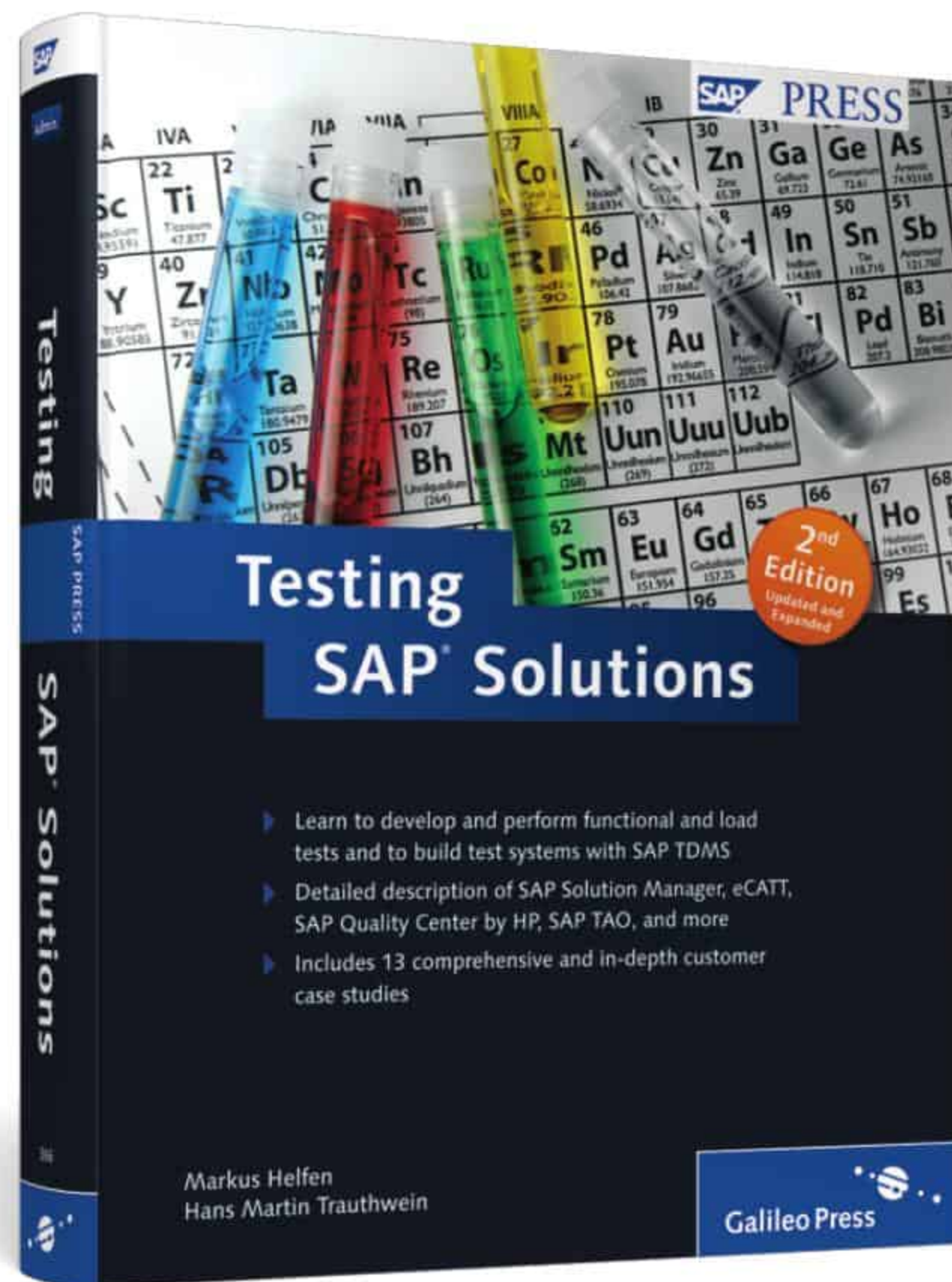


Markus Helfen and Hans Martin Trauthwein

## Testing SAP® Solutions



Galileo Press 

Bonn • Boston

# Contents at a Glance

1	Introduction .....	31
<b>PART I Methodology .....</b>		<b>41</b>
2	Theory of Software Testing .....	43
3	Test Methodology .....	77
<b>PART II Functional Testing .....</b>		<b>123</b>
4	Test Management with SAP Solution Manager .....	125
5	Project-Related Testing with SAP Solution Manager and SAP Quality Center by HP .....	275
6	Supporting Test Automation with SAP TAO .....	337
7	Economic Aspects of Test Automation .....	361
8	Test Automation with eCATT .....	411
9	SAP Test Data Migration Server .....	531
<b>PART III Performance Tests .....</b>		<b>567</b>
10	Project Outline of a Performance Test .....	569
11	SAP LoadRunner by HP .....	595
12	Monitoring a Performance Test .....	627
<b>PART IV Test Center .....</b>		<b>649</b>
13	Test Center .....	651
<b>Appendices.....</b>		<b>691</b>
A	SAP Solution Manager Test Workbench vs. SAP Test Organizer .....	693
B	SAP NetWeaver Knowledge Warehouse—Functionality in SAP Solution Manager .....	697
C	Recommended Reading .....	701
D	The Authors .....	703

# Contents

Preface to the Second Edition .....	15
Foreword to the Revised and Extended Second English Edition ....	19
Foreword to the Second Edition .....	21
Foreword to the First Edition .....	27
<b>1 Introduction .....</b>	<b>31</b>
<b>PART I Methodology .....</b>	<b>41</b>
<b>2 Theory of Software Testing .....</b>	<b>43</b>
2.1 System Changes Necessitate Testing .....	43
2.2 Test Types .....	47
2.3 Test Stages .....	49
2.3.1 Test Stages in the General V-Model .....	49
2.3.2 Test Stages in SAP Projects .....	53
2.4 Black-Box Testing and White-Box Testing .....	56
2.5 Test Case Design for Black-Box Testing .....	57
2.5.1 Equivalence Class Partitioning .....	58
2.5.2 Boundary Value Analysis .....	60
2.5.3 Error Guessing .....	61
2.6 Test Data .....	62
2.7 Basic Procedures for Planning and Execution .....	62
2.7.1 Test Preparation .....	63
2.7.2 Test Case Creation .....	65
2.7.3 Test Planning and Execution .....	65
2.7.4 Test Evaluation .....	66
2.8 Selecting Test Tools .....	66
2.8.1 Types of Test Tools .....	66
2.8.2 Tool Selection and Implementation .....	72
2.9 Conclusion .....	75

<b>3</b>	<b>Test Methodology .....</b>	<b>77</b>
3.1	Roadmaps in SAP Solution Manager .....	78
3.1.1	Project Phases .....	80
3.1.2	Roadmaps .....	85
3.2	Project Preparation .....	90
3.3	Business Blueprint .....	95
3.3.1	Mapping Business Process Structure .....	95
3.3.2	Test Standards .....	98
3.4	Realization .....	101
3.5	Final Preparation .....	109
3.6	Go-Live & Support and Run .....	119
3.7	Conclusion .....	121
<b>PART II</b>	<b>Functional Testing .....</b>	<b>123</b>
<b>4</b>	<b>Test Management with SAP Solution Manager .....</b>	<b>125</b>
4.1	Testing in the Context of SAP Enterprise Support, Run SAP, and SAP Standards for Solution Operations ....	127
4.2	SAP Solution Manager Enterprise Edition .....	131
4.2.1	Application Management Lifecycle .....	131
4.2.2	Work Center .....	137
4.2.3	Adapter and Functional Enhancements .....	141
4.2.4	Projects and Solutions .....	143
4.3	SAP Solution Manager—Basic Settings .....	146
4.4	Project-Related Testing with SAP Solution Manager .....	152
4.4.1	Creating a Project .....	152
4.4.2	Creating the Process Structure .....	161
4.4.3	Integrating Test Cases .....	164
4.4.4	Creating Test Plans and Test Packages .....	167
4.4.5	Extended Functionality for Creating Test Plans and Packages .....	171
4.4.6	Test Execution .....	182
4.4.7	Status Analysis .....	186
4.5	Integration Scenarios .....	193
4.5.1	Test and Service Desk .....	194
4.5.2	Test and Change Request Management .....	198
4.5.3	Test and Quality Gate Management .....	201

4.5.4	Test and Diagnostics .....	203
4.5.5	Test and IT Reporting .....	206
4.6	Solution-Related Testing .....	209
4.7	Summary .....	212
4.8	Customer Report by SOKA-BAU .....	215
4.9	Customer Report by SEWAG .....	229
4.10	Customer Report by BSH Bosch und Siemens Hausgeräte GmbH .....	241
4.11	Update of the Customer Report by BSH Bosch und Siemens Hausgeräte GmbH .....	251
4.12	Customer Report by Reno Fashion & Shoes GmbH .....	260
4.13	Update of the Customer Report by Hamm-Reno- Group GmbH & Co. KG .....	268

**5 Project-Related Testing with SAP Solution Manager  
and SAP Quality Center by HP ..... 275**

5.1	SAP Quality Center by HP .....	276
5.2	SAP Solution Manager Adapter for SAP Quality Center by HP .....	279
5.3	Test Management in the SAP Quality Center by HP .....	282
5.3.1	Creating a Project and Process Structure in SAP Solution Manager .....	282
5.3.2	Managing Requirements .....	285
5.3.3	Creating Test Cases .....	289
5.3.4	Test Planning and Execution .....	290
5.3.5	Creating and Managing Error Messages .....	292
5.3.6	Versioning and Traceability .....	294
5.3.7	Status Analysis .....	302
5.3.8	Transferring the Test Results to SAP Solution Manager .....	308
5.4	Summary .....	309
5.5	Customer Report by Endress+Hauser Group .....	309
5.6	Customer Report by DB Systel GmbH .....	323

<b>6</b>	<b>Supporting Test Automation with SAP TAO .....</b>	<b>337</b>
6.1	Basic Principle: Automation Through Composition .....	338
6.2	Components of and Prerequisites for SAP TAO Technology .....	340
6.3	Using SAP TAO to Create Test Cases .....	343
6.3.1	Using the Process Flow Analyzer to Create Test Components and Test-Case Designs .....	345
6.3.2	Using the Inspection Procedure to Create Test Components .....	348
6.3.3	Composing Test Cases .....	351
6.3.4	Defining Test Data .....	353
6.3.5	Consolidating Test Scripts .....	354
6.3.6	Planning and Executing the Tests .....	356
6.4	Using SAP TAO to Maintain Test Cases .....	356
6.5	Summary .....	360
<b>7</b>	<b>Economic Aspects of Test Automation .....</b>	<b>361</b>
7.1	Cost Model for Software Testing .....	363
7.1.1	Cost of Designing Test Cases .....	363
7.1.2	Cost of Test Tools .....	364
7.1.3	Costs of Implementing Test Automation .....	369
7.1.4	Costs of Maintaining Test Cases .....	370
7.1.5	Costs of Implementing a Test Cycle .....	372
7.1.6	Testing Costs When Creating Component- Based Test Cases .....	372
7.1.7	Individual Test Cost Model .....	377
7.2	Cost Model for Software Errors .....	378
7.2.1	Errors Not Detected During Software Testing ....	379
7.2.2	Errors Detected During Software Testing .....	381
7.3	Overall View .....	382
7.4	Summary .....	383
7.5	Customer Report by INVISTA Resins & Fibers GmbH ....	383
7.6	Update to Customer Report by INVISTA Resins & Fibers GmbH .....	397

<b>8</b>	<b>Test Automation with eCATT .....</b>	<b>411</b>
8.1	Implementation and Prerequisites .....	412
8.1.1	Architecture of the Test Landscape and eCATT Fundamentals .....	412
8.1.2	Structure of the eCATT Test Scripts .....	418
8.1.3	Technical Requirements .....	422
8.1.4	Summary .....	425
8.2	Creating and Running UI-Driven Tests .....	426
8.2.1	Testing Transactions without Controls (TCD) .....	426
8.2.2	Testing Transactions with Controls (SAP GUI) .....	434
8.2.3	Testing Web Dynpro Applications .....	449
8.2.4	Summary .....	457
8.3	Creating Tests Via Direct Program Control .....	457
8.3.1	Testing Global ABAP Object Classes .....	457
8.3.2	Function Modules and BAPIs .....	458
8.3.3	Inline ABAP .....	459
8.3.4	Database Accesses .....	460
8.3.5	Summary .....	461
8.4	Creating Tests for Web Services .....	461
8.5	Integration with External Test Tools .....	465
8.6	Implementing Checks .....	470
8.6.1	Testing Parameters .....	471
8.6.2	Direct Testing of Values .....	473
8.6.3	Message Handling .....	474
8.6.4	Parameterization of Message Handling .....	480
8.6.5	Checking Tables .....	482
8.6.6	Summary .....	483
8.7	Managing Test Data .....	483
8.7.1	Test Configuration .....	483
8.7.2	Variants .....	485
8.7.3	Test Data Container .....	485
8.7.4	Runtime Data Container (RDC) .....	491
8.7.5	Test Data Container Programming Interface (TDC-API) .....	492
8.7.6	Summary .....	493
8.8	Modularizing Test Scripts .....	493
8.9	Additional eCATT Commands .....	497

8.10	Starting, Logging, and Analyzing Test Executions .....	503
8.10.1	Running eCATT Scripts .....	503
8.10.2	Logging eCATT Tests .....	507
8.10.3	Log Archiving .....	510
8.10.4	Automated Performance Analysis .....	511
8.10.5	Error Analysis for eCATT Scripts .....	515
8.10.6	Summary .....	518
8.11	Overview of the eCATT Versions .....	519
8.12	Further Steps .....	521
8.13	Summary: Advantages of the Integration of eCATT in the SAP System .....	521
8.14	Customer Report by Zürcher Kantonalbank .....	524
<b>9</b>	<b>SAP Test Data Migration Server .....</b>	<b>531</b>
9.1	Data Privacy for Test Data Collection .....	533
9.2	Process Types .....	535
9.3	Architecture and System Landscape .....	543
9.4	Data Transfer and Extraction .....	545
9.5	Creation and Refresh of Test Systems .....	547
9.6	Summary .....	550
9.7	Customer Report by Behr GmbH & Co. KG .....	551
9.8	Customer Report by Infineon Technologies AG .....	559
<b>PART III</b>	<b>Performance Tests .....</b>	<b>567</b>
<b>10</b>	<b>Project Outline of a Performance Test .....</b>	<b>569</b>
10.1	Load Test—Stress Test—Volume Test .....	571
10.2	Roles in the Performance Test Project .....	572
10.3	Phase Model of a Performance Test .....	574
10.4	Planning .....	575
10.4.1	Process Analysis .....	576
10.4.2	Data Analysis .....	581
10.4.3	Selecting the Load Test Tool .....	583
10.4.4	The Load Profile .....	583
10.5	Performing the Load Test .....	584
10.5.1	Data and System Preparation .....	585
10.5.2	Single-User Test .....	586

10.5.3	Multi-User Tests .....	587
10.5.4	Result Analysis .....	588
10.5.5	Optimization .....	588
10.6	Performing the Stress Test .....	589
10.7	Completing .....	590
10.7.1	Executive Summary .....	590
10.7.2	Action Plan .....	591
10.7.3	Description of the Test Structure .....	591
10.7.4	Description of the Test Goals .....	592
10.7.5	Documentation of the Test Execution .....	592
10.7.6	Lessons Learned .....	592
10.8	Summary .....	593

## 11 SAP LoadRunner by HP ..... 595

11.1	LoadRunner Virtual User Generator .....	597
11.2	LoadRunner Controller and LoadRunner Agent .....	600
11.3	LoadRunner Analysis .....	602
11.4	SAP Performance Center by HP .....	603
11.5	Summary .....	607
11.6	Customer Report by HeidelbergCement AG .....	608
11.7	Customer Report by Sanofi-Aventis .....	616

## 12 Monitoring a Performance Test ..... 627

12.1	Sample Process In Case Performance Problems Occur ...	628
12.2	Localization of Performance Bottlenecks .....	630
12.3	Transactions for Technical Monitoring .....	634
12.3.1	AL08: List of all Users Logged On .....	634
12.3.2	SM04: User List .....	635
12.3.3	SM12: Lock Entry List .....	636
12.3.4	SM21: System Log .....	637
12.3.5	SM66: Global Work Process Overview .....	638
12.3.6	STAD: Transaction Analysis .....	639
12.3.7	ST02: Overview of SAP Buffers .....	640
12.3.8	ST04: Database Overview .....	642
12.3.9	ST05: SQL Trace Analysis .....	643
12.3.10	ST06: OS Monitor .....	645

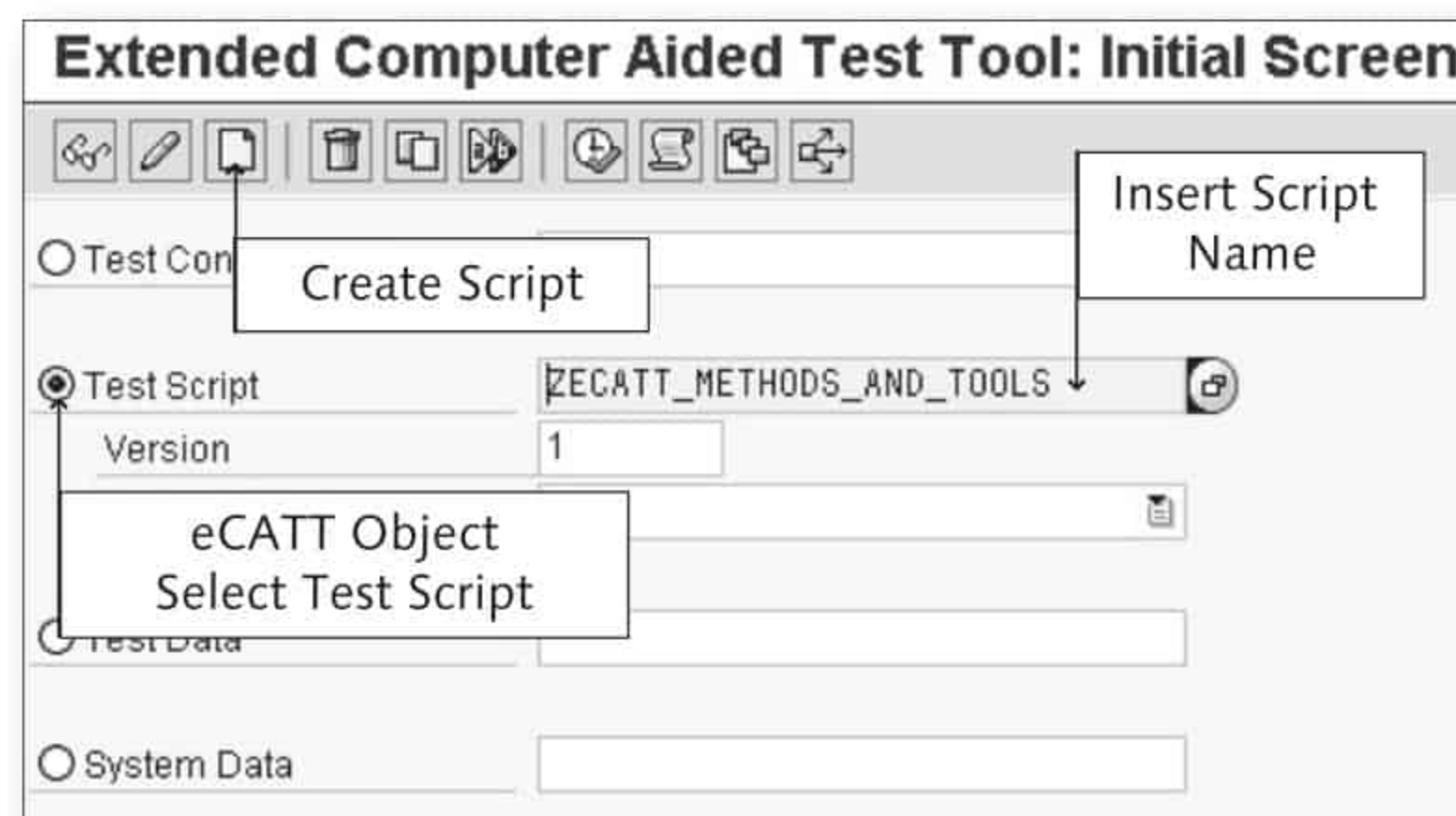
12.3.11	ST22: ABAP Runtime Error .....	647
12.4	Summary .....	648
<b>PART IV</b>	<b>Test Center .....</b>	<b>649</b>
<b>13</b>	<b>Test Center .....</b>	<b>651</b>
13.1	Setting Up a Test Center .....	653
13.2	Test Center Services .....	656
13.3	Summary .....	660
13.4	Customer Report by Deutsche Telekom AG .....	660
<b>Appendices</b>		
A	SAP Solution Manager Test Workbench vs. SAP Test Organizer .....	693
B	SAP NetWeaver Knowledge Warehouse—Functionality in SAP Solution Manager .....	697
C	Recommended Reading .....	701
D	The Authors .....	703
	Index .....	705

## 8.2 Creating and Running UI-Driven Tests

Using eCATT, you can test the following UI-based applications: transactions with and without controls as well as Web Dynpro applications.

### 8.2.1 Testing Transactions without Controls (TCD)

To create an eCATT script, first start eCATT (Transaction SECATT). In the initial screen (see Figure 8.11), select the TEST SCRIPT item, and enter a name. Note that the name must reside in the customer-specific namespace.



**Figure 8.11** eCATT Initial Screen

**Script editor** The script editor is the central tool for creating eCATT scripts. Its interface is divided into three areas (see Figure 8.12). The lower area is occupied by the command editor. There you edit the script logic that is composed of the different commands.

The upper area provides an input possibility for creating the parameters of a script. If you want to edit a command interface, the structure editor is opened to the right. It can be enabled by double-clicking on the name of a command interface in the command editor.

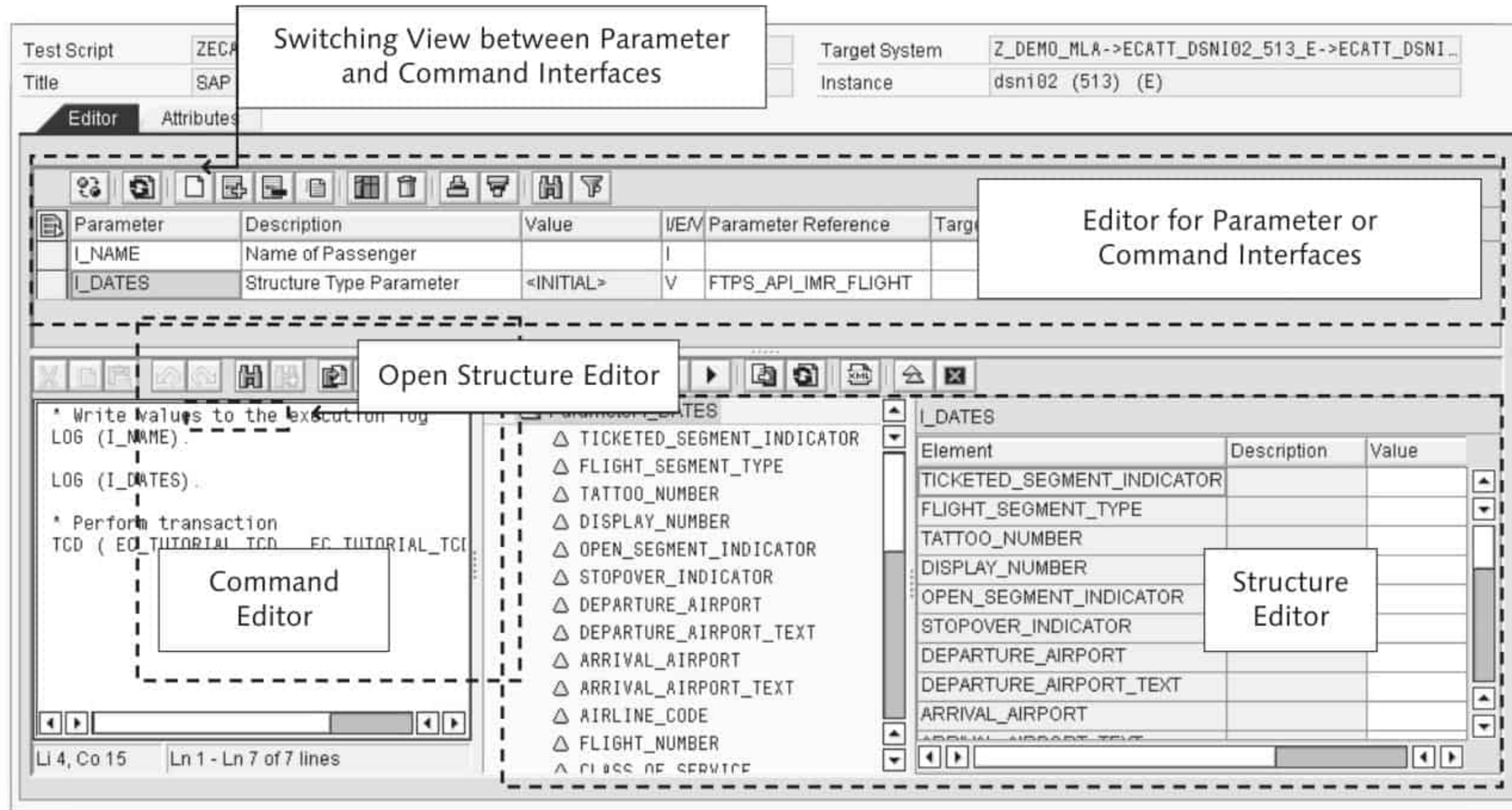


Figure 8.12 Script Editor

The first step in creating the script logic is recording a transaction. Use the RECORDER function of the script editor for this purpose.

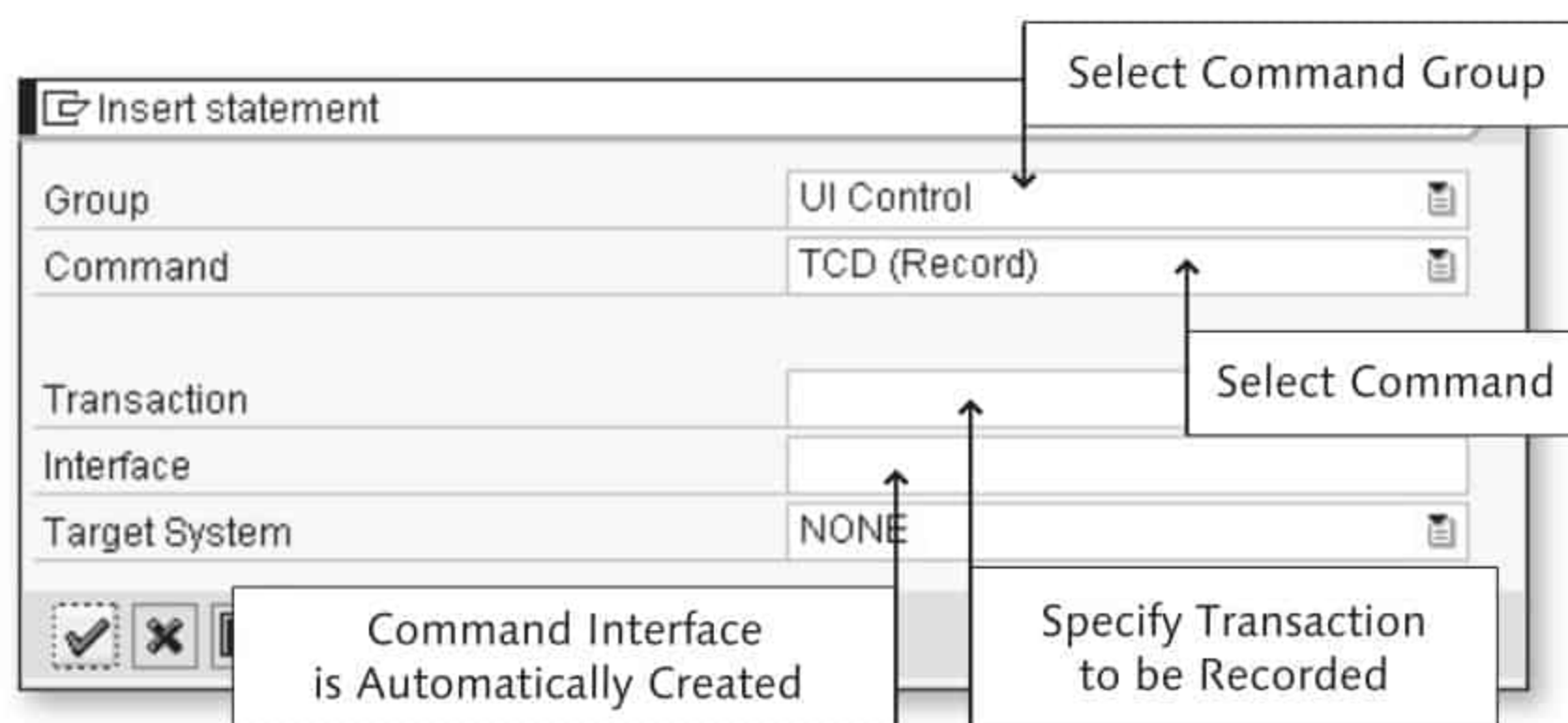


Figure 8.13 "Start recording" Dialog

To start recording, in the script editor select the PATTERN button. A dialog is displayed where you can specify the necessary settings for the recording (see Figure 8.13). The available commands are sorted by functions and divided into groups. First select the desired UI ADDRESSING function group and then the test driver. In the following description, we assume that you use the TCD driver for recording transactions without

Start recording

controls. As soon as you select the driver, you can select the transaction to be tested. An appropriate interface is then automatically created by the script editor.

eCATT now opens the transaction. Perform the transaction as usual and then close the transaction window **F12**. eCATT asks you if you want to accept the data. If you confirm with OK, a new command line with the TCD command is displayed in the editor. The entire recorded transaction is now included in the corresponding command interface.

**"TCD" Command**

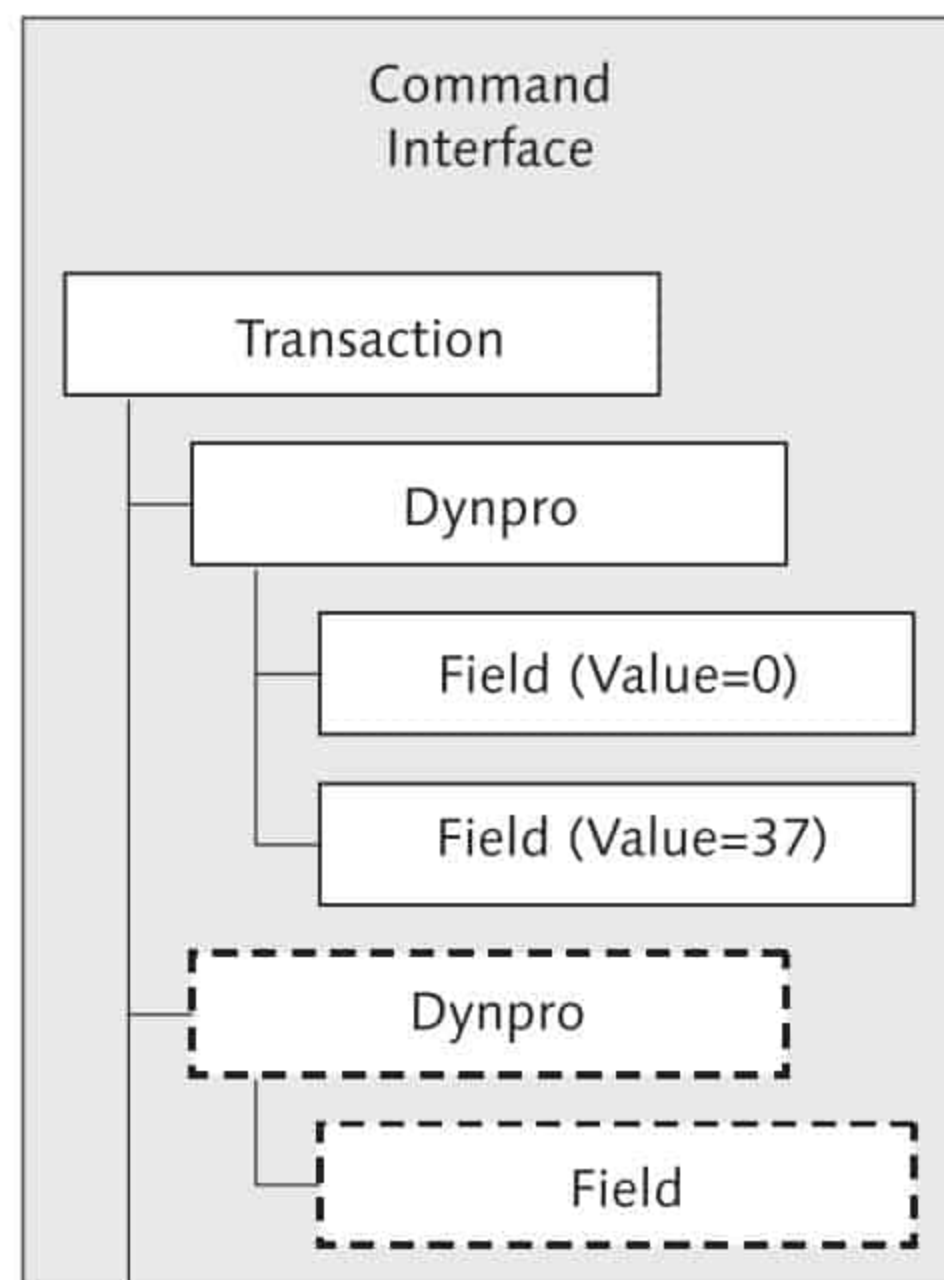
The TCD command is designed for addressing the TCD driver. It has the following format:

```
TCD ( <transaction code>, <command interface>,
      [<target system>] ).
```

A TCD command must always be created via a recording.

**Command interface**

When a TCD driver is used, the command interface records all dynpros with all fields that have been displayed in the transaction (see Figure 8.14). The data you entered and the default values for fields you did not fill in are recorded.

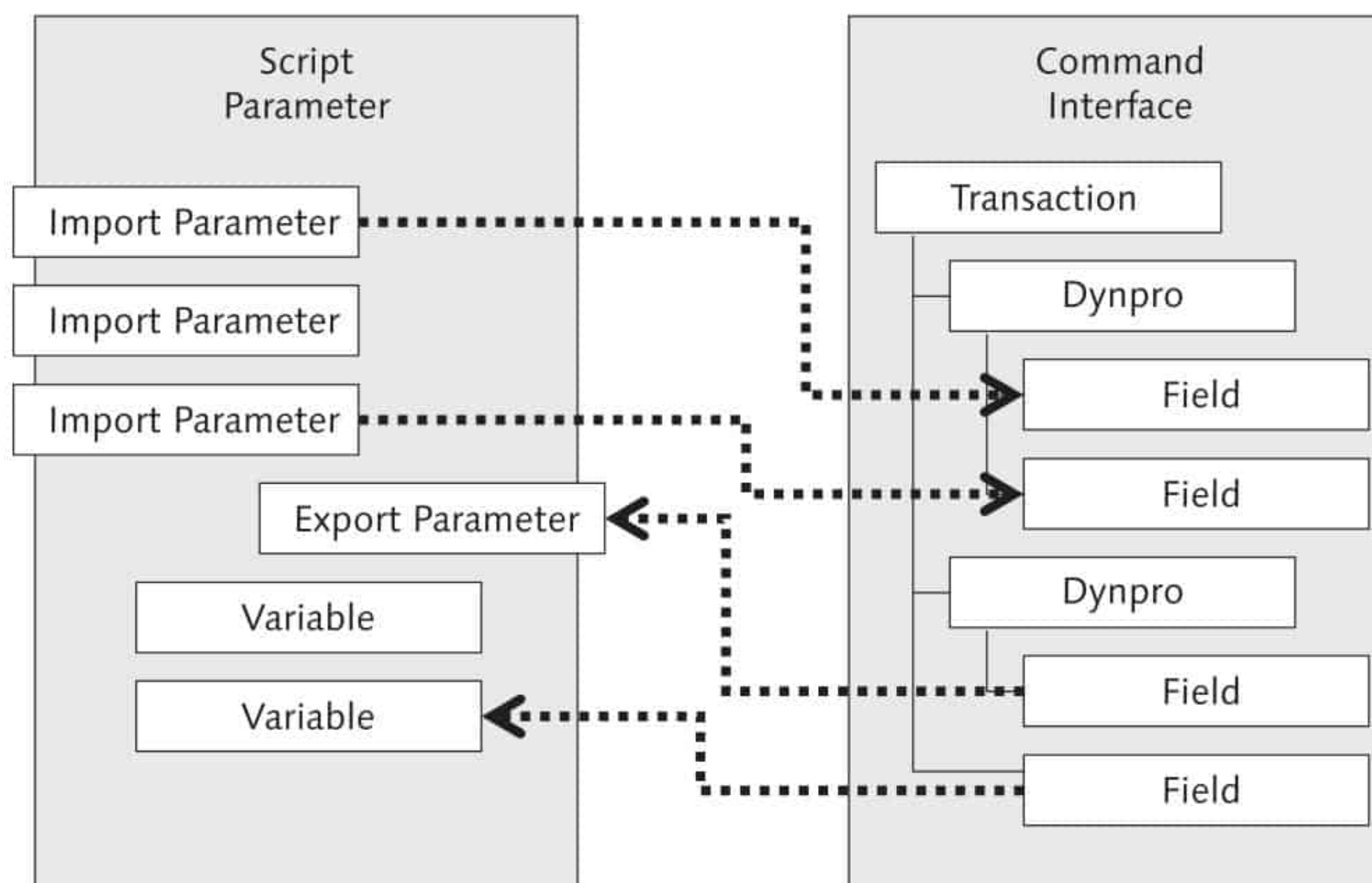


**Figure 8.14** Structure of the Command Interface

The values you entered are displayed as fixed values in the command interface. You can use the search function later to find these fixed values during parameterization. No fixed values are recorded for fields you kept at their preset default values. During parameterization, you therefore you might have difficulties with identifying these fields. When recording, ensure that you enter input values for all fields that are relevant to the test case.

If the test case is to be executed with values other than those used during the recording, the corresponding fixed values need to be replaced with parameters (see Figure 8.15). During execution of the test script, the parameter value is then inserted in the appropriate place. This procedure links a part of the command interface of the TCD command to parameters that are visible from the outside. In other words, fields of the command interface can be linked to parameters to return results of the TCD command. Use import parameters to transfer values to input fields. Use export parameters or local variables to process results from the command interface.

Script  
parameterization



**Figure 8.15** Replacing Fixed Values Entered with Parameters

The automatic creation of parameters is supported as of SAP Web AS Release 6.40. You can also manually create the parameters required. For this purpose, complete a row in the parameter editor to add a parameter (see Figure 8.16).

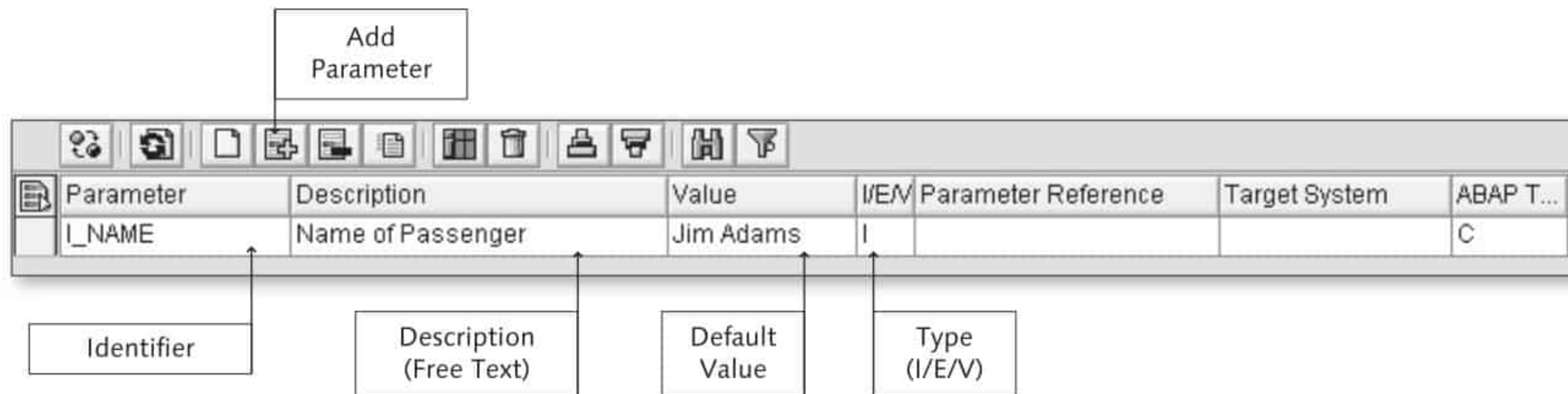


Figure 8.16 Adding a Parameter

Recommended procedure for naming conventions

Every parameter has a unique name for identification. To improve the readability of the scripts, we recommend keeping a consistent notation. A well-established procedure is to have all import parameters start with "I\_," all export parameters with "E\_," and all local variables with "V\_."

Parameter types

Once the names of the parameters have been defined, you specify their visibility. In the parameter editor, specify "I" for import or "E" for export parameters. If you need local variables you can create them now as well. Set the type to "V."

To parameterize a recorded TCD command, open the related command interface in the structure editor (double-click). Look for the entered fixed values. When you have found the corresponding field, replace the value with the name of the parameter (see Figure 8.17).

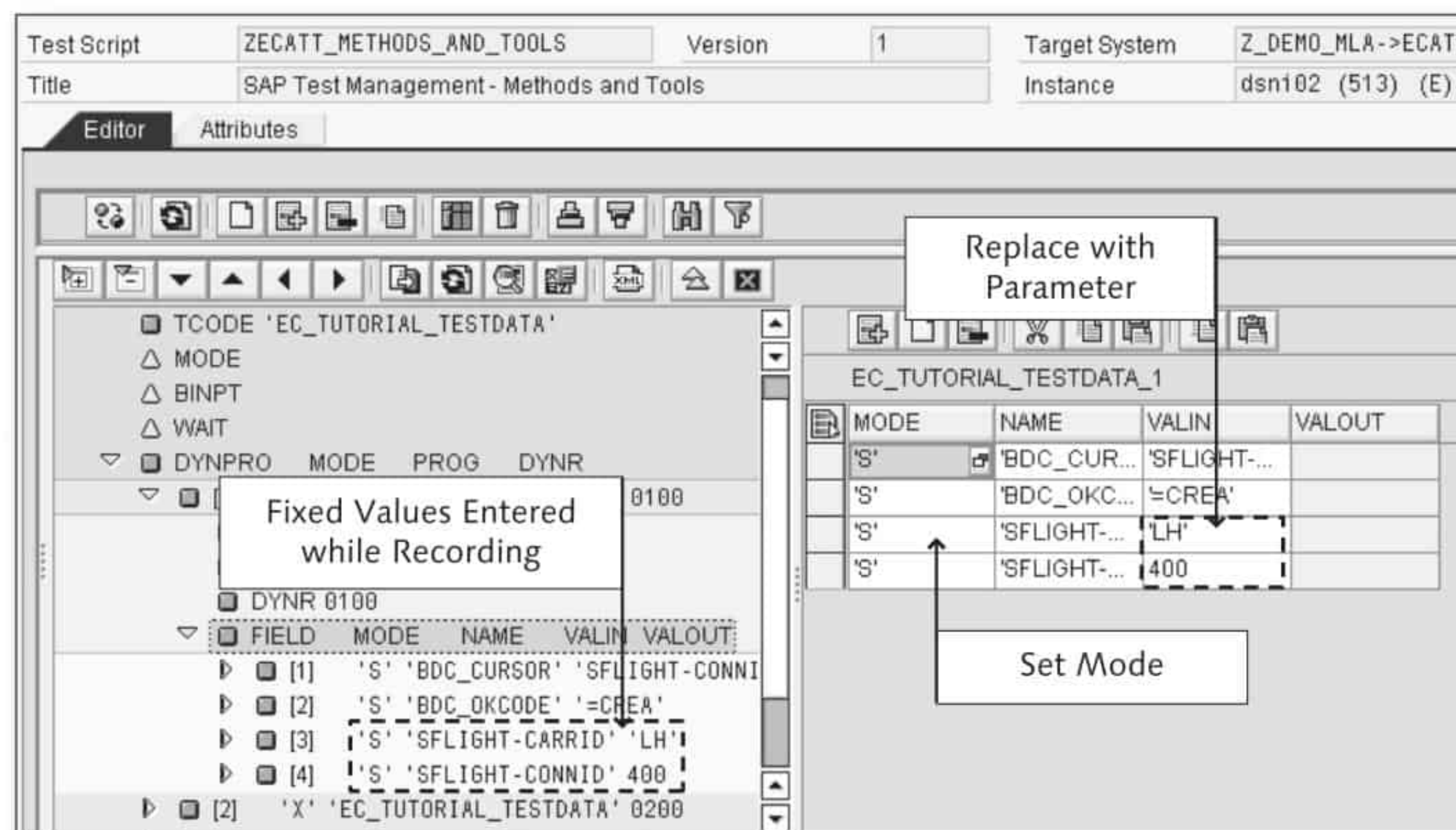


Figure 8.17 Script Editor During Parameterization

eCATT supports different kinds of parameterization. We distinguish between three actions. Two passive modes are also available. In the **MODE** column, you can set the appropriate mode: Parameter mode

**1. S (set value)**

This mode is used for transferring import parameters or local variables to transaction fields. If you select this mode, the parameter value is transferred to the corresponding dynpro field during script execution, just like user input.

**2. G (get value)**

This mode is designed for exporting values from the command interface. A value calculated by the transaction is transferred to an export parameter or a local variable after the `TCD` command has been executed.

**3. C (check value)**

The third mode permits a simple verification of the results. The value returned by the `TCD` command is compared to the specified parameter. If the comparison fails, the test case is regarded as faulty. Note that the possibilities of this test are rather limited. For more complex conditions, you should use the "G" mode to first read the field value and then check it later using the `CHEVAR` command. More information on this topic can be found in Section 8.8, *Modularizing Test Scripts*.

**4. I/O (passive)**

The "I" mode refers to an input field that is not changed by eCATT. Via user parameters, the application can predefine the field with values, though.

The "O" refers to an output field that is neither read nor checked by the eCATT script.

The dynpro simulator is an alternative to the value maintenance in the field lists. To open the simulator, select the desired dynpro in the list of recorded dynpros and then the `SIMULATE DYNPRO` function. Dynpro simulator

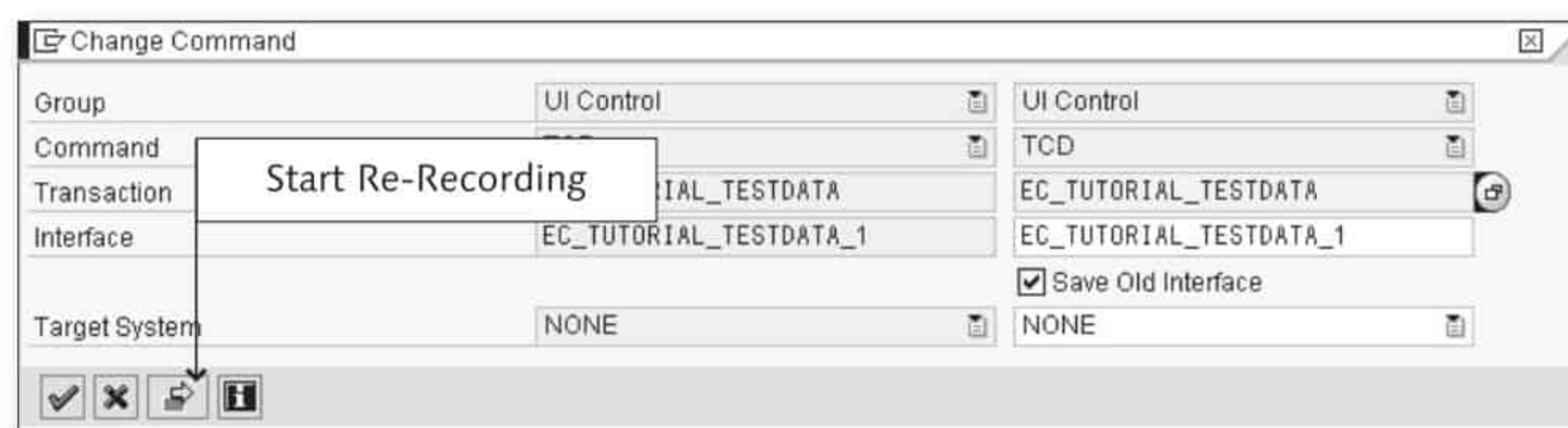
The dynpro of the application is simulated on the screen and provided with eCATT-specific functions; for instance, `INSERT PARAMETER`. You can also navigate between the recorded dynpros so that you can edit the entire recording.

The relevant values, parameters, and mode entries are copied to the field list when you exit the dynpro simulator.

**Re-recording** The TCD driver enables you to re-record driver calls that have already been parameterized. This option is used if the test script encounters an error after the underlying transaction has been changed; for example, after installing a support package. Such an error can have two causes. The obvious cause is that the change has caused an error in the application logic. In this case, the error must be corrected in the application. Although the joy at a successful test—testing is always successful when errors are found—is usually muted by the awareness of difficulties for the operation or the project, valuable insights can still be gained.

The second possible error source consists of an incompatible change in the structure of the recorded transaction for the existing test script. Because the structure of a transaction changes more frequently than its fields (a field can be assigned to a different dynpro or another group but is hardly ever renamed or deleted), you can re-record an existing TCD driver call while maintaining the parameterization.

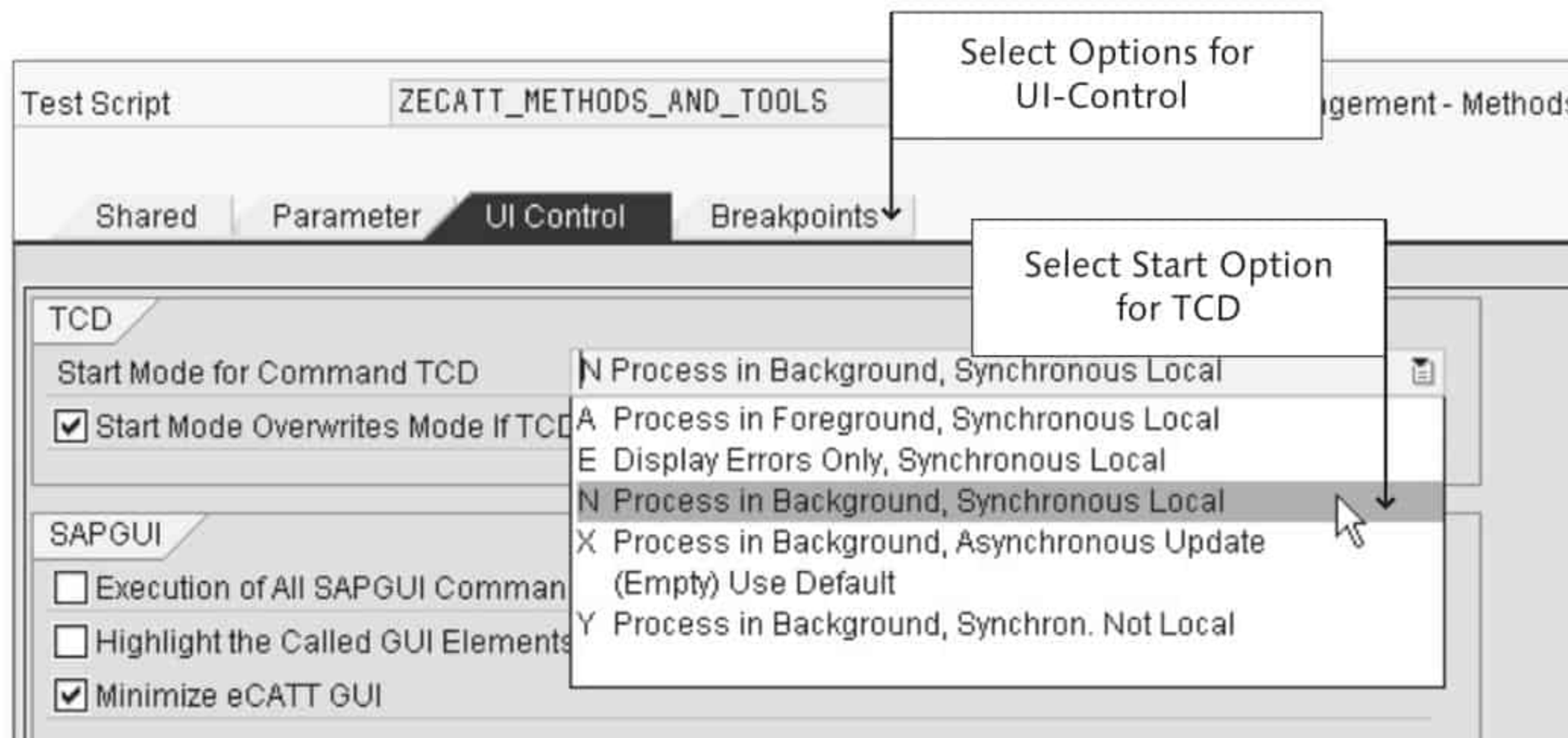
Double-click on the command to open the CHANGE COMMAND dialog (see Figure 8.18). Trigger the re-recording of the transaction, and record the transaction as usual. Fields that have already been parameterized are taken over from the old command interface according to their field names. In most cases, the transaction is already fully parameterized immediately after the recording. Only newly added fields must be completed during the parameterization. This functionality is exclusively available in the TCD driver and makes the recordings performed using this driver extremely easy to maintain.



**Figure 8.18** “Change Command” Dialog Window

For processing a script, you are provided with various start options for selection. The options relevant to the TCD driver can be found under the UI CONTROL tab in the TCD area (see Figure 8.19).

Start options for the TCD driver



**Figure 8.19** Start Options for Processing a TCD Recording

Start options are selected before a script is processed. You have the following options:

► **Process in Foreground, Synchronous Local**

The script is processed in the foreground; that is, with a user interface. All actions of the script can be observed on screen. The database is updated synchronously. This option ensures that all input values have been updated in the database before the next step in the script is executed. For tests using eCATT, you should always select one of the options with synchronous update for the reason mentioned above.

► **Display Errors Only, Synchronous Local**

This option processes the script in the background; that is, without a user interface. If an error occurs during the execution, the corresponding position is displayed in the user interface. The error can now be manually corrected. The script is then continued in the background until another error occurs or the test case has been completed. The database is updated synchronously.

► **Process in Background, Synchronous Local**

This option processes the script completely in the background. There is no output to the screen. Errors are not reported immediately, but they can be viewed in the automatically created log of the eCATT run

after the script has been executed. The database is updated synchronously.

► **Process in Background, Asynchronous Update**

This option processes the script completely in the background. The database is updated asynchronously. This means that the control flow might return to the script before the updater has changed all values in the database. For a subsequent step of the script, therefore, there is no guarantee that a change from a previous step has been implemented in the database.

The option can be used when eCATT is implemented to provide mass data in the system. This can happen either during a data migration or when test data is created for preparing manual tests or trainings. By disabling a synchronous update, you can often considerably increase the speed.

► **Process in Background, Synchronous Not Local**

The script is processed in the background, and the update takes place synchronously but via a different work process than the transaction. This option is obsolete and is only supported for downward compatibility.

► **Use Default**

The option stored in the command interface of the command is used.

Regarding the message handling within the TCD driver, please refer to Section 8.6.3, *Message Handling*.

### 8.2.2 Testing Transactions with Controls (SAP GUI)

Starting with SAP basis Release 4.5B, transactions are able to present more complex and user-friendly graphical user interfaces via SAP GUI controls. One characteristic of this way of programming is the requirement that a part of the application logic is run on the front end.

Because the TCD driver immediately interferes with the application server, application parts running on the front end are outside its reach. Therefore, eCATT provides its own test driver for recording transactions with controls. The SAPGUI driver works with the SAP GUI for Windows and interferes with the SAP system at a different level than does the

TCD driver. An important difference is that the SAPGUI driver does not connect to the application server but to the front end.

**"SAPGUI" Command**

The SAPGUI command is designed for addressing the SAPGUI driver. It has the following format:

```
SAPGUI ( <command interface>, [<target system>] ).
```

In contrast to the TCD command, the SAPGUI command only allows the transfer of values to the application. There are separate commands for reading and testing values.

While the TCD driver records the result of the input in the record fields, the SAPGUI driver registers events. These events refer to changes of the state of screen elements; for instance, the selection of a value in a listbox, the input of text in a text field, or the expansion of a branch in a tree control. In particular, this means that the SAPGUI driver only records information about the fields that the user actually changes during recording. Another difference is that the SAPGUI driver uses different commands for reading and querying screen elements while the TCD driver serves all actions of the TCD command (see Table 8.2).

Events

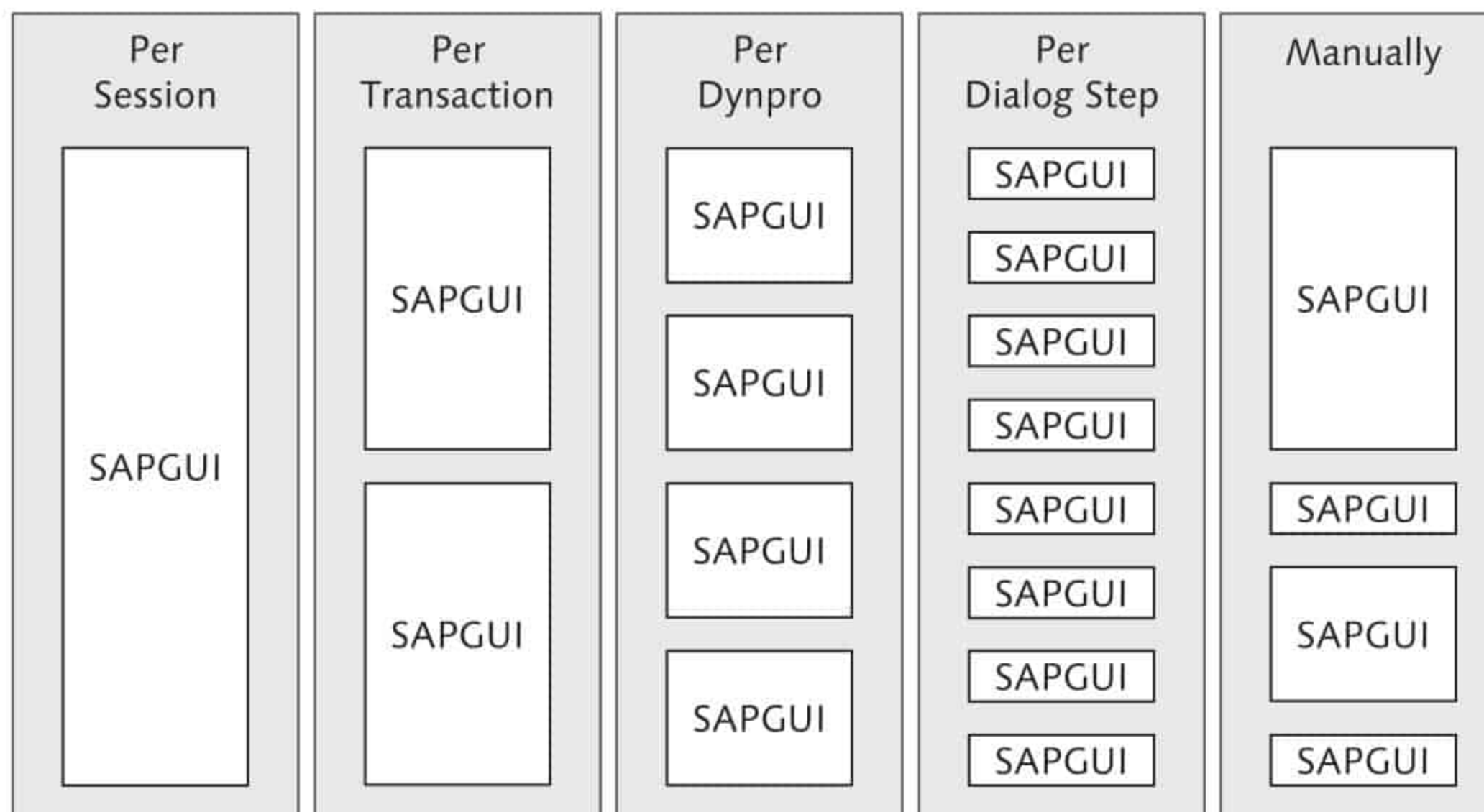
Function	TCD Driver	SAPGUI Driver
Parameterizing	TCD (mode S)	SAPGUI
Reading	TCD (mode G)	GETGUI (as of SAP Web AS 6.40)
Check	TCD (mode C)	CHEGUI (as of SAP Web AS 6.40)
Passive (output)	TCD (mode O)	
Passive (input)	TCD (mode I)	

**Table 8.2** Commands for Parameterizing, Reading, and Checking Field Values

Because this type of recording generates many events for complex transactions you must specify the appropriate level of granularity prior to recording. This level of granularity determines the number of individual commands into which the script is subdivided. The higher the level of granularity for the recording, the better the overview, reusability, and maintainability of the script. It is also easier to insert descriptive comments between the individual steps. Figure 8.20 shows the different

Recording  
granularity

levels of granularity, with the level of granularity increasing the farther you go to the right.



**Figure 8.20** Granularity Levels of the SAP GUI Recording

The various granularity levels of the SAP GUI recording have the following meanings:

**1. Per dialog step**

For every GUI event (every roundtrip between front end and back end), a separate row containing one `SAPGUI` command is inserted into the script.

**2. Per dynpro**

Events referring to the same dynpro are joined to form one command.

**3. Per transaction**

Events referring to the same transaction are joined, even if they span several dynpros.

**4. Per session**

All events between starting and ending an SAP GUI session are joined to form one command.

**5. Manual**

The creation of an `SAPGUI` command is explicitly triggered by the user, who activates an appropriate button during the recording. If you use

this option, you should add a comment to the creation of every command to maintain a better overview.

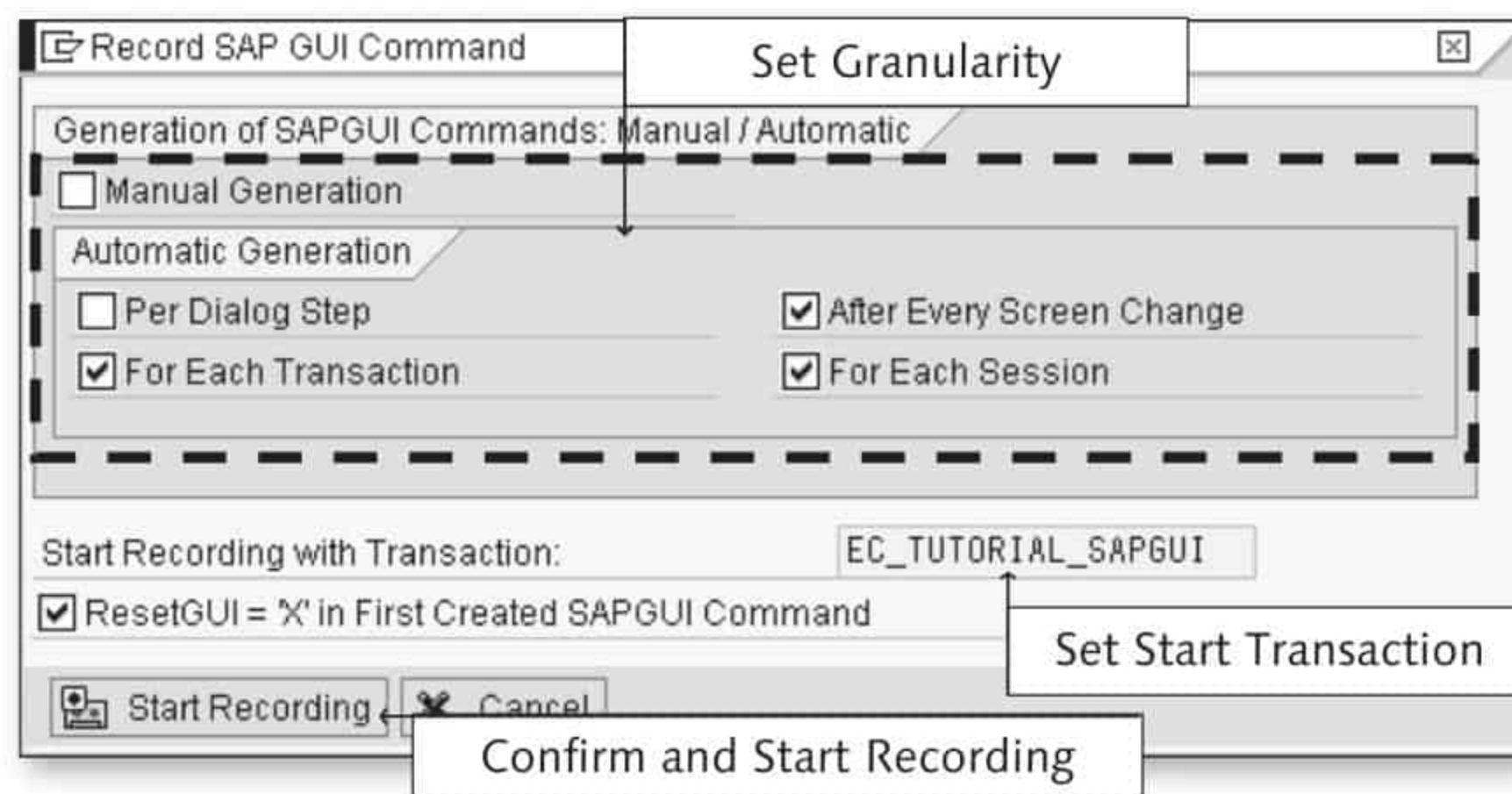
In general, the granularity level per dynpro as proven as the most appropriate solution for most requirements.

If you want to change the granularity of your script subsequently, you can join steps using the `Join` command or split them using `Split`. If it should become necessary to change the test script, for example, to adapt it to a new support package level, you can use the `SAPGUI (Attach)` command to re-record the affected part of the script, rather than the entire sequence. These commands are described in more detail in the course of this chapter.

To prepare recording of one or several `SAPGUI` commands, go to Transaction `SECATT`, create a new test script, and then select the `PATTERN` button. In the `INSERT STATEMENT` dialog window, select the `UI CONTROL` group and then the `SAPGUI (RECORD)` command. If you leave the `[GENERATED]` presetting in the `INTERFACE` field unchanged, `eCATT` generates a name for the command interface based on the selected granularity. For example, if you selected the granularity level per transaction, the generated name includes the transaction code followed by a number. If the generated name does not match your naming concept, you can enter the desired name in the `INTERFACE` field.

Prepare recording

When you confirm the entries, the system shows a dialog window in which you set the granularity of the recording (see Figure 8.21).

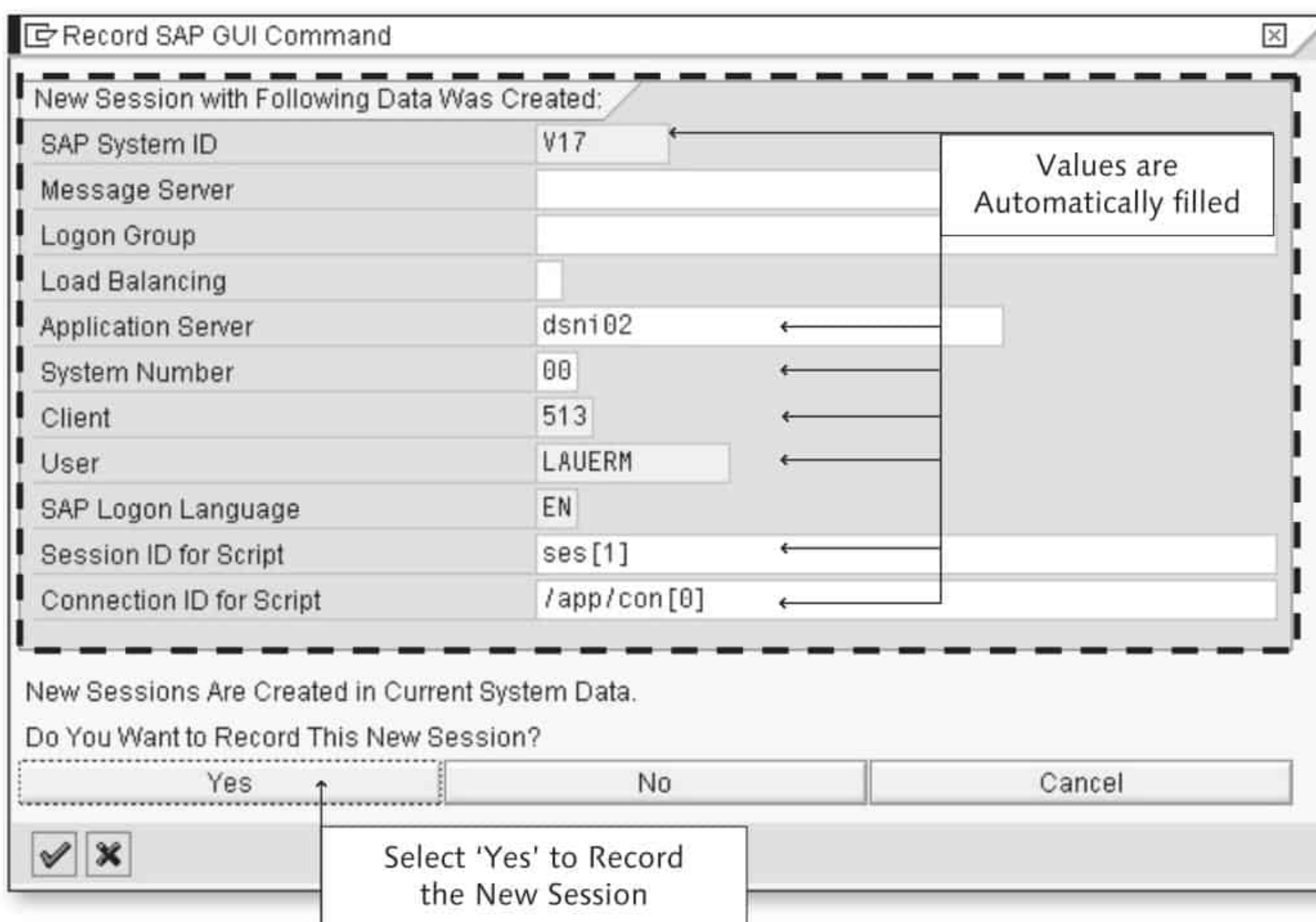


**Figure 8.21** Setting the Granularity Levels of the SAP GUI Recording

With SAP Web AS Release 6.40, you must specify a start transaction. Then, the recording starts directly with the first screen of the transaction selected. In SAP Web AS Release 6.20, by contrast, you must start the transaction by entering the appropriate transaction code during recording.

**ResetGUI** If you've selected the RESETGUI option, a `ResetGUI = 'X'` flag is generated in the first generated command interface of the recording. This means that the command has a similar effect during processing as an `"/n"` prefixed to the transaction code. To explain further: A correct processing of the SAP GUI commands is only ensured if every subsequent command starts precisely at the point where the previous command has stopped. This condition is met with everything that you're recording in one single step. But if you record a test script in several steps, this might not be the case. For this reason, you can select the `ResetGUI` flag in the first `SAPGUI` command of every transaction when you process an SAP GUI flow with several individual transaction changes.

After you've made your selection, click **START RECORDING** to confirm. The system displays the **RECORD SAP GUI COMMAND** window (see Figure 8.22).



**Figure 8.22** Dialog Window for Starting the Recording in a New Session

eCATT generates a new session when a recording is started. It automatically recognizes the values of this newly generated session and displays them in the dialog window. In most cases, it is therefore sufficient to confirm the preset values with YES. SAP Note 1307732 provides further information on exceptions.

Based on the *connection ID* and the *session ID*, the GUI sessions are uniquely identified during the recording of SAPGUI commands. By default, these different IDs are also used for processing to address different sessions. *Session* always refers to a mode and *connection* to a target system. If all SAPGUI commands are supposed to be run in the same session, all SAPGUI command interfaces must have the same session ID and connection ID. Different combinations, in turn, stand for different sessions. This is significant if you work in different modes in parallel for a recording; for instance, if you record an application in one mode and open another mode in which information is to be checked and recorded at the same time. During processing, eCATT determines which command must be executed in which session, based on the different session IDs.

You have two ways to override these ID values. You can change values for multiple commands in the script editor simultaneously by selecting EDIT • PARAMETER/COMMANDO INTERFACES • REPLACE IDs IN SAPGUI INTERFACES. Alternatively, you can select the EXECUTION OF ALL SAPGUI COMMANDS IN A SINGLE SESSION PER DESTINATION option in the start options for all commands of a destination (that is, a target system). With this option, the different IDs are ignored and all SAP GUI commands are executed in the same session.

If you confirm the values in the RECORD SAP GUI COMMAND window, the system displays the RECORDING RUNNING window (see Figure 8.23). In this window, you can implement different actions; for instance, inserting commands manually and stopping recording again. The functions of this window are described later on.

In parallel to the RECORDING RUNNING window, the start transaction for recording is opened in another mode, the recording mode (not in SAP Web AS 6.20, where you must still start the transaction manually). Execute the transaction as usual. If you selected the manual mode, you must change back to the recording window (see Figure 8.23) and trigger the creation of SAPGUI commands using the appropriate button. After the

Session ID and  
connection ID

Run recording

transaction has finished, go back the RECORDING RUNNING window and terminate the recording.

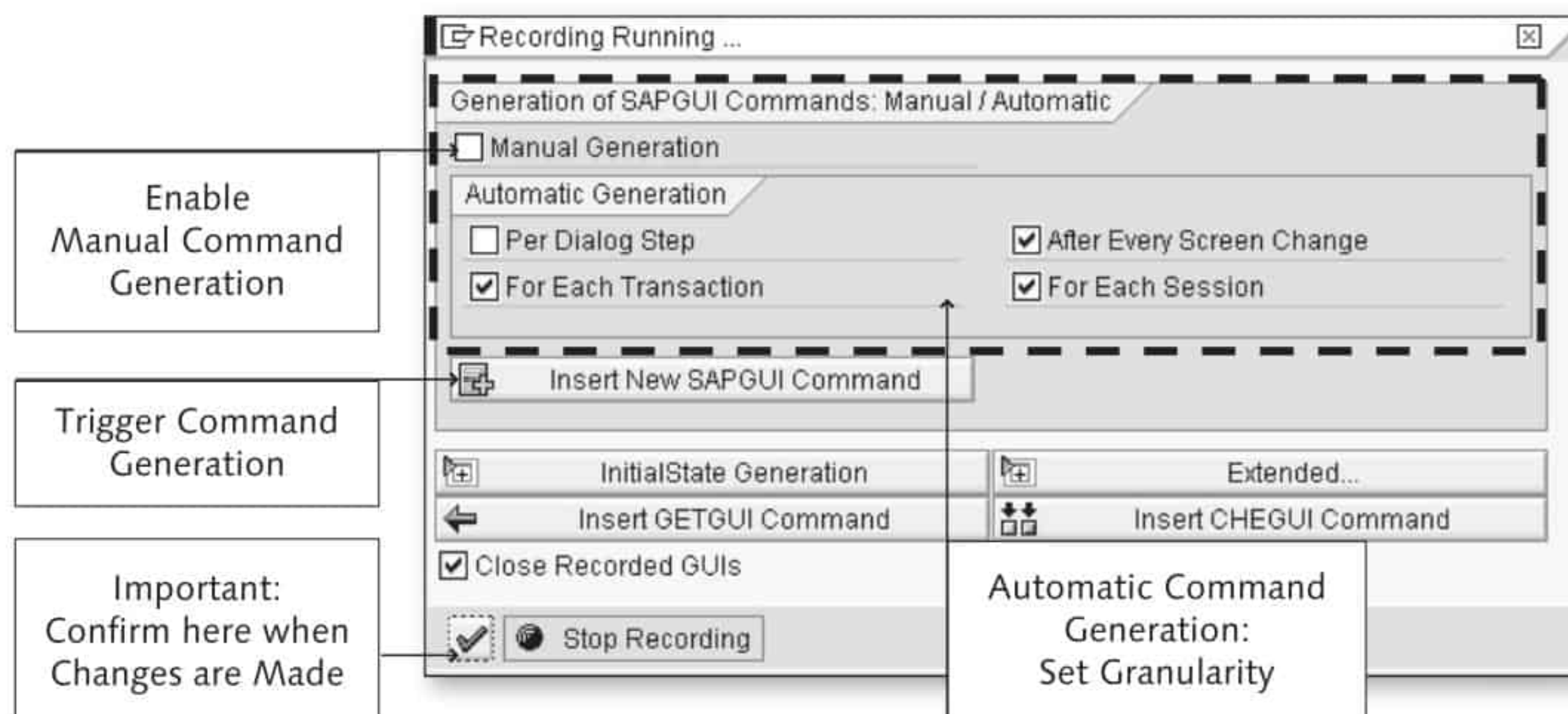


Figure 8.23 Dialog Window for Controlling the Recording

Using "CHEGUI" and "GETGUI"

In your scripts, you will often have to copy values from the screens or check contents. In eCATT, this is implemented with the commands, GETGUI and CHEGUI, which are available as of SAP Web AS 6.40. GETGUI reads the values of a GUI element, for instance, for a text field. CHEGUI reads and checks.

At the point where you want to determine a value, go back from the application mode to the recording dialog and select INSERT GETGUI COMMAND. If you want to additionally check the field content, select INSERT CHEGUI COMMAND (see Figure 8.24).

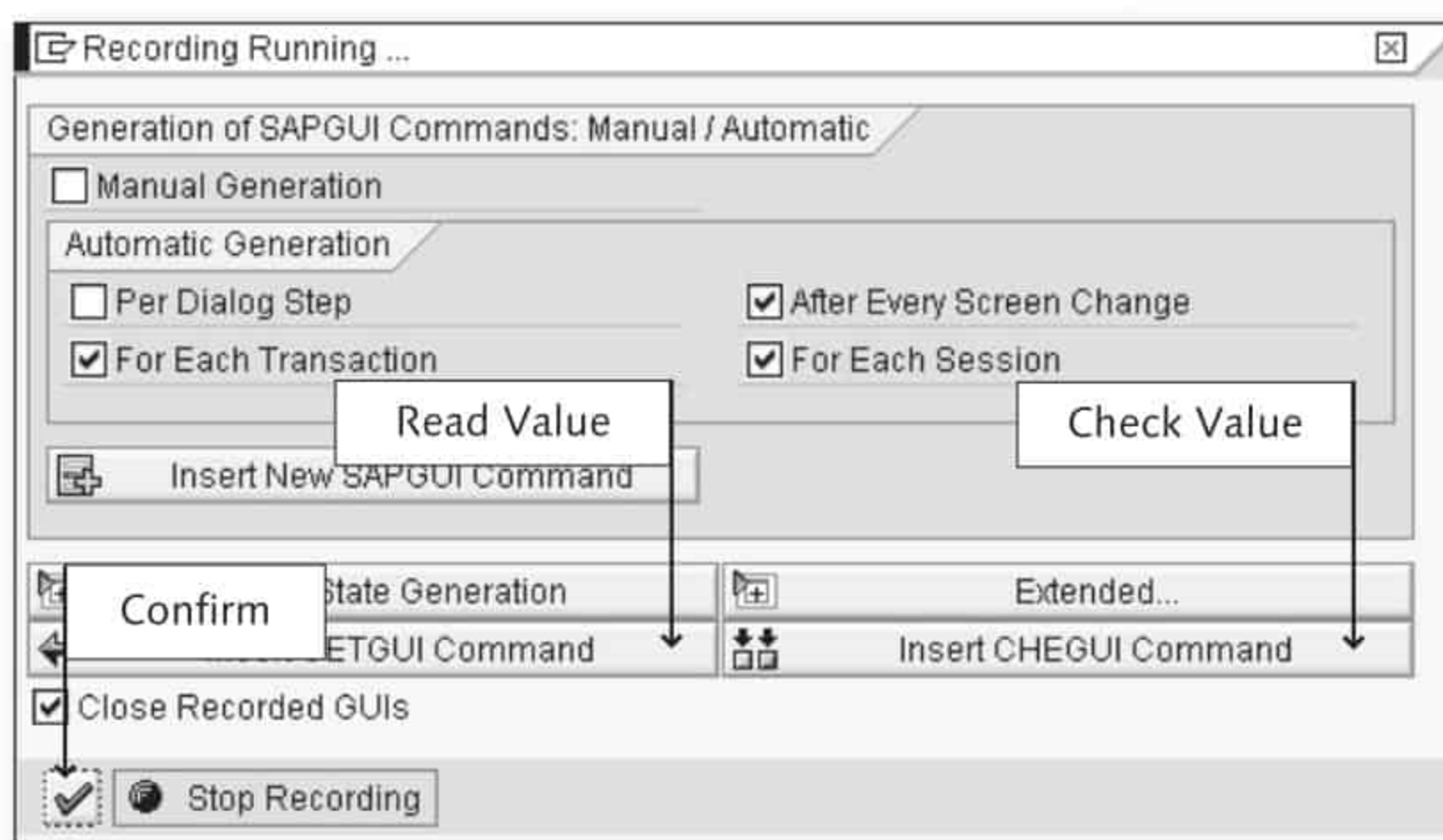
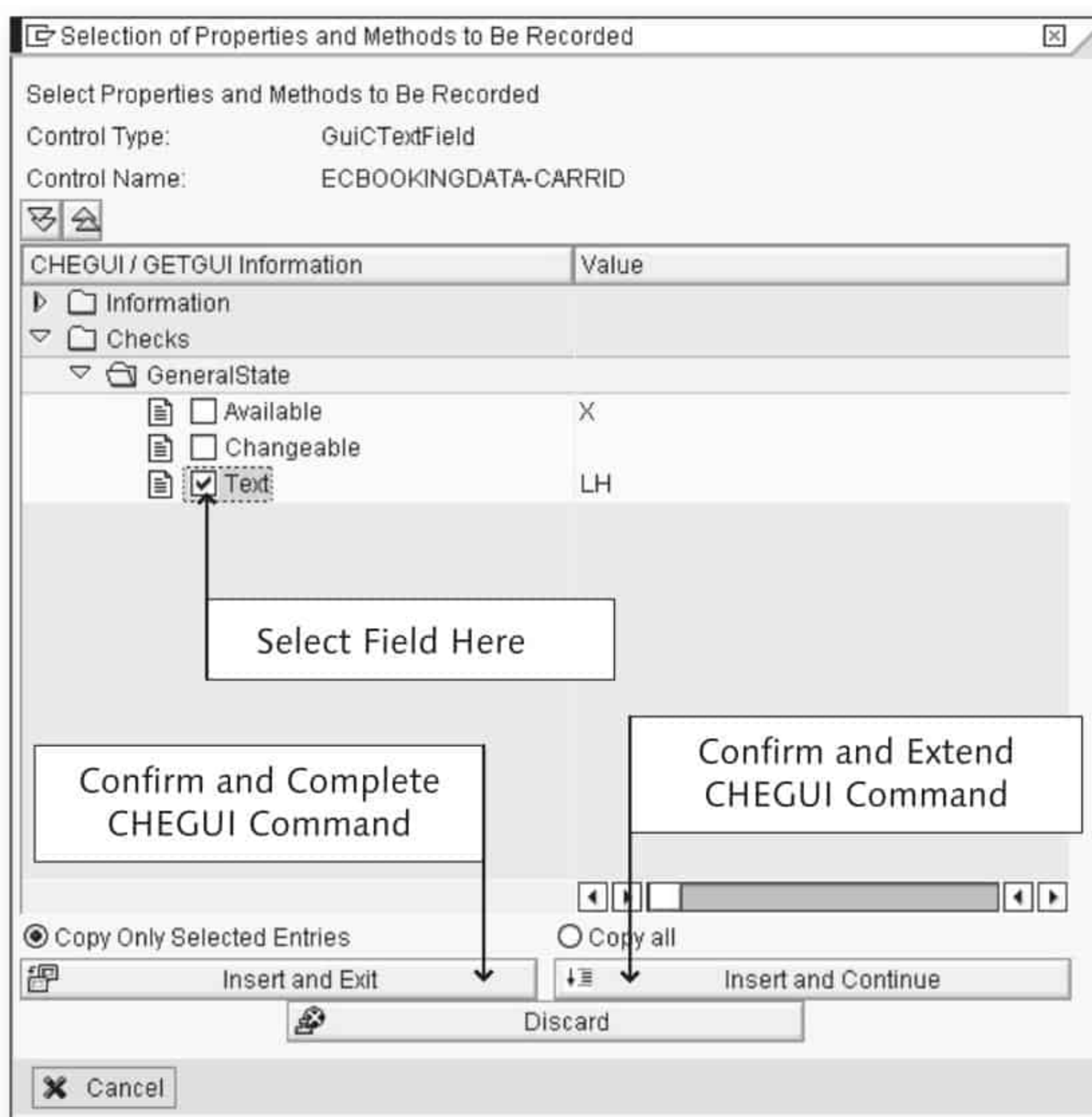


Figure 8.24 "Recording Running" Dialog Window

After you've selected the relevant option, the control returns to application mode. The subsequent steps are identical for the two commands.

You can use the mouse to select the area whose value you want to determine. Because SAP GUI is in selection mode, the selected area is indicated with a colored frame if you move the mouse over it. The selected area is copied using the left mouse button.

After you have selected the corresponding field, the system takes you to the dialog for editing the `GETGUI/CHEGUI` command (see Figure 8.25).



**Figure 8.25** Dialog for Processing a "CHEGUI"/"GETGUI" Command

Because every screen element contains quite a number of properties, you must decide which of these properties you want to check. Usually this is the input value of the field. For text fields, this is the `TEXT` property to be found under `GET/GENERALSTATE/TEXT`. For some other screen elements, like list fields, for example, the input value is named `VALUE` and can be found in the same place.

Selecting properties

In some situations it might make sense to check the availability of a screen element before it is accessed. For this purpose, there is an `AVAILABLE` property. It always has the value "X" at the time of recording. For processing, it only has a value of "X" if the relevant screen element is accessible.

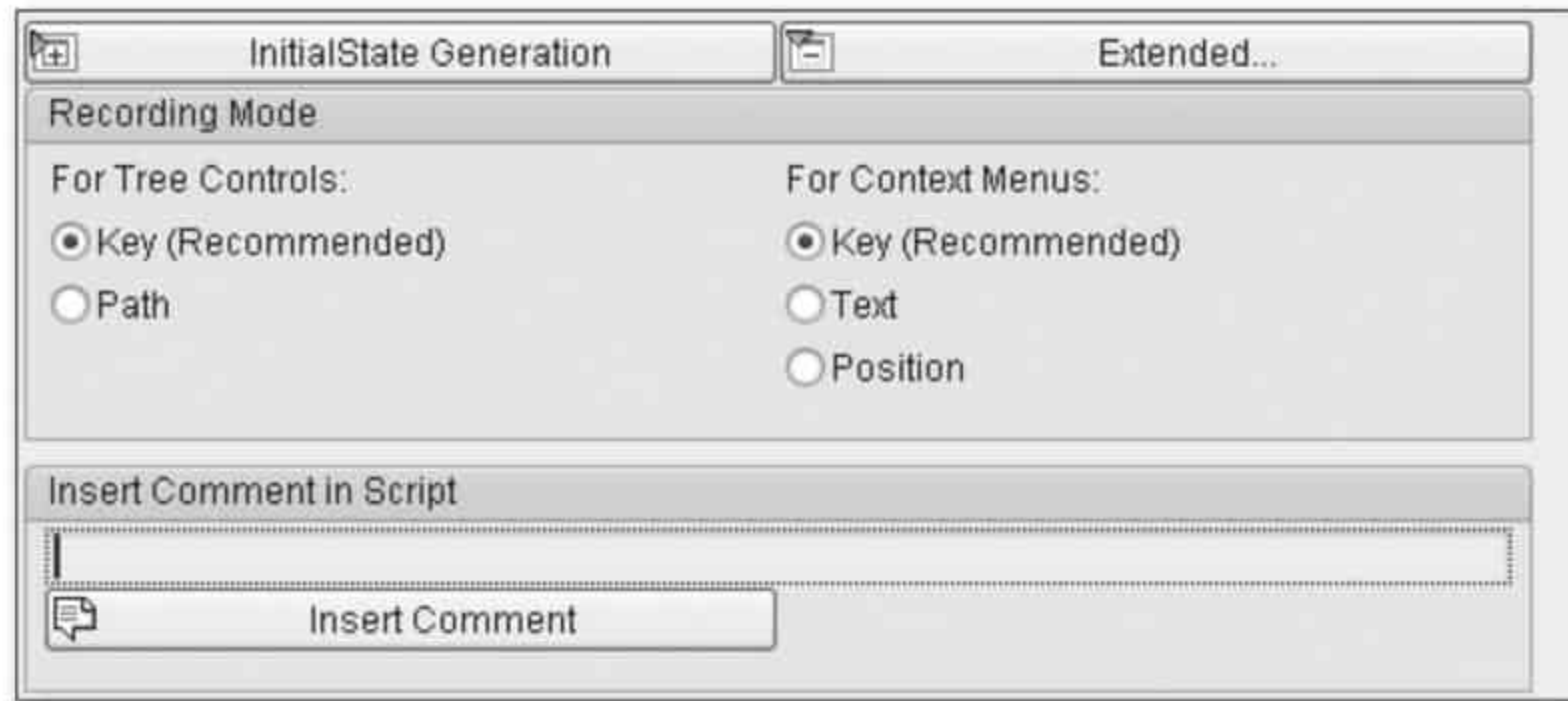
After you've selected the attribute that you want to read or want to read and check, you must select from the following options:

- ▶ **Insert and Exit**  
Enters the attributes in the command interface of the `GETGUI/CHEGUI` command and returns to the recording.
- ▶ **Insert and Continue**  
Enters the attributes in the command interface of the `GETGUI/CHEGUI` command and returns to the recording. You can insert additional fields in the command interface of the same `GETGUI/CHEGUI` command.
- ▶ **Reject**  
You exit the dialog, and the `GETGUI/CHEGUI` command is not inserted into the script.

**Initial state recording** Because the eCATT commands `CHEGUI` and `GETGUI` were not introduced until SAP Web AS Release 6.40, a different option of accessing field values has to be used in Web AS Release 6.20. For this purpose, the initial state recording is implemented. For SAP Web AS 6.20, this is the only option to read values. As of SAP Web AS 6.40, it is no longer recommended to use this function.

**"Extended" button** Last but not least, the recording window also provides the `EXTENDED` button. If you select this option, you are provided with the following additional functions (see Figure 8.26):

- ▶ In the `RECORDING MODE` area, you can specify in `FOR TREE CONTROLS` whether the recording is supposed to be run via a key or path and in `FOR CONTEXT MENUS` whether the recording is supposed to be implemented based on a key, text, or position.
- ▶ In the `INSERT COMMENT IN SCRIPT` area, you can insert a comment which is inserted into the currently recorded script in the background.



**Figure 8.26** Dialog with Expanded Extension Functions

After recording, you can find all actions performed by the user in the relevant command interface with all details. This interface contains different nodes depending on the command used (SAPGUI, GETGUI, CHEGUI).

Command interfaces

The SAPGUI command interface comprises the following main nodes (see Figure 8.27):

"SAPGUI" command interface

- ▶ The first node below the main folder contains the system information. The CONNECTIONID and SESSIONID nodes specify the window in which the system runs the SAPGUI command. It is essential that all SAPGUI commands have exactly the same values here in order to execute them in the same window. You can directly change the values for connection ID and session ID here. For more details on the IDs, refer to the relevant sections in this chapter.
- ▶ Each PROCESSEDSCREEN node corresponds to an executed dynpro within the SAPGUI command.
- ▶ Below the PROCESSEDSCREEN node, you can view the USERCHANGED-STATE section which lists all actions at GUI element level executed by the user. At this point, you can parameterize the GUI elements; for instance, input fields. For example, the blue arrow pointing to the right just before the term "text" indicates that the user made an entry in an input field. These fields can be parameterized, or more precisely only those dynpro fields can be parameterized that you've changed via user entries during recording. This also means that F4 input helps are not recorded, but only the actually selected value. If in this example we assume that the name "Printer1" was entered as the value, then, a parameter ( I\_PRINTERNAME in this case) can be assigned to this value in the window on the right.



In the USER CHANGEDSTATE section, you can only set the values. With GETGUI and CHEGUI, you can read and check values of output fields.

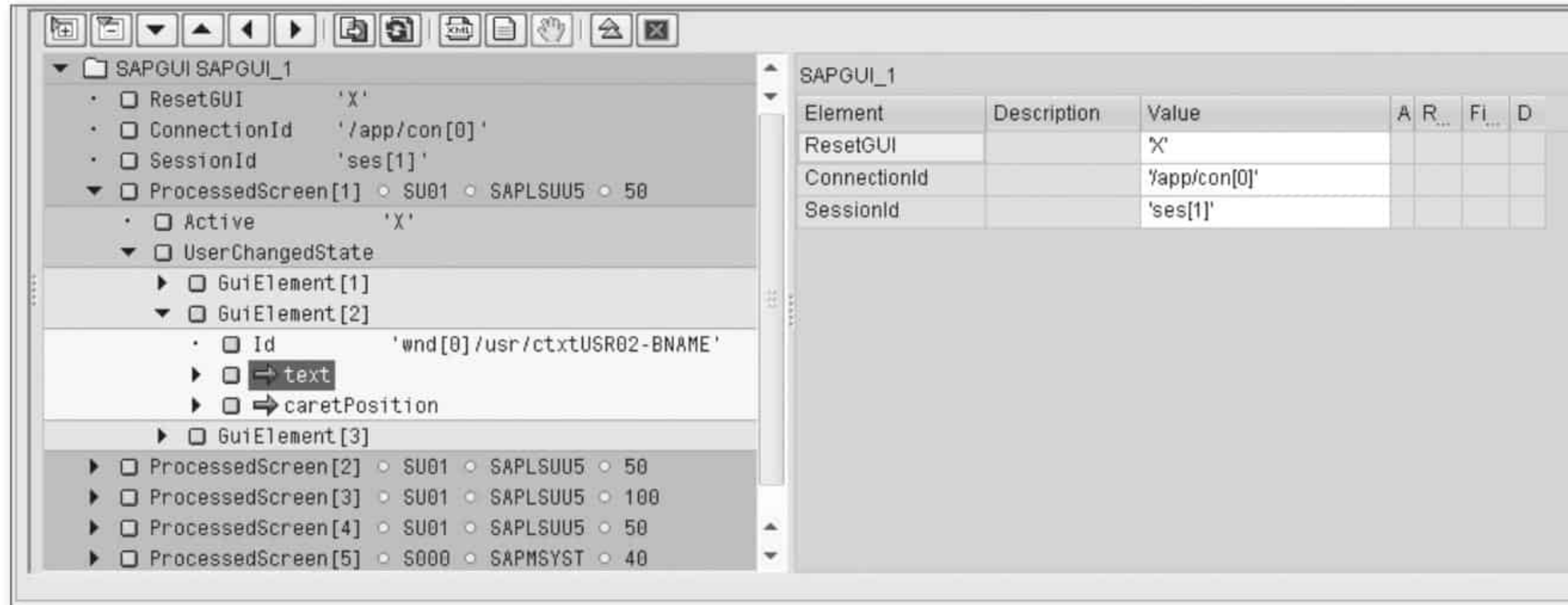


Figure 8.27 "SAPGUI" Command Interface

"GETGUI" command interface

If you inserted a GETGUI command for the recording, you can parameterize the corresponding field in the command interface so that you can exchange the values at runtime.

For this purpose, expand the GUIELEMENT node that contains the selected field, and enter the parameter or variable to be used in the VALUE row (see Figure 8.28).

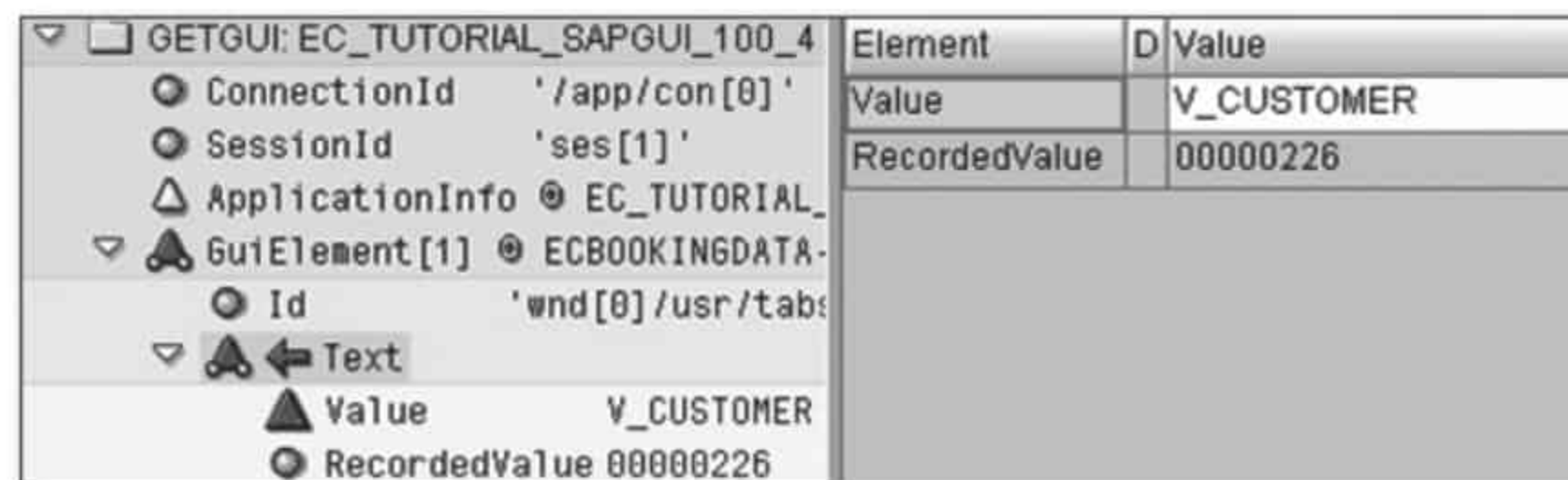


Figure 8.28 "GETGUI" Command Interface

"CHEGUI" command interface

The command interface of CHEGUI is very similar to the GETGUI command interface (see Figure 8.29).

In line with your selection, a CHEGUI command is inserted into your script. To parameterize a field, expand the GUIELEMENT node that contains the selected field. Select a relational operation (=, <>, ...) in the

CHECKACTION row, and enter the parameter or variable to be used for the check in the EXPECTEDVALUE row. You can also parameterize the VALUE field; there, you find the actual value at the time of execution.

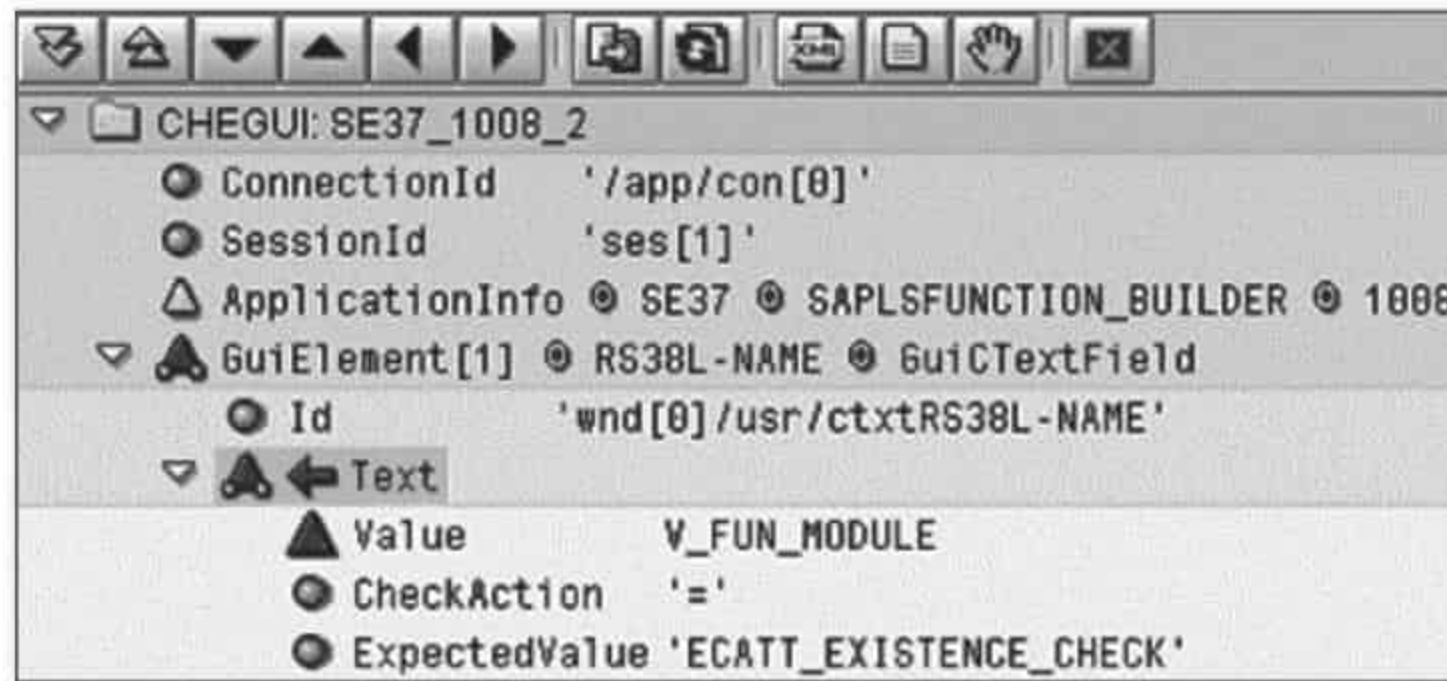


Figure 8.29 "CHEGUI" Command Interface

As of SAP Web AS Release 6.40, there are a number of functions for revising SAPGUI commands that have already been recorded. In addition to improving the usability, they primarily simplify the maintenance of the scripts.

Revising "SAPGUI" commands

An important aspect is that SAPGUI commands recorded as of SAP Web AS Release 6.40 can be extended at a later stage. To extend a script, you first need to ensure that you are at that point in the recorded session where you want to insert the extension. Then select PATTERN, and specify UI CONTROL as the group and SAPGUI (ATTACH) as the command. In the RECORD SAP GUI COMMAND WINDOW, define the granularity and whether the command is supposed to be recorded manually or automatically.

SAPGUI attach

After you've activated the ATTACH mode, the system displays a list of all sessions that you can record. This list contains all sessions on the local PC on which scripting is activated in the front end and the back end (except for the eCATT session). Choose the sessions to be recorded; you can activate multiple sessions at a time. Run the script recording as usual. The existing script is extended by adding the new steps.

If you want to use the extension function without working directly in the correct position in the application to be recorded, temporarily comment-out the commands not desired. Then execute the script. Ensure that the DO NOT CLOSE CREATED SESSIONS option has been selected in the script execution dialog. As soon as the session has reached the appropriate state

(in other words, the end of the existing part of the script), return to the script editor. Select SAPGUI (ATTACH) from the pattern dialog.

**"Join" and "Split" commands**

The function of splitting or joining SAPGUI commands after recording has also been added to the SAP Web AS Release 6.40. This is very useful for inserting CHEGUI/GETGUI commands at a later stage and improving the readability of the script.

**Joining "SAPGUI" commands**

To join SAPGUI commands, highlight them in the script editor and open the context menu. From the menu, select the JOIN item. A new SAPGUI command is created, and its command interface comprises the actions of all highlighted commands. The commands marked are commented out.

**Splitting "SAPGUI" commands**

Split functionality divides an SAPGUI command into several commands. This ensures a better overview, but is even more important in a re-recording when you want to split existing commands and insert new steps for re-recording. For example, if you forgot during a recording to fill a text field, you can split the command at the point where the text field needs to be filled. To split a long SAPGUI command, highlight the relevant command in the script editor. Open the context menu, and select the SPLITTING AT submenu. Then select the granularity. The following granularity levels are available:

- ▶ Transaction change
- ▶ Dynpro change
- ▶ Dialog step
- ▶ Methods/property

If you split a command in the script editor, the number of new commands depends on the granularity selected.

As of SAP NetWeaver 7.0, you can split in any place. If you split a command in the structure editor, two new commands are created for the resulting parts. Select the SPLIT COMMAND INTERFACE option from the context menu.

The original command is commented out, and the new commands are inserted with the new command interfaces. The original command interface is not lost. You can delete it if you are sure that you no longer require it.

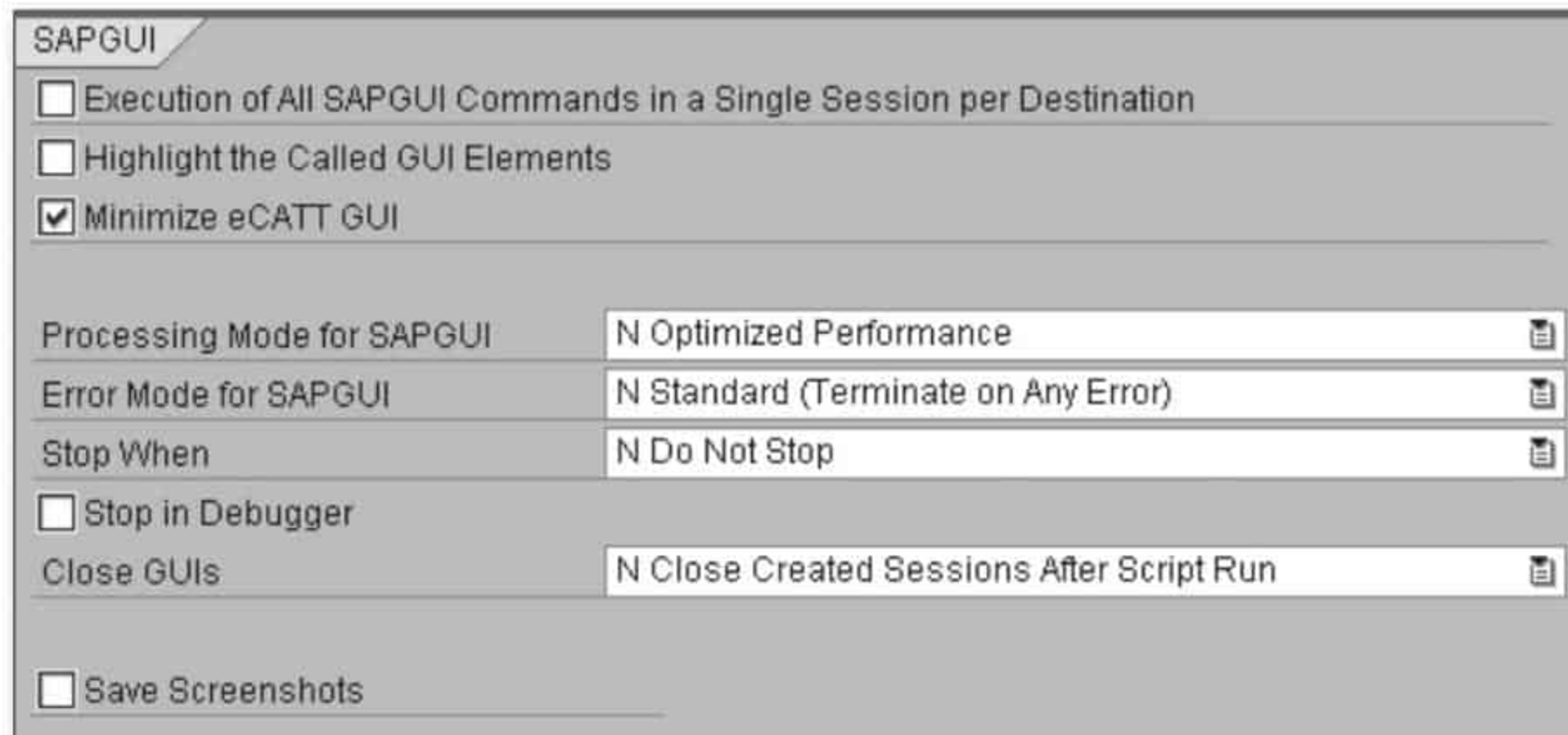
There are several alternatives for flexibly designing SAPGUI commands. This flexibility enables you to cover several test variants with different details in a single test script. For example, you can set the editing of a dynpro to optional. In the command interface of the SAPGUI command, set the `ProcessedScreen[n]\Active` value from "X" to "O" for optional. The dynpro is edited only if it is displayed by the application. Otherwise, this part of the script is skipped. The common use of this option is to handle popup windows in the script that, depending on the input data or context, are either displayed or not.

Flexibility of the recording

You can create more complex constructs to cover alternative paths using a dynpro. For this purpose, select a fine granularity for the recording (for instance, `METHOD`) and then toggle between different SAPGUI commands using conditionals (see the `IF` command; Section 8.9, *Additional eCATT Commands*).

When processing a script using SAPGUI commands, you are provided with separate SAPGUI-specific start options (see Figure 8.30).

Start options for the SAPGUI driver



**Figure 8.30** Start Options for Executing "SAPGUI" Commands

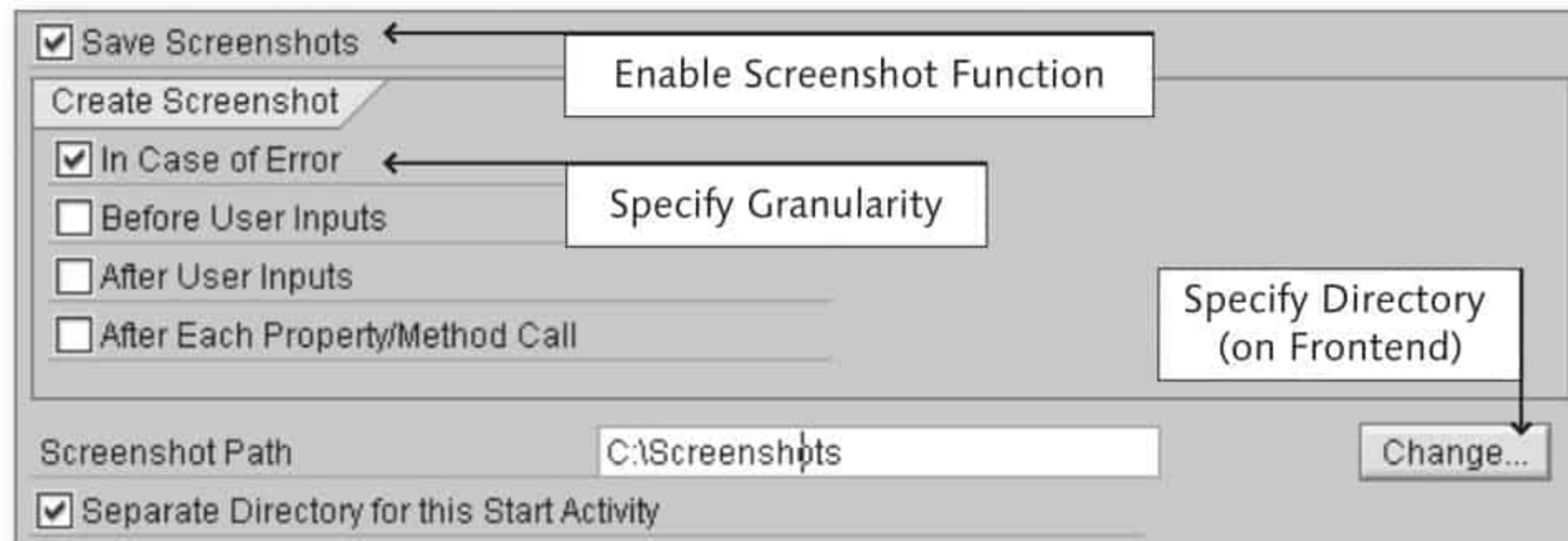
- ▶ The option `EXECUTION OF ALL SAPGUI COMMANDS IN A SINGLE SESSION PER DESTINATION` causes one session to be used per destination (target system). This is useful if you have different combinations of connection ID and session ID in your script or if you encounter difficulties with opening a new session due to a stored limitation of the number of sessions (default: 6).
- ▶ If the `HIGHLIGHT THE CALLED GUI ELEMENTS` option is enabled, the active screen element is highlighted with a red frame while the script

is being executed. This option can be very useful for debugging scripts.

- ▶ The `MINIMIZE eCATT GUI` option minimizes the SAP GUI window running eCATT to an icon on the task bar.
- ▶ The `PROCESSING MODE FOR SAPGUI` option is a performance parameter. The following modes can be selected:
  - ▶ **Optimized Performance**  
In this mode, the GUI updates are processed via the *automation queue*, which forwards them to the GUI. On the one hand, this improves the performance. On the other hand, a possible error in the script may not be displayed in the eCATT log directly where it occurs. Nevertheless, this is the recommended execution mode.
  - ▶ **Synchronous GUI Control**  
Bypasses the automation queue and sends GUI updates directly to the front end. In this mode, GUI updates are sent to the GUI synchronously.
- ▶ The `ERROR MODE FOR SAPGUI` option controls the behavior of eCATT in the case of an error. The selections are self-explanatory.
- ▶ The `STOP WHEN` option causes eCATT to interrupt the script execution in specific places. It is not continued until the user confirms it. This option is particularly useful during script development. In case of an interruption, the execution control operates via a popup window.
- ▶ If you additionally enable `STOP IN DEBUGGER`, the eCATT script switches to the debug mode whenever it is interrupted. In this case, the execution control is via the debugger and not via a popup window. More information can be found in Section 8.12, *Further Steps*.
- ▶ The `CLOSE GUIs` option enables you to specify whether and when the modes that have been automatically created during the script execution are automatically closed again.

#### Screenshot functionality

As of SAP NetWeaver Release 7.00, functionality for automatically creating and saving screenshots is provided (see Figure 8.31). This functionality was designed primarily for covering the documentation requirements in an environment where validation is mandatory. However, a sequence of screenshots can also be useful for tracing the individual steps of test runs for troubleshooting purposes.



**Figure 8.31** Screenshot Options in the Start Options

To enable the functions for automatically creating screenshots, select the **SAVE SCREENSHOTS** option in the start options. The screenshot options are displayed. Specify the granularity and a directory where the screenshots are to be stored. The specification of the granularity level defines the application events for which the screenshots are to be created. The screenshots are saved in .jpg format.

Because the functionality for automatically creating screenshots requires the availability of a user interface, it is only available for the SAPGUI driver. In an environment requiring validation, you should consider this when you select the test driver. It may make more sense in such a case to use the SAPGUI driver even for recording a transaction without controls, even though you would normally use the TCD driver due to its better maintainability and performance.

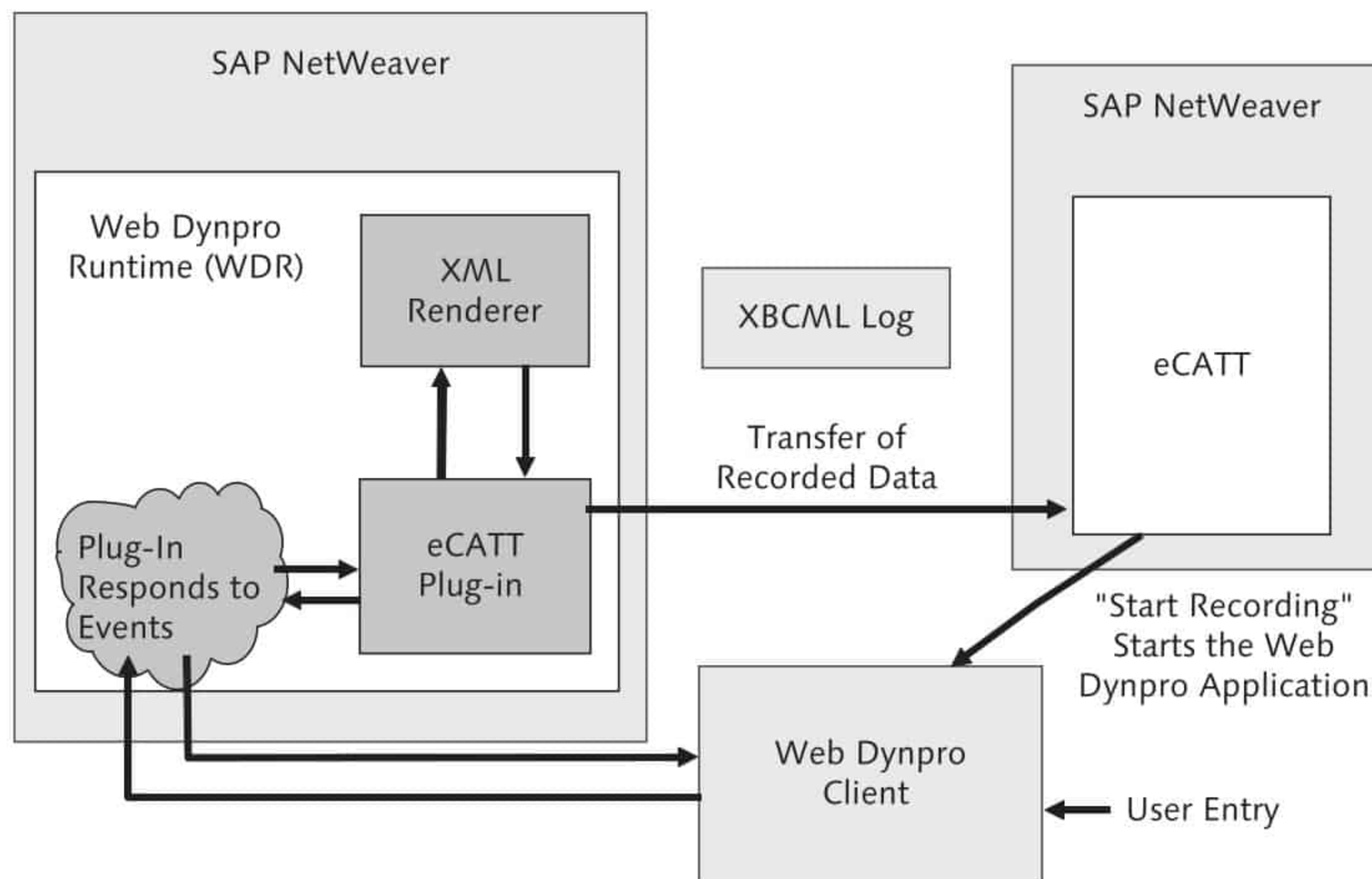
Regarding the message handling within the SAPGUI driver, please refer to Section 8.6.3, *Message Handling*.

### 8.2.3 Testing Web Dynpro Applications

As of SAP Web AS 6.40, eCATT supports the direct testing of Web Dynpro Java-based applications. For SAP NetWeaver 7.02 and as of SAP NetWeaver 7.20, the testing of Web Dynpro ABAP-based applications is also supported.

When recording a Web Dynpro application, the user operates it either in a Web Dynpro client (for Web Dynpro ABAP and Web Dynpro Java) or via the browser (only for Web Dynpro Java). Figure 8.32 shows the recording via the Web Dynpro Business Client.

Architecture



**Figure 8.32** Recording Web Dynpro Commands

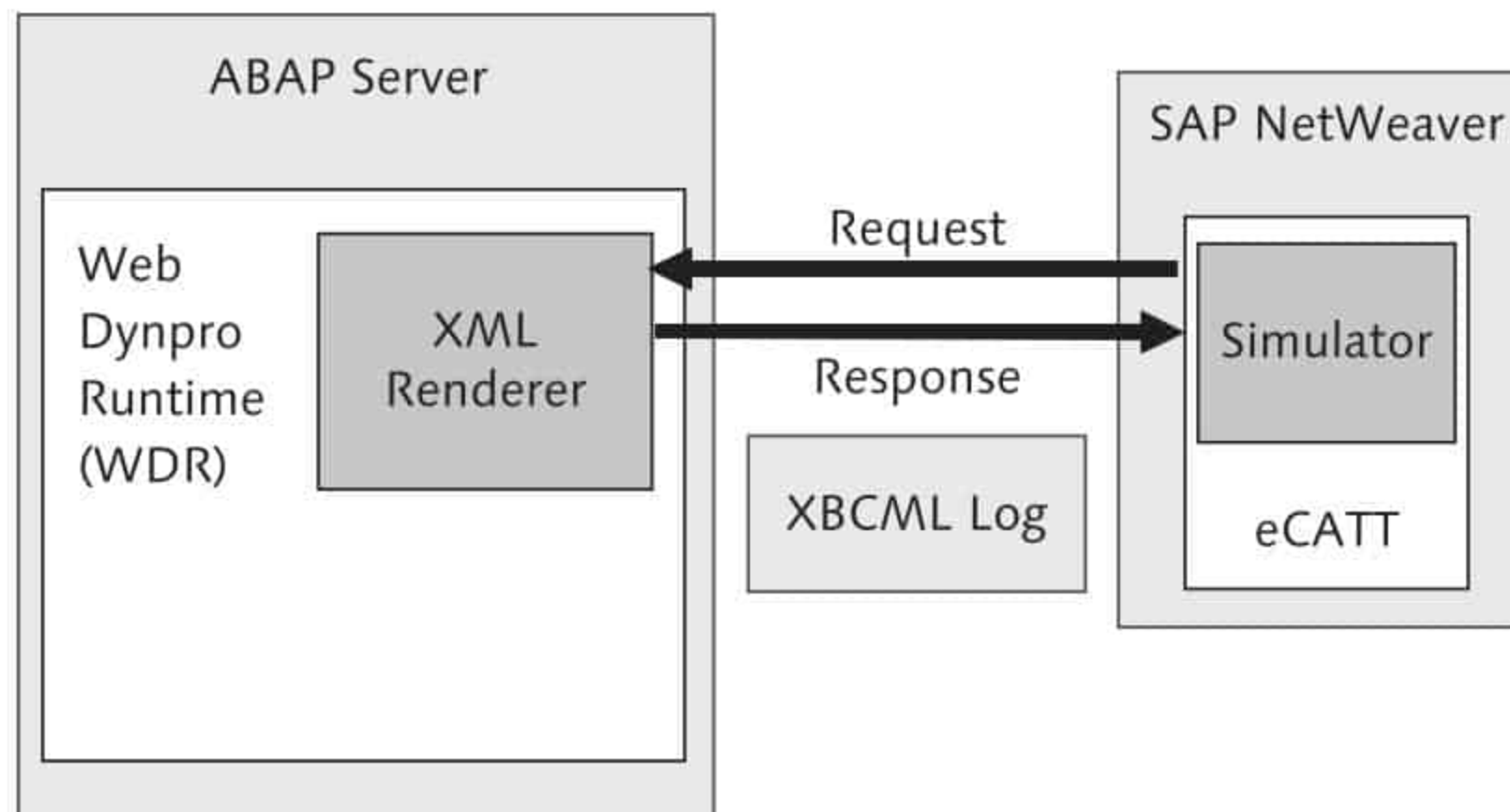
**Communication while recording**

When you start the recording of a Web Dynpro in Web Dynpro ABAP, the Web Dynpro Client opens. In Web Dynpro Java, the user can select between the recording in the Web Dynpro Business Client and recording via the browser. The user operates the application in the Web Dynpro Client or in the browser. As soon as changes are made to the Web Dynpro context (back end) that is, if data change or actions are executed the plug-in is informed about this and then records the operations. In Web Dynpro, eCATT records the business logic of the application and not the events in the interface as in the SAP GUI with controls, for example. The data recorded is then sent to eCATT as an XML file in the background.

**Communication while processing**

The processing of Web Dynpro applications is done in the background via HTTP without a browser.

During processing, the request is sent to the target system via HTTP in XML format. The request corresponds to an input in the UI plus the Web Dynpro action. An XML description of the next page to be displayed is returned as the response. The description contains the structure of the window or the subareas of a window as well as the data. If, for example, an input field is filled with values, this value is also part of the client's response. The processing usually occurs in the background. Using the simulator, you can trace the processing of the transactions on screen.



**Figure 8.33** Processing a Web Dynpro Transaction

The URL for addressing the Web Dynpro application is structured as follows:

Structure of the URL

`HTTP://<Server>:<Service>/<URL extension>/<Application>`

The meanings of server (name or IP address) and service (port) are familiar. The URL extension looks different depending on whether the Web Dynpro runtime environment is based on the Java or the ABAP stack. If you are using the ABAP stack, the extension is:

`<ABAP URL extension> = sap/bc/webdynpro`

If the Web Dynpro runtime environment is based on Java, the extension has the following format:

`<Java URL extension> = webdynpro/dispatcher`

The first step for recording a Web Dynpro-based application is the creation of the targets for the HTTP connections. Use Transaction SM59 for this purpose. Java-based applications require an HTTP connection to an external server (connection type G). For ABAP applications, you should create an HTTP connection to the SAP system (connection type H).

Creating the connections

Enter host name and service (port) of the target system as shown in Figure 8.34.

As soon as you have created and successfully tested the HTTP connection, insert a logical target in the system data container. In the HTTP DESTINATION column, store the newly created connection for this target system.

**Figure 8.34** Creating a Connection Transaction SM59

#### Recording Web Dynpro transactions

To record a WEBDYNPRO command, go to the script editor. Select PATTERN, and from the UI CONTROL group select the WEBDYNPRO command. A dialog box for specifying the Web Dynpro application is displayed. This dialog box differs, depending on whether you want to record Web Dynpro ABAP or Java. In case of Web Dynpro Java, you have the option of starting the recording in the Web Dynpro Client or in the browser. In case of Web Dynpro ABAP, only the recording button is available, and the recording is started automatically in the Web Dynpro Business Client when you select this button. Select a target system (the target system with the HTTP connection you previously created; see Figure 8.35).

In the APPLICATION input field, complete the basis URL of the connection with an application-specific section. This section depends on the application to be recorded; for example, CarRental. The address sections are then merged by eCATT. Then, in the case of Web Dynpro Java, select the recording type (in the browser or Web Dynpro Business Client). Subsequently, click the START RECORDING button. Note that the Web Dynpro Business Client must be installed to be able to start the recording in the Business Client (see SAP Note 773899).

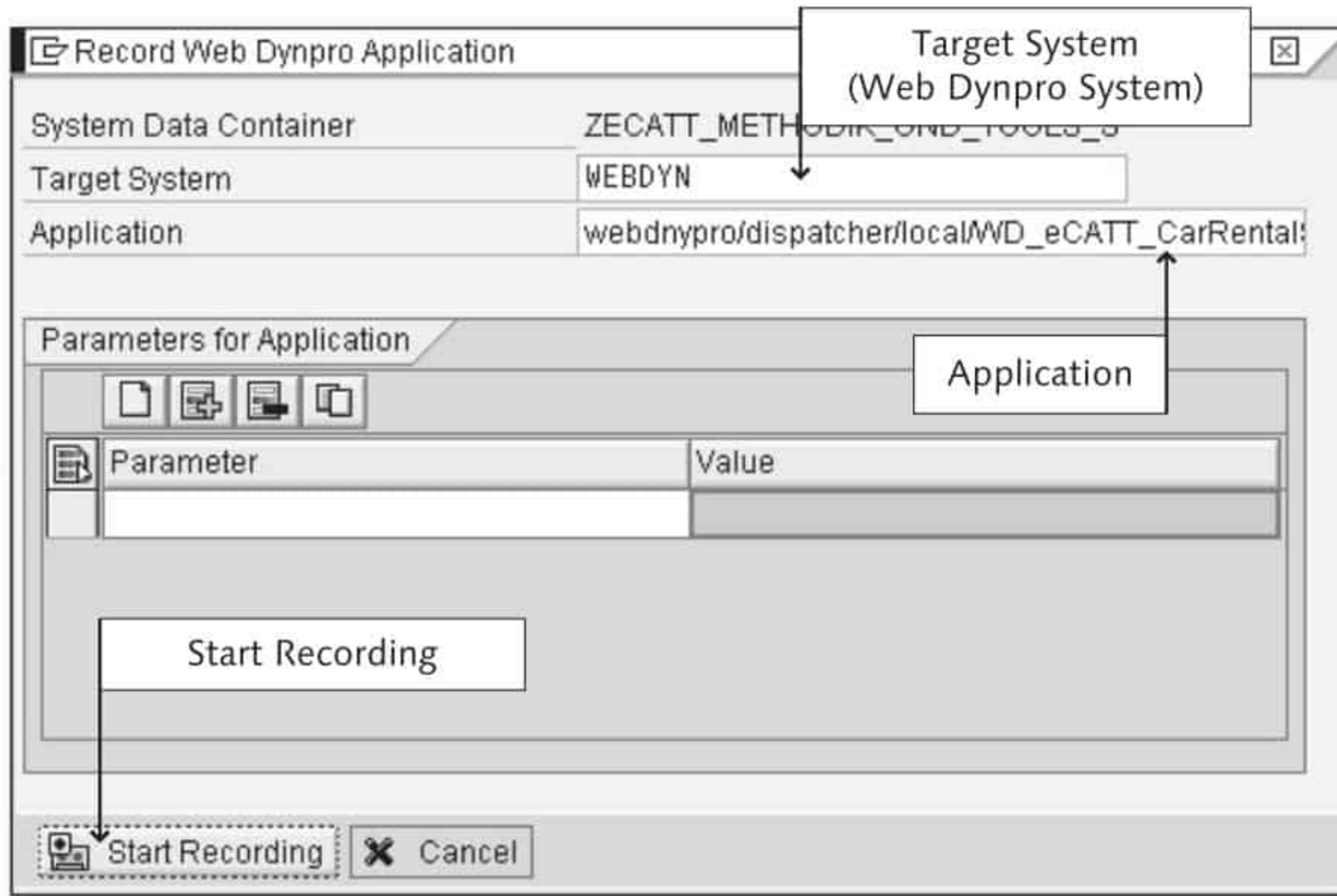


Figure 8.35 Starting the Web Dynpro Java Recording

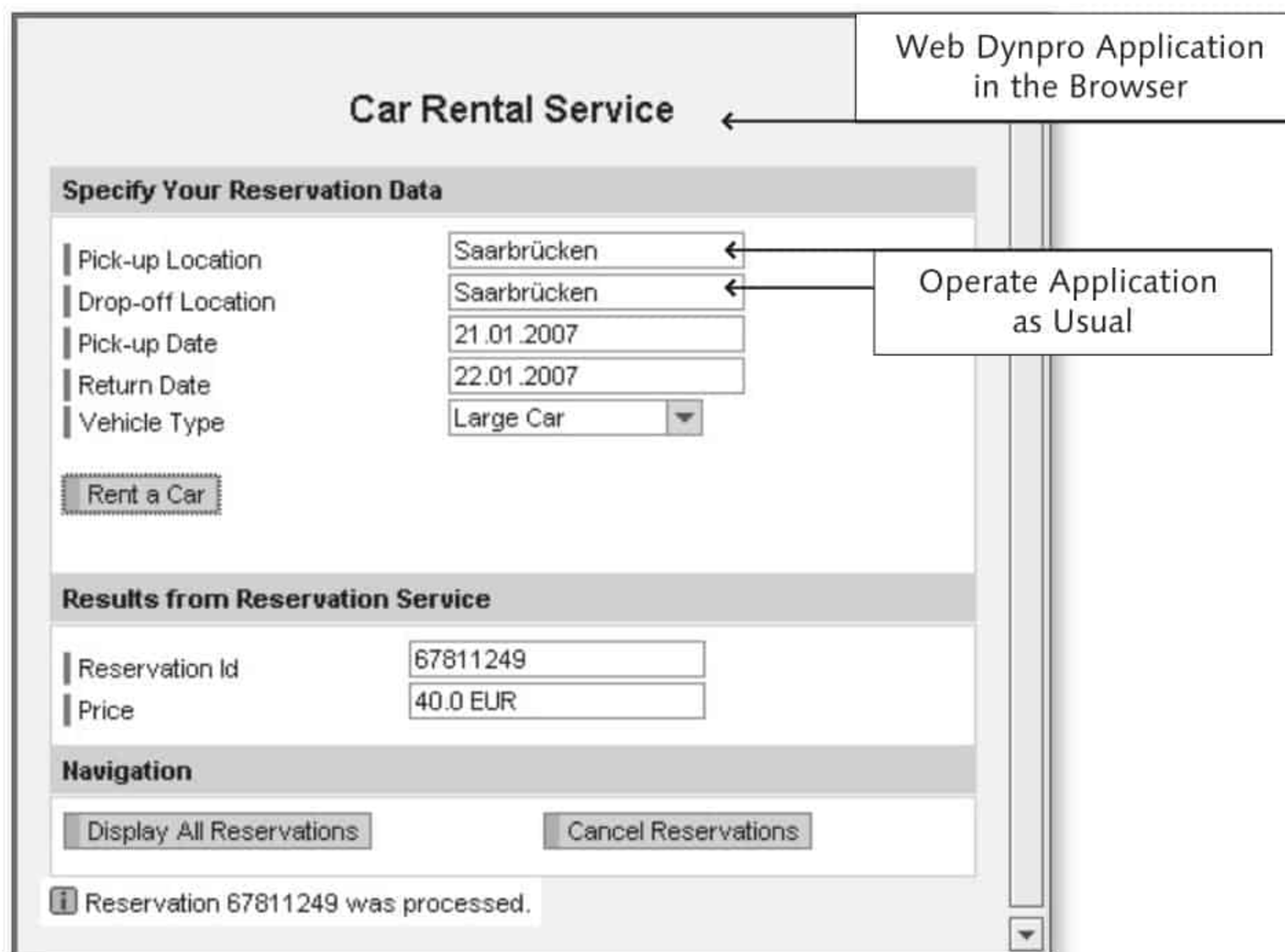
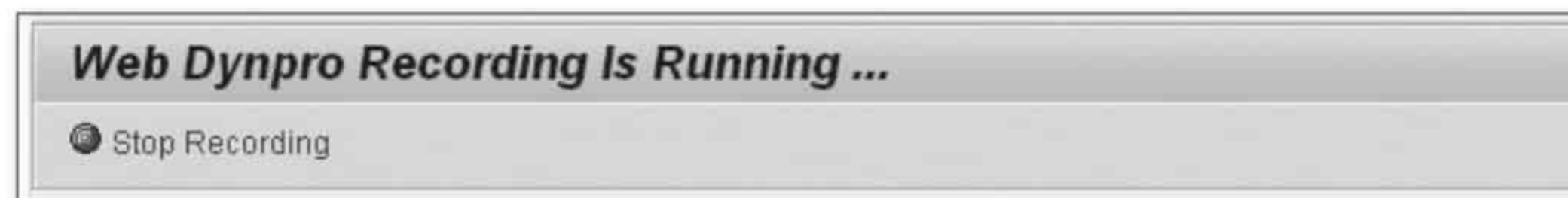


Figure 8.36 Browser Window with Web Dynpro Application During Recording

As soon as you start recording, eCATT automatically opens a browser window or Web Dynpro Business Client window containing the Web Dynpro application (see Figure 8.36). In this window, you operate the application as usual and populate the dialog elements with values. Your

input is recorded and is later available in the command interface of the WEBDYNPRO command.

At the same time, another window opens in which you can stop recording again (see Figure 8.37).



**Figure 8.37** Dialog Window for Stopping the Recording

As soon as you have stopped the interaction with the Web Dynpro application, use the task bar to change to this window and stop the recording. Now the WEBDYNPRO statement is displayed in the script editor.

#### "WEBDYNPRO" Command

The WEBDYNPRO command is designed for addressing the Web Dynpro driver. It has the following notation:

```
WEBDYNPRO (<Command interface>, [<Target system>] ).
```

The target system must be an HTTP connection of the G type (for Java Server) or the H type (for ABAP Server).

**Command interface** The command interface of the WEBDYNPRO command comprises the following nodes with recording details (see Figure 8.38):

- ▶ Under SCREEN • DATA, you can find the XML description of the page returned. This description is relevant if the pages are supposed to be displayed in the eCATT Web Dynpro simulator.
- ▶ Under DATACHANGES, you can find the values entered during the recording. Like with an SAPGUI command, you can parameterize the recording to link the Web Dynpro command to the test data.
- ▶ Under ACTION, you can find the actions that trigger a round trip. For example, data changes do not become effective until the relevant action triggers the round trip.
- ▶ Under GETS AND CHECKS, the system displays the details on checks or value determinations. This node is only visible if such checks/value determinations have been started from the simulator (see next section).

- ▶ Under PAGE • SCREEN • MESSAGES, you can find the messages sent during recording. You can process messages from Web Dynpro applications via a MESSAGE ... ENDMESSAGE block as well (see Section 8.6.3, *Message Handling*).



**Figure 8.38** Command Interface of the Web Dynpro Command

The `WEBDYNPRO` command includes a page simulator which enables you to determine values from output fields or execute checks on their content. To start the simulator, open a `WEBDYNPRO` command interface in the structure editor, and select the magnifying glass button. In the simulation, select the field to be checked, and then choose `INSERT CHECK`. If you want to determine the value of a field, select `INSERT GET` instead. In both cases, a new node, `GETS_AND_CHECKS`, is inserted in the command interface. In this node, you can now determine or test the values of a page. The parameterization is similar to that from the `GETGUI` and `CHEGUI` commands.

Simulator



You can also directly go to the simulator display from the log if you double-click the `PAGE` node. This way, you can immediately determine to which screen a possible error refers. Even in the debugger, you go to the simulation if you press the `[F5]` key. Note that the SAP NetWeaver Business Client must be installed to use the eCATT Web Dynpro simulator. SAP Note 773899 provides more detailed information.

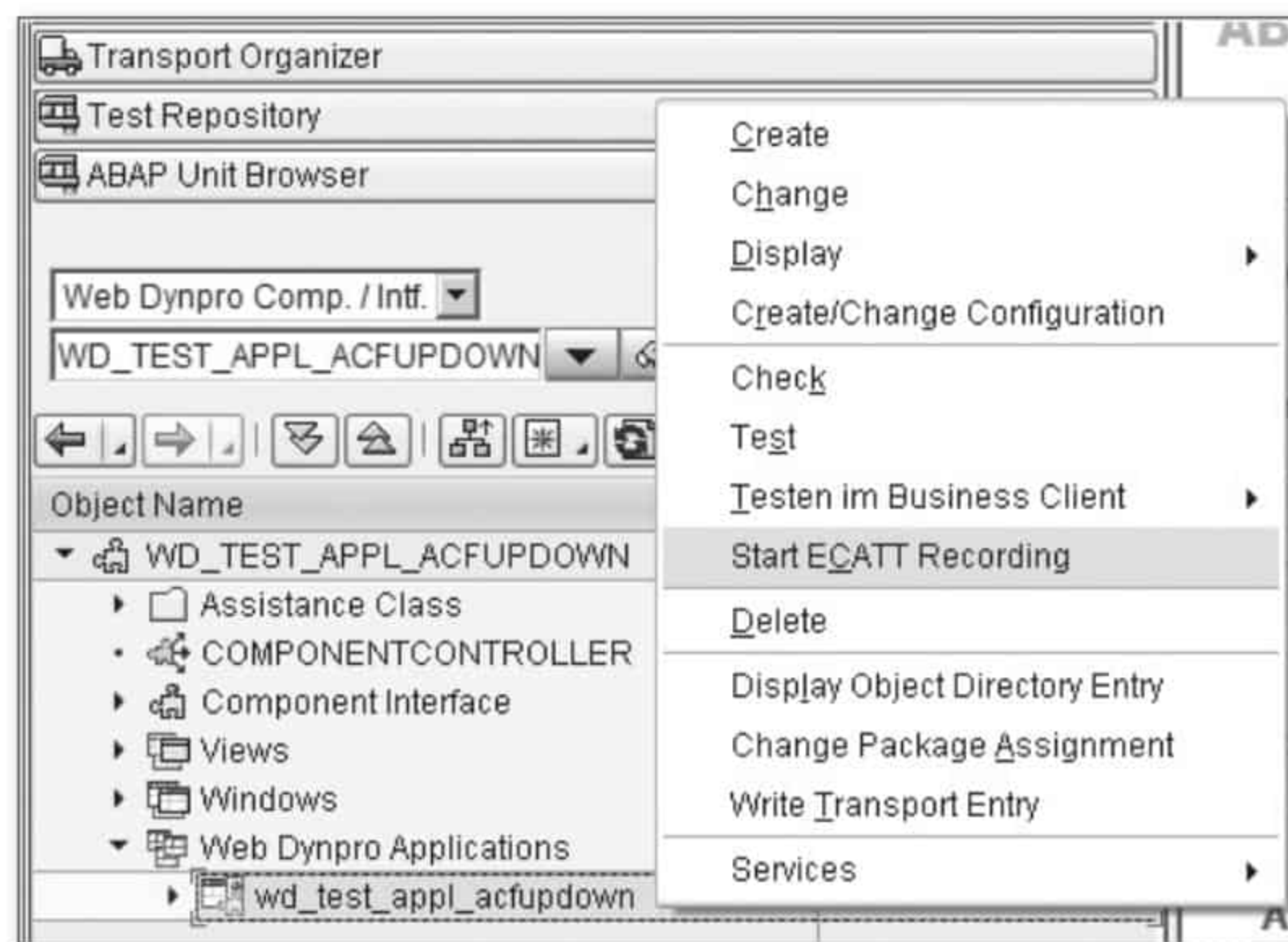
When the Web Dynpro driver is used, you have the following start options:

Start options for the Web Dynpro driver

- ▶ If you select the `PROCESS IN BACKGROUND` option, the script is processed without being displayed in a user interface. The progress of the script cannot be traced on screen. However, this option ensures the best performance.

- ▶ If you select the PROCESS IN FOREGROUND option, the progress of the script is displayed in a user interface. The PROCESS IN FOREGROUND (DISPLAY RECORDED PAGE IN PARALLEL) option causes the progress of the test case in the application at the time of recording to be displayed in a second window in addition to the actual script. Because it enables comparison, this option is very helpful for the troubleshooting process during the script development. For both options for processing in the foreground, you can specify how long the respective screen is supposed to be displayed.

**SE80 connection** You can also start the eCATT recording of a Web Dynpro application directly from the object navigator (Transaction SE80; see Figure 8.39). Here, the benefit is that you don't need to specifically call the eCATT Transaction SECATT. An XML file is returned.



**Figure 8.39** Starting the Recording from Transaction SE80

If you select the Pattern button in Transaction SECATT, two additional Web Dynpro commands are available for selection within the UI CONTROL group: WEBDYNPRO (attach) and WEBDYNPRO (import). The latter option refers to the Web Dynpro Java applications. If a Web Dynpro Java application is recorded in SAP NetWeaver Developer Studio, the system generates an XML file that can be uploaded to eCATT. Then, the corresponding Web Dynpro commands are generated in eCATT. WEBDYNPRO (attach) enables you to extend scripts from the Web Dynpro simulator.

### 8.2.4 Summary

UI-based applications can be tested with eCATT via different test drivers.

Transactions without controls can be recorded by the TCD driver: comparable to a macro recorder that you might know from Microsoft Excel. During parameterization, you replace the fixed values entered during the recording process with parameters and thus can create a flexible, usable script. Changing the assignment mode enables you to read and check the actual values of a parameter.

Transactions with controls can be recorded using the SAPGUI driver. The selection of the correct granularity is very important, although it can be corrected at a later stage in more recent eCATT versions. For reading and checking values, the `GETGUI` and `CHEGUI` commands are available. Moreover, you can join and split script commands using the `JOIN` and `SPLIT` commands.

Web Dynpro applications can be recorded just like SAP GUI transactions. The URL is opened automatically in the Web Dynpro Business Client or a browser. The recording takes place in parallel while the Web Dynpro client or the browser is being operated by the SAP application.

## 8.3 Creating Tests Via Direct Program Control

eCATT enables you to test the following program controls:

- ▶ Global ABAP object classes
- ▶ Function modules and BAPIs
- ▶ Inline ABAP
- ▶ Database accesses

### 8.3.1 Testing Global ABAP Object Classes

The eCATT command set comprises commands for testing global classes. You can find the set of commands if you open the desired test script in transaction SECATT, click the `PATTERN` button, and select the `ABAP OBJECTS` group.

To be able to access instance attributes and methods of a class, you must create an instance of the class. As a prerequisite, there must be a param-

eter of the type to which the object can be assigned. The parameter must contain the type of the class and a class or an interface from which the class inherits.

By default, an object is created using the `CREATEOBJ` command. However, you also can create an object by using a function module or a method that provides an object.

#### Instance attributes and static attributes

To access instance attributes, you first need to instantiate the object. The following table lists commands available for accessing instance attributes and static attributes of a class.

Command For Accessing Instance Attributes	Command For Accessing Static Attributes	Description
CHEATTR	CHESTATIC	Compares the current value of an attribute with the value specified.
GETATTR	GETSTATIC	Calls the current value of an attribute and assigns a parameter to this value.
SETATTR	SETSTATIC	Changes the current value of an attribute to a value specified.

**Table 8.3** Commands for Accessing Instance and Static Attributes

**Methods** You use `CALLMETHOD` to call an instance method of a class. For this purpose, you must first instantiate the object. To call a static method, use `CALLSTATIC`.

### 8.3.2 Function Modules and BAPIs

In the following cases, it could be necessary to test or use function modules:

- ▶ In a test in which you test an individual function module.
- ▶ In a background process in which you test a chain of function modules which represent a complete process by transferring results of a module to the next one. You can perform this work step in a single script or in several consecutive test scripts.
- ▶ As a utility within a script; for instance, to call data from the database which you want to use in transactions or to perform complex plausibility checks via already existing function modules.

**"FUN" Command**

FUN

The eCATT command required to call function modules is named `FUN`. After opening a test script in Transaction `SECATT`, you can select the `PROGRAM CONTROL` group via the `Pattern` button.

The command interface corresponds to the interface of the function module. In the structure editor, it is subdivided into areas for the different parameter types—import, export, change, and tables—with an additional area for exceptions. For the transfer to a function module, you assign the relevant literal or variable to the correct parameter in the command interface. To call a value from an export parameter of the function module, you assign a variable to the parameter.

**8.3.3 Inline ABAP**

Inline ABAP provides the option of inserting complex programming elements into an eCATT script. An inline ABAP block is initiated with the `ABAP` script statement and ends with `ENDABAP`. This statement can also be selected via the `Pattern` button in the `PROGRAM CONTROL` group.

Within the inline ABAP block, you can use (almost) any ABAP commands. Inline ABAP is particularly required for complex database accesses, but it can also be used for all calls of function modules and ABAP OO routines.

**Note**

When you use inline ABAP, you should avoid any user interactions as well as mode-generating or mode-ending operations (for instance, `CALL FUNCTION STARTING NEW TASK`) because otherwise an automatic test execution is no longer possible.

The connection to the eCATT script is established via the local variables (parameter type `V`). All local variables are automatically declared within inline ABAP and are provided with the current values from the script execution. After executing the inline ABAP block, all local variables are in turn provided with the values that were assigned within the inline ABAP.

**Tip**

You should outsource inline ABAP scripts to custom scripts in order to keep the number of local variables as low as possible. Transfer to and from inline ABAP can hurt performance when complex variables and large tables are used.

### 8.3.4 Database Accesses

CHETAB

#### "CHETAB" Command

The CHETAB command compares field values of a database table with values which you define in the command interface; the command interface has the same structure as the table.

Using the CHETAB command, you can check the following:

- ▶ Database updates after transactions
- ▶ The existence of specific table entries
- ▶ Entries in customizing tables

You maintain the values of fields in the command interface. For this purpose, you aren't limited to the key fields, and you don't need to specify the complete key either. However, you shouldn't leave any key field empty; we recommend entering an asterisk in each key field to which you don't assign a value.

If no data record is found that meets the search criteria, an error message is written to the log. If one or more data records are found, the check is entered as successful in the log.

You can specify the system in which the database table can be found. If you specify a target system, a corresponding system container must be assigned to the test script or the test configuration.

GETTAB

#### "GETTAB" Command

The GETTAB command reads a data record of a database table and assigns the values of the data record to the corresponding fields of a command interface. The command interface has the same structure as the table.

In general, the same conditions as apply to CHETAB apply to the maintenance of field values. For this reason, if you don't specify the full key, multiple data records can meet the selection criteria. GETTAB reads the first data record that meets the selection criteria.

### 8.3.5 Summary

Besides UI-based applications, you can use eCATT to test the following program-controlled applications: ABAP object classes, function modules and BAPIs, and inline ABAP, as well as database accesses.

## 8.4 Creating Tests for Web Services

The SAP NetWeaver Application Server enables enterprises to extend their solutions by integrating web services and providing them to their users. It supports the XML, SOAP (Stateless, Stateful, and Security), WSDL, and UDDI standards.

As of SAP NetWeaver 7.0, eCATT supports the automated testing of web services. A web service is then called in the same way as an internal ABAP function module. The necessary ABAP proxy classes are generated automatically.

Because the functionality of the eCATT web service driver is not limited to testing web services provided via the SAP NetWeaver Application Server, you indirectly gain an interesting alternative for testing third-party solutions. As long as the third-party system in your process chain has a web service interface, you can integrate it seamlessly in the test coverage via eCATT scripts.

Testing third-party solutions

To test a web service, first generate an HTTP connection using Transaction SM59. Then, in the eCATT script editor, click on the PATTERN button and from the ENTERPRISE SERVICES group select the WEBSERVICE command. Figure 8.40 shows the dialog field that is displayed.

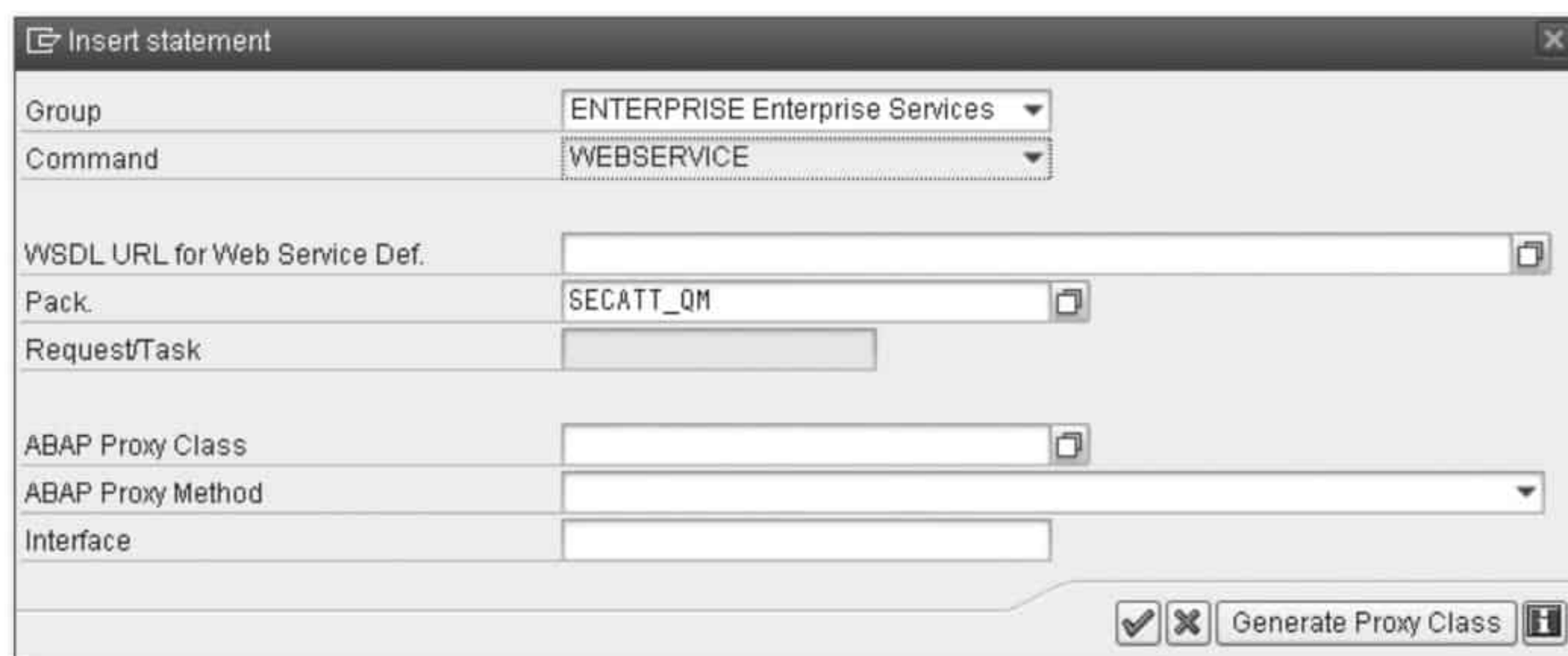


Figure 8.40 Inserting a Web Service Test

Because web services are function-like objects that do not allow for direct user communication, no recording occurs in this step. Instead, you specify a method call that is submitted to the web service.

#### ABAP proxy objects and WSDL

If the web service is implemented on an SAP NetWeaver Application Server and you know the ABAP proxy class on which it is based, you can select the corresponding ABAP proxy class from the directory. Once you've confirmed your entries, the appropriate values are inserted in the fields, ABAP PROXY METHOD and INTERFACE.

If there is no appropriate ABAP proxy class yet, eCATT supports you by automatically creating an appropriate ABAP proxy class. In the WSDL URL FOR WEB SERVICE DEFINITION field, enter a URL where a web service description can be found. The WSDL description (Web Service Description Language) specifies the functions provided by the service. Usually, you will want to obtain the description directly from the used server. In that case, the URL normally uses the following format:

```
HTTP://<Server>:<Port>/<Web Service Name>/Config?wsdl
```

As soon as the correct address for the service description has been entered, click on the GENERATE PROXY CLASS button; eCATT queries the capabilities of the web service and generates an ABAP proxy class. This generated class is entered directly in the ABAP PROXY CLASS field. You need to assign the class to a package.

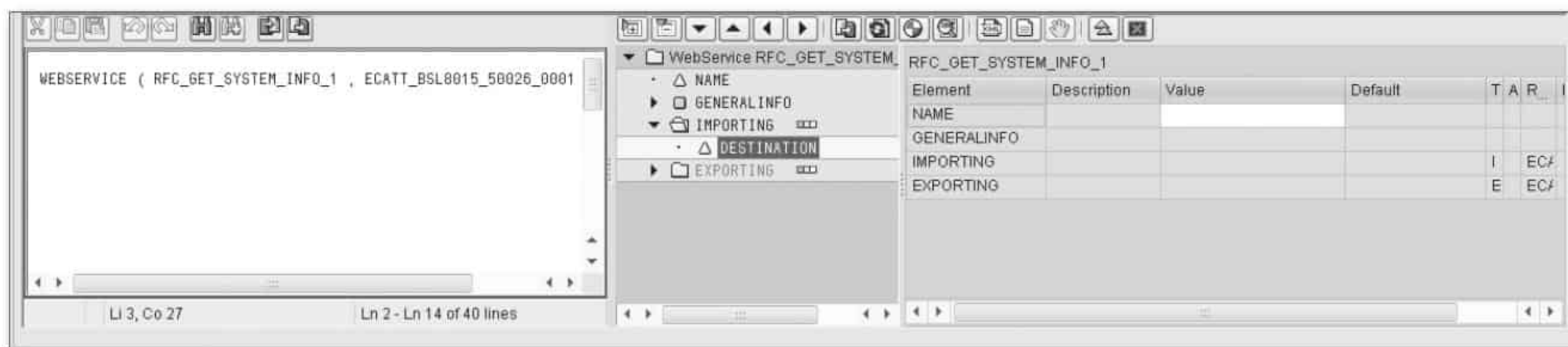
Once you have a functioning proxy class, you can select the visible methods of the class in the ABAP PROXY METHOD field. Select the wanted method from the list, and close the dialog box. eCATT then generates a WEB SERVICE command in the script editor.

#### "WEB SERVICE" Command

The WEB SERVICE command is used for addressing web services. It has the following format:

```
WEB SERVICE(<Command interface>, [<Target system>] ).
```

The command interface (see Figure 8.41) corresponds to the interface of the selected function from the ABAP proxy class. Usually, it is generated automatically from the WSDL. Therefore you do not need to worry about the structure of the transferred parameters. Just populate the command interface with appropriate values.



**Figure 8.41** Command Interface of the "WEB SERVICE" Command

With SAP NetWeaver 7.1, you can upload web service test scenarios as XML files into eCATT to test WEB SERVICE commands with eCATT. For this purpose, you need the Composition Environment as of Release 7.1. Moreover, the Web Service Navigator must be configured accordingly. Prior to the uploading process, you can make presettings for parameterization. You can select which import or export parameters should be created.

Uploading web service test scenarios

The following upload options are available:

- ▶ Upload of the test scenario and simultaneous generation of eCATT objects (test configuration and test script). For this purpose, open Transaction SECATT, enter the name of the test script you want to create or overwrite, and select eCATT OBJECT • OTHER FEATURES • UPLOAD WEB SERVICE TEST SCENARIO DATA.
- ▶ Upload of the test scenario and simultaneous generation of a new web service command interface. For this purpose, open Transaction SECATT of the wanted test script, and select the PATTERN button. The INSERT PATTERN selection screen is displayed. In this screen, select ENTERPRISE SERVICES as the group and WEB SERVICE (UPLOAD) as the command.
- ▶ Upload of the test scenario in an already existing command interface. For this purpose, open in transaction SECATT the wanted test script, and double-click the web service command interface that is supposed to be uploaded to the file. The command structure is displayed in the subscreen on the right. Then select the UPLOAD WS TEST SCENARIO option.
- ▶ Upload of a test scenario to create test configuration variant(s). For this purpose, open the test script in Transaction SECATT and double-click the web service command interface that is supposed to be uploaded

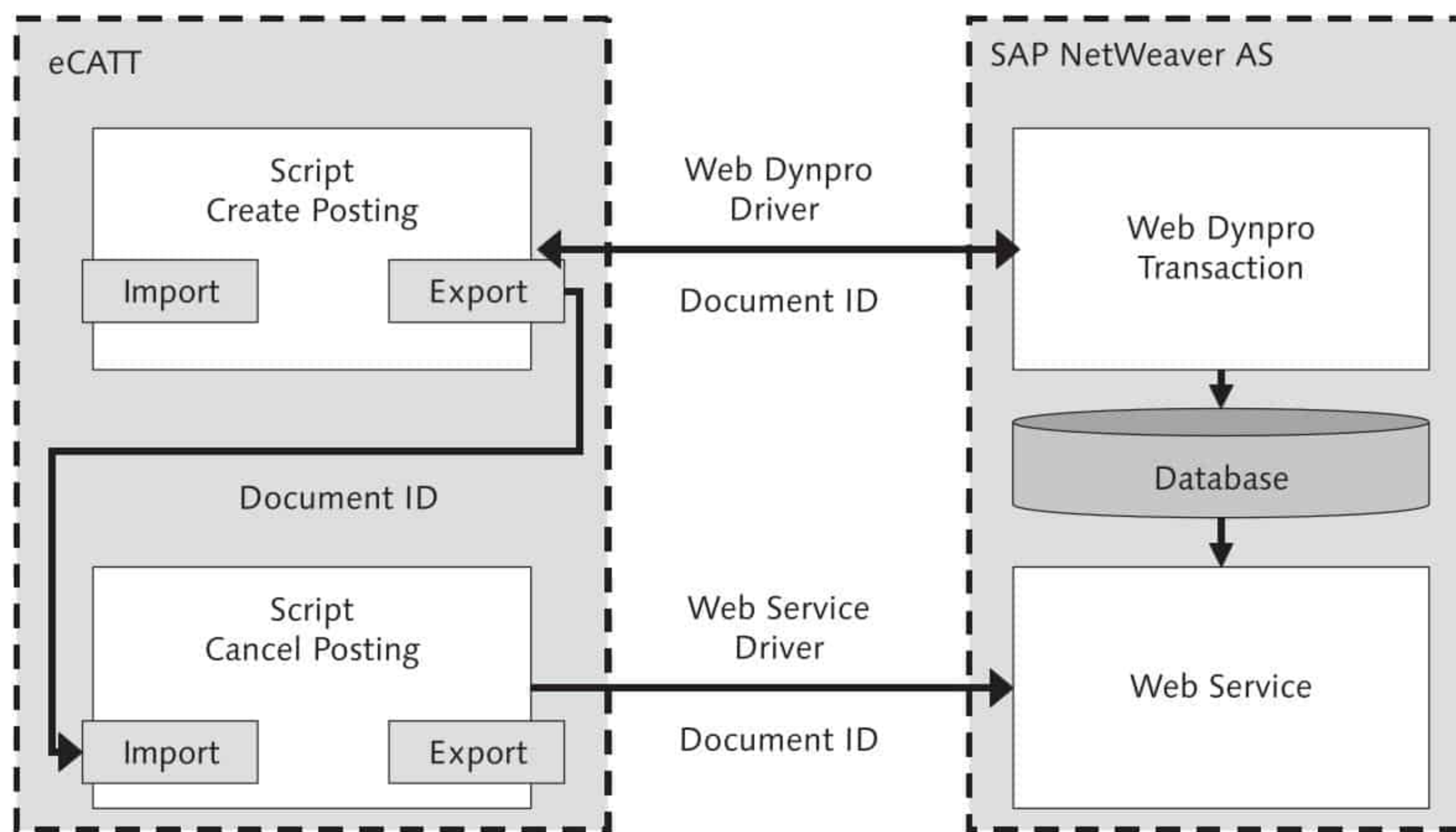
to the file. The command structure is displayed in the subscreen on the right. Then select the CREATE CONFIGURATION VARIANT option.

Except for the option to upload to an existing command interface, all upload variants are “capable of mass processing;” that is, multiple data records can be selected for processing.

After the upload process, the script editor displays a log including all originally selected settings and possible error messages. This enables you to reproduce the upload process.

#### Integration of Web Dynpros and web services

An interesting application scenario is created when web services are tested that are integrated with Web Dynpro-based transactions. For example, a data record can be entered via an automated Web Dynpro transaction to check the correct update of the data in the test system via an appropriate web-service call. This enables a continuous test of service-oriented solution landscapes.



**Figure 8.42** Example of a Test Scenario with Web Dynpros and Web Services

#### Conclusion

The web-services test is fully supported and integrated with eCATT as of SAP NetWeaver 7.0. Because ABAP proxy classes are generated

automatically from WSDL descriptions, addressing a web service is just as simple and comfortable as calling a function module. In combination with Web Dynpros, this results in very elegant possibilities for testing modern service-oriented solution landscapes via different forms of access.

## 8.5 Integration with External Test Tools

The driver for external test tools plays a special role among the eCATT test drivers. An external tool is a program of a third-party provider that uses the implementation of the BC-eCATT interface to interact with eCATT. SAP can certify the successful interaction. Under <http://www.sap.com/ecosystem/customers/directories/SearchSolution.epx>, you can find a list of the test tools certified by SAP for the interaction with eCATT. Then select BC-ECATT 6.2WIN in the first dropdown list.

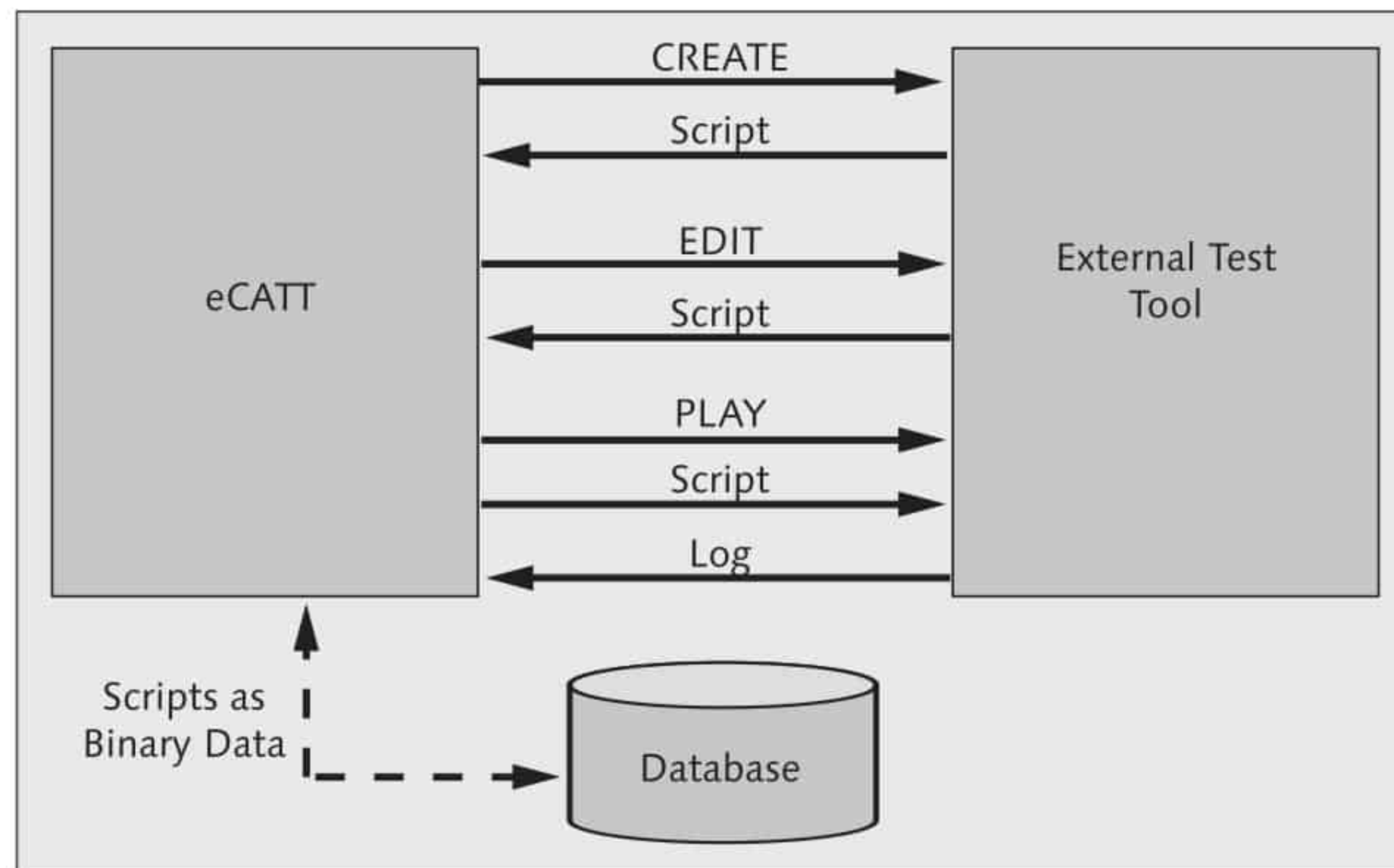
In heterogeneous solution landscapes, business processes can be effectively tested automatically with eCATT. The external test tool then covers those process steps that do not use a SAP GUI for Windows or Web Dynpro as a user interface.

Such an external test tool must be installed on the front end to be tested, and registered in the back end. The documentation of the external test tool contains information on the registration. In registration, the external program works as an adapter for the software to be tested. The work process—that is, the recording, editing, and processing—of user interactions is controlled via the external tool (see Figure 8.43). For this purpose, the tool is addressed by eCATT via the BC-eCATT interface. Recorded scripts are transferred via BC-eCATT and stored and managed in the central repository like native eCATT scripts. This ensures central, continuous, and consistent data storage even for the integration of external tools.

For recording, a distinction is made between two different processes. For the integrated recording scenario, eCATT is the driver that calls the external tool. In the standalone scenario, however, the recording is initially run in the external tool and then transferred to eCATT.

BC-eCATT

Recording



**Figure 8.43** Integrating an External Tool with eCATT

#### Integrated recording

The procedure for integrated recording is as follows.

1. When you create the script in Transaction SECATT, enter the desired test script name and select the external tool to be used.  
The system takes you to an editor window whose functions are highly reduced because the recording itself takes place in the external test tool.
2. Click the SCRIPT button to start the recording in the external tool.
3. After the interaction with the application to be tested is completed, the user stops the recording, defines transfer parameters if required, and selects the save function of the respective external test tool.
4. The recorded script is transferred via BC-eCATT and stored in the database.

#### Standalone recording

For standalone recording, the recording takes place in the external test tool independent of eCATT. If you want to transfer the data to eCATT after recording, then—depending on the implementation in the used external test tool—you must save the data in such a way that it can be transferred to eCATT. You then create the corresponding test script and a test configuration in eCATT.

The script is edited and maintained in the external tool. For this purpose, eCATT sends the script back to the external tool. After it has been edited, the script can be uploaded again.

Editing

During the execution, eCATT interacts with the external tool in the same way, except that in this case no further user interaction is required other than starting the eCATT test configuration. eCATT sends the script to the external tool. The tool is then started by eCATT, receives the possibly transferred parameter from eCATT, and executes the script. After the script has run, result values can be returned to eCATT in the form of parameters and can then be further processed. Additionally, the log of the external tool is created, transferred to the SAP system via BC-eCATT, and stored by eCATT in the database. In contrast to the script format, the log format is defined by the BC-eCATT standard. These logs can thus be fully integrated in the log of the eCATT test configuration and be displayed, searched, and analyzed just like the logs of native eCATT scripts.

Execution

Depending on whether this function was implemented in your external test tool, you have the option in eCATT to define a Uniform Naming Convention (UNC) path to the log in the external tool. If you've defined a UNC path and a script is executed, the complete log is stored in the external tool at the location referenced by the UNC path. Additionally, a node is added to the eCATT log that specifies the UNC path as a link. If you follow this link, the log is displayed in the log viewer of the external tool. This functionality is beneficial, for example, if you store screenshots in the external system, together with the log, which are not supposed to be transferred to eCATT for performance reasons.

UNC path to logs

The actual user interaction is not stored in the command interface when an external test driver is called. Instead, every external driver call has its own script that is marked as external. Analogous to the REF command that references to another test script, the external script is referenced via the REFEXT command.

Calling external test drivers

#### "REFEXT" Command

The REFEXT command serves for calling a test script recorded by an external tool:

```
REFEXT ( <external script>,
         <command interface>,
         <version> ).
```

Start options for the external driver	<p>When an external driver is implemented, you have the following start options.</p> <ul style="list-style-type: none"> <li>▶ <b>Normal</b> The external tool is not visible during the execution. Only the test execution, by interacting with the application to be tested, can be watched on screen.</li> <li>▶ <b>Debug mode</b> The debugger of the external tool is started, and the test script stops and remains in debug mode before the first statement. Depending on the capabilities of the implemented external tool, the test script can now be executed and analyzed step-by-step in the debugger.</li> <li>▶ <b>With the interface of the external tool</b> The external tool is visible during the test execution. The actual test execution is the same as for the NORMAL start option.</li> </ul>
Start options and start profile	<p>If the external test tool has its own authorization concept, you can store a user name and a password in the start options. These are then used to log on to the external tool. However, because you may forget to set these entries in the start options, a storage alternative is available: the start profile. If you then run a test from the Test Workbench, specify which test configuration is supposed to be executed with which start profile. A fast way to create a start profile is to copy the current settings in the start page (START OPTIONS • SAVE START PROFILE).</p>
"SENDEXT" command	<p>Let's assume that your script contains a REFEXT command with a DO loop, and the external script is supposed to be run 100 times. Normally, the REFEXT command would send the script to the external tool for every single execution. To avoid this unnecessary data transfer, the REFEXT command interface has the DO NOT SEND option. When you select this option, a SENDEXT command is put at the start so that the data is transferred only once at the beginning.</p>
External tools and process chains	<p>Let us now look at some implementation scenarios for the external tool driver. A typical case is the integration of a web application with an SAP system, such as the implementation of a third-party web shop as a part of a front end for an ERP system.</p> <p>The key aspect from the tester's point of view is that the browser-based application of the web shop is not based on a user interface directly sup-</p>

ported by eCATT. It is out of reach for the test drivers presented so far. Because most relevant business processes also affect the web shop as an integral component, it isn't useful to exclude it from the test.

Because complete business processes are to be tested end-to-end in the context of an integration test, a separate test of the web shop is not very useful either. You need to completely and automatically test the integration with the other components of the solution landscape as well. The objective of implementing the external test tool is to close the gap in the automated test chain.

One scenario assumes that we are dealing with an SAP-dominated system landscape with few foreign components. In this situation, an eCATT-based test using an external test tool is recommended. A major part of the testing then benefits from the deep integration of eCATT with the SAP applications; the gaps can then be elegantly closed by the external driver.

If a system landscape is dominated by systems of one or several third parties, SAP applications might only be used sporadically. The advantages of an eCATT-based automation solution seem less striking. Before making a decision, however, we should weigh the integration possibilities of SAP Solution Manager. In the sense of a holistic application management, it enables you to integrate steps of non-SAP applications with the Business Process Repository (BPR) and to build the test organization across systems in a process-oriented way. In addition to this organizational advantage, we should consider the chance to minimize the total cost of operation for the automation solution. For the non-integrated implementation of external test tools, in addition to the actual tool costs, there are usually costs for the infrastructure required by the tool, that is, databases, file services and their license and administration costs. With integration in eCATT, this cost can be avoided almost entirely. Even in this situation, you should always check the alternative of eCATT-based test automation.

In the SAP Developer Network (SDN) on the certification page of BC-eCATT, you can obtain information on the BC-eCATT interface as well as the test tools that can be integrated. This page is available via the following link: <https://www.sdn.sap.com/irj/sdn/interface-certifications>.

SAP-dominated  
solution landscape

Heterogeneous  
system landscape  
with SAP share

Test tools that can  
be integrated

### Conclusion

eCATT treats third-party programs implementing the BC-eCATT interface as external test tools. These external programs are addressed like native eCATT test drivers; however, they do not store the recorded information in command interfaces but in specific scripts. The external scripts themselves can be called from eCATT using the `REFEXT` command. For the standalone recording variant, the test scripts are a direct part of the test configuration. The scripts, their test data, and the execution logs are all managed within the SAP system.

## 8.6 Implementing Checks

An eCATT script typically consists of two logical blocks: the call of the test driver, and the subsequent checking of the results. The possibilities of recording, parameterizing, and revising commands for the various test drivers have been presented in the previous sections. We now look at the actual checking process: the comparison of the result returned by the driver and an expected value.

An important consideration is that scripts do not necessarily fail just because the recorded transaction returns an error. Conversely, a script is not necessarily successful just because the transaction returns a value. To clarify these statements, let's look at the following example:

In a transaction, a passenger is to be booked on a flight. The flight is determined by a flight number and a date. If a reservation on the specified flight is possible, it should be entered in the database, and a reservation number should be returned. The most obvious case is that a flight number and a date are entered and a reservation number is returned. However, there are a number of other possible outcomes.

- Unexpected errors
1. The desired flight exists, and there is a seat available. In this case, the reservation is to be completed and a reservation number should be returned.

If an error is still reported by the software, there might be a programming or customizing error. If this error is detected during the test, we refer to an unexpected error.

# Index

## A

---

ABAP, 370  
ABAP exceptions, 647  
ABAP Object classes  
    *global*, 457  
ABAP OO driver, 418  
ABAP proxy object, 462  
ABAP runtime error, 647  
Accelerated SAP, 77, 80, 86  
Accelerator, 88  
Acceptance test, 51  
Action plan, 591  
Adapter, 279  
AGS\_WORKCENTER, 138  
allow (message), 476  
Anonymization, 540  
Anonymization of test data, 535  
Application lifecycle, 44  
Application Lifecycle Management (ALM), 125, 129, 686  
Application Management Lifecycle, 131  
Application server, performance analysis, 632  
Archiving, 507  
Archiving flag, 510  
ARIS, 670, 671  
ARIS for SAP NetWeaver, 141  
ARIS Synchronization, 164  
ASAP, 77, 80, 86  
ASAP Roadmap, 52, 63, 77, 78, 651  
    *phases*, 80  
    *upgrade*, 137  
Attribute, custom, 161  
Authorizations, 106  
Automated testing  
Automation, 382, 386  
    *advantages*, 361  
    *end-to-end*, 679  
    *error management*, 682  
    *prerequisites*, 364  
    *Test Center*, 657  
Automation team, 115

## B

---

Background processes, 581  
Backup & recovery tests, 682  
BAPs, 458  
BC-eCATT, 465  
BCSET command, 502  
Black-box, 56  
Borland SilkPerformer, 681  
Bottom-up strategy, 53  
Boundary value analysis, 60  
Breakpoint, 516  
Buffer, 640  
Build and Test phase, 133  
Business Blueprint (project phase), 81, 95  
Business Components, module, 317, 351  
Business Process Change Analyzer, 80, 112, 120, 139, 170, 172, 176, 211, 227  
Business process, granularity, 162  
Business Process Library (BPL), 536, 546  
    *business contexts*, 537, 538  
Business process monitoring, 135, 670, 685  
Business Process Repository (BPR), 77, 162, 232, 469  
Business process structure, mapping, 95  
Business Process Testing, 118, 275, 315, 337, 339  
BW reporting, 186, 191, 192, 676

## C

---

Capability Maturity Model Integration (CMMI), 323  
Capture and replay, 69  
CATT, 496  
    *migration*, 265  
CCMS, 684  
CCMS performance monitors, 628  
Change analysis, 69

- Change impact analysis, 175
  - Change Management, 379
    - Test Center*, 657
  - Change Request Management, 119, 130, 136, 143, 198
  - Changes, 45
    - business process-driven*, 45
    - IT-driven*, 45
    - to business processes*, 45
  - Changes to infrastructure, 46
  - Check logic, 374
  - Checks, 470
    - semantic*, 374
  - CHEGUI command, 440, 444, 473, 520
  - CHETAB command, 460, 482
  - CHEVAR command, 471, 481
  - Client, accelerate, 564
  - Client copy, 561
  - Client maintenance, 422
  - Client server layer, 631
  - Collaboration platform, 133
  - Command (eCATT)
    - CHETAB*, 526
    - CHEVAR*, 526
  - Company mergers, 45
  - Component-based test case creation, 372
  - Component, logical, 151
  - Component test, 51
  - Configuration editor, 503
  - Connection ID, 439
  - Consolidate, 354
  - Cookies, 600
  - Cost model, 363, 377
    - software errors*, 378
  - CPU time, 639
  - CPU utilization, 646
  - CREATEOBJ command, 458
  - Cross-client customizing, 423
  - Custom Development Management Cockpit, 137
- D**
- 
- Data analysis, 581
  - Database accesses, 460
  - Database analysis, 630
  - Database overview, 642
  - Database response time, 631
  - Database server, performance analysis, 631
  - Database time, 639
  - Data Dictionary, 547
  - Data privacy, 533
  - Data protection, 107
  - Data reduction, 535
  - DBA-Cockpit, 204
  - Debugger, 72, 516
  - Defect Management, 68, 292, 319, 332
    - Test Center*, 657
  - Defect management process, 100
  - Degree of test coverage, 103
  - DELSTORE command, 501
  - Deploy phase, 133
  - Developer test, 53, 200
  - Development activities, 49
  - Development-oriented testing, 72
  - DO command, 497
  - Documentation of test types, 90
  - Documentation tools, 70
  - Document management, 695
  - Dummy system, 157
  - Dynpro, optional, 447
  - Dynpro simulator, 431
- E**
- 
- E2E alerting, 205
  - E2E change analysis, 204
  - E2E monitoring, 205
  - E2E trace, 205
  - E2E trace analysis, 685
  - EarlyWatch Alert, 135
  - eCATT, 92, 114, 223, 258, 328, 362, 366, 368, 387, 411, 670
    - ATTACH command*, 445
    - BCSET command*, 502
    - CHEGUI command*, 440
    - CHETAB command*, 460, 483
    - CHEVAR command*, 471
    - client maintenance*, 422
    - DELSTORE command*, 501
    - DO command*, 497, 498

- driver for external tools*, 465
  - encapsulation*, 415
  - EXIT command*, 499
  - FILL\_RDC command*, 491
  - FUN command*, 459
  - GETGUI command*, 440
  - GETTAB command*, 460, 482
  - IF command*, 497
  - implementation*, 262
  - JOIN command*, 446
  - kinds of parameterization*, 431
  - LOG command*, 500
  - logging*, 507
  - LOGMSG command*, 481
  - LOGTEXT command*, 500
  - MESSAGE command*, 262, 475
  - modularity*, 415
  - parameter*, 420
  - PERF command*, 514
  - REFCATT command*, 496
  - REF command*, 493
  - REFEXT command*, 467
  - RESCON command*, 501
  - RETRIEVE command*, 501
  - SAPGUI command*, 434
  - script editor*, 426
  - SENDEXT command*, 468
  - SPLIT command*, 446
  - STORE command*, 501
  - system data container*, 415
  - TCD command*, 428
  - TCD tests*, 426
  - test configuration*, 415
  - test data generation*, 62
  - test driver*, 416
  - test objects*, 414
  - test script*, 118, 414, 418
  - test script commands*, 420
  - versioning*, 506
  - versions*, 519
  - Web Dynpro*, 449
  - WEBDYNPRO command*, 452
  - WEBSERVICE command*, 461
  - Effort estimation, 64
  - Email notification, 221
  - Encapsulation, 415
  - End-to-end automation, 679
  - End-to-end solution operations, 79, 129
  - End-to-end test cases, 667, 679
  - End User Experience Monitoring, 206
  - Equivalence class partitioning, 58
  - Error analysis, 515
  - Error correction, costs, 380
  - Error costs, 379
  - Error guessing, 61
  - Error management, 675
  - Error message, 221, 292, 481
  - Error recording, 184
  - Errors, 470
  - Execution control, 509
  - Execution log, 628
  - Execution time, 639
  - Executive summary, 590
  - EXIT command, 499
  - exit (message), 477
  - Expected errors, 471
  - expected (message), 476
  - Expert mode, SAP TDMS, 540
  - Export parameter, 419
  - Extended memory, 642
  - External test tools, 465
- ## F
- 
- fail (message), 477
  - FILL\_RDC command, 419
  - Filter, 475
  - Final Preparation (project phase), 83, 109
  - Flash-copy, 553
  - Full copy, 532
  - FUN command, 459
  - Functional test, 48, 525
  - Functional upgrades, 45
  - Function modules, 458, 527
  - FUN driver, 418
- ## G
- 
- General V-model, 49
  - GETGUI command, 440, 444, 474, 520
  - GETTAB command, 460, 482

Global ABAP object classes, 457  
 Global work process overview, 638  
 Go-Live and Support (project phase),  
 84, 119

## H

---

hardware utilization, 629  
 Heterogeneous System Landscape, 129  
 History, 185  
 Hit ratio, 641  
 HP Quality Center, 332  
 HP Quality Center Dashboard, 325  
 HP QuickTest Professional, 92, 116, 275  
   *SAP TAO*, 341

## I

---

IBM WebSphere, 684  
 ICT Lifecycle Management, 323  
 IEE 829, 90  
 IEEE 829, 64  
 IF command, 447, 497  
 IMG project, 157  
 Implementation project, 144  
 Implementation/Upgrade, work center,  
 80, 85  
 Import parameters, 419  
 Incident Management, 130, 379  
 Index, missing, 587  
 Initialize Script, 353  
 Initial state recording, 442  
 Inline ABAP, 370, 459  
 Input data combinations, 57  
 Instance attributes, 458  
 Integrated recording, 466  
 Integration test, 51, 54, 201, 271  
 Interfaces, 527  
 International Software Testing  
 Qualifications Board (ISTQB), 43, 651  
 ISO/IEC 25000, 47  
 ISTQB, 43, 651  
 ITIL, 88, 379, 651  
 IT Infrastructure Library (ITIL), 88, 379,  
 651

## J

---

JOIN command, 446  
 Joint process test, 255

## K

---

Kaizen, 110  
 Key accounter, 671  
 Key user concept, 389  
 Knowledge Warehouse, 695, 697

## L

---

Laufzeitdatencontainer, 491  
 License costs, 244  
 License fees, 364  
 List of all users logged on, 634  
 Load creation, 596, 601  
 Load injector  
   *performance analysis*, 630  
 Load profile, 576, 577, 583, 612, 619  
   *documentation*, 579  
   *forecast*, 581  
 LoadRunner Agent, 600  
 LoadRunner Analysis, 602  
 LoadRunner Controller, 600  
 LoadRunner Virtual User Generator, 597  
 Load test, 569, 571, 575, 584, 598, 610,  
 682  
   *execution*, 612  
   *production system*, 624  
 Local variables, 419  
 Lock, 637  
 Lock entry list, 636, 637  
 Locks, forgotten, 587  
 Log, 503  
 Log archiving, 510  
 LOG command, 500  
 Logical component, 150  
 Logical operator, 473  
 Log ID, 512  
 LOGTEXT command, 500  
 LOOP command, 498, 520  
 Lotus Notes, 364

**M**

---

Maintenance, 44  
 Maintenance fees, 364  
 Maintenance Optimizer, 137  
 Maintenance project, 144  
 Mass data test, 569, 572  
 Master script, 389  
 Mercury, 276  
 Message  
   *determine*, 479  
   *parameterize handling*, 480  
 Message class, 475  
 MESSAGE command, 262  
 Message filter, 475  
   *create*, 478  
 Message handling, 474  
 Message mode, 476  
 Message rule, 477  
 Message type, 475  
 Migration Workbench, 545  
 Milestone, 158  
 Modes per user, 636  
 Modularity, 415  
 Modularization, 370, 388  
 Module test, 54, 200  
 Monitoring, 134, 675  
 Monitor, performance, 627  
 Multi-user test, 587, 628

**N**

---

Negative test, 471, 480  
 Network diagnosis, 633

**O**

---

Object-based test case creation, 70  
 Offshore, 407  
 Operate phase, 134  
 Operating system analysis, 629  
 Optimize phase, 136  
 Organizational unit, 158  
 OS monitor, 645  
 OTA API, 333

Outsourcing, 651, 660  
 Outtasking, 654  
 Overview of SAP buffers, 640

**P**

---

Parameterization, LoadRunner, 598  
 Peak load, 578  
 PERF command, 514  
 Performance analysis, 627  
   *automated*, 511  
   *global*, 512  
   *localization*, 630  
 Performance test, 48, 55, 71, 367, 569, 620  
   *analysis*, 602  
   *browser-based application*, 599  
   *budget*, 574  
   *completion*, 590  
   *data analysis*, 581  
   *data preparation*, 585  
   *documentation*, 592  
   *implementation period*, 574  
   *load profile*, 583  
   *monitoring*, 627  
   *multi-user test*, 587  
   *optimization*, 588  
   *perform*, 584  
   *planning*, 575  
   *process analysis*, 576  
   *result analysis*, 588  
   *SAP LoadRunner by HP*, 595  
   *single-user test*, 586  
   *Test Center*, 658  
   *tool*, 583  
 Performance test project  
   *phases*, 574  
   *roles*, 572  
 Personal data, 533  
 Personnel administration, data selection, 538  
 Pilot project, 74  
 PMI PMBOK, 88  
 Popup window, 447  
 Portal, 617  
 Pretty Printer, 520

- Problem Management, 379
  - Problem message, 89, 237, 247
  - Process analysis, 576
    - result*, 580
  - Process documentation, 398, 399
  - Processing time, 631
  - Process structure
    - create*, 161
    - creation*, 282
  - Production data, 62
  - Production system copy, 532
  - Program analysis, 630
  - Program buffer, 641
  - Project, 143
    - create*, 154
    - creation*, 282
  - Project documentation, 96
  - Project Preparation (project phase), 80, 90
  - Project Standards, 158
  - Project types, 144
  - Proof of concept, 74
  - Protocol, SAP LoadRunner, 596
- Q**
- 
- QALoad, 621
  - Quality Gate Management, 134, 201
  - Quality objectives, 90
- R**
- 
- Random tests, 104
  - Rational Unified Process, 63
  - RDC, 491
  - Realization (project phase), 82, 101
  - Real-time monitoring, 628, 629
  - REFEXT command, 467
  - Refresh, 546, 549, 565
  - Refreshing the test system, 549
  - Regression test, 55, 111, 238, 258, 382
    - automation*, 113
    - risk analysis*, 112
    - scope*, 111
  - Regular correction, 200
  - Relational operators, 472
  - Release Management, 379
  - Reporting, 66, 68, 92, 109, 238, 319, 400
  - Requirement module, 288
  - Requirements and design phase, 133
  - Requirements-based testing, 97
  - require (message), 476
  - Re-Recording, 432
  - RESCON command, 501
  - ResetGUI, 438
  - Resource planning, 64
  - RETRIEVE command, 501
  - Retroactive analysis, 628
  - Reusability, scripts, 118
  - Reutilization, 370
  - Revision security, 130
  - RFC connection, trusted, 422
  - Risk analysis, 64, 112
  - Roadmaps, 77, 85
  - Roadmap select, 155
  - ROI analysis, 361, 363
  - Root cause analysis, 685
  - Rule, 475
  - Run (project phase), 85, 121
  - Run SAP, 127, 130, 651
  - Run SAP methodology, 63
  - Run SAP Roadmap, 79
    - test management*, 91
  - Runtime data container, 419
  - Runtime Data Container (RDC), 491, 520
- S**
- 
- Sandbox system, 562
  - SAP Active Global Support, 127, 135
  - SAP Best Practices, 590
  - SAP buffer, 640
  - SAP Central Process Scheduling by Redwood, 143
  - SAP (Click and Script), protocol, 597
  - SAP Consulting
    - offering*, 94
    - Test Center*, 655
  - SAP CRM, 252, 665

- SAP Developer Network (SDN), 702
- SAP Enterprise Support, 127, 128, 131
- SAP for Retail, 261
- SAP GoingLive Check, 623
- SAPGUI (Attach) command, 445
- SAPGUI command, 473, 519
  - granularity levels*, 436
- SAPGUI driver, 417
  - events*, 435
- SAPGUI, protocol, 596, 597
- SAP GUI scripting, 423
- SAP Help Portal, 701
- SAP Knowledge Warehouse, 153, 165, 330
- SAP LoadRunner by HP, 595, 611, 620, 628
  - components*, 596
- SAP MaxAttention, 127
- SAP NetWeaver Business Client (NWBC), 138
- SAP NetWeaver Business Warehouse, 270, 610, 665
- SAP NetWeaver BW 7.0, 252
- SAP NetWeaver Portal, 252, 617
  - performance test*, 599
- SAP Notes, 45
- SAP product, 150
- SAP ProductivityPak by RWD, 114, 143, 402
- SAP Quality Center by HP, 92, 97, 102, 108, 126, 143, 275, 276, 312, 325
  - and Service Desk*, 292
  - creating a test case*, 289
  - data synchronization*, 285
  - project and SAP Solution Manager*, 285
  - risk analysis*, 288
  - SAP TAO*, 340
  - structure*, 278
  - test management*, 282
  - test planning*, 290
  - test scripts*, 118
  - transfer test results to SAP Solution Manager*, 308
- SAP Quick Sizer, 578
- SAP Reference IMG, 148
  - Test management*, 160
- SAP Safeguarding, 128
- SAP SEM, 610
- SAP Service Desk, 246, 254, 264, 330
- SAP Service Marketplace, 702
- SAP Solution Manager, 125, 130, 399
  - adapter*, 141
  - Adapter for SAP Quality Center by HP*, 102, 279
  - and application management*, 132
  - and ARIS*, 141
  - and eCATT*, 142
  - and SAP Quality Center by HP*, 275
  - Change Request Management*, 198
  - configuration*, 147
  - data export*, 332
  - Diagnostics*, 136, 203
  - Enterprise Edition*, 131
  - execute test scripts*, 504
  - integration scenarios*, 193
  - project*, 143
  - project and SAP Quality Center by HP*, 285
  - project-related testing*, 152
  - RFC connections*, 149
  - roadmaps*, 77, 78, 85
  - root cause analysis*, 685
  - SAP TAO*, 340
  - SAP TDMS*, 543
  - Service Desk*, 194, 246, 254
  - solution*, 143
  - solution-related testing*, 209
  - test case creation*, 101
  - test case description*, 102
  - Test Center*, 669
  - test evaluation*, 308
  - test scripts*, 118
  - Test Workbench*, 673
  - Work Center*, 137
- SAP Solution Manager Adapter for SAP Quality Center by HP, 211, 315, 329, 331
  - SAP TAO*, 340
- SAP Standards for Solution Operations, 127, 130
- SAP TAO, 92, 119, 321, 337, 367
  - components*, 340
  - create test case*, 343

- test case maintenance, 356*
  - SAP TDMS, 107, 531
    - and SAP Solution Manager, 543*
    - architecture, 543*
    - data reduction, 538*
    - data selection, 538*
    - for CRM, 542*
    - for HCM, 532, 538, 560*
    - for SAP NetWeaver BW, 542*
    - hardware, 544*
    - performance, 563*
    - process types, 535*
  - SAP Test Management Consulting, 610, 622, 687
  - SAP Test Workbench, 92, 107, 126, 239, 247, 262, 327
  - SAP Tutor, 389
  - SAP - Web, protocol, 596
  - Sarbanes-Oxley Act (SOX), 669
  - Scenario testing, 54
  - Scrambling rules, 540
  - Screenshot, 448
  - Script creation, costs, 369
  - Script developer, performance test, 573
  - Script editor, 426, 503
  - Script recording, LoadRunner, 597
  - Scripts, reusability, 118
  - Semantic checks, 374
  - SENDEXT command, 468
  - Service Desk, 141, 194, 292, 380, 401
  - Service Level Agreements, 570
  - Service-Oriented Architecture (SOA), 651, 663
  - Session ID, 439, 600
  - Shell creation, 536, 542, 548
  - Shortdump, 647
  - Signature, digital, 695, 699
  - SilkPerformer, 681
  - Single-user test, 569, 586, 628
  - Six Sigma, 110
  - SOA, 664
    - automation, 679*
  - SOAP, 461
  - Software error, costs, 378
  - Solution, 143
  - Solution Documentation Assistant, 103, 163, 227
  - Solution landscape, 147
  - Solution-related testing, 209
  - SPLIT-command, 446
  - SPRO, 159
  - SQL statement, poor, 587
  - SQL trace, 629
  - SQL trace analysis, 643
  - Standalone recording, 466
  - Static attributes, 458
  - Status analysis, 109, 186, 239, 302
  - Status information system, 400
  - Status overview, 188
  - Status schema, 178, 228
  - ST-ICO, 133
  - Stress test, 569, 575, 589
  - Structural data selection
    - SAP TDMS, 539*
  - Structure editor, 426
  - Support Packages, 44, 45, 119
    - automated testing, 371*
  - Sustainability, 251
  - Swap memory, 646
  - Swaps, 641
  - Synthetic test data, 62, 525
  - System changes, 43
  - System data container, 415
  - System failure, 570
  - System landscape, 156, 592
    - mapping, 151*
  - System Landscape Directory, 148
  - System load, 601
  - System log, 637
  - System role, 151
  - System shell, 548
  - System test, 51, 54
  - System trace, 112
- T**
- 
- Tables, check, 482
  - TCD command, 473
  - TCD driver, 328, 366, 416, 427
    - re-recording, 432*
    - start options, 433*
  - TDC-API, 492
    - exceptions, 492*

- Technical upgrades, 46
- Template, 158
- Template project, 144
- Test administration system, central, 412
- Test analysis, 140
- Test automation, 69, 113, 130, 142, 223, 256, 272, 328, 367, 387
  - costs*, 338, 369
  - goals*, 370
  - implementation*, 373
  - process flows*, 116
  - reusing scripts*, 117
  - Test Center*, 657
- Test case
  - create*, 289
  - insert*, 164
  - maintenance*, 375
  - maintenance costs*, 370
  - modification*, 46
  - parameterize*, 353
- Test case attribute, 99, 166
- Test case creation, 65, 70, 101, 671
  - component-based*, 367, 372
- Test case description, 98, 102, 159, 363
  - automated*, 167
  - manual*, 165
  - Test Center*, 657
- Test case design, 57
  - costs*, 363
- Test case document, 672
- Test case documentation, 364
- Test case sequence, 108, 228, 253, 271
- Test case template, 98
- Test catalog, 693
- Test Center, 651, 653
  - implementation*, 654
  - services*, 656
  - setup*, 666
- Test component
  - create*, 316
- Test concept, 63, 90
- Test configuration, 415
- Test coordinator, 671
- Test cost model, 377
- Test coverage, 412
  - SAP Quality Center by HP*, 286
- Test cycle
  - costs*, 372
  - implement*, 376
- Test data, 62, 106, 129, 414
  - anonymization*, 535
  - create*, 107
  - data privacy*, 533
  - manage*, 483
  - quantity*, 582
  - requirements*, 531
  - sensitive*, 531
  - TDMS*, 531
  - tools*, 71
  - verify*, 582
- Test data container, 223, 419, 485, 520
  - central*, 486
  - create*, 487
  - parameter*, 487
  - programming interface*, 492
  - separated by contents*, 486
  - variant maintenance*, 489
- Test documentation, 183, 329, 366
- Test driver, 416
- Test effort, 245
- Test end criteria, 100
- Tester handout, 108
- Testers, 105
  - assign*, 170
- Tester worklist, 140, 182
- Test evaluation, 66
- Test execution, 108, 182
  - documentation*, 592
- Test Factory, 653
- Test focus, 114, 129
- Test goals
  - description*, 592
  - performance test*, 571
- Testing, 401, 674
  - cost model*, 363
  - effort and benefits*, 361
- Testing by Composition, 372
- Testing Maturity Model (TMM), 110
- Testing Option 1, 152, 283
- Testing Option 2, 275, 331, 340
- Testing process, review, 109
- Test Lab, 653
- Test lead, performance test, 572

- Test log, 371
- Test management, 67, 130, 398, 400, 657
  - accelerators*, 90
  - review*, 223
  - SAP Quality Center by HP*, 282
- Test management cockpit, 677
- Test management system, central, 422
- Test management tools, 67
- Test management, work center, 80
- Test methodology, 69
- Test note, 160, 183
- Test notice
  - configuration*, 185
- Test object, 167, 414
- Test organization, 315
  - central*, 653
- Test Organizer, 263, 511, 693
  - archiving flag*, 510
- Test package, 170
- Test plan, 220
  - create*, 167
  - generate*, 170
  - SAP Quality Center by HP*, 290
  - status analysis*, 189
- Test plan attribute, 168
- Test plan management, 140, 167, 510
- Test Plan, module, 351
- Test planning, 356, 673
- Test planning and execution, 65, 68
- Test preparation, 63, 139
- Test process
  - optimization*, 252
  - standardization*, 258
- Test report, 186
- Test report, create, 187
- Test reporting, 100, 401, 669, 675
- Test script, 414, 418
  - commands*, 420
  - consolidate*, 354
  - create*, 586
  - error analysis*, 515
  - maintenance*, 338
  - modularization*, 388, 420
  - modularize*, 493
  - run*, 503
  - sequence*, 494
  - single*, 493
  - top script*, 494
- Test sequence, 171, 172
  - create*, 180
  - workflow*, 180
- Test series, 161
- Test set
  - automated*, 371
  - regular execution*, 382
- Test specification, 70
- Test stages, 49, 51
  - in SAP projects*, 53
- Test standards, 98, 115, 159
- Test strategy, 64
- Test structure, description, 591
- Test system, 105, 552
  - creation*, 547, 560
- Test tool options, 92
- Test tools, 91
  - automation*, 69
  - change analysis*, 69
  - costs*, 364
  - development-oriented testing*, 72
  - performance testing*, 71
  - piloting*, 74
  - selection*, 66, 368
  - test case creation*, 70
  - Test Center*, 657
  - test data generation*, 71
  - test management*, 67
  - test specification*, 70
- Test types, 47
- Test Workbench, 193, 289, 670, 673
  - archiving flag*, 510
  - reporting*, 676
- Think time, 598
- Time-out dump, 647
- Tool costs, 373
- Tool selection, 72, 366
  - Testing by Composition*, 373
- Tool specification, 73
- Top script, 494
- Traceability, 130
- Trace analysis, 643, 685
- Trace list, 644
- Training concept, 222
- Transaction

- AI\_SPS, 152  
 AL08, 629, 634  
 DNO\_CUST04, 197  
 ME21N, 98, 173  
 RMMAIN, 85  
 SARA, 510  
 SE80, 456  
 SECATT, 426  
 SM04, 629, 635  
 SM12, 629, 633, 636  
 SM21, 629, 637  
 SM50, 638  
 SM51, 634  
 SM59, 451  
 SM66, 629, 631  
 SMSY, 150  
 SOLAR01, 173  
 SOLAR02, 173  
 SOLAR\_EVAL, 308  
 SOLAR\_PROJECT\_ADMIN, 152, 284, 699  
 SOLMAN\_SETUP, 146  
 SOLMAN\_WORKCENTER, 137  
 SPPFCADM, 179  
 SPRO, 147  
 ST02, 634, 640  
 ST04, 629, 631, 642  
 ST05, 629, 643  
 ST06, 629, 631, 645  
 ST22, 629, 647  
 ST30, 512, 513  
 STAD, 629, 631, 639  
 STWB\_2, 140, 510  
 STWB\_INFO, 140, 511  
 STWB\_SET, 140, 159  
 SU3, 424  
 Transaction analysis, 639  
 Troubleshooting, 382  
 Trusted RFC connection, 149, 422
- U**
- 
- UDDI, 461  
 UI Scanner, 350  
 Unexpected errors, 470  
 Unit testing, 54  
 Upgrade, 385  
 Upgrade project, 144  
 Urgent correction, 200  
 User acceptance, 570  
 User acceptance test, 55, 623  
 User list, 635
- V**
- 
- Validation system, 225  
 Variant  
   *assistant*, 490  
   *create*, 488  
   *external*, 489  
   *manual*, 485  
   *with test data container*, 490  
 Variant wizard, 520  
 Versioning, 505  
 Versioning editor, 507  
 Visibility, 430  
 V-model, 49, 667  
   *SAP version*, 52  
 V-model XT, 63  
 Volume test, 569, 572
- W**
- 
- Waterfall model, 49  
 Web (Click and Script), protocol, 597  
 Web Dynpro, 449  
 Web Dynpro application, 449  
   *and web services*, 464  
 WEBDYNPRO (attach) command, 456  
 WEBDYNPRO command, 452, 454  
   *page simulator*, 455  
 Web Dynpro driver, 417  
   *start options*, 455  
 Web (HTTP/HTML), protocol, 596  
 WEB SERVICE command, 461  
 Web service driver, 417, 461  
 Web services, 461, 663  
 WebSphere, 684  
 White-box, 57  
 Work center, 137, 227, 253, 258  
   *assistant for solution documentation*,

## Index

164  
*Change Management*, 199, 201  
*Implementation/Upgrade*, 152  
*MyHome*, 138  
*Root Cause Analysis*, 203  
*structure*, 138  
*Test Management*, 138, 168, 182,  
186  
*Test Management*, 153

*Workflow*, 172, 177, 179  
*Work process overview*, global, 638  
WSDL, 461, 462

## **X**

---

XML, 461