p_name TYPE progtname
           p_type
           CHANGING p_wb_request.
DATA: l_scope_objects TYPE rinfoobj,
      l_scope_objtypes TYPE rseutypes,
      l_object_type TYPE seu_objtype,
      l_object_name TYPE rsfind.
      l_object_type = p_type.
      l_object_name = p_name.
      CALL METHOD
        cl_wb_infosystem=>create_where_used_list_request
        EXPORTING
          p_object_type = p_object_type
          p_object_name = l_object_name
          p_scope_objects = l_scope_objects
          p_scope_object_types = l_scope_objtypes
        IMPORTING
          p_wb_request = p_wb_request
        EXCEPTIONS
          execute_in_batch = 1
          action_cancelled = 2
          error_occured = 3
          OTHERS = 4.
      ENDFORM. "create_request
```

You must specify the additions `TABLES`, `USING`, `CHANGING`, and `RAISING`, in this order (see Listing 9.8 and Listing 9.9 for an example of multiple types). Specifying `TABLES` after `USING` or `CHANGING` creates a formal parameter called `TABLES`. Likewise, specifying `USING` after `CHANGING` creates a formal parameter called `USING`.

```
IF cursor_field = 'SOURCE_ID' AND cursor_value NE space.
  PERFORM create_request
    USING cursor_value
      abap_pgeditor->abap_editor->context->context_type
      CHANGING l_wb_request.
  ELSE.
```

**Listing 9.9 The Subroutine (FORM Routine)**
With the addition of RAISING, class-based exceptions can be passed, which are triggered in the subroutine or propagated to the subroutine by the ABAP runtime environment or by using the statement RAISE EXCEPTION, but are not handled in a TRY block.

Internal tables should be passed with either USING or CHANGING. The other advantage of using one of these is that you can pass internal tables other than standard tables, for example sorted or hashed. We recommend that all parameters for any type of modularization unit be typed. Typing provides additional syntax checking during activation and provides faster execution during runtime.

**ABAP Function Module**

A function module is a routine or reuse component that is stored centrally in the Function Library of an SAP system. Each function module has an interface with parameters for importing or exporting data. The main purpose of a function module is its reusability. It therefore belongs to the reuse components. Function modules can also be accessed from outside the system by external systems or processes if they are a remote-enabled module.

Function modules are assigned to a function group. Each function group is a collection of function modules that have similar functions or process the same data. A function group can contain the same components as an executable program. (From a system perspective, they are a type of program but are not executable directly.) Table 9.2 shows the types of components that can be placed in a function group.

### Table 9.2 Function Module Elements

<table>
<thead>
<tr>
<th>Component</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data object</td>
<td>These are global in relation to the function group; that is, they are visible to and can be changed by all function modules within the group.</td>
</tr>
<tr>
<td>Subroutine</td>
<td>These can be called from all function modules in the group.</td>
</tr>
<tr>
<td>Screens</td>
<td>These also can be called from all function modules within the group.</td>
</tr>
</tbody>
</table>

As for subroutines, a function module can contain its own local types and data object definitions. These can only be seen within the function module. The interface of a function module can contain the elements shown in Table 9.3. Figure 9.6 shows a function module with all elements. You should type interface parameters with types from the ABAP Dictionary.

### Table 9.3 Function Module Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import parameter</td>
<td>Values or variables of the calling program can be transferred to non-optional import parameters when calling the function module. The optional parameters do not need to be supplied during the call.</td>
</tr>
<tr>
<td>Export parameter</td>
<td>The calling program accepts the output of the function module by assigning a receiving variable to an export parameter. Export parameters are always optional.</td>
</tr>
<tr>
<td>Changing parameter</td>
<td>You can transfer variables from the calling program to the function module, where they are changed and returned to the calling program.</td>
</tr>
<tr>
<td>Tables parameter (obsolete)</td>
<td>This parameter is similar to the changing parameter in that the table provided can be changed. However, it can only be used for standard tables. The formal parameter specified by this parameter within the function module will always have a header line.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>These can be raised by the function module in error situations and provide information regarding the processing error. Exceptions should always be caught and handled by the calling program.</td>
</tr>
</tbody>
</table>

You can use the logical expression IS SUPPLIED within a function module to see if an actual parameter was set for a formal parameter during a call. The older logical expression IS REQUESTED should no longer be used. The expression IS REQUESTED checked only the output parameter, whereas the logical expression IS SUPPLIED can be used with all optional parameters. The expression IS INITIAL should not be used because it does not take into account that a default value was used.
If a program calls a function module, the entire corresponding function group is loaded into the same internal session as the calling program, and the function module is executed. The function group remains loaded in memory until the calling program is closed. When you call another function module of this group, it is processed without needing to be reloaded, and the same global data exists. Therefore, if you call a function module that stores values in a global data object of the function group, other function modules in the same group can access this data when they are called during the same program.

You should note the **Optional** and **Pass Value** checkboxes (which in the example are shortened to **Opt...** and **Pa...**) in Figure 9.7, which control if the parameter is required and if it is copied (pass by value). In Figure 9.8, you should note that the **Optional** checkbox does not exist because all export parameters are optional. Figure 9.9 shows the changing parameters. Finally, Figure 9.11 shows the exceptions for this function module.

---

**Figure 9.6** A Function Module Interface Documented as Comments

If a program calls a function module, the entire corresponding function group is loaded into the same internal session as the calling program, and the function module is executed. The function group remains loaded in memory until the calling program is closed. When you call another function module of this group, it is processed without needing to be reloaded, and the same global data exists. Therefore, if you call a function module that stores values in a global data object of the function group, other function modules in the same group can access this data when they are called during the same program.

You should note the **Optional** and **Pass Value** checkboxes (which in the example are shortened to **Opt...** and **Pa...**) in Figure 9.7, which control if the parameter is required and if it is copied (pass by value). In Figure 9.8, you should note that the **Optional** checkbox does not exist because all export parameters are optional. Figure 9.9 shows the changing parameters. Finally, Figure 9.11 shows the exceptions for this function module.

---

**Figure 9.7** Function Module Import Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Associated Type</th>
<th>Default Value</th>
<th>Opt</th>
<th>Pass Val</th>
<th>Shortled</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBSELECT_DATE</td>
<td>LIKE</td>
<td>BTSCDO_DATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOBNAME</td>
<td>LIKE</td>
<td>JRSNAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRNO</td>
<td>LIKE</td>
<td>ERRNO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECON</td>
<td>LIKE</td>
<td>SELECON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST1</td>
<td>LIKE</td>
<td>TEST1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.8** Function Module Export Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Associated Type</th>
<th>Pass Val</th>
<th>Opt</th>
<th>Shortled</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBSELECT_DATE</td>
<td>LIKE</td>
<td>BTSCDO_DATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOBNAME</td>
<td>LIKE</td>
<td>JRSNAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRNO</td>
<td>LIKE</td>
<td>ERRNO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECON</td>
<td>LIKE</td>
<td>SELECON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST1</td>
<td>LIKE</td>
<td>TEST1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.9** Function Module Changing Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Associated Type</th>
<th>Optional</th>
<th>Shortled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_CODE</td>
<td>TYPE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.10** Function Module Tables Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Associated Type</th>
<th>Optional</th>
<th>Shortled</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBSELECT</td>
<td>LIKE</td>
<td>BTSCDO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOBNAM</td>
<td>LIKE</td>
<td>JRSNAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USERNAME</td>
<td>LIKE</td>
<td>USERNAME</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.11** Function Module Exceptions
ABAP Classes and Methods

In the course of extending the ABAP language to include object-oriented concepts, global classes were introduced that use methods to provide functions. Like function modules, methods also have an interface, known as a signature, which consists of importing, exporting, changing, and returning parameters and exceptions.

In addition to methods, classes have other components. They contain global data objects known as attributes. In the same way that global data objects of a function group can be accessed by all the function modules in the group, all methods can access the attributes of their classes.

With function groups, the global data objects are not visible outside the function group. This is the encapsulation of data in a function group. Attributes are also normally encapsulated within the class and can therefore only be read or changed using methods of the same class. However, in contrast to function modules, classes also allow you to make specific attributes visible to the users of the class. A distinction is therefore made between public and private attributes.

This distinction is applied not only to attributes, but also to methods. Whereas all function modules can be called from outside the function group, only public methods are available outside the class. Private methods can only be called by other methods of the same class and are thus similar to subroutines (form routines) within a function group.

The major difference between global classes and function groups is that a function group with its global data objects can only be loaded once to the program context for each main program, but the data of a global class can be loaded as many times as you like. This is known as multiple instantiation of the class. In practice, it means the global data object values are the same for all function module calls, because you have only one copy of each of the data objects of a function group. A class can have several instances, each of which is stored separately in the program context of the main program. Each instance can therefore have different attribute values. A method can see different values in the attributes depending on the instance from which you called it.

Attributes that can have different values for each instance are known as instance attributes to distinguish them from static (or class) attributes. Static attributes exist only once for each program context, regardless of how many class instances are generated. If instance methods access a static attribute, all instances see the same value.

Note the PASS VALUE and OPTIONAL checkboxes (which, in Figure 9.13, are shortened to P... and O...), which control if the parameter is copied (pass by value) and is required.

Figure 9.12 shows the attributes of a method. Figure 9.13 shows the parameters for the method. Figure 9.14 shows the exceptions for the method. The beginning of the method is shown in Listing 9.10.
method convert_number_base.
data:
lv_factor type P length 10,
lv_converted_number type P length 10,
lv_src_nbr type C length 99,
lv_iterations type I,
lv_position type I,
lv_max_value type I,
lv_digit type I,
lv_exponent type I.
field-symbols:
<digit> type C.
constants:
lc_starting_factor type I value '19',
lc_number_base type C length 36
value '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
if source_base is initial or
destination_base is initial.
raise base_not_identified.
endif.
lv_iterations = strlen( lc_number_base ).
Listing 9.10 Method Coding

Note
You should note that the terms instance and object are often used interchangeably. Further details regarding ABAP object-oriented programming is discussed in Chapter 14.

As with function modules and subroutines, you have the capability to import, export, change, and produce exceptions. Methods also provide the ability to return a value, which allows the call to be placed within either a COMPUTE statement or a logical expression. Another difference is that every parameter is required to be typed. This is different from both subroutines and function modules, where it is a recommendation that the parameters be typed.

ABAP Selection Screen
In general, selection screens are used for entering selection criteria for data selection. From a technical perspective, selection screens are Dynpros. However, they are not designed by the developer directly, but are generated in accordance with declarative statements in the source code.

The selection screen has the following standard functions:

- Text on the selection screen can be maintained in several languages. At runtime the text is automatically displayed in the user’s logon language.
- The system checks types automatically.
- In addition to single value entries (PARAMETERS), you can also implement complex selections (SELECT-OPTIONS) on the selection screen. With a complex selection (SELECT-OPTIONS), the user can enter multiple values, intervals, comparative conditions, or even patterns as restrictions.
If the input field is defined using an ABAP Dictionary element, the field documentation can be displayed on the input field by pressing the \( F1 \) function key. The value help attached to the ABAP Dictionary type displaying possible inputs can be called up by pressing the \( F4 \) function key. It is also possible for the label (selection text) to use the long text of the data element.

You can easily save the values (or calculation in the case of a dynamic date or time calculation) of complicated selection screens as selection screen variants for reuse or in background processing.

Listing 9.11 shows the coding for a selection screen, and Figure 9.15 shows the resulting generated selection screen.

```abap
***-------------------------------- Parameters/Select-Options
selection-screen begin of block c01
  with frame title text-c01.
  parameters:
    p_custzg as checkbox default 'X',
    p_wrkbch as checkbox default 'X',
    p_repair as checkbox default 'X',
    p_others as checkbox default 'X'.
  selection-screen end of block c01.
selection-screen begin of block c02
  with frame title text-c02.
  parameters:
    p_r_po radiobutton group sort,
    p_e_s_r radiobutton group sort,
    p_r_s_d radiobutton group sort,
    p_d_s_r radiobutton group sort,
    p_r_d_s radiobutton group sort.
  selection-screen end of block c02.
selection-screen begin of block c03
  with frame title text-c03.
  select-options:
    s_trkorr for gs_outtab-trkorr.
  selection-screen end of block c03.
selection-screen begin of block c04
  with frame title text-c04.
  select-options:
    s_user for gs_outtab-as4user modif id c04.
  selection-screen end of block c04.
```

Further details regarding selection screens are discussed in Chapter 13.

Authorization Checks

Critical data and parts of the functional scope of the SAP system must be protected from unauthorized access. You have to implement authorization checks in
your program so that users can only access areas for which they are authorized. To code an authorization check, use the \texttt{AUTHORITY-CHECK} statement to check whether the current user has the authorization required for executing the function in his master record. Depending on the check result returned in the SY-SUBRC, you can continue accordingly. If the return code equals 0 (zero), then the user has the required authorization. Authorization objects can also be defined in report transaction codes to provide additional restrictions other than simply by transaction code.

Under normal circumstances, the definition of authorization objects is part of data modeling and the creation of database tables. Implementing the authorization check is one of the tasks of the developer who programs access to the database tables. The subsequent steps, such as defining the authorizations and user profiles and assigning them to the user master records, are the tasks of the administrator or security team.

Before you can implement the required authorization check in your program, you must first determine the structure (the fields) of the corresponding authorization object. An object usually consists of the \texttt{ACTVT} (activity) field and one or several other fields, which specifies the data type to be protected (for example, material number, organizational unit, account number, and so on).

\begin{itemize}
\item \textbf{Tip}
\end{itemize}

If you do not want to carry out a check for a field, either do not enter it in the \texttt{AUTHORITY-CHECK} statement or enter \texttt{DUMMY} as the field value. \texttt{DUMMY} is a predefined description entered without quotation marks.

Consider this example of a suppressed field check: When a change transaction is called, the system should always check immediately whether the user has any change authorization. If the check fails, an appropriate message is to be output to the user immediately. (See the first \texttt{AUTHORITY-CHECK} in Listing 9.12 for an example of the syntax; Figure 9.16 shows the authorization object.)

\begin{verbatim}
if p_s_develop-devclass = space.
  authority-check object 'S_DEVELOP'
    id 'DEVCLASS' field p_s_develop-devclass
    id 'OBJTYPE' field p_s_develop-objtype
    id 'OBJNAME' field p_s_develop-objname
    id 'P_GROUP' dummy
    id 'ACTVT' field l_develop_actvt.
  endif.
else.
  endif.
\end{verbatim}

\textbf{Listing 9.12} Call to an Authorization Check

Again, to avoid spelling errors in object and field names, you can click on the \texttt{Pattern} button to generate the appropriate \texttt{AUTHORITY-CHECK} statement.

\textbf{ABAP Dynpros}

As mentioned earlier, standard screens, or Dynpros, are designed by the developer directly through the use of the Screen Painter. A screen consists not only of
its layout with input and output fields, buttons, and other screen elements, but also a processing logic known as flow logic. The fact that the ABAP Dictionary is integrated into the system provides automatic consistency checks for screen input fields. These checks include type checks, foreign key checks, and fixed value checks.

The checks we mentioned can be complemented with other program checks. Techniques are available for screens that allow you to control the order in which checks are performed. If an error occurs, the appropriate fields are made input-ready again.

The layout can be designed very flexibly, with input fields, output fields, radio buttons, check fields, and buttons with which corresponding functions of the program can be executed.

Further, screens have the same formatting options as lists and selection screens: Fixed point numbers and dates are formatted according to the settings in the user master record, times are formatted as HH:MM:SS, currency amounts are formatted according to the currency, and physical measurements (lengths, weights, quantities, etc.) are formatted according to their unit fields.

In theory, there are two options for starting a screen sequence:
- **By calling the first screen (using the CALL SCREEN statement) from a processing block in your program**
- **By creating a dialog transaction that references the program and the first screen**

After a screen is processed, the statically or dynamically defined screen sequence is processed. A formal next screen of 0 (zero) returns processing to the point where the screen was called or ends the dialog transaction.

Screens can appear in executable programs, function groups, or module pools:
- **Executable program**
  Executable programs (reports) use screens to display data in addition to the list output, or to replace the list output display completely, and sometimes for the display of an ALV Grid. You can also use screens to enter and change data in the list. For the purpose of reusability and data encapsulation, you should no longer create screens directly in reports, but use screens in function groups instead.

- **Function group**
  Function groups often use screens to encapsulate a screen or screen sequence for reuse. SAF provides a vast number of such encapsulated screens or screen sequences. You can find various SAP standard function modules in the Repository Information System simply by looking for “popup” as part of the function module name. These encapsulated screens exist for displaying messages, accepting values, and selecting between options.

- **Module pool**
  Module pools can only be started with a transaction code; they cannot be tested as other types of executable programs directly in the ABAP Workbench (Transaction SE80) or ABAP Editor (Transaction SE38). In contrast to screens in function groups, you cannot encapsulate module pools or provide an external interface.

When a screen is displayed for a user, the presentation server performs all interactions until a function is selected. A function corresponds to either a button on the application toolbar, a menu, one of the function keys (F1 through F12), or the Enter key. Each of these actions by a user assigns the function code defined by the programmer to a special field associated with every screen. This field is normally referred to as the OK_CODE. You normally name this field and provide a global field with the same name to receive the value in your program. The field should be defined to use the type SYUCCOMM. This transport to this variable is automatic. During the Process After Input (PAI) you can determine what action the user performed through the contents of this variable.

The statement CALL SCREEN accesses the Dynpros of the relevant main program of the current program group, and these use the global data and dialog modules of the main program. If the specified Dynpro does not exist in the main program of the program group, an untreatable exception occurs.

You use the addition STARTING AT to open a new popup level and to display all screens of the called Dynpro sequence in a modal dialog box rather than as a full screen. The upper-left corner of the dialog window is determined by the values following the addition STARTING AT for the column and line. The values refer to the window with popup level 0. The lower-right corner is set automatically from the size of the window, or you can specify it after ENDING AT. For column and line number (for either the top-left or bottom-right corner), data objects of type I are expected. The values of the top-left corner should be smaller than those of the
bottom-right corner; otherwise, the behavior is undefined. The maximum popup level is 9.

If, during the processing of a modal dialog box, a new Dynpro sequence is called, it must be started in another popup level. You cannot use the statement CALL SCREEN without adding STARTING AT in this case. When calling a Dynpro in a dialog window, specify the window as a modal dialog window in its properties and set an appropriate GUI status beforehand. We recommend that a Dynpro sequence in a modal dialog window consist of only one Dynpro.

Further details regarding Dynpro screens are discussed in Chapter 12.

**Dialog Messages**

You use the MESSAGE statement to send dialog messages to the users of your program. When you do this with an actual message defined from a message class, you must specify at least a three-digit message number, the message class, and the type of the message. It is also possible to provide the text of the message or just the type of the message.

Message number and message class clearly identify the message to be displayed. You use the message type (either A, E, I, S, W, or X) to define how the ABAP runtime should process the message. Table 9.4 offers an explanation of these message types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>General Program Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Termination</td>
<td>The message appears in a dialog box, and the program terminates. When the user has confirmed the message, control returns to the next-highest area menu.</td>
</tr>
<tr>
<td>E</td>
<td>Error</td>
<td>Depending on the program context, an error dialog appears or the program terminates.</td>
</tr>
<tr>
<td>I</td>
<td>Information</td>
<td>The message appears in a dialog box. Once the user has confirmed the message, the program continues immediately after the MESSAGE statement.</td>
</tr>
<tr>
<td>S</td>
<td>Status</td>
<td>The program continues normally after the MESSAGE statement, and the message is displayed in the status bar of the next screen.</td>
</tr>
</tbody>
</table>

Table 9.4 Message Types and Their Behavior

If the specified message contains placeholders (specified as either & or &#, where # is 1, 2, 3, or 4), you can supply them with values from your program by using the WITH addition. Instead of the placeholders, the transferred values then appear in the displayed message text. There can only be up to four placeholders, and if a number is part of the placeholder, it specifies the order in which the placeholders are filled. This allows for text translation that may require the placeholders in a different order (see Listing 9.13 for an example). Messages can either be displayed in modal dialog boxes or in the status bar of the screen based on the user’s settings. How a message is processed depends on its type and on the context in which it is sent.

If the introductory statement of a program contains the addition MESSAGE-ID id and your message to be displayed is of the same class, you can use the short form of the message statement without specifying the class. Listing 9.14 contains two examples of messages using this form, and Listing 9.15 shows the introductory statement.

```vba
call function 'ENQUEUE_ESRDIRE' 
  exporting 
    mode_trdir = l_enq_mode 
    name = incl2 
    _x_name = ' ' 
    _scope = 'Z' 
    _wait = ' ' 
  exceptions 
    foreign_lock = 01 
    system_failure = 02. 
if sy-subrc ne 0 and rs38l-extern = space. 
message id sy-msgid 
key Concepts Refresher Chapter 9 293
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>General Program Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>Warning</td>
<td>Depending on the program context, a warning dialog appears or the program terminates.</td>
</tr>
<tr>
<td>X</td>
<td>Exit</td>
<td>No message is displayed, and the program terminates with a short dump. Program terminations with a short dump normally only occur when a runtime error occurs. Message type X allows you to force a program termination. The short dump contains the message ID.</td>
</tr>
</tbody>
</table>

Table 9.4 Message Types and Their Behavior (Cont.)
type 'E'
number sy-msgno
with sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
endif.

Listing 9.13 Message Specifying All Options

call function 'CHANGEDOCUMENT_OPEN'
exporting
objectclass = 'WFRULES'
objectid = objectid
planned_change_number = planned_change_number
planned_or_real_changes = planned_or_real_changes
exceptions
sequence_invalid = 1
others = 2.
case sy-subrc.
  when 1. message a600 with 'SEQUENCE INVALID'.
  when 2. message a600 with 'OPEN ERROR'.
endcase.

Listing 9.14 Two Messages in the Shortened Form

function-pool wfrc
  message-id cd
.

Listing 9.15 Introductory Statement Specifying the Message Class

This general behavior is ultimately driven by the context of where the call to the message statement occurs. One unique context is when you have no screen. This includes the following processing blocks; all other processing blocks are associated with screen processing and are reacting to user input:

- The program constructor LOAD-OF-PROGRAM
- PBO modules (PBO of screens); see Chapter 12 for more information
- The selection screen event AT SELECTION-SCREEN OUTPUT (PBO of a selection screen)
- The reporting events INITIALIZATION, START-OF-SELECTION, GET, and END-OF-SELECTION
- The list events TOP-OF-PAGE and END-OF-PAGE

To make Table 9.5 more concise, these processing blocks are structured into two groups:

- **Group 1**
  LOAD-OF-PROGRAM, PBO module of screens, AT SELECTION-SCREEN OUTPUT

- **Group 2**
  Reporting and list events (INITIALIZATION, START-OF-SELECTION, GET, END-OF-SELECTION, TOP-OF-PAGE, and END-OF-PAGE)

<table>
<thead>
<tr>
<th>Type</th>
<th>Display</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Dialog box</td>
<td>Program terminates, and control returns to last area menu.</td>
</tr>
<tr>
<td>E</td>
<td>Group 1: Dialog box</td>
<td>Group 1: Behaves like type A.</td>
</tr>
<tr>
<td></td>
<td>Group 2: Status line of current window</td>
<td>Group 2: Program termination and display of an empty screen with empty GUI status. After the user action: Return to the calling position of the program.</td>
</tr>
<tr>
<td>I</td>
<td>Group 1: Status line of the next screen</td>
<td>Program continues processing after the MESSAGE statement.</td>
</tr>
<tr>
<td></td>
<td>Group 2: Dialog box</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Status line of next screen</td>
<td>Program continues processing after the MESSAGE statement.</td>
</tr>
<tr>
<td>W</td>
<td>Group 1: Status line of the next screen</td>
<td>Group 1: Behaves like type S.</td>
</tr>
<tr>
<td></td>
<td>Group 2: Status line of the current window</td>
<td>Group 2: Program termination and display of an empty screen with empty GUI status. After the user action: Return to the calling position of the program.</td>
</tr>
<tr>
<td>X</td>
<td>None</td>
<td>Triggers a runtime error with short dump.</td>
</tr>
</tbody>
</table>

Table 9.5 General Message Type Behavior

Another unique context includes all situations where a screen is being processed; that is, the program is reacting to user input. In ABAP programs, this means all Process After Input (PAI) modules. Table 9.6 shows the message behavior during PAI.
A unique context includes all situations where a selection screen is being processed; that is, the program is reacting to user interaction with lists. In ABAP programs, this corresponds to the `AT SELECTION-SCREEN` statement. The system then restarts selection screen processing directly after the `MESSAGE` statement. You cannot use error messages with the `ON HELP-REQUEST`, `ON VALUE-REQUEST`, and `ON EXIT` additions. Instead, a runtime error occurs.

Table 9.8 shows the behavior during list processing.

<table>
<thead>
<tr>
<th>Type</th>
<th>Display</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Dialog box</td>
<td>Program terminates, and control returns to last area menu.</td>
</tr>
<tr>
<td>E</td>
<td>Status line</td>
<td>Selection screen processing terminates, and the selection screen is redisplayed. The screen fields specified through the additions to the <code>AT SELECTION-SCREEN</code> statement are ready for input. If there are no additions to the statement, then all screen fields are ready for input. The user must enter a new value.</td>
</tr>
<tr>
<td>I</td>
<td>Dialog box</td>
<td>Program continues processing after the <code>MESSAGE</code> statement.</td>
</tr>
<tr>
<td>S</td>
<td>Status line</td>
<td>Program continues processing after the <code>MESSAGE</code> statement.</td>
</tr>
<tr>
<td>W</td>
<td>Status line</td>
<td>Like type E, but the user can confirm the message by pressing <code>[Enter]</code> without having to enter new values. The system then resumes selection screen processing directly after the <code>MESSAGE</code> statement. You cannot use warnings with the <code>ON HELP-REQUEST</code>, <code>ON VALUE-REQUEST</code>, and <code>ON EXIT</code> additions. Instead, a runtime error occurs.</td>
</tr>
<tr>
<td>X</td>
<td>None</td>
<td>Triggers a runtime error with short dump.</td>
</tr>
</tbody>
</table>

Table 9.7 Messages During List Processing (Cont.)

A unique context includes all situations where a list is being processed; that is, the program is reacting to user interaction with lists. In ABAP programs, this includes the following processing blocks:

- `AT LINE-SELECTION`
- `AT USER-COMMAND`
- `AT PFnn`
- `TOP-OF-PAGE DURING LINE-SELECTION`

Table 9.8 shows the behavior during list processing.
The last unique context is in function modules and methods. Messages have two different functions in function modules and methods:

- Normal message
- Triggering an exception

If you use messages in function modules and methods without the RAISING addition in the MESSAGE statement, the message is handled normally according to the context in which it is called within the function module or method. In other words, if it is not caught, it behaves as it would from one of the other contexts, depending on where the call for the function module or method occurs.

If you use messages in function modules and methods with the addition RAISING exc in the MESSAGE statement, the way the message is handled depends on whether the calling program catches the exception exc or not:

- If the calling program does not catch the exception, the message is displayed and handled according to the context in which it occurs in the function module or method from which it was called. This, again, is as if the error occurred where the call to the function module or method was made.
- If the calling program catches the exception, the message is not displayed. Instead, the procedure is interrupted in accordance with the message type, and processing returns to the calling program. The contents of the message are placed in the system fields SY-MSGID, SY-MSGTY, SY-MSGNO, and SY-MSGV1 through SY-MSGV4.

### Important Terminology

For the certification, you should know that repository objects are development objects such as programs or classes in the ABAP Workbench. Each repository object is assigned to a package. A change request contains at least one task for each developer working on the change. A change request records and manages all changes made to repository objects and customizing settings during a development project. A task is the information carrier for entering and managing all changes to repository objects and customizing settings performed by developers within a development project. Simply put, development is assigned to a task that is part of a change request.

You should understand that global declarations appear in a section after the introductory program statement of an ABAP program in which the data types, classes, and data objects that are visible in the whole program can be declared.

A modularization unit of an ABAP program is something that cannot be separated or nested. Processing blocks are procedures, dialog modules, and event blocks. They contain statements that are structured using control structures in statement blocks. Every non-declarative statement of an ABAP program is part of a processing block. A procedure interface consists of formal parameters and states the possible exceptions of the procedure.

With pass by reference, you pass only the address of the actual parameter to the formal parameter. Formal parameters do not occupy their own memory space. Within the subroutine, you work only with the field from the calling program. If you change the formal parameter, the field contents in the calling program change, too.

In pass by value, a local data object is created as a copy of the actual parameter, and when the procedure is called, the value of the actual parameter is passed to it. Changed formal parameters are only passed to the actual parameter if the procedure ends without errors.

The term event can mean either a component of a class or interface declared using (CLASS) EVENTS or an event of the ABAP runtime environment. We discussed the latter in this chapter. More details of the former can be found in Chapter 14. Events in classes trigger event handlers, and events of the ABAP runtime environment trigger event blocks. An event block is a processing block without a local data area that can be defined in every ABAP program—except for subroutine pools, class pools, and interface pools—and is processed when a specific event of the ABAP runtime environment occurs. It begins with an event keyword and ends with the start of the next modularization.

<table>
<thead>
<tr>
<th>Type</th>
<th>Display</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Status line</td>
<td>Program continues processing after the MESSAGE statement.</td>
</tr>
<tr>
<td>W</td>
<td>Status line</td>
<td>Like type E.</td>
</tr>
<tr>
<td>X</td>
<td>None</td>
<td>Triggers a runtime error with short dump.</td>
</tr>
</tbody>
</table>
Practice Questions

These practice questions will help you evaluate your understanding of the topic. The questions shown are similar in nature to those found on the certification examination. Although none of these questions will be found on the exam itself, they allow you to review your knowledge of the subject. Select the correct answers and then check the completeness of your answers in the following solution section. Remember that you must select all correct answers and only correct answers on the exam to receive credit for the question.

1. ABAP is a programming language that... (Select all that apply.)
   - [ ] A. Executes on all three levels of the three-tier architecture.
   - [ ] B. Controls the business logic.
   - [ ] C. Processes and formats data.
   - [ ] D. Interacts with the user.
   - [ ] E. Separates program code from language text.

2. Which events can exist in all types of programs that actually contain executable statements?
   - [ ] A. LOAD-OF-PROGRAM
   - [ ] B. INITIALIZATION
   - [ ] C. START-OF-SELECTION
   - [ ] D. AT LINE-SELECTION
   - [ ] E. AT USER-COMMAND
   - [ ] F. AT PF##

3. Dynpros can be placed in which program types? Select all that apply.
   - [ ] A. Executables
   - [ ] B. Module pools
   - [ ] C. Function groups
   - [ ] D. Class pools

4. A change request is part of a task.
   - [ ] A. True
   - [ ] B. False

5. Which statements about ABAP are true?
   - [ ] A. Each statement must begin with a keyword.
   - [ ] B. Each statement must end with a period.
   - [ ] C. ABAP keywords and additions must be in uppercase.

6. A development object can be assigned to only one package.
   - [ ] A. True
   - [ ] B. False

7. A development object can be assigned to only one change request.
   - [ ] A. True
   - [ ] B. False

8. Each ABAP program that actually contains executable statements... (Select all that apply.)
   - [ ] A. Is divided into processing blocks.
   - [ ] B. Assigns every executable statement to a processing block regardless of it being in a processing block.
   - [ ] C. Only assigns executable statements in a processing block to a processing block.
   - [ ] D. Uses event blocks to trigger events in ABAP.
   - [ ] E. Has declarative statements outside of processing blocks that are considered local.
   - [ ] F. Has declarative statements inside of processing blocks that are considered local.
   - [ ] G. Can be tested from the ABAP Workbench by pressing (F8).
9. Which modularization units can raise an exception? Select all that apply.
   - [ ] A. Function modules
   - [ ] B. Methods
   - [ ] C. Subroutines (FORM routines)

10. Which types of programs or parts of programs can be tested directly from the ABAP Workbench or ABAP Editor? Select all that apply.
   - [ ] A. REPORT
   - [ ] B. PROGRAM
   - [ ] C. FUNCTION-POOL
   - [ ] D. FUNCTION MODULE
   - [ ] E. CLASS-POOL
   - [ ] F. METHOD
   - [ ] G. INTERFACE-POOL
   - [ ] H. TYPE-POOL
   - [ ] I. INCLUDE

11. Which method of passing parameters is preferred for its performance?
   - [ ] A. Pass by reference
   - [ ] B. Pass by value

12. Which modularization units are global? Select all that apply.
   - [ ] A. Function modules
   - [ ] B. Subroutines (FORM routines)
   - [ ] C. Methods within a local class in the program
   - [ ] D. Methods within class pools

13. FORM routines (subroutines) can be used in which program types? Select all that apply.
   - [ ] A. Executables
   - [ ] B. Module pools

14. You can use the logical expression IS SUPPLIED for any formal parameter passed to which modularization unit? Select all that apply.
   - [ ] A. Subroutine (FORM routine)
   - [ ] B. Function module
   - [ ] C. Static method
   - [ ] D. Instance method

15. A selection screen can only be defined in an executable program.
   - [ ] A. True
   - [ ] B. False

16. Subroutines provide which types of parameters? Select all that apply.
   - [ ] A. Input
   - [ ] B. Output
   - [ ] C. Input/output (changing)
   - [ ] D. Return values
   - [ ] E. Exceptions

17. Function modules provide which types of parameters? Select all that apply.
   - [ ] A. Input
   - [ ] B. Output
   - [ ] C. Input/output (changing)
   - [ ] D. Return values
   - [ ] E. Exceptions
18. Methods provide which types of parameters? Select all that apply.

- A. Input
- B. Output
- C. Input/output (changing)
- D. Return values
- E. Exceptions

19. It is not possible to test a function module if another function module of the same function group contains a syntax error.

- A. True
- B. False

20. Each button on a Dynpro (screen) requires the assignment of a function code. This function code...

- A. Is used to define global variables that receive a value when the button is clicked.
- B. Can be used to identify when the button is clicked by looking for the function code in the screen’s OK_CODE field.
- C. Prevents the function code from be assigned to a menu item.

21. Which message types behave the same regardless of the context in which they are called? Select all that apply.

- A. A
- B. E
- C. I
- D. S
- E. W
- F. X

22. Authorizations are handled automatically during access to the database interface to restrict data access of the user.

- A. True
- B. False

Practice Question Answers and Explanations

1. Correct answers: B, C, D, E

   ABAP does not, however, execute on the database server. Although it can be considered to interact with the database through the database interface, it does not execute on the database level. Nor does it actually execute on the presentation server. It interacts with the presentation server normally through the SAP GUI, but the only execution occurs on the application servers.

2. Correct answer: A

   The only event common to all executable programs is LOAD-OF-PROGRAM.

3. Correct answers: A, B, C

   Dynpros are not possible in methods or classes.

4. Correct answer: B

   A change request holds a task.

5. Correct answer: B

   All ABAP statements must end with a period. Whereas it is certainly true that ABAP requires the statement to begin with a keyword, there are cases where the keyword is optional. The most obvious is the COMPUTE keyword, which we have never seen coded in a program (though it certainly helps to know the
keyword exists when you are trying to find help on an arithmetic function while in the editor). ABAP does not care if the case of the statements is upper, lower, or mixed.

6. Correct answer: A
A development object can only be assigned to a single package. It can be reassigned to a different package, but it is only ever assigned to one package at a time.

7. Correct answer: B
A development object can only be assigned to multiple projects or change requests. Only one project or change request can be active at a time, but over time, it is common for development objects to be assigned to different projects and change requests, and they are changed.

8. Correct answers: A, B
A program is divided into processing blocks, and every executable statement is assigned to a processing block either explicitly or implicitly. Any statement not explicitly included in a processing block is implicitly included in START-OF-SELECTION. Declarative statements inside a modularization unit are almost always considered local, whereas those outside the modularization units are considered global; the exception that invalidates F is that MODULES and event block declarations are considered global. Event blocks are called when the ABAP runtime triggers them.

Whereas almost every ABAP program can be tested simply by pressing (F8) while in the editor, one type (a module pool) cannot be started directly from the editor or ABAP Workbench. Module pools (dialog programs) must start with a particular screen, and the only way is to run the transaction code that associates the first screen to process. To be able to test in the ABAP Workbench, you must first navigate to the transaction code, which you can then execute by pressing (F8).

9. Correct answers: All options
It is currently possible to raise a class-based exception from all three of these modularization units.

10. Correct answers: A, D, E, F
Reports, function modules, classes, and methods can all be executed and tested directly. Programs (module pools) can be indirectly tested by testing their corresponding transaction code. Function groups, interface pools, type pools, and includes cannot be tested independently.

11. Correct answer: A
Pass by reference is always preferable when performance is an issue, whereas pass by value is more suitable for situations where robustness and data consistency are important.

12. Correct answers: A, D
Any modularization that can be reused is considered global. A method defined within a program cannot be reused, nor should a subroutine (form routine). Both function modules and methods defined in a class pool are considered global.

13. Correct answers: A, B, C, F
Subroutines cannot be used in classes. Interface pools and type pools cannot have any executable code.

14. Correct answers: B, C, D
Subroutines do not have optional parameters, and therefore the logical expression IS SUPPLIED cannot be used.

15. Correct answer: B
Selection screens can be defined in any program that allows screens. Originally, only one selection screen (screen 1000) was allowed in an executable program, but with later releases it is now possible to define as many as you want. This capability is why it is possible to place selection screens in programs other than just executable programs.

16. Correct answers: C, E
Subroutines technically only provide input/output (CHANGING) types of parameters (through USING and CHANGING) and exceptions.

Function modules provide input (IMPORTING), output (EXPORTING), and input/output (CHANGING) types of parameters and exceptions (EXCEPTIONS).
18. Correct answers: **All options**

Methods provide input (**IMPORTING**), output (**EXPORTING**), input/output (**CHANGING**), and return value (**RETURNING**) types of parameters and exceptions (**EXCEPTIONS**).

19. Correct answer: **A**

ABAP looks at the whole program during both generation and execution. A syntax error in any part will prevent it from executing.

20. Correct answer: **B**

The function code of any menu, button, or function key is always placed into the **OK_CODE** field associated with the screen. It can also be found in the system variable **SY-UCOMM**.

21. Correct answers: **A, F**

The abort or terminate message type always causes a program termination, and control returns to the last area menu. The exit message type always triggers a runtime error with short dump.

22. Correct answer: **B**

It is the programmer’s responsibility to program authority checks to perform the necessary authorization validation.

23. Correct answers: **B, D**

The introductory statement must be the first executable line or the first statement executed. It is possible to have comments precede the introductory statement, or the statement may be placed in an **INCLUDE**, which may be the first line in the program.

The introductory statement is often modified, most frequently to assign a default error class to the program.

24. Correct answer: **B**

No one outside those assigned to the request will have the ability to change the object, but anyone assigned to the request will have the ability to make a change (provided it is not actively being worked on).

**Takeaway**

You need to understand how transports are organized. Remember that a change request is commonly called a transport and contains tasks assigned to specific developers. It is also important to recognize the various types of programs: Executable programs (also known as reports), module pools (also known as dialog programs), function groups (which contain function modules), and class pools (which contain methods) are the primary program types you need to understand. Within each of these, you need to understand the types of modularization and how events are triggered.

You also need to be able to distinguish which types of screens, when displayed, use the SAP GUI versus a web browser. Classical screens, or Dynpros, and selection screens use the SAP GUI, whereas Web Dynpro and Business Server Pages, for example, use a web browser. Web Dynpro is discussed in detail in Chapter 20. You are expected to be able to identify how a classical screen (Dynpro) handles fields and buttons and their use in function modules. Included with this classical screen handling are both selection screens and modal dialog boxes.

You will need to understand the use of authorization checks and modularization units. This includes how to call them and options for providing data.

Finally, you must know how messages work in different contexts. You will need to understand where messages are displayed and the behavior of the program following the message production.

**Refresher**

You must understand various types of modularization blocks. Events, subroutines, methods, function modules, and dialog modules will all play a part in the certification examination. You must know how they are defined, how they are called (including their interface or signature), and how they behave with the production of a dialog message. You must have a thorough understanding of the techniques of both pass by value and pass by reference.

Classical screen (Dynpro) handling is also essential knowledge. You must understand how screens are called and what distinguishes a screen from a modal dialog box and from a selection screen. You should understand how to define all three and in which types of programs they can be produced.

Table 9.9 shows the key concepts of basic ABAP programs and interfaces creation.
Chapter 9  Basic ABAP Programs and Interface Creation

Tips

As with the majority of the subjects covered in the certification examination, it is important to have as much practical experience with the subject as possible. Unlike the majority of subjects found in the certification examination, this is an area where, if you have been programming in ABAP, you will already have sufficient experience.

Whereas the vast majority of the concepts presented in this chapter should be second nature, it is important that you understand the behavioral differences of interfaces (or signatures) and of message production.

Summary

You should now be able to create various types of ABAP programs. You should also have an understanding of the different types of encapsulation that a program can use and the interaction between a user and the program through the use of screens and messages. These skills will allow you to successfully complete this portion of the certification exam.
# Contents

Acknowledgments ......................................................................................... 15  
Preface .......................................................................................................... 17  

## PART I: General Introduction

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABAP Development Certification Track: Overview</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Certification Exam Scoring Changes</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Certification Levels</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Advanced Certification Levels</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Becoming an SAP Certified ABAP Development Associate: Overview</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Associate Examination Specifics</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Competency Areas</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Courses and Experience</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Training Courses for ABAP</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Sources of Information</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Strategic Understanding</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>SAP Examination Strategies</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>General Examination Strategies</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>45</td>
</tr>
</tbody>
</table>

## PART II: Exam Preparation

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>SAP NetWeaver: Overview</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Objectives of this Portion of the Test</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Key Concepts Refresher</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>SAP Products in a Nutshell</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Product Evolution</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>SAP NetWeaver Architecture</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Kernel and Administration Services</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Software-Oriented View</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>User-Oriented View</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>Structure of a Work Process</td>
<td>65</td>
</tr>
</tbody>
</table>
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Important Terminology</td>
<td>69</td>
</tr>
<tr>
<td>Practice Questions</td>
<td>70</td>
</tr>
<tr>
<td>Practice Question Answers and Explanations</td>
<td>72</td>
</tr>
<tr>
<td>Takeaway</td>
<td>73</td>
</tr>
<tr>
<td>Refresher</td>
<td>74</td>
</tr>
<tr>
<td>Tips</td>
<td>75</td>
</tr>
<tr>
<td>Summary</td>
<td>75</td>
</tr>
</tbody>
</table>

## 4 ABAP Workbench Usage

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives of this Portion of the Test</td>
<td>78</td>
</tr>
<tr>
<td>Key Concepts Refresher</td>
<td>79</td>
</tr>
<tr>
<td>ABAP Workbench</td>
<td>79</td>
</tr>
<tr>
<td>Repository Browser</td>
<td>85</td>
</tr>
<tr>
<td>Repository Information System</td>
<td>86</td>
</tr>
<tr>
<td>Workbench Settings</td>
<td>89</td>
</tr>
<tr>
<td>ABAP Editor and Workbench Settings</td>
<td>90</td>
</tr>
<tr>
<td>ABAP Workbench Tools in Detail</td>
<td>94</td>
</tr>
<tr>
<td>Enhancement Information System</td>
<td>96</td>
</tr>
<tr>
<td>Packages and Their Attributes</td>
<td>99</td>
</tr>
<tr>
<td>Transport Organizer</td>
<td>102</td>
</tr>
<tr>
<td>Practice Questions</td>
<td>105</td>
</tr>
<tr>
<td>Practice Question Answers and Explanations</td>
<td>108</td>
</tr>
<tr>
<td>Takeaway</td>
<td>111</td>
</tr>
<tr>
<td>Refresher</td>
<td>111</td>
</tr>
<tr>
<td>Summary</td>
<td>112</td>
</tr>
</tbody>
</table>

## 5 ABAP Debugger Program Usage

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives of this Portion of the Test</td>
<td>115</td>
</tr>
<tr>
<td>Key Concepts Refresher</td>
<td>115</td>
</tr>
<tr>
<td>New and Classic Debugger</td>
<td>116</td>
</tr>
<tr>
<td>New Debugger Tools and UI</td>
<td>120</td>
</tr>
<tr>
<td>Assertions and Breakpoints</td>
<td>130</td>
</tr>
<tr>
<td>New Debugger Customization and Settings</td>
<td>131</td>
</tr>
<tr>
<td>Important Terminology</td>
<td>137</td>
</tr>
<tr>
<td>Practice Questions</td>
<td>138</td>
</tr>
<tr>
<td>Practice Question Answers and Explanations</td>
<td>141</td>
</tr>
<tr>
<td>Takeaway</td>
<td>143</td>
</tr>
<tr>
<td>Refresher</td>
<td>144</td>
</tr>
</tbody>
</table>

## 6 ABAP Types and Data Objects

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives of this Portion of the Test</td>
<td>149</td>
</tr>
<tr>
<td>Key Concepts Refresher</td>
<td>149</td>
</tr>
<tr>
<td>ABAP Types and Data Objects</td>
<td>150</td>
</tr>
<tr>
<td>ABAP Data Types</td>
<td>157</td>
</tr>
<tr>
<td>Local Data Types</td>
<td>162</td>
</tr>
<tr>
<td>Global Data Types</td>
<td>165</td>
</tr>
<tr>
<td>Data Object Visibility</td>
<td>167</td>
</tr>
<tr>
<td>Important Terminology</td>
<td>168</td>
</tr>
<tr>
<td>Practice Questions</td>
<td>168</td>
</tr>
<tr>
<td>Practice Question Answers and Explanations</td>
<td>171</td>
</tr>
<tr>
<td>Takeaway</td>
<td>173</td>
</tr>
<tr>
<td>Refresher</td>
<td>173</td>
</tr>
<tr>
<td>Summary</td>
<td>174</td>
</tr>
</tbody>
</table>

## 7 Internal Table Definition and Use

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives of this Portion of the Test</td>
<td>176</td>
</tr>
<tr>
<td>Key Concepts Refresher</td>
<td>177</td>
</tr>
<tr>
<td>Internal Table Definition and Use</td>
<td>177</td>
</tr>
<tr>
<td>Defining ABAP Internal Tables</td>
<td>182</td>
</tr>
<tr>
<td>Using ABAP Internal Tables</td>
<td>191</td>
</tr>
<tr>
<td>Important Terminology</td>
<td>209</td>
</tr>
<tr>
<td>Practice Questions</td>
<td>209</td>
</tr>
<tr>
<td>Practice Question Answers and Explanations</td>
<td>213</td>
</tr>
<tr>
<td>Takeaway</td>
<td>215</td>
</tr>
<tr>
<td>Refresher</td>
<td>215</td>
</tr>
<tr>
<td>Summary</td>
<td>216</td>
</tr>
</tbody>
</table>

## 8 SQL Statements Including Update Strategies

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives of this Portion of the Test</td>
<td>219</td>
</tr>
<tr>
<td>Key Concepts Refresher</td>
<td>219</td>
</tr>
<tr>
<td>Data Modeling</td>
<td>219</td>
</tr>
<tr>
<td>Data Retrieval</td>
<td>222</td>
</tr>
<tr>
<td>Performance of Database Access</td>
<td>225</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Takeaway</td>
<td>580</td>
</tr>
<tr>
<td>Refresher</td>
<td>580</td>
</tr>
<tr>
<td>Tips</td>
<td>581</td>
</tr>
<tr>
<td>Summary</td>
<td>581</td>
</tr>
<tr>
<td><strong>18 Enhancements and Modifications</strong></td>
<td>583</td>
</tr>
<tr>
<td>Objectives of this Portion of the Test</td>
<td>584</td>
</tr>
<tr>
<td>Key Concepts Refresher</td>
<td>585</td>
</tr>
<tr>
<td>Enhancing SAP Applications</td>
<td>585</td>
</tr>
<tr>
<td>Enhancement Techniques</td>
<td>586</td>
</tr>
<tr>
<td>Enhancement Framework</td>
<td>614</td>
</tr>
<tr>
<td>Modification</td>
<td>616</td>
</tr>
<tr>
<td>Practice Questions</td>
<td>619</td>
</tr>
<tr>
<td>Practice Question Answers and Explanations</td>
<td>622</td>
</tr>
<tr>
<td>Takeaway</td>
<td>624</td>
</tr>
<tr>
<td>Refresher</td>
<td>624</td>
</tr>
<tr>
<td>Summary</td>
<td>625</td>
</tr>
<tr>
<td><strong>19 Table Relationships</strong></td>
<td>627</td>
</tr>
<tr>
<td>Objectives of this Portion of the Test</td>
<td>628</td>
</tr>
<tr>
<td>Key Concepts Refresher</td>
<td>629</td>
</tr>
<tr>
<td>Table Relationships</td>
<td>629</td>
</tr>
<tr>
<td>Value Help</td>
<td>637</td>
</tr>
<tr>
<td>Practice Questions</td>
<td>642</td>
</tr>
<tr>
<td>Practice Question Answers and Explanations</td>
<td>646</td>
</tr>
<tr>
<td>Takeaway</td>
<td>647</td>
</tr>
<tr>
<td>Refresher</td>
<td>647</td>
</tr>
<tr>
<td>Tips</td>
<td>648</td>
</tr>
<tr>
<td>Summary</td>
<td>648</td>
</tr>
<tr>
<td>The Authors</td>
<td>649</td>
</tr>
<tr>
<td>Index</td>
<td>651</td>
</tr>
</tbody>
</table>
Index

A

ABAP
basic statement, 273
data object, 150
data type, 148, 157, 173
data type, domain, 350
definition, 79, 261
event blocks, 268
global declaration, 262, 299
global definition, 262
obsolete additions, 262
procedural, 262
processing block, 264
processor, 66
runtime environment, 261, 262, 268, 273, 566, 568
statement, 273, 558
subroutine, 276
syntax rules, 273
text translation, 262
type, 324
type of program, 262, 263
ABAP application server, 50, 61, 63, 64, 67, 222
resource optimization, 68
ABAP class, 459
attribute, 468
class events, 481
constructor, 477
create object, 468
event, 481
global class, 461
local class, 459
method, 471
private, 461
private instantiation, 466
private section, 464
protected, 461
protected instantiation, 466
protected section, 464
public, 461
public section, 464
ABAP class (Cont.)
raise exception, 477
read-only, 469
returning, 475
static method, 480
ABAF Debugger, 90, 114, 116
architecture, 115
classic, 116, 117, 120, 121
flow control, 131
new, 117, 120, 121, 131, 138
process call, 275
start methods, 118
starting, 115, 117, 119
stopping, 119
switching, 120
work areas/desktops, 121
ABAF Dictionary, 67, 95, 313, 369, 545, 628, 638
activate table definition, 220
append structure, 369
check table, 628, 629, 630, 634, 637, 638
create view, 638
customizing include, 369
data element, 291, 505, 512, 545, 628, 630, 638, 640
data element, individual field, 221
data type, 148, 169
domain, 221, 630
fixed value, 630, 632
foreign key, 628, 634, 635, 636
input field, 286
interface parameters, 279
key concepts, 355
overview, 315
program development support, 629
search help, 628, 629, 630, 638
search helps, 338
structure enhancement, 369
table, 355
text table, 636, 639
tool, 317
transparent table, 220
value table, 628, 630, 632, 635
ABAP Dictionary (Cont.)
value table to check table, 634
ABAP Dictionary type, 324
ACCP, 324
CLNT, 324
DATS, 324
FLTP, 324
predefined, 165
ABAP Editor, 94, 127, 132
configure settings, 90
new ABAP Editor, 77, 90, 93
open code, 116
ABAP List Viewer → ALV
ABAP messaging channels (AMC), 56
ABAP Objects, 116, 123, 262, 276, 282, 456
definition, 459
reference, 116
ABAP program
development tasks, 259
ABAP push channels (APC), 57
ABAP Test Cockpit, 83
ABAP Unit Browser, 83
ABAP Workbench, 78, 111, 530
change request creation, 260
introductory statement, 263
main screen, 79
settings, 89
tools, 79, 94
Abstraction, 458
Access mode, 331
Alias, 570
ALV, 494
change layout, 503
custom container, 494, 495, 496, 498, 508, 516
currency, 505
custom container, 497
data sorting, 503
display data table, 500, 509
display variant, 500, 502, 503
event handling, 494, 495, 506, 517
event registration, 506, 507, 518
field catalog, 500, 501, 502, 503, 504, 512, 518
field group, 501
field label, 630
filtering, 501
full screen, 494, 495, 499, 508
ALV (Cont.)
grid control, 494, 496, 499, 500
layout, 501
object model, 495, 512, 514
print output, 501
sorting, 501, 512
unit of measure, 505
ALV Grid
sort data, 500
ALV Grid Control
create instance, 499
data order, 503
ALV Object Model, 507
American Standard Code for Information
Interchange → ASC
Append, 191, 192, 215
Application layer, 61, 226, 231, 261
ASCIL, 360
Assert, 115, 128
Assertions, 130
Assigning, 198, 199
Associate certification, 29
Associate examination
competency area, 32
specifics, 32
At exit-command, 404
At selection-screen, 448
on, 448
on block, 448
on help-request for, 448
on radio button, 448
on value-request, 449
output, 439
ATC Result Browser, 83
Attribute, 282, 470, 491
instance, 496, 508, 559
static, 283, 508, 559, 566
Authorization check, 258, 287
Authorization object, 288
☐
Back-End Editor, 90
BAdI, 585, 602
definition, 604
Filter-depend, 603
implementation, 609
BAdI (Cont.)
menu enhancement, 611
multiple use, 602, 604
new, 615
screen enhancement, 612
searching, 605
BAPI, 60
definition, 330
Begin of block
title, 443
with frame, 443
Binary search, 199, 211
Breakpoint, 115, 123, 128, 130, 137, 266
types, 129
BSP, 55
FIBF, 600
process, 602
process interface, 599
publish and subscribe, 599
search, 602
Buffer, 67
Buffering, 629
Open SQL, 227
type, 336
Business Application Programming Interface
→ BAPI
Business object, 457
Business Process Platform, 51
Business Server Pages → BSP
Business Transaction Event → BSP
Byte and character string processing, 370
Byte type, 360, 364, 366, 370, 372, 373
C
Call method, 478
Cardinality, 635
Central instance, 232
Certification
associate, 29
professional, 30
certification exam
scoring, 28
certification level, 29
advanced, 31
Change and Transport System (CTS), 102
Change request, 258, 260, 299
assign objects, 260
project timeline, 261
task, 260, 299
Character type, 158, 360, 364, 365, 367, 369, 370, 373
Certification
associate, 29
Professional, 30
view controller, 536
window controller, 536
Fixed point arithmetic, 159
Flat structure, 147, 174, 353
type, 162
Flow logic, 290
Foreign key, 334, 634
domain equality, 635
field types, 636
relationship cardinality, 635
table, 634
Fragment view, 360, 364, 367, 369, 373
Free, 207
Friendship => class
Front-End Editor, 90
Full buffering, 336
Function Builder, 95
Function keys, 407
Function module, 278, 316
Gateway, 65
GUI
Graphical Screen Painter, 386
Global class, 459, 461
public instantiation, 466
Graphical Screen Painter, 386
GUI
creating GUI status, 407
creating GUI title, 407
setting GUI status, 410
setting GUI title, 410
status, 380, 402, 427
title, 380
title design, 402
Hashed table, 181, 184, 188, 331
internal, 216
Help view, 348, 354
Hexadecimal type, 159
Hook methods, 536
Identifying error, 114, 115
Index, 316, 337
Index table, 181, 184
Inheritance, 458, 512, 563, 569, 574
friend, 562
semantic difference, 569
tree, 564
Initialization, 438, 447
Inline declaration, 155
Inner join, 345, 346
Input check, 317, 629
in ARAP dialog, 397
Insert, 191
lines, 193
lines in internal table, 192
table, 183
Instance, 57
attribute, 470
component, 467
Integer type, 158
Interface, 568, 574
Internal table, 176, 215, 278
access mode, 183
append, 180
append line to, 191
append summarized lines, 195
append/modify, 198
assigning, 198
at...endat, 202
binary search, 199
clear, 206
collect, 195
column, 178
define, 182
define hashed, 189
define in program, syntax, 186
define sorted hashed, 188
definition, 177
delete adjacent duplicates, 204
delete lines, 204
empty table key, 182
endloop, 200
features, 177
hashed, 278
hashed table, 181
Internal table (Cont.)
header line, 179
index access, 181
insert size, 187
insert lines, 192
key access, 181
line type, 179
loop, 200
modify, 203
modify lines, 202
MOVE-CORRESPONDING statement, 193
non-unique key, 186
occurs, 210
populate, 180
process lines sequentially, 200
read lines, 197
read table, 197
refresh, 206
secondary key, 182
sort, 205
sorted table, 180, 278
standard, 278
standard key, 179
standard table, 180
table key, 179
table row, 178
table type, 183, 330
transporting, 197
uniqueness of key, 181
user-defined key, 179
with header line, 187, 189
without header line, 179
work area, 179
Internet Communication Manager, 54
Internet Transaction Server, 54
IS NOT BOUND condition, 498
Java Connector, 56
Join, 345
Kernel and administration services, 59
Key, 331
Layout
FlowLayout, 541
GridLayout, 542
MatrixLayout, 542
RowLayout, 541
Like, 157
Line type, 331
Literal
constant, 152
numeric, 152
string, 152
Literals, 151
Local class, 459
Lock object, 51, 218, 219, 232, 314, 316, 342, 629
Dequeue, 629
decqueue, 231, 233, 237, 241
Enqueue, 629
denqueue, 231, 233, 237
enqueue work process, 69
function modules, 343
local lock container, 234
lock mode, 345
maintain, 233
Lock parameter
collect, 234
mode, 234, 235
scope, 234, 236
wait, 234, 236
Log data change, 337
LUW, 51, 218, 219, 237
commit, 230, 232, 237, 240, 241, 242
database, 65, 68, 229, 231, 232
definition, 229
implicit commit, 230, 232
lock database, 242
place changes in dialog step, 230
rollback, 229, 239, 240
SAP, 68, 239
update module error, 242
Maintenance, 345
Maintenance view, 316, 348, 354
N
Narrowing cast → Down cast
Nested structure, 174
type, 163
Node cardinality, 537
Non-cumulative lock, 345
Non-Unicode system, 361
Numeric literal, 152
Numeric type, 158
O
Object, 468, 492
context menu, 81
instantiation, 283
Object Navigator, 80, 81
Object-oriented programming, 456
concept, 457
On, 391
Open SQL
buffering, 227
read database records, 223
SQL Parser, 244
statements, 228
Outer join, 346
Overloading, 565
P
Package, 99
globalSwitch, 102
interface, 101
nesting, 100
pakparam, 101
visibility, 100
Package Builder, 99, 100
PAI, 381, 383
Paramater, 150, 157, 266, 276
as checkbox, 433
changing, 265, 267, 276, 277, 279, 282, 557
default, 433
input, 265, 267, 279, 282, 506, 640
memory ID, 433
obligatory, 432, 433
output, 265, 267, 276, 279, 282, 557, 640
Parameter (Cont.)
parameter ID, 330, 433
pass by reference, 265, 266, 268, 276, 299
pass by value, 265, 266, 268, 276, 280, 283, 299
radiobutton group, 433
returning, 265, 282, 285, 557
table, 265, 277, 279
value check, 434
Parameters statement, 432
PBO, 381, 383
Personalization, 585, 586
Physical lock, 231
Plug, 533, 534
POH, 383, 427
Polymorphism, 459, 566
Pooled table, 354
POV, 383, 427
Predefined function for internal table, 208
Presentation layer, 62, 230, 231, 261
user view, 63
Primary index, 337
Private attribute, 469
Procedure
block, 268
ext exception, 265, 278, 279, 282, 568
parameter, 265
Process After Input → PAI
Process Before Output → PBO
Process interface, 601
Process On Help-Request → POH
Process On Value-Request → POV
Processing block, 262, 264, 294, 299
dialog module, 264, 641
event block, 264, 268
procedure, 264
Professional certification, 30
Program attribute, 362
Program context, 231
Program exits, 592
Program type, 293
class, 263, 264, 282
executable, 263, 290
function group, 263, 264, 278, 280, 291
interface, 263, 264
module pool, 263, 291
type group, 263, 264
Projection view, 348, 352, 354
Public attribute, 470
Q
Quality management, 84
R
Radio button, 434
Read lock, 345
Read table, 183, 197, 199, 200
Redefinition, 564
Refactoring Assistant, 563, 571
Reference data type, 353
Reference table, 334
Reference type, 165
REFRESH, 206
Relational operator, 371
Remote Function Call → RFC
Repository Browser, 85, 111
create package, 99
Repository Information System, 77, 86, 111
customize selection screen, 88
Repository object, 61, 70, 75, 85, 260
create/edit, 86
search, 86
Reuse component, 223, 274, 278, 531
Reuse Library, 81
RFC, 60, 230, 262
Root component, 539
S
SAP application server, 50, 231, 232
SAP applications
enhance, 585
SAP Business All-In-One, 51
SAP Business ByDesign, 51
SAP Business One, 51
SAP Business Suite, 51
SAP Certified ABAP Development Associate, 32
SAP Community Network (SCN), 39
SAP examination
Strategy, 42
SAP GUI, 60, 62, 64, 66
SAP HANA, 61
as standard database, 243
primary database, 62
SQLScript, 244
view, 243
SAP Help Portal, 39
SAP List Viewer -> ALV
SAP Logon, 63
SAP NetWeaver, 51
server architecture, 54
SAP NetWeaver Application Server, 50, 361, 405, 530
ARAP, 56, 57, 60, 61, 63, 64, 65, 67
ARAP + Java, 54, 57
Java, 56, 57
SAP Notes Assistant, 618
SAP Products, 51
SAP Notes Assistant, 618
Screen, 63, 278, 289, 417, 496
chain, 391
caller control, 496, 497, 499, 516
control, 496, 499
define layout, 386
design, 382
Dynpro, 65, 258, 285, 289, 496, 498, 517, 529
element attribute, 386
endchain, 391
endloop, 391
exit, 596
field, 393
field help, 398
function code, 389
function type, 404
loop, 391
modal dialog box, 258, 291
modify dynamically, 417
module calls, 389
on-chain-input, 394
on input, 394
on request, 394
process after input, 381
process before output, 381
processing, 391
processor, 65
Screen (Cont.)
show element list, 388
start sequence, 290
structure, 418
type, 385
value help, 398
values, 391
Screen flow logic, 389, 390
at cursor-selection, 395
at exit-command, 395
FI Help with dialog module, 400
keywords, 391
process after input, 390
process before output, 390
table control, 414
Screen Painter, 339, 380
create screen for dialog programming, 382
design screen layout, 384
screen attribute definition, 384
Search help, 314, 317, 338, 353
append, 339, 342
attach to data element, 641
attach to field, 640
attach to table field, 333
attachment, 638, 640
collective, 339, 341
data element, 329
dialog behavior, 638, 639
elementary, 339
interface, 638, 640
parameter, 339, 341
selection method, 638, 642
value list, 638, 639
Secondary index, 226, 338
Secondary table key
hash/sorted key, 182
Select, 223, 226, 227, 228, 642
array fetch, 225
field list, 224
from clause, 223
into clause, 223, 224, 225
join column, 228
join condition, 227
join table, 227
loop, 224, 225
select clause, 223
single, 224
Select (Cont.)
where clause, 223, 225, 226
Selection method, 339
Selection screen, 150, 157, 258, 285, 382, 430, 431, 435
as sub-screen, 445
begin of block, 443
begin of line, 443
call selection-screen, 442
code, 287
design, 431
design before output, 443
design after output, 443
formatting, 442
group, 436
high, 436
memory ID, 439
no-display, 439
no-extension, 438
obligatory, 439
option, 435
parameter, 285, 430, 431, 432
processing, 447
selection text, 432
selection before output, 432
select-options, 285, 430, 431
sign, 435
skip, 444
tab strip control, 446
text element, 432
uline, 444
variant, 442
SELECT-OPTIONS statement, 435
Server, 58
Session, 63
Set handler, 484
Shared lock, 345
Shared memory, 65
Singleton, 560, 562
SLAD -> Debugger:Software Layer Aware
Debugging
Sort key
specify dynamically, 205
Sorted table, 180, 184, 188, 331
internal, 216
Specialization, 563, 569
SQL, 218
access method, 225
ABDC, 222
buffering, 67, 221, 222, 226, 262
CDS, 245
dataset, 221, 224, 225
dataset SQL, 66, 68, 222
Open SQL, 66, 67, 223, 262
optimize, 225
process rows, 228
secondary index, 221, 226
SQL expressions, 245
SQL parser, 244
Standard table, 180, 184, 331
internal, 215
Static attribute, 470
Static component, 467
String, 159
String literal, 152
Structure, 221, 224, 316, 318, 330
Syntactically forbidden, 571
System, 59
T
Table, 167, 288, 316
buffering, 336
DBTABPRT, 337
Field, 333
join, 227
key field, 221
primary key, 221
secondary key, 221, 226
transparent, 220
transparent vs structure type, 222
Table control
attribute, 413
programming, 412
screen flow logic, 414
structure, 416
Table Control Wizard, 413
Table expression, 207
using table key, 207
Table type, 163, 183, 316, 330
key category, 185
line type, 183
Tag Browser, 83
Index

Target → Select
Technical platform, 52, 54
Technical settings, 336
Test Repository Browser, 83
Text field literal, 152
Text symbol, 153
Tool, 122, 128
change how opened, 133
in work areas, 131
object history, 124
service, 124
Variable Fast Display, 132
Transaction
CMOD, 591
FIBF, 600
SE51, 384
SE80, 85
SMOD, 590
UCHECK, 360, 362
Transparent table, 315, 332
attributes, 221
deep structure, 350
definition, 221
Transport Organizer, 102, 105, 112
Transport → Change request
Transporting, 201
Type, 157, 162

U
Understand
Comprehension, 41
Effective memory, 41
Mnemonics, 42
Testing, 41
Unicode, 273, 358
alignment, 359, 364, 366, 368
byte order, 366
character set, 361
check, 358, 360, 362
compliant, 359
enforce check, 359
fragment view, 369
program, 262, 361, 362, 364, 365, 369
syntax, 359, 360, 362
system, 361, 362, 372
tool, 360, 362

Value, 157
Value help, 629, 637
Variable, 154
inline declaration, 155
STATICS keyword, 156
View
container, 541
context, 532, 535, 544
controller, 532, 535, 544
define, 346
layout, 539, 541, 544
type, 345
UI element, 532, 537, 538, 539, 540, 541,
543, 545
Visibility section, 564, 569
private, 560, 562, 564, 565
protected, 562, 565
public, 560, 562, 565, 569

Warning message, 396
Watchpoint, 115, 116, 128, 137
Web Application Builder, 96
Web Dynpro, 55, 63, 528
application, 539
component, 531, 534, 535
component interface, 531, 539
tool, 360, 362
controller, 531, 532, 535, 536
event handling, 534, 538
Explorer, 96
Text Browser, 82
view, 531, 533, 534
window, 531, 534
Widening cast → Up cast
Puneet Asthana is a recognized expert in ABAP development; he has 18 years of development experience, and has worked for SAP for over 12 years. Apart from ABAP development, he is also an expert in IDoc, ALE, EDI, workflow, SAP PI, SAP Fiori, SAP WorkManager, and mobility. He has worked on a number of projects as a technical lead and has worked as a technical architect on a number of SAP Retail implementation projects. He has extensive experience with integrating SAP and non-SAP systems and with leveraging SAP Integration Technologies (including SAP PI) to deliver innovative technical solutions.

David Haslam is a recognized expert in ABAP development, having worked for SAP for over 20 years. David has led or participated in more than seven full lifecycle implementations of SAP, which include several multiple-phase projects and four large development projects. He was awarded the prestigious status of SAP Platinum Consultant more than a decade and a half ago, and he enjoys helping others through workshops and white papers, and sharing his knowledge and experience. In addition, David participates in development reviews, project estimates, and customer project issue escalations.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.

© 2015 by Rheinwerk Publishing, Inc. This reading sample may be distributed free of charge. In no way must the file be altered, or individual pages be removed. The use for any commercial purpose other than promoting the book is strictly prohibited.