

# CATT: Enhanced Mode (BC-CAT-TOL)



HELP.BCCATTOL

Release 4.6C



## Copyright

© Copyright 2001 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft<sup>®</sup>, WINDOWS<sup>®</sup>, NT<sup>®</sup>, EXCEL<sup>®</sup>, Word<sup>®</sup>, PowerPoint<sup>®</sup> and SQL Server<sup>®</sup> are registered trademarks of Microsoft Corporation.

IBM<sup>®</sup>, DB2<sup>®</sup>, OS/2<sup>®</sup>, DB2/6000<sup>®</sup>, Parallel Sysplex<sup>®</sup>, MVS/ESA<sup>®</sup>, RS/6000<sup>®</sup>, AIX<sup>®</sup>, S/390<sup>®</sup>, AS/400<sup>®</sup>, OS/390<sup>®</sup>, and OS/400<sup>®</sup> are registered trademarks of IBM Corporation.

ORACLE<sup>®</sup> is a registered trademark of ORACLE Corporation.

INFORMIX<sup>®</sup>-OnLine for SAP and Informix<sup>®</sup> Dynamic Server<sup>™</sup> are registered trademarks of Informix Software Incorporated.

UNIX<sup>®</sup>, X/Open<sup>®</sup>, OSF/1<sup>®</sup>, and Motif<sup>®</sup> are registered trademarks of the Open Group.

HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C<sup>®</sup>, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA<sup>®</sup> is a registered trademark of Sun Microsystems, Inc.

JAVASCRIPT<sup>®</sup> is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

## Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

## Contents

<b>CATT: Enhanced Mode (BC-CAT-TOL)</b> .....	<b>7</b>
<b>General information</b> .....	<b>8</b>
<b>New functions in Release 4.0, 4.5 and 4.6</b> .....	<b>9</b>
<b>Modular test cases</b> .....	<b>12</b>
<b>Plan tests</b> .....	<b>13</b>
<b>Test phases</b> .....	<b>14</b>
<b>CATT Management</b> .....	<b>14</b>
<b>Authorization</b> .....	<b>15</b>
<b>Original language handling</b> .....	<b>16</b>
<b>CATT in the Repository Information System</b> .....	<b>16</b>
<b>Naming convention</b> .....	<b>16</b>
<b>Test case</b> .....	<b>17</b>
<b>Test Case Types</b> .....	<b>18</b>
Use of Automatic Test Cases.....	18
Use of Manual Test Cases.....	19
Use of Referring Test Cases.....	19
Using CATT for R/2 test cases.....	20
R/2: Create Simple Test Case .....	21
R/2: Create Transaction-Independent Test Case.....	22
Using External Applications .....	22
Use External Applications.....	23
Run External Application.....	25
Using an External Test Tool.....	25
Setup external test tool interface .....	26
Call External Test Tool From CATT.....	26
Edit External Test Tool Test Script .....	27
Parameter interface between CATT and external test tool.....	27
Using function module tests .....	28
Create Function Module Test .....	29
Parameterize Function Module Test.....	29
Edit Compare Parameters .....	30
<b>Maintenance</b> .....	<b>30</b>
Record test case .....	31
Create test cases manually.....	32
Maintain test cases .....	32
Copy Test Cases.....	33
Delete Test Cases.....	34
Simulating Controls .....	36
Change Management (Rerecord) .....	36
<b>Attributes</b> .....	<b>37</b>
<b>Functions</b> .....	<b>39</b>
Function editor .....	40
Maintain functions .....	41
TCD – test transactions .....	42

COMMIT WORK Handling in CATT .....	43
Maintain screens.....	44
Get system data.....	47
Specify screens and field names dynamically.....	48
REF - Refer to test case .....	49
FUN – use function module .....	51
Use function modules .....	51
TXT - Enter comment.....	52
CHEERR - Check system message .....	52
Extended System Message Check.....	56
CHETAB – check table contents .....	56
CHEVAR – check variable contents .....	57
SETVAR - Assign values to variables and parameters .....	58
SETTAB - Set customizing table.....	61
Using SETTAB and RESTAB .....	62
RESTAB – reset table.....	63
DO n... (EXIT)... ENDDO loops.....	63
EXIT – conditional termination .....	64
IF ... ENDIF Conditions .....	64
Use of conditions .....	65
Schedule Execution Wait Time .....	66
<b>Parameterization .....</b>	<b>67</b>
Parameters.....	68
Define parameters .....	68
Assign values to parameters .....	70
Variants .....	72
Edit Variants in R/3 .....	73
Edit Variants Externally.....	74
Create where-used list .....	75
Special variables .....	76
Message variables .....	76
SET/GET parameters .....	77
Text variables.....	77
System variables.....	78
CATT special variables .....	78
Date variables .....	79
<b>Execution .....</b>	<b>80</b>
Individual Execution .....	81
Collective execution .....	82
Remote execution .....	84
<b>Logging .....</b>	<b>85</b>
Display log.....	86
Analyze Log.....	86
Common Log Errors.....	90
Running test cases from the log.....	91
Process Logs in Groups.....	91
Archiving logs.....	92

---

Notes and Recommendations.....	93
Test case creation tips.....	93
Rules for creating and maintaining test cases.....	94
Clients for creating and running test cases .....	95
External TCD data .....	95
Programming notes .....	97
Tips and tricks for creating test cases.....	98

## CATT: Enhanced Mode (BC-CAT-TOL)

### Use

The Test Workbench contains the Computer Aided Test Tool (CATT) to create manual or automatic test cases.

#### Manual test cases

Manual test cases are most useful for acceptance tests. They are descriptions of tests which a tester must perform manually on the system.

#### Automatic test cases

Automatic test cases are performed by the R/3 System without user dialog, and are most useful for function tests. The result of an automatic test case is a detailed log which documents the test. The use of automatic tests can considerably reduce the test effort.

Test cases test individual transactions or whole business transactions. Test cases are constructed modularly, to minimize the creation and maintenance effort, above all for business transactions.

The procedure for testing one transaction or function is described in test modules. The creation of test modules is greatly simplified by the CATT recording functionality.

Test procedures can be constructed from the test modules. Test modules are referred to and linked in the test procedure.

In modular test procedures:

- n test modules can be reused
- n changes to test modules are immediately effective in the test procedures which use them.

#### Simple test case creation

The transaction test procedure is recorded with a transaction recorder for the test module. The creator runs the application as in normal dialog operation. At the end of the transaction, you go back to CATT.

You can make test cases flexible by subsequently replacing the values input when recording by parameters.

You can store frequently used parameter settings in variants which you use in tests.

This functionality has been available since 4.6 in an additional [CATT maintenance mode \[Ext.\]](#) which can be used by users without technical background knowledge.

### Implementation notes

Test cases usually make R/3 System database changes, so they should not run in the productive system. To control this centrally, the [client table maintenance \[Page 95\]](#) has a flag which allows tests to be run in the system.

[Authorizations \[Page 15\]](#) are also checked in CATT.

## General information

### Integration

The ABAP Workbench infrastructure provides general functions such as correction and transport, the Repository information system, and multi-lingual applications.

In addition, it guarantees the correctness of a test run by updating synchronously, thereby explicitly refreshing the table buffer. This is especially important for transaction chains, where transactions use the results of previously executed transactions.

You can also test processes from one central R/3 system in a different R/3 or R/2 satellite system. The test procedure runs in the central R/3 system and controls the transactions via Remote Function Call (RFC) in the satellite systems.

The CATT logs created contain all information relevant for the test run and are stored centrally in the database of the executing R/3 system.

### Functions

CATT can:

- test transactions
- check system messages
- check authorizations (user profiles)
- testing results and database updates
- setup customizing tables
- test the effect of customizing setting changes

These processes are less suited for CATT:

- lists and display results
- menu paths
- Online help (F1, F4)
- editor functions

### Restrictions

You cannot use CATT for the following transaction types:

- Transactions that contain the statement LEAVE TO TRANSACTION

## General information

This section contains general information about CATT.

## New functions in Release 4.0, 4.5 and 4.6

### New Maintenance Mode (4.6)

The new [Maintenance mode](#) is a further development of the Release 4.5 Easy Mode. This maintenance mode provides CATT functionality in an environment which requires little or no training.

The user can create test cases by recording the transactions to be tested. If a transaction chain has been recorded in a test case, the user can chain the transactions together. This means that the result of a transaction can be passed to subsequent transactions.

You can also create variants. A variant is a set of runtime transaction input field values.

Field contents and error messages can also be checked against constants or parameters.

### Rationalization of Variants (4.6)

The distinction between "internal" and "external" [variants \[Page 72\]](#) has been dropped in 4.6.

### Extended System Message Checks (4.6)

Whereas you could previously only check the message number, you can now check all information in a message. You can specify up to three messages, of which at least one must be sent at runtime for the test to be successful, as [expected system messages \[Page 56\]](#).

### Check Field Values (4.6)

The check function checks whether the runtime value of a field is the expected value, at the end of the transaction. The check value can be a constant or a parameter (see *Define field check* in [Maintain screen \[Page 44\]](#)).

### Read Field Values (4.6)

You define a parameter in the test case detail screen maintenance for the field which is to be read, into which the field value is put after the transaction run (see *Read field value* in [Maintain screen \[Page 44\]](#)).

### Simulation of Controls (4.6)

The use of Custom Controls in applications requires a [CATT functionality enhancement \[Page 36\]](#) if they are to be tested with CATT. The data flows to and from Custom Controls are now recorded as well as screen input values when test cases are recorded. The data flows are those of the Data Provider and the Automation Queue.

When the test case runs, the Custom Controls are deactivated and the user actions are simulated by reading the recorded data flows.

### Namespace Extension (4.0)

CATT test case names are being lengthened as part of the Basis [namespace extension \[Page 16\]](#). At the same time CATT will switch from internal number assignment to external name assignment.

---

**New functions in Release 4.0, 4.5 and 4.6****CATT test case upgrade change management (4.5B)**

CATT test cases navigate through transactions in a fixed screen sequence. If the screen sequence is changed, test cases must be modified accordingly. The [Change Management \[Page 36\]](#) function has been implemented to make this easier.

**Screen simulation mode (4.5)**

Parameters can be maintained in screens in [screen simulation mode \[Page 44\]](#). It allows the maintenance of nested subscreens, Table Controls and TabStrips.

**Function module test (4.0)**

[Function modules can be tested \[Page 28\]](#) using type F test cases. These test cases can be managed in the Test Workbench.

**Waiting times in test cases (4.5B and 4.6)**

You can control whether the transactions in a transaction chain run directly after one another (within a work process), or are separated by a specified [waiting time \[Page 66\]](#) (work process is released during waiting), in a [parameter \[Page 66\]](#).

In 4.6 you can define a waiting time between screens. The waiting time applies to all background transactions. The system waits for the specified time between each screen. The waiting time depends on the number of fields transferred and the characters which they contain.

**CATT mode without COMMIT WORK termination (4.0)**

CATT has previously considered a transaction to be finished when COMMIT WORK was called. Transactions which contained several COMMIT WORKs could not be tested with CATT. The new [CATT mode \[Page 43\]](#) allows transactions which contain several COMMIT WORKs to be tested.

The previous mode is still supported.

**Release-independent test module call (4.0)**

Test cases usually use the screen definition for the transactions tested in the R/3 System which called them. This can cause problems when testing transactions in various test systems from a central System, if transactions are tested which have different screen definitions in the central System. The [release-independent call \[Page 49\]](#) function has been implemented for this case.

**Making test modules and test procedures more similar (4.5)**

In Release 4.5A test modules and test procedures are made more similar.

For this reason, the term test case is used in the documentation to include both test modules and test procedures. These terms are now only used where the distinction between test module and test procedure needs to be made.

**Retest from the Log (4.0)**


To ease the creation of complex test cases, individual test cases can be [retested \[Page 91\]](#) from the log.

When a test case is retested, its import interface has the same values as in the logged test. The retest is in a new session.



Test cases from systems with at least Release 4.5A must not be transported into systems previous to Release 4.5A.

## Enhanced CATT – Initial screen with Favorites

The Favorites list is now also in Enhanced CATT as well as standard CATT: You no longer need to select subobjects such as parameters or attributes directly. . You go to the attributes with the attribute pushbutton  in the application toolbar.

The Change/Display pushbuttons are now in the application toolbar in the initial screen.

## Delete Test Cases

This function is under **Utilities** → **Delete test cases** in the CATT menu.

You can select test cases individually or together in the following screen. All selected test cases are listed. All deletable selected test cases can be deleted together.



The list can be longer than the screen display: Check by scrolling that only test plan logs which you want to process are selected.

The procedure is described in [Delete test cases \[Page 34\]](#).

## Copy test cases with references

This function is under **Utilities** → **Copy test cases** in the CATT menu.

You can select test cases individually or together in the following screen. All selected test cases are listed and they can be copied together. The names of copied test cases can be changed.

The procedure is described in [Copy test cases \[Page 33\]](#).

## Process CATT test plan logs in groups

You can now process several CATT test plan logs together. You can now select logs to be processed in the Enhanced CATT overview list. You can choose individual test cases or all of them. You can also deselect all. You can process all selected logs together, e.g. assign expiry date, flag for archiving or delete.



The list can be longer than the screen display: Check by scrolling that only test plan logs which you want to process are selected.

The Enhanced CATT test plan log list now contains a column *B* which shows which CATT test cases ran in the background: They can be used for validation because they were certainly not changed at runtime.

The procedure is described in [Process logs in groups \[Page 91\]](#).

## Modular test cases

### Modular test cases

SAP recommends that complex test cases (test procedures) be assembled modularly from simple test cases (test modules). One test module usually tests one transaction.

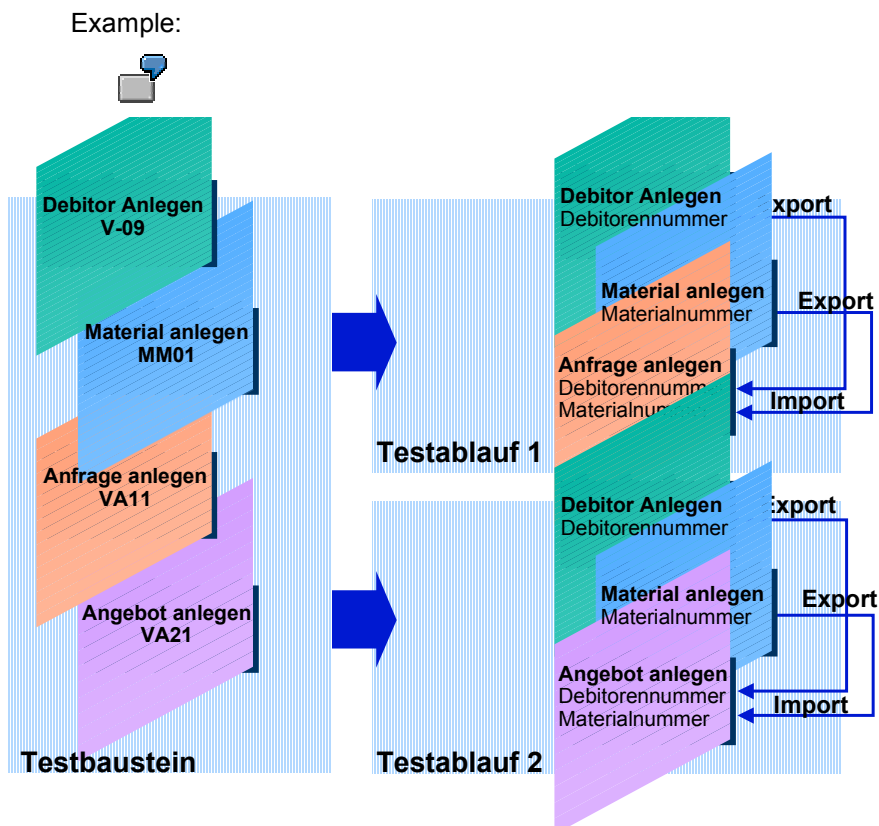
Test modules are test cases for a transaction, and test procedures are test cases for a process.

Create test modules using the recording function. You can use the [easy transaction \[Ext.\]](#) or the [Expert transaction \[Page 31\]](#). You can make test cases flexible by subsequently replacing the values input when recording by parameters. You pass the test module result (e.g. a customer number) in an export parameter.

After these steps, you have created a test module with an import and export interface, to which you can now refer in test procedures. The test procedure provides the import interface values and reads the export interface. The values read can, e.g. be passed to the import interface of a following test module.

This concept has the following advantages:

- you can reuse test modules in different test procedures.
- test module changes are immediately effective in the test procedures which use them.



You want to test the SD transaction “Create Inquiry”. The transactions “Create ordering party” and “Create Material” are prerequisites for “Create Inquiry”. First check the ABAP/3 Repository Information System for the existence of these modules. You can then use them in your test procedure. You must create test

**Plan tests**

modules for the other transactions. The three test modules are then assembled into a test procedure.

In a second step you want to test the transaction "Create Quotation". This transaction also requires the transactions "Create ordering party" and "Create Material". Since, for the first step, you already created the test modules "Create ordering party" and "Create Material", you must create only the test module "Create Quotation".



The only technical difference between a test procedure and a test module is that test procedures can have a test status.

The term test case is used in the documentation for both test procedures and test modules, except where they need to be distinguished.

## Plan tests

The success of automated tests depends on the quality of the test cases. It is therefore important to plan the test steps and gather the information needed, before creating the test cases. The following points are intended to help you to plan your test:

1. What is to be tested ?
  - Which process chains are to be modeled with CATT?
  - Which application areas are involved?
2. Restrictions
  - Are the tests restricted by, e.g. the language, the country, the test sequence, or the system environment?
3. Structure
  - Which test cases are needed, and how must they be parameterized so that they can be used as flexibly as possible?
  - Are variables required?
  - Which screens are called?
4. Run-time characteristics
  - Must customizing settings be changed?
  - If so, you must check that they are changed back at the end of the test run.
  - Must database changes be checked?
  - Must error messages be checked?

## Test phases

- Are procedure variants to be defined?
- They require import parameters on procedure level.

## Test phases

A test case usually has the following phases:

### 1. Prerequisites

You generate the master data which is required by the transactions which are to be tested.

### 2. Settings

Setup customizing tables, CATT variables and CATT parameters.

### 3. Transaction test

Run the test case. Procedures are usually tested by performing one or more transactions, which change data and return the result.

### 4. Checks

Transaction messages and database table contents are tested. Test plans can be programmed in function modules.

## CATT Management

The management tools are in the *Utilities* menu under the following headings:



For further information, call the Extended help (*Help* → *Application help*) in the CATT management program selection screen.

### • Collective execution

Run several test procedures (not test modules) by various selection criteria.

### • Transport preparation

Put several test procedures in one transport request. All test modules used by the chosen test procedure are put in the same request. You can remove test procedures from the request by double-clicking on them in the displayed list.



Test cases from systems with at least Release 4.5A must not be transported into systems previous to Release 4.5A.

- **Overview list**

A list of all test logs which satisfy the selection criteria is displayed. You can edit these test cases from the list, run them, and examine the log.

## Authorization

CATT is part of the ABAP Workbench. CATT authorizations use the “ABAP Development Workbench” (S\_DEVELOP) authorization object. This object has five fields, for which the following settings are checked:

1. Development class for Change & Transport Management (DEVCLASS)  
checked at creation. Not checked at runtime.
2. Authorization group ABAP program (P\_GROUP)  
not checked
3. Development object type (OBJTYPE)  
Check for value ‘SCAT’
4. Object name (OBJNAME)  
The test case name is checked. You can therefore assign authorizations to individual test cases or groups of test cases.
5. Activity (ACTVT)  
The following values are checked:
  - 01 when **creating** test cases
  - 02 when **creating, changing, and deleting** test cases
  - 03 when **displaying** test cases
  - 16 when **running** test cases
  - 70 when **managing** test procedures

This authorizes you to set the check flag for test procedures. To be able to set test status, the CATT *test status* flag must be set in the user defaults (*System* → *User profile* → *Own data*).

## Original language handling

### Original language handling

The integration of CATT in the ABAP Workbench makes the original language handling consistent. Test cases have an original language, as do all Workbench objects. When you create a test case, it has your logon language as its original language. All test case texts always exist in their original language. When a test case is copied, the new test case inherits its original language, as it is normally edited in this language.

If the original language of the test case being processed is different from your logon language, you can:

- proceed in the original language
- change the original language.



If you want to change the original language, ensure that the texts have already been translated into the new original language. If this is not the case for all texts, you can first copy the non-existent texts into the new language, but then ensure that they are translated as soon as possible.

### CATT in the Repository Information System

All attributes and objects used by the test cases are managed in the Repository Infosystem. You can search for test cases by specified criteria. Choose *Environment* → *Repository Infosystem*. You go to the Repository Infosystem initial screen. Go to the CATT test case search selection screen via *Repository* → *Environment* → *Automatic tests* → *Test cases*. For a detailed description of the Repository Information System, see the [ABAP Workbench: tools \[Ext.\]](#) documentation.

### Naming convention

When you create a test procedure or module, give it a unique name of up to 30 characters, as follows:

1. "Y" or "Z" (customer namespace) or a namespace prefix
2. unique name

## Test case

### Definition

The test case models a complete business or administrative process, with all its prerequisites for a particular test. "Create standard order" requires, e.g. "Create customer", "Create material" and "Specify conditions".

### Use

A test case can use (refer to) any other test case for all or part of an application. The test case can be given a test status if appropriate settings have been made (see "Management data" below).

Test cases are client-independent, and can be created in any client. Test cases can, however, only run in certain clients, as customizing settings are required.

### Structure

- **Referred test case (optional)**  
A referred test case usually tests a small, testable entity, which is usually a transaction, e.g. "Create customer". It modularizes a test case and makes individual tests reusable.
- **Attributes**  
Test case flags and management information which can be used as search criteria in the Repository Info System.
- **Management data**  
If the CATT *Test status* flag is set in the user defaults (*System* → *User profile* → *Own data*), test case management data containing processing and test data are saved after each test procedure run. (See "Test status" in [Collective execution \[Page 82\]](#)).
- **Functions**  
Individual test actions performed in a test case.
- **Parameters and variables**  
for the flexible use of test cases, passing values between referred test cases and performing calculations.
- **Variants**  
a test-specific set of test case import parameter values.
- **Initial screen** (individual execution)  
Contains the run conditions (log type, processing mode, variants) and executes a test case.
- **Log**  
Test case runs are logged. The log lists any errors in the tested procedures. You can display the transaction error log.

## Test Case Types

# Test Case Types

## Use

The Test Workbench has the following test case types:

Type	Meaning
<a href="#">CATT [Page 18]</a>	Automatic R/3 transaction test
<a href="#">Function module test [Page 28]</a>	Automatic function module test case
<a href="#">Remote R/2 test [Page 20]</a>	Remote R/2 transaction test
<a href="#">External application [Page 22]</a>	To load files locally on to the presentation system, where they can be edited by external programs Test cases of this type can save test data in the format of <a href="#">external manufacturers' test tools [Page 25]</a> .
<a href="#">Manual test case [Page 19]</a>	Manual test instructions
<a href="#">Referring [Page 19]</a>	Test case to refer to test cases in other R/3 Systems. The test case can only be called from the Test Organizer.

The test case type is specified in the test case [attribute screen \[Page 37\]](#).

## Integration

All types can be created, maintained and managed in the [Test Organizer \[Ext.\]](#).

# Use of Automatic Test Cases

## Use

Type CATT test cases are used to test R/3 transactions automatically. The transactions usually run without user interaction. Tests are logged.

Type CATT test cases have two maintenance modes:

- [CATT \[Ext.\]](#) Create test cases.
- [Extended CATT \[Ext.\]](#): Create test cases with complex check logic: all [CATT functions \[Page 41\]](#) are available.

## Procedure

1. Create test cases. Use the recording functionality. You can create transaction chains that correspond with your business processes.

## Use of Manual Test Cases

2. Define test rules in the test cases.
3. Create test case variants.
4. Manage the test cases in the Test Organizer.
5. Run the test cases and analyze the logs. You may have to retest test cases in which errors occurred.

## Use of Manual Test Cases

### Use

Type *Manual test case* test cases contain the instructions for manual tests. Test instructions are written as SAPscript long texts for the test case.

*Manual* test cases cannot use the automatic CATT functions.

### Procedure

1. Create a **Manual test case**.
2. Write a long text for this test case and save it.
3. Manage the test case in the Test Organizer.

## Use of Referring Test Cases

### Use

A *Referring* type test case refers to an existing automatic test case in another R/3 System. This type is useful when you use a central test management system which manages test catalogs of test cases and tests various R/3 Systems (with different Releases).

Create a test case containing a test routine in each system. It must have the same name in all systems.

Create a test case of type **referring**, which calls the test case in the target system, in the central test system.

The target system is specified in the Test Organizer when the test plan is created. You can create a test plan for each R/3 System to be tested and specify the correct R/3 System in each test plan.



Test cases of this type can only be called by the Test Organizer.

## Using CATT for R/2 test cases

### Procedure

1. Create an automatic test case in the target system.
2. Create a new test case in the central test system.
3. Assign type **Referring** to the test case in the attribute maintenance *Management data* tab.
4. Assign the referred test case in the target system to the test case in the attribute maintenance *General data* tab.
5. Save the test case.
6. Put the test case in a test catalog in the Test Organizer.
7. Specify the target system when you create the test plan.

## Using CATT for R/2 test cases

### Use

*Remote R/2 test* type test cases are used to test R/2 transactions automatically. The transactions run without user interaction. You can check system messages and test database changes. All tests are logged.

*Remote R/2 test* cases have all [CATT functions \[Page 41\]](#). They are maintained in [CATT extended mode \[Ext.\]](#).

The following restrictions apply:

FUN	The function module runs in R/3 but it can call function modules remotely in R/2
CHETAB SETVAR <tab>	All tables that are not in ATAB must have a data read function module in the R/2 table TC40B.
SET cursor	Has no effect
SET/GET parameters	Not available

### Prerequisites

The R/2 test system must be Release 5.0G or later and the following SAP Basis components must be installed:

- BS2000:
  - Remote Function Call
  - Host communication with TCP/IP
  - 153 SAP host link
- MVS and VSE systems:

R/2: Create Simple Test Case

- Remote Function Call
- Host communication with other LU 6.2 systems
- 153 SAP host link

There must be an RFC connection between the R/3 system and the R/2 test system. The R/2 user must be type 'CPIC' and must have the necessary authorizations.

**Procedure**

1. Create simple **Remote R/2 test** cases (test modules) to test individual transactions.
2. Assemble these simple test cases into procedures. Create *Remote R/2 test* cases that use the test modules in a meaningful sequence. You can create transaction sequences that correspond with your business processes.
3. Save the test case.
4. Create test case variants.
5. Manage the test case in the Test Organizer.
6. Run the test case and analyze the log.

**R/2: Create Simple Test Case**

**Prerequisites**

You must have an RFC link to the target system before creating a *Remote R/2 test* case.

**Procedure**

1. Record a BDC session of the transactions to be tested in the R/2 test system with the transaction SIRC.
2. Check that the BDC session is batch-input-compatible with the transaction SBDC.

You can use the following utility programs:

RSCATB00	Display screen definition
RSCATB10	CALL TRANSACTION with spooler data
RSCATB30	Display BDC session in R/3 format.

3. Call the CATT maintenance transaction in the R/3 System and create a new test case.
4. Enter the type **Remote R/2 test** in the test case attribute screen. You are prompted for the R/2 remote system if you have not yet specified it. Specify the System.
5. Go to the function editor. Choose *Edit* → *Enter TCD*.
6. Enter the file and session names in the *Read R/2 BDC session* dialog box. Choose *Continue*. A new TCD entry is made in the function editor.
7. [Parameterize \[Page 67\]](#) the R/2 transaction input fields.

## R/2: Create Transaction-Independent Test Case

8. Save the test case.

### Message check via SAPMSCEM 0001 also in R/2

The message check via screen SAPMSCEM 0001 is now also recommended for the R/2 System, particularly if the expected message numbers are between 001 and 010 because these numbers are also used for system errors with the message ID (work area) TT. In such cases a transaction with errors could pass as 'OK' if only the numbers and not the message ID are checked, so you should append the message screen SAPMSCEM 0001 to the screen list. Define the check for message number *and* work area in this message screen. This replaces *Check Error* which only checks the message number with a combined message number and work area check.

See [Extended system message checks \[Page 56\]](#).

## R/2: Create Transaction-Independent Test Case

### Prerequisites

You must have an RFC link to the target system before creating a *Remote R/2 test case*.

### Procedure

1. Call the CATT maintenance transaction in the R/3 System and create a new test case.
2. Enter the type **Remote R/2 test** in the test case attribute screen. You are prompted for the R/2 remote system if you have not yet specified it. Specify the System.
3. Go to the function editor.
4. Refer to the simple test cases from which you want to assemble the transaction-independent test case, with the REF function.
5. Insert other CATT functions in the test case (e.g. table check).
6. Save the test case.
7. Manage the test case in the Test Organizer.
8. Run the test case and analyze the log.

## Using External Applications

### Use

You can put external applications in test cases that are used in other test cases. You can, for example run external test tools or display manual test documentation, which was not created in the SAP system, in external programs.

## Prerequisites

Frontend:

- Windows NT, Windows95; Windows 3.x or OS/2
- Installation of the application program
- Association of filename extension with the application program

## Procedure

1. Create a new **External application** type test case.
2. Put the files in the file list.
3. Pass import parameters to the external applications and read the export parameters back from it.
4. Use the test case in other test cases.

## Use External Applications

1. Go to the CATT maintenance transaction initial screen and create a new test case.
2. Maintain the attributes. Specify type **External application**. This flags the test case as an external application.
3. Go to the file list via *Goto* → *Functions*.  
All files for the module are listed in the *CATT - External application file list* window.



You cannot define functions for external applications.

4. Process the file list using the functions below.
5. Mark the object which is to be executed when the test case runs locally on your computer. Unmarked files are downloaded before running the external application and are thus also available.
6. Save the test case.



If you want to use a data file as initial file, you must ensure that the initial filename extension is associated with the application program on your computer.

### File list functions:

#### Delete line:

Delete marked test module objects.

## Use External Applications

### Insert URL:

To call URL addresses, choose *Insert URL*. Enter the URL address in the input line. When this line is chosen, a web browser is called and goes to this address.

### Import file:

*Import* local files into the test module.

### Export file:

Store files from the test module locally on the presentation system.

### Refresh file:

You can refresh an existing file by uploading an up-to-date version from your presentation system and overwriting the old version.

## Parameter interface between test module and external application:

Parameters can be defined to allow test cases to communicate with an external application. Import parameters are downloaded in a text file and can be read by the external application. The text file is downloaded first, before the external application files.

The external application can put export parameters in this text file, which is uploaded by the test case after the external application has run. The parameters can be processed further in the test case. The CATT export parameter names are downloaded with the import parameters.

### Name of the parameter passing text file

The default name for the parameter passing text file is `CATSTART.TXT`. If you want a different file name, define an import parameter with the name `&PARAMFILE` in the type *External application* test module. Assign the desired file name to this parameter.

### Text file structure

Each parameter has a text record with three fields:

- 1: Parameter type 1-character, "I" = Import-, "E" = Export
- 2: Parameter name max. 12-character, without leading "&"
- 3: Parameter value max. 132-char.

The first two fields are separated by a space. The 2nd field is output as 12 characters. If the parameter name is shorter, the field is filled with spaces. The last two fields are separated by " = " (space, equals sign, space).



```
I IMPORT1   = Test
E EXPORT   = Test end
```

## Run External Application

E EXPORT1 = Test successful



Export parameters can be returned without spaces.

A parameter text line cannot exceed 148 characters.

## Run External Application

If a type C test case calls an *External application type* test case, it always runs in the foreground. When running the test case, a dialog box appears that tells you how to proceed.

If there is a long text containing test instructions for the external application test case, choose *Long text* in the dialog box to display it.

After performing all external test case steps, choose *Continue* in the R/3 system dialog box. The test case runs as described in [Run test case \[Page 80\]](#).

## Using an External Test Tool

### Use

Test cases in the format of the test tools of external manufacturers (e.g. Mercury Interactive), can be used in CATT via a standardized CATT interface. The test cases must be of type *External application*, and the external test tool must be installed on the frontend where the SAPGUI runs. The interface between CATT and the external test tool is based on Control Enabling Technology. You can call the external test tool and create, edit and execute a test case from CATT. You can use all CATT functions to manage and analyze the test case.

The CATT interface is a functionality enhancement to use external test tools in CATT test cases. Common Internet Browsers are also supported, so SAP Internet application components can also be tested.

### Prerequisites

To test SAP Internet application components, the external test tool must have a special SAP enhancement. See the manufacturer documentation for whether this enhancement is possible.

## Setup external test tool interface

# Setup external test tool interface

## Prerequisites

The external test tool installation on your frontend must support the CATT interface. See the manufacturer specification.

## Procedure

### Customizing:

1. Go to customizing (*Tools* → *Business Engineer* → *Customizing*).
2. Go to the application component hierarchy (*Implementation projects* → *SAP reference IMG*).
3. Select the chapter
  - Application-independent components
    - CATT tool
      - Use external test tool
4. Run this customizing transaction.
5. Enter your interface parameters as described in the IMG chapter documentation. You can, for example specify the external test tool name with which you subsequently call the test tool in the CATT maintenance screen.



When you have configured the interface in CATT, you can use all the external test tool functions from CATT.

## Call External Test Tool From CATT

### Procedure

1. Go to the CATT maintenance transaction initial screen and create a new test case.
2. Maintain the attributes and specify type **External application**. This flags the test case as an external application.
3. Go to the file list. Next to the file inclusion function pushbuttons (see [Using external applications \[Page 22\]](#)), there is a pushbutton labeled external test tool, or with the name of which you specified in [Customizing \[Page 26\]](#).
4. Call the external test tool by pressing this pushbutton. If the interface was defined correctly, the external test tool runs on your presentation computer.

**Edit External Test Tool Test Script**

5. Record a test case with the external test tool and save it with a test tool function (see manufacturer documentation).
6. After saving, the external test object appears in the CATT test case file list. In the same line next to the test object name is a two-character manufacturer code which was defined in customizing to identify the external test tool format.



Communication with CATT is only possible if the external test tool was called from the CATT file list (from the CATT maintenance transaction with *Go to* → *Functions*). The communication is canceled when you leave the CATT maintenance screen or call a CATT function. To reestablish communication, you must call the test tool from CATT again.

## Edit External Test Tool Test Script

The CATT interface can save a test case as a test object in a *External application* type CATT test module. Only one external test tool test case can be saved per CATT test case.

After the test object has been created and saved in the external test tool, the test script appears in the CATT maintenance screen file list.

See the documentation of the external test tool for the use of the save function in the CATT interface.

When the test object appears in the file list, save the CATT test case. This interrupts the communication to the external test tool.

The test script of an external test case cannot be deleted with the file list functions. To delete a test script, you must delete the CATT test case.

To edit the test script, call the external test tool with the right-hand pushbutton above the file list, and edit it locally in the external test tool. The updated test script is imported into the CATT test case again after editing.

## Parameter interface between CATT and external test tool

CATT has import and export parameters for passing data between test modules and external applications (see also [Use external applications \[Page 23\]](#)) which the external test tool can use. They are passed to the CATT interface with the test case when the external test tool is called. The test tool provides import and export parameter read and set functions.

The parameters have the following format:

## Using function module tests

The parameter type is either import or export.

The parameter name is maximum 12-character and case-insensitive.

The parameter value is maximum 132-character and case-sensitive.

You can also redefine import and export parameters which were added to the existing CATT test case parameters after saving the CATT test case, from the external test tool .



Communication with CATT is only possible if the external test tool was called from the CATT file list (from the CATT maintenance transaction with *Go to* → *Functions*). The communication is canceled when you leave the CATT maintenance screen or call a CATT function. To reestablish communication, you must call the test tool from CATT again.

## Using function module tests

### Use

You can create and run function module tests with type *Function module test*. Function module dialogs can also be tested.

When you create a function module test, a parameter set is created for the function module parameters, structures and tables to be passed. The function module then runs. If it contains dialogs, they are recorded. At the end of the function module, its result is added to the parameter set as a compare parameter. You can then choose parameters, structures and tables from among the compare parameters, whose contents should match the recorded values in future tests.

### Procedure

1. [Create a function module test \[Page 29\]](#).
2. [Parameterize the function module test case \[Page 29\]](#), if it contains dialogs.
3. [Edit the list of compare parameters \[Page 30\]](#).
4. Save the test case.
5. Use the test case to test function modules automatically.

## Create Function Module Test

### Procedure

1. Create a *Function module test* type test case:
  - Create a new **Function module test** type test case in the Test Organizer and choose *Record*.
  - Choose *Record function module* in the CATT maintenance transaction initial screen.
2. Enter the function module name in the dialog box and choose *Copy*.
3. Pass test values to the function module import, changing and table interfaces.
4. Run the function module.
5. Perform any function module dialogs.
6. You get the result at the end of the function module. You can now choose which comparisons are to be carried out for subsequent function module tests by selecting entries. The current result is your reference.

## Parameterize Function Module Test

### Prerequisites

You can only parameterize dialog function modules. You can only parameterize the dialog entries.

### Procedure

1. Go to the *Function module test* type test case functions.
2. Position the cursor on the function module to be tested and choose *Choose*. You go to the dialog screen overview.
3. Position the cursor on a screen and choose *Choose*. The simulated screen appears.
4. Maintain the parameters, as described in [Maintain screen \[Page 44\]](#).
5. Save the changes.

## Edit Compare Parameters

# Edit Compare Parameters

At the end of a function module test, the tested function module returns a result to its interface (Export, Changing, Tables). You can compare this result with a set of compare parameters. If the result of the function module test is different from the compare parameters, the test is flagged as failed.

You can exclude specified compare parameters from the result comparison.

## Procedure

1. Go to the *Function module test* type test case functions.
2. Choose *Compare parameters*.

A list of the compare parameters, which was created when the function module test was recorded, appears.

3. Mark the result parameters (Export, Changing, Tables) which are to be compared with the current result, after the function module test. Position on the parameter names and choose *Select/Deselect*.
4. Save the changes.



You can only compare complete structures or internal tables. You cannot choose individual structure or internal table fields.

## Maintenance

### Use

Test case maintenance is mainly the maintenance of automatic test cases. It includes maintenance of attributes (for all test case types), CATT-specific functions, and CATT parameters.

### Prerequisites

You need test case maintenance [authorizations \[Page 15\]](#).

### Functions

You maintain the attributes in the attribute screen subviews.

You maintain CATT-specific functions in a CATT-specific editor.

You maintain CATT parameters at various points in the test case maintenance. The parameter maintenance screen contains an overview of all the test case parameters.

You create simple automatic test cases with a transaction recorder.

## Activities

The CATT maintenance transaction is in the ABAP Workbench menu, under *Test* → *Test Workbench* → *CATT* → *Edit test case*.

## Record test case

You can use the recording function to create test cases for transactions:


1. Enter a test case name in the CATT maintenance transaction initial screen and choose *Record module*.
2. Enter the transaction for which you want to record a test case, in the next dialog box.

Confirm your entries with `ENTER`.

You go to the initial screen of the transaction you want to record.



Specify the transaction recording [CATT mode \[Page 43\]](#) in the transaction entry dialog box. The transaction is recorded in “without COMMIT WORK end” mode by default. It can be appropriate in particular cases to record the transaction in “with COMMIT WORK end” mode (e.g. Batch Input data transfer transaction sequence test). To change the mode, choose “COMMIT WORK end” (`F7`) with the right-hand mouse key in the dialog box.

3. Make sure that the window size is set to the default by choosing  → *Default size*.  
This ensures that recorded test cases run correctly in the background. This is necessary, e.g. for step loop fields whose length depends on the screen size.
4. Run the transaction as you want it to be tested later with the CATT test case.
5. After ending the transaction, you automatically go to the CATT maintenance transaction. A dialog box tells you that the recording is finished. Choose `ENTER`.
6. You go to the CATT maintenance transaction attribute maintenance screen. Check the attributes and change them if necessary.
7. Choose *Goto* → *Functions* to go to the [Function editor \[Page 40\]](#).
8. Enter additional [functions \[Page 41\]](#) and save them.
9. [Parameterize \[Page 67\]](#) the test case.



Read the [test case creation tips \[Page 93\]](#) before creating test cases.

You can also create test cases manually, as described in [Create test cases manually \[Page 32\]](#).

## Create test cases manually



You must confirm type 'I' system messages with ENTER before you can continue. This also applies to messages that only appear in the status bar.

If an OK code of type 'T' is entered on a screen of the transaction to be recorded, the recording terminates without an error message.

## Create test cases manually

Use this procedure if you want to collect individual test modules (transaction test cases) in a test procedure (e.g. to test a business process) (see also [Modularizing test cases \[Page 12\]](#)).

1. To create a test case manually, enter a name and choose *Test procedure* → *Create* in the CATT maintenance transaction initial screen.
2. Maintain and save the [attributes \[Page 37\]](#). When you save the module or procedure for the first time, a dialog box prompts you to assign it to a development class.

Set the *Test module* attribute for test modules.

3. Choose *Goto* → *Functions* to go to the [Function editor \[Page 40\]](#).
4. Enter [functions \[Page 41\]](#).
5. Parameterize the test case.



Read the [test case creation tips \[Page 93\]](#) before creating test cases.

You can create test cases for a transaction quickly and easily with the CATT [recording functionality \[Page 31\]](#).

## Maintain test cases

To maintain an existing test case, enter the test case name in the CATT maintenance transaction initial screen, mark the desired subobject and choose *Change*. You go to the associated maintenance screen.



If you do not know the name of the test case, you can use the [Repository Infosystem \[Page 16\]](#) to search by various criteria.

You can display a list of the test cases which refer to the current test case by choosing *Utilities* → *Use of test module*.

## Copy Test Cases

You can select all test cases which are referred to, but do not yet exist, and all unused test cases, by choosing *Utilities* → *Use of references* in the maintenance transaction initial screen. The selection is performed over all test modules, it cannot be restricted.

You can go to a window in which you can enter an existing transport number by choosing *Utilities* → *Requests*. This skips the development class and correction number dialog box.

You can list the test cases which are in a procedure by choosing *Utilities* → *Graph. output* → *Test mod.hierarchy*.

You can list the functions and parameters in a test case by choosing *Utilities* → *Graph. output* → *Test procedure*. You can print this list by choosing *Test procedure* → *Print*.

## Copy Test Cases

### Use



The function *Copy test cases* copies one or more test cases.

### Prerequisites

You are in the Extended CATT mode.

### Procedure


#### Copy with search by test case names

1. If you are in the Extended CATT initial screen, enter the name of a test case to be copied.
2. Choose **Utilities** → **Copy test cases**.
3. To select several test cases, replace parts of the test case name in the new screen with wildcards.
4. Assign the copies to a development class if it is different from that specified
5. Choose a prefix for the name of the test case copy and choose . The test cases found are listed.
6. Choose the test cases to be copied.
7. Change the proposed name of the copy or copies, if required.
8. Choose  to create the copy or copies.

#### Copy with search by other criteria



1. If you are in the Extended CATT initial screen, enter the name of a test case to be copied.
2. Choose **Utilities** → **Copy test cases**.
3. Put the cursor in the name field and choose **F4**.

## Delete Test Cases

4. Enter your search criteria in the following screen and choose . The test cases found are listed.




You can search with [wildcards \[Ext.\]](#).

5. Select the test cases to be copied and choose .
6. Choose a prefix for the name of the test case copy and choose .
7. Choose the test cases to be copied.



The list can be longer than the screen display: Check by scrolling that only test cases which you want to process are selected.

8. Change the proposed name or names, if required.
9. Choose  to create the copy or copies.

## Result

Copies of all selected test cases have been created.

## Delete Test Cases

### Use


The function *Delete test cases* deletes one or more test cases.

### Prerequisites

You have created test cases which are to be deleted. You are in the Extended CATT mode.

### Procedure


#### Search by test case names

1. Enter a test case to be deleted in the *Test case* field in the initial screen.
2. Choose **Utilities** → **Delete test cases**
3. Extend the test case name search with wildcards in the following screen if you want to delete several test cases.
4. Search with . The test cases found are listed.
5. Select test cases.



The list can be longer than the screen display: Check by scrolling that only test cases which you want to process are selected.

Deletion is irreversible.


6. Delete your selection with  and confirm the following dialog box.

### Search for test cases by other search criteria




1. Enter a test case to be deleted in the *Test case* field.
2. Choose **Utilities** → **Delete test cases**. You go to a new screen, the test case is entered in the appropriate field.
3. Choose **F4** and enter your search criteria in the following screen.



You can use incomplete test case names with [wildcards \[Ext.\]](#).

4. Search with . The test cases found are listed.
5. Select test cases to be deleted





You can select and deselect all test cases together with  and  respectively. You can display selected test cases with .



The list can be longer than the screen display: Check by scrolling that only test cases which you want to process are selected.

Deletion is irreversible.

6. Copy all selected test cases with .
7. Confirm your selection in the next screen.
8. Delete your selection with  and confirm the following dialog box.



Deletion is irreversible.

### Result

All selected test cases have been deleted.

## Simulating Controls

# Simulating Controls

When you record transactions that use Controls in a test case, the Control data flows between application server and frontend are recorded.

The Controls are simulated when the test case runs. The Controls are deactivated and the recorded data flows are read in. You can then test the reaction of the transaction to Control events (e.g. double-click on a node in an SAP Tree Control).

The recorded Control data must be the same as the test data. A data change could invalidate the test (e.g. the node for a double-click no longer exists).



The Control functions are only tested when recorded, not at test runtime.



Manual modification of the test case to reflect transaction or data changes is often problematic. You should [rerecord \[Page 36\]](#) the transaction.

## Change Management (Rerecord)

### Use

Test cases that model a transaction can be revised with the Change Management at upgrade to a new SAP Release. This new function retains the parameters from the previous test case screen sequence, in the new screen sequence, as far as possible.

If a test case is to be revised with the Change Management, the transaction is rerecorded using the recording function. When it has been recorded, the parameterization is compared with the previous parameterization. Parameters that already existed are put in the newly-recorded transaction. The technical names of the input fields are used to identify the parameters.

### Integration

The parameters can only be reused if the technical name of the input fields is the same in the transaction before and after the release change. Parameters that could not be assigned are listed by the Change Management function.

### Prerequisites

You must choose a transaction call (**TCD**) for revision in the test case.

### Functions

When the function has been performed, the user can change the transaction code in a dialog box.

The CATT mode can also be changed in this dialog box. "Without COMMIT WORK end" mode is the default. It can be changed in the dialog box context menu.

## Attributes

When it has been recorded, the parameterization is compared with the previous parameterization. The technical names of the parameterized input fields are compared in the new and previous recordings. If more than one input field with this technical name is found in the new recording, only the first input field is parameterized.

Parameters that could not be assigned are listed by the Change Management function, and must be processed manually.

## Activities

Go to the Change Management function in the CATT maintenance transaction in the function editor.

1. Edit the test case in the test case function list. Position the cursor on the line which calls the transaction (TCD <transaction code>).
2. Choose *Rerecord*.
3. The transaction to be revised is displayed in a dialog box. Confirm, go to the transaction, and record.
4. At the end of the transaction, the system automatically parameterizes the rerecorded transaction using the previous parameterization.
5. Complete manually, if necessary (see Integration).

## Attributes

### Use

The test case attributes contain organizational (e.g. tested components) and technical (e.g. test case type) information. All attributes can be used as test case selection criteria (e.g. in the Test Workbench).

### Functions

There are four groups of test case attributes:

#### Management data

- *Test case*  
Name of the current test case.
- *Title* (required field)  
Enter a short description of the test case.



It should start with the transaction code of the transaction to be tested for test modules (transaction test case). This helps you to find test modules to use in test procedures (business process test case).

## Attributes

- **Type:**  
[Test case type \[Page 18\]](#)
- **Person responsible**  
The creator of the test case is automatically entered in this field. You can change the entry if the creator is not the person responsible for this test case.
- **Development class**  
A dialog box prompts you for the development class when you save.
- **Component**  
SAP application hierarchy component name.
- **Status**  
The status (active or inactive) is set automatically when you save.
- **Release**

## General data

- **Test case**  
Only for test cases of type '[Referring \[Page 19\]](#)'. Specifies the test case in the target system.
- **Time**  
Estimated test case time.
- **Priority**  
Test case priority.
- **Test objects**  
Test objects can be transactions (type T), reports (type R) and function modules (type F). Enter the name of the test object in the *Name* field.  
If the test tests several test objects, you can specify several test objects.

## Restrictions

- **Release**  
If the test case is only valid for certain releases, you can specify a validity interval. If you try to run the test case in a Release which is not in the specified interval, it does not run and an error appears in the log.  
When the test case is created, the current Release is entered as the lower limit. If you have made changes to the test case which are not backwards-compatible, replace this value with the last Release for which the test case runs.
- **Language**  
Flags a test case as language-dependent.  
Enter the language to which the test procedure or module applies. If the test case runs in another language, it is flagged as an error in the log and it is canceled immediately.
- **Country**  
Flags test cases as country-specific.  
Enter the country to which the test case applies. This flag is not checked at runtime.
- **Customizing**  
If the test case is customizing-dependent, specify the customizing required.

- *Database*  
If the test case is database-dependent, specify the database for which it was created.
- *Platform*  
If the test case is platform-dependent, specify the platform for which it was created.
- *Frontend*  
If the test case is frontend-dependent, specify the frontend for which it was created.
- *Use:*  
Areas of use of the test case.

### CATT-specific

- *Test module*  
The object being created is a test module.
- *Termination flag*  
The test case terminates upon the occurrence of the first error. Otherwise it continues despite errors.  
The termination flag is only effective in directly called test cases. The termination flag of the calling test case applies to referred test cases.
- *Context*  
The context-dependent definition of test procedures is relevant for collective tests ([See: Collective execution \[Page 82\]](#) )  
Set this flag if you want to specify a sequence when you call several test procedures. Enter a sequence value (**1 - 4** for leaders or **6 - 9** for trailers).

## Functions

### Use

You can create automatic R/3 and R/2 test cases with the CATT functions in a function editor tailored for CATT requirements.

### Prerequisites

Functions can only be created for type C, R and L test cases.

### Functions

The following functions exist:

- [TCD \[Page 42\]](#): run and test transactions
- [REF \[Page 49\]](#): refer to other test cases
- [FUN \[Page 51\]](#): call ABAP function modules
- [TXT \[Page 52\]](#): comment
- [CHETAB \[Page 56\]](#): check database contents

## Function editor

- [CHEVAR \[Page 57\]](#): check parameter contents
- [CHEERR \[Page 52\]](#): check system messages
- [SETVAR \[Page 58\]](#): assign values to parameters
- [IF ... ENDIF \[Page 64\]](#): conditions
- [DO ... ENDDO \[Page 63\]](#): loop
- [SETTAB \[Page 61\]](#) / [RESTAB \[Page 63\]](#): change customizing tables

## Function editor

Go to the function editor with *Goto* → *Functions*. Maintain test case CATT functions.

## Editor commands

You enter editor commands in the C column. Editor commands:

Command	Effect
r(epeat)	the contents of the current line are copied into the following line.
i(nsert)	an empty line is inserted after the current line.
d(elete)	deletes the current line.
c(opy) m(ove)	copy or move the current line. You must specify a destination, i.e. you must flag the line before the target line with <b>a</b> (after).
> resolve reference	copy the contents of a referred test module ( <b>See: <a href="#">REF - Refer to test module [Page 49]</a></b> ) to the current position in the current test case.
*	puts the current line in the first line of the current page.
Position	

## Block mode

You can edit blocks of lines in the function editor. You go to block mode with the *Select* function, in which you can delete, move, copy and save blocks. At first, the chosen line is the complete block. You can expand it with *End selection*.

You can move blocks between procedures with the *Export* and *Import* functions. The buffer used is retained when you leave CATT or logoff. It is client-independent in the current system.



You can also use the block functions, except *Export* and *Import*, in the CATT function **TCD** (test transaction) detail maintenance screen, which lists the program names and screen numbers of the transaction.

Block functions:

- **Select**
  - a) Position the cursor on the first line to be selected and choose *Select*.

Maintain functions

- b) Position the cursor on the last line to be selected and choose *Select End*. The selected block is highlighted.  
Reset block mode with *Goto* → *Back*.
- **Copy/Move**
  - a) Select the block as described under *Select*.
  - b) Position the cursor on the line at which the block is to be inserted and choose *Copy* or *Move*. The block is inserted at the current cursor position.
- **Delete**
  - a) Select the block as described under *Select*.
  - b) Choose *Delete*. The selected block is deleted.
- **Store**

You can buffer a block with the Store function. You can use this buffer in subsequent logon sessions in the same system.

  - a) Select the block as described under *Select*.
  - b) Put the block in the CATT buffer with *Edit* → *Block* → *Store*.
  - c) Insert the stored block at the current cursor position with *Edit* → *Block* → *Read*.

## Maintain functions

1. Enter the desired functions in the *Funct.* column in the [Function editor \[Page 40\]](#), or choose from the F4 possible entries help.
2. For the following functions, enter the required object in the following column, and confirm your entries.

Function	Object
<a href="#">TCD [Page 42]</a>	<transaction code>
<a href="#">REF [Page 49]</a>	<test module name>
<a href="#">SETVAR &lt;table&gt; [Page 58]</a>	<table>
<a href="#">SETTAB [Page 61]</a>	<table>
<a href="#">RESTAB [Page 63]</a>	<table>
<a href="#">DO [Page 63]</a>	<n>
<a href="#">CHEERR [Page 52]</a>	<error number>
<a href="#">CHETAB [Page 56]</a>	<table>

For table objects and transaction codes, an associated short text is entered in the *Text* column.

3. For the following functions, complete the *Text* column:

Function	Text
<a href="#">SETVAR [Page 58]</a>	<value assignment>

**TCD – test transactions**

<a href="#">IF [Page 64]</a>	<condition>
<a href="#">DO [Page 63]</a>	<loop counter>
<a href="#">EXIT [Page 64]</a>	<condition>
<a href="#">FUN [Page 51]</a>	<function module name>
<a href="#">TXT [Page 52]</a>	<comment text>
<a href="#">CHEVAR [Page 57]</a>	<value check>

4. For the following functions, you must perform an additional step. You enter the required values for the test function in an extra maintenance screen:

- [SETTAB \[Page 61\]](#)  
table field values
- [SETVAR <table> \[Page 58\]](#)  
key field values and variables for the desired values
- [TCD \[Page 42\]](#)  
values for all transaction screen fields
- [REF \[Page 49\]](#)  
referred test case interface
- [FUN \[Page 51\]](#)  
values for import and export parameters and exceptions
- [CHETAB \[Page 56\]](#)  
table field values

To go to this maintenance screen, double-click on a line, or position the cursor on a line and choose *Edit* → *Select*.

## TCD – test transactions

You put transactions in the test case with the function **TCD**. You maintain the required input values for the input fields of all screens in the transaction which is to be tested, in a detail maintenance screen.

### Generate TCD from recording

You can generate a TCD statement in two ways with the recording function:

- Choose *Record module* in [Create new test case \[Page 31\]](#).
- Place the cursor on the line where a TCD statement is to be inserted and choose *Edit* → *Record and Insert* when you are editing an existing test case in the function editor. Proceed as described in Record test case. The only difference in the procedure is in step 1.

## Other settings

You can also make the following settings in the TCD function detail screen (double-click on the TCD function in the function editor):

- You can enter the number of an error message that can occur in this transaction without causing the transaction to be flagged as failed in the log, in the *Allowed message* field.
- You can choose the processing mode for this transaction in the *Processing mode* field. If you make an entry (except ' '), this TCD statement always runs in this processing mode.

## Edit TCD manually



If the transaction uses Controls, the test case created cannot run because the Controls data flows to the SAPGUI cannot be input manually. You should use the recording functionality whenever possible.

## Create TCD

1. Enter `TCD` in the *Funct.* field and the transaction code in the *Object* field. Confirm your entries. The transaction text is displayed in the *Text* column.

## Edit TCD

1. Choose *Edit* → *Choose* (or double-click on the line). The transaction screen sequence input detail screen appears.
2. Enter the program name in the *Program* field, and the [screen number \[Page 47\]](#) in the *Screen* field (see [Get system data \[Page 47\]](#)).



Note that **all** screens must be entered in the test procedure in the order in which they are called by the transaction.

4. Enter the values and parameters required for the input fields for each screen:  
Position the cursor on a screen and choose either *Edit* → *Choose* (see [Field value input via screens \[Page 44\]](#)) or *Edit* → *Field list* (see [Field value input via field list \[Page 44\]](#)).
5. Go back to the function editor by choosing *Back* twice, and save the test case.

## COMMIT WORK Handling in CATT

In the detail maintenance for the TCD function, you can choose whether CATT should terminate when it reaches an implicit or explicit COMMIT WORK. Select *End CATT session without COMMIT WORK* on the detail maintenance screen.

## Maintain screens

### Effect of End CATT session without COMMIT WORK

Checkbox	Result
Not selected	The CATT session ends at any of the following events: <ul style="list-style-type: none"> <li>• a <b>COMMIT WORK</b> statement</li> <li>• a <b>LEAVE TO TRANSACTION</b> statement</li> <li>• a transaction call via menu entry (type T)</li> <li>• transactions which create batch input sessions</li> </ul>
Selected	Only <b>LEAVE TO TRANSACTION</b> of the above events ends the CATT session.



If you do not set the flag, the memory flag CATTAKTIV is set. The system field SY-BINPT is no longer initial. (see [programming notes \[Page 97\]](#))

If the flag is set, the memory flag CATTAKTIV is not set. The system SY-BINPT is initial. To show that a CATT is running, the system sets the CATTNEU memory flag.

## Maintain screens

The screen is displayed in a simulation mode in the screen flow logic detail view (for both the TCD function and the function module test). This mode replaces the screen list display from up to release 4.0B. It has the following additional functions:

- possible entries help (F4) and field help (F1) as in the original screen.
- TabStrips, table controls and nested subscreens.
- input field entry format checks.
- all radio buttons in a group are parameterized at once. Only one radio button can be selected at a time.
- The entire radio button group is always presented in radio button detail maintenance.
- when a pushbutton or tab title is selected, its OK code can become the new OK code.

### Field value assignment via screens

All fields set to input-enabled in the Menu Painter are ready for input in the screen simulation, even if the transaction dynamically set them to input-disabled.



To ensure that you only enter values in the input fields, you should put values in all fields when recording the transaction/function module.

1. Enter values in the required fields. You can either enter a value directly, or call the individual field processing.



Space characters (to delete field default values) can be passed by entering two apostrophes ( ' ' or " ", i.e. with or without space characters between the apostrophes).

2. Enter the required OK code, e.g. (/11), with the *OK code* function.
3. Return to the function maintenance screen and save your entries.

### Individual field processing

If you choose a screen field and choose *Field contents*, an individual field processing dialog box, which contains information about any field parameters and its current value, appears.


### Parameterization

Mark a screen field and choose *Insert import parameter*.

### Read field values

You define an export parameter to contain the value of the field that is to be read, with the *Read field value* function in test case screen detail maintenance.

Proceed as follows:

1. Position the cursor on the field whose value is to be read.
2. Choose .
3. Specify a parameter name. The contents of the field are passed in this parameter. If this parameter does not exist, it is automatically created as an export parameter.




The parameter is filled at the end of the transaction, so you cannot use it in the same transaction as an import value for a subsequent screen.

### Define field check

The check function checks whether the runtime value of a field is the expected value, at the end of the transaction.

The check value can be a constant or a parameter. If the check value is a parameter, the test case creator must ensure that it contains the correct check value at runtime. If a non-existent parameter is specified, an import parameter is automatically created.

You define a check with the screen detail maintenance function *Check field*, as follows:

1. Position the cursor on the field to be checked.
2. Choose .
3. Define the check in the following dialog box:
  - If you enter a value in the *Field contents* field, it is checked.
  - If you enter a parameter name in the *Parameter name* field, it is checked. If this parameter does not exist, it is automatically created as an import parameter.

You can similarly define a check on the last system message: See „[Extended system message checks \[Page 56\]](#)“.

## Maintain screens

### Field checks with patterns

Screen field checks are defined in the CATT function TCD. The check value can be a constant or the current contents of a parameter.

From 4.6C generic checks are also possible. If a check value or the current check parameter contains an '\*' at the end of the strings, a longer field would still give a positive check result.



The check is made according to the ABAP condition CP (contains pattern) rules, so it is not case-sensitive.

All wildcard characters, e.g. '\*', '+', '#' can be used with the same meaning as in ABAP conditions with the comparison operator CP between strings. See *CP (Contains Pattern)* in [Comparing strings \[Ext.\]](#)

To define a check with 'Contains Pattern' similarly to a field check:

1. Position the cursor on the field to be checked.
2. Choose
3. Define the check in the following dialog box:
  - if you enter a partial value in the *Field contents* field and add wildcard characters, a check is made against this constant.
  - If you enter an incomplete parameter name containing wildcard characters in the *Parameter name* field, it is checked. If this parameter does not exist, it is automatically created as an import parameter.



You check against the *Message text* field of the message screen SAPMSCEM 0001: The entry `USER*` in transaction FD01 Create user has a positive check result for both the messages "User created" and "User already exists".

### Cursor position

You can change the cursor position during the test with the *Cursor position* function. Enter the name of the field on which the cursor is to be positioned in the test, in the dialog box. The current cursor position is proposed (field name or line/column).

### Function codes

Use the function keys or the transaction Menu Painter function codes. [Get system data \[Page 47\]](#) tells you how to find these.

### Subscreen

Subscreens are recorded with a transaction/function module. To specify subscreen areas:

1. Display the screen simulation.
2. Choose *Subscreens*.
  - The subscreen areas are displayed by boxes.
3. Position the cursor on the top left-hand corner of the subscreen (on the period).
4. Choose *Detail*.

**Get system data**

5. Enter the module pool name and the subscreen number on the next dialog screen. Confirm your entries.

The input fields are now visible in the subscreen area.

6. Go back to the screen simulation and maintain the subscreen fields, if necessary.



You can get the name and screen number of the subscreen by calling the transaction, positioning the cursor on a subscreen field, and choosing *System* → *Status*.

**Field value input via field list**

Alternatively to entering field values on the screen, you can maintain them in the field list. It lists the fields of the selected screen, with the option of entering new field contents. You can use variables or parameters here too.

In exceptional cases, you can also use internal variables as field names (see ). [Process dynamic screens and field names \[Page 48\]](#).



Before you can edit the fields of a subscreen in the field list, you must define the subscreen. Proceed as described in Subscreen (above).



You can find the assignment of field names to the screen fields by positioning the cursor on the associated screen field in the transaction, and pressing the help key. A field help window appears. You can display the table name and the field name as well as the program name and the screen number, with the *Technical Info* function.

**Get system data**

To program or modify test cases manually you need system data. You need the program name and screen number to control the test case screen sequence. You need the function code to maintain individual screens.

To get system data for a test module, open an additional session. Run the transaction for which you want to modify the test module in this session. Run the transaction until you reach the maintenance screen for which you need the system data.

**Get program name and screen number:**

Choose *System* → *Status* in the transaction maintenance screen. The system displays all system data for this screen. The *Program (Screen)* field contains the program name and the *Screen number* field contains the screen number.

## Specify screens and field names dynamically

If subscreens are used in the screen, you get the subscreen module pool and screen number by positioning on a subscreen field and choosing *System* → *Status*. The *Program (Subscreen)* field contains the program name and the *Screen number* field contains the screen number.



If the maintenance screen is a modal dialog box, press F1 on dialog box fields to get system data. The help information appears in another dialog box. Choose *Technical info*. The data you need is in the *Program name* and *Screen number* fields.

## Get function code:

If a function key is assigned to the function, press the right mouse key in the work area to find out which one it is.

If no function code exists, click the function in the menu with the left-hand mouse key. Keep the mouse key depressed and press F1. A dialog box appears that displays the function code in the *Function* field.



Fast paths are language-dependent, and should be avoided.

Functions which require an EXIT-COMMAND (PAI at EXIT-COMMAND), are not entered with, for example =DELE in the *OK code*, but with /EDELE.

## Specify screens and field names dynamically

In exceptional cases, you may need to use an internal variable or a parameter instead of module pool name and screen number. For example, generated screens whose names or field names contain the client or table entries.



Module pool name	Screen number
SAPMF02D	0105

You can choose between:

Module pool name	Screen number
&V01	

in which case the value 'SAPMF02D0105' must be assigned to the variable &V01 in the function editor.

Module pool name	Screen number
&V01	0105

## REF - Refer to test case

in which case the value 'SAPMF02D' must be assigned to the variable &V01 in the function editor.

The screen field values can then only be input in the field list.

To create a field name choose *Create field* and assign a field name. You can also enter the field names and field values as variables or parameters. These variables and parameters must have the appropriate values at runtime.



The screen number can only be defined dynamically using the same variables as for the module pool name.

## REF - Refer to test case

The REF function is used to [modularize \[Page 12\]](#) test cases. It puts the functionality of the referred test case in the current test case. Data is passed between the current test case and the referred test case via its import/export interface.

Use the REF function to use a test case in another test case.

1. Enter **REF** in the *Funct.* field and the called test case name in the *Object* field.
2. Go to the called test case detail maintenance by positioning the cursor on this line, and double-clicking or choosing *Choose*.
3. Maintain the import interface as follows:
  - assign a variable, constant or parameter to the calling test case.
  - assign a variant of the called test case from which the import interface can be filled.
  - no assignment, in which case the default called test case import parameter value is used, if there is one.



If a combination of the parameterization described above is used, the values assigned by the calling test case overwrite all other assignments.

An assignment by a variant only overwrites the default values of the called test case.



To use an external variant to fill the import interface, specify it as a five-character number, where e.g. the first external variant (fourth line in the text file) is 00001.

4. Maintain the export interface. Choose *Export*. You can read this interface into variables and parameters in the calling test case.
5. Specify the processing mode of the called test case in the *Processing mode* field.

## REF - Refer to test case

### Go to the referred test case

To see the functions of the referred test module, position the cursor it in the function editor and choose *Goto* → *Test module*. You go to the test case display. Go back to the original test case with *Goto* → *Back*.

### Resolve reference

The REF function can also be used as a copy function. You can copy the functions of a referred test case into the current line by resolving it.

You can do this in one of two ways:

- Position the cursor in the function editor and choose *Edit* → *Resolve reference*.
- Enter ">" in the editor column and choose ENTER.

### Remote call of test cases

From Release 4.0 test cases can be run locally in the remote system. A remote flag is set and an RFC destination is specified in the REF command. The version of the test case in the remote system then runs in the remote system, not the local version. The parameter interface is assigned as usual for the REF command.

The remote run log is saved in the local system. Only the referred test case and the RFC destination are specified in the local System. Click on the RFC destination to display the remote system log.

#### Create remote call:

1. Go to the test case function editor.
2. Insert a REF command in your procedure in the editor.
3. Enter the test case to be referred to.
4. Ignore the warning TT022 'CATT <test case> not found'. Choose ENTER.
5. Choose *Edit* → *Remote REF target*.
6. Define an import parameter for the remote system at runtime. Enter the usual RFC destination as default.
7. Go to the interface of the remotely-referred test case.
8. Choose *Read interface* to read the remote system import and export interface definition.
9. Enter interface values as usual.

## FUN – use function module

You can call function modules which get values from various application areas for further processing in the test case using the FUN <function module name> function, (see also [Using function modules \[Page 51\]](#)).

1. Enter the function FUN in the *Funct.* column, and the function module name in the *Text* column in the function editor.
2. Position the cursor on this line and double-click. You go to the detail maintenance screen, which lists the function module interface parameters.
3. Enter parameter values:
  - *Import* (import parameter)  
Enter constants, parameters or variables in the fields.
  - *Export* (export parameter)  
Enter variables and parameters which are to be assigned values by the function module.
  - *Exception* (exception condition)  
If the occurrence of an exception during the run should cause termination, enter “X” in the *Value* field.
  - *Changing*  
Enter variables and parameters which are to be changed by calling the function module.
  - *Tables* (table structures)

### Import

You can only specify one record from this table (not a complete table).

### Export

You can only export the header record of this table.

You can check for expected exceptions in the function modules with SY-SUBRC.



You will find the function module interface documentation with the function module (transaction SE37).

## Use function modules

You can call function modules to check database entries, set tables, delete data, etc. in a test case using the CATT function FUN. The calls of function modules in FUN commands are generated dynamically at runtime.

If changes are made to a function module which is used in test cases which change the parameters, you must change the CATT function FUN in the procedure. Other changes are made

## TXT - Enter comment

automatically because of the dynamic generation of the CATT interfaces. Run the program **RSCATINI** to get an overview of all function modules in CATT.

The program produces a log of generated function module CALL interfaces.

## TXT - Enter comment

You can enter comment keywords to document procedure steps, with the TXT function. This is useful to explain the variables and parameters in reusable test cases.

**TXT**

Comments on the use of variables

**See also:** "Data flow documentation" in [Rules for creating and maintaining test cases \[Page 94\]](#).

## CHEERR - Check system message

You can check whether a transaction sends an expected system message in the test case with the **CHEERR <error number>** function.

Enter the **CHEERR** function directly before the **TCD** function which should send the system message. If CHEERR is used before the function **REF** it applies to the first **TCD** in the referred test module.



You expect the error message 286 in the transaction FB01:

*Period is not open for account type & and G/L &.*

Funct.	Object
CHEERR	286
TCD	FB01

The transaction is cancelled when an error occurs (processing mode **n**). If the error which occurred is the expected system message, the transaction result in the log is positive, despite the error.

If the expected system message does not occur, an error message is entered in the log. If the termination flag is set, the procedure is cancelled if the expected error does not occur.

## Allow all error messages

The following statement allows all error numbers. You can check whether the result is as you expected by checking the **SY** fields and. &M variables after the transaction has run:

Funct.	Object
--------	--------

CHEERR	*
--------	---

### Special test case system message variables

If a transaction sends more than one system message, the message variables &M01 to &M04 contain the variable parts of the last message issued. You can get the data from previous messages as follows:

#### Position on a message:

Insert the following line after the transaction call.

Funct.	Object
IF	&MSG = 'XYYnnn'

where:

X	Message type (one character: A, E, I, S or W)
YY	Message ID (two-character)
nnn	Message number (three-digit number)

The selection criterion can be entered incomplete. They are identified by the length and numbers.



&MSG = 'X'	Message type
&MSG = 'YY'	Message ID
&MSG = 'nnn'	Message number
&MSG = 'XYY'	Message type and ID
&MSG = 'Xnnn'	Message type and number
&MSG = 'XYYnnn'	Message type, ID and number

#### Position on the first message sent:

Funct.	Object
IF	&MSG = '**'

#### Position on the last message sent:

Funct.	Object
IF	&MSG = '='

#### Scroll in the list of messages sent

When you have positioned on a message with one of the above conditions, you can scroll in the list of all messages using other conditions.

**CHEERR - Check system message**

The following table contains the statements which you can use in [IF conditions \[Page 64\]](#) to scroll in the list:

Operator	Go to
&MSG = '+'	next message
&MSG = '-'	previous message
&MSG = '>'	next message according to last, incl. incomplete, selection &MSG = 'XYYnnn'

**Further information about a message**

When a messages has been found by one of the above conditions, the following variables contain further information:

Message variable	Meaning
&MS1	1. variable part (50-character)
&MS2	2. variable part (50-character)
&MS3	3. variable part (50-character)
&MS4	4. variable part (50-character)
&MST	Message type (one character:
&MSI	Message ID (two-character)
&MSN	Message number (three-digit number)
&MSP	Module pool (eight-character)
&MSD	Screen number (four-character)
&MSX	Message table index (four-character)
&MSL	Total number of messages (four-character)



The following example assumes that the following messages were sent by the TCD statement:

STT123

IAA222

EBB333

IAA344

You can go to the first message, e.g. with the following statement:

Funct.	Object
IF	&MSG = '*'
CATT statements	

CHEERR - Check system message

ENDIF	
-------	--

The message TT123 is put in the message variables.

To go to the first message in the message class AA:

<b>Funct.</b>	<b>Object</b>
IF	&MSG = 'AA'
CATT statements	
ENDIF	

The message AA222 is put in the message variables.

To go to the second message in the message class AA:

<b>Funct.</b>	<b>Object</b>
IF	&MSG = 'AA'
ENDIF	
IF	&MSG = '>'
CATT statements	
ENDIF	

The message AA344 is put in the message variables.



Call test case B2000106 after the TCD statement for a compressed transaction message log display:

Function	Object	Text
TCD	...	
REF	B2000106	...

## Message Screen SAPMSCEM 0001

In CATT Easy Mode (SCEM) a message screen with the result of the transaction is created at the end of a transaction. This message screen SAPMSCEM 0001 is now appended as the last screen of Enhanced CATT (SCAT) as well. It contains the last message in the transaction. You can make checks against this message.

See [TCD – Test transactions \[Page 42\]](#).

## Extended System Message Check

# Extended System Message Check

## Use

After each transaction, the last system message sent is logged. You can define other checks in addition to the [CHEERR – Check system message \[Page 52\]](#) check, by inserting a particular screen at the end of the screen sequence of a transaction.

## Procedure

1. Go to the CATT function editor.
2. Go to the screen sequence of a transaction (function **TCD**).
3. Go to the last screen.
4. Insert **SAPMSCEM 0001** as last screen, if this is not already the case.



The screen **SAPMSCEM 0001** is automatically appended to all test cases which are newly-recorded in Release 4.6C.

5. Double-click on this screen.
6. Define a message check. You can check the message number or the combination of message class (*Message ID*) and message number.  
Put the cursor on the field number and delete the displayed value. Choose *Check field*. Enter the expected message number or a parameter that will contain the expected message number at runtime, in the following dialog box.  
To check the message class, delete the *Message ID* field contents and choose *Check field*. Enter the message class in the following dialog box.
7. You can define other allowed messages by performing step 5 using the *Other allowed messages* fields.

## Result

You have defined system message checks. If these system messages are sent, the test is a success, even if the transaction ended with an error (check error). If you have defined several allowed messages, the transaction must send one of them during the test for it to be a success.

## CHETAB – check table contents

The function CHETAB compares database field values with specified values to check database updates after transactions. You can also check with CHETAB whether a particular table entry exists. You can also check customizing table entries.

CHETAB requires specification of the complete key.

## CHEVAR – check variable contents



If the key is not completely specified, those key fields which are not specified take the initial value for their type (spaces for CHAR, "0" for NUM).



In those exceptional cases where you need to select by non-key fields, you can call appropriately programmed function modules in test cases, with the function FUN. If the tables to be read are buffered, you should call the **CAT\_TABLE\_BUFFER\_RESET** function module with the function FUN, before the read function module. This resets a table in the table buffer.

Proceed as follows:

1. Enter CHETAB in the *Funct.* column, and the table name in the *Object* field.
2. Position the cursor on this line and double-click. You go to the field contents maintenance detail screen. At first only the key fields are listed.
3. Assign values to all the key fields (ARG). You can use constant values, or variables and parameters which already contain values.
4. Choose *Display all fields* to list the function part fields as well.
5. Enter the function field (FKT) target values, for which a target/actual comparison should be made.



Database table values must be input in the fields in internal format. Values which fail the target/actual comparison are output as errors in the log.



If the CHETAB function does not find any entries for the specified values in the corresponding table in special cases, because e.g. an account has not yet been updated, an error occurs, because CATT assumes that this is an error. You can avoid this by reading the corresponding table entry with the function SETVAR<table>. If the entry does not exist, SY-SUBRC <> 0, and this can be tested in a condition.

## CHEVAR – check variable contents

You can check the value of a variable or parameter with CHEVAR <condition> (see [Use condition \[Page 65\]](#) ).

If the specified condition is not satisfied, an error message is logged (see [Display log \[Page 86\]](#)) and if the termination flag is set, the run is terminated.

## SETVAR - Assign values to variables and parameters

# SETVAR - Assign values to variables and parameters

Values can be assigned to variables and parameters in a test case using the function SETVAR. These values remain valid for the entire test case, unless you overwrite them.

Value assignment types:



Constants must always be enclosed in inverted commas ' '.

The decimal sign is a point (period). Other separators (e.g. thousands) are not allowed.

If you need leading zeros, enter them.

## Conventional value assignment

A constant value is assigned to the variable as follows:

- to assign a constant value to the parameter or variable **&EXAMPLE**.



Funct.	Text
SETVAR	&EXAMPLE = '1000'

Result:

```
SETVAR &EXAMPLE = '1000'
&EXAMPLE = 1000
```

- Continuing the example, add the constant 20 to &EXAMPLE and assign the result to the variable or parameter &EXAMPLE2.



Funct.	Text
SETVAR	&EXAMPLE2 = &EXAMPLE + '20'

Result:

```
SETVAR &EXAMPLE2 = &EXAMPLE + '20'
&EXAMPLE2 = 1000 + 20
&EXAMPLE2 = 1020
```

- to assign the value of the first message variable (0000100211) to the parameter or variable &EXPORT.

SETVAR - Assign values to variables and parameters



Func.	Text
SETVAR	&EXPORT = &M01

Result:

```

SETVAR &EXPORT = &M01
&EXPORT = 0000100211
    
```

- to assign the string in the first message variable (0000100211) with offset (+2) and length (8) to the parameter or variable **&EXPORT**.



Func.	Text
SETVAR	&EXPORT = &M01+2(8)

Result:

```

SETVAR &EXPORT = &M01+2(8)
&EXPORT = 00100211
    
```

- to assign the text 'Insert' with offset (+2) length (6) to the internal variable or parameter **&EXAMPLE**.



Func.	Text
SETVAR	&EXAMPLE+002(006) = 'Insert'

Result:

```

SETVAR &EXAMPLE+2(8) = 'INSERT'
&EXAMPLE+2(8) = INSERT
&EXAMPLE      =  INSERT
    
```

- to append the text 'Append' to the internal variable or parameter **&EXAMPLE** from the previous example:



Func.	Text

**SETVAR - Assign values to variables and parameters**

SETVAR	&EXAMPLE+132 = 'Append'
--------	-------------------------

Result:

```

SETVAR &EXAMPLE+132 = 'APPEND'
&EXAMPLE+132 = APPEND
&EXAMPLE      = INSERTAPPEND

```

**Leading zeros and decimal places**

The **SETVAR** statement operand sequence is significant for decimal places and leading zeros in arithmetical operations.

The leading zeros are lost and the decimal point position does not behave as required by database queries in numerical variables.

Avoid this problem by setting the first operand on the right-hand-side of SETVAR value assignments and arithmetic operations correctly.

**Decimal places**

In arithmetical operations, the decimal places are set in the result according to the decimal places in the first operand.



Func.	Text
SETVAR	&V01 = '01.11' / '0000002.22222'

Func.	Text
SETVAR	&V01 = '01.11' / '0000002.22222'

(calculated: 0,4995004995...)

(result: &amp;V01 = 00.50)

If the output format truncates decimal places, commercial rounding is used.

**Assigning values from a table (SETVAR <table>)**

You can assign database table values to variables during the test procedure. You must specify the complete key.

1. Enter SETVAR in the *Func.* column, and the table name in the *Object* column.
2. Position the cursor on this line and double-click. You go to the detail maintenance screen where the table fields are listed.



For database tables, only the key fields (ARG) are initially displayed. You can display the function part (FKT) fields by choosing *Edit* → *Display all fields*.

3. Enter the complete key in the arguments part of the table (ARG). These values are passed to the table key fields during the test. You can use constants, parameters and variables.

## SETTAB - Set customizing table



If the key is not completely specified, those key field which are not specified take the initial value for their type (spaces for CHAR, "0" for NUM).

4. Enter the variables in the fields into which you will read database values, in the function part of the table (FKT).



The table variable assignment function can now calculate the sum of field contents. Enter the same variable in all the fields which you want to sum. The test procedure sums the values of these fields (e.g. sum of G/L account postings) and assigns the total to the specified variable.

You can identify SETVAR table calculations in the log by a preceding plus sign (+).

## SETTAB - Set customizing table

The SETTAB <table name> function changes customizing tables (other database tables **cannot** be changed). The previous values are saved in an internal backup table, and can be restored with RESTAB.

SETTAB can:

- change existing table records
- enter new records.



SETTAB changes table contents at test case runtime, but it does not lock the table.

1. Enter SETTAB in the *Funct.* field and the table name in the *Object* field.
2. Double-click on the line. You go to the detail maintenance. All table key fields are listed.
3. Enter the complete keys (arguments) in the *ARG* table fields.



The client table field always contains the current client.

4. Choose *Display all fields*.
5. If you want to change particular fields, enter the new value in the function parts of these fields (*FKT*) only.



After entering the complete key, the function part field contents can be read in automatically with the *Edit* → *Tables* → *Read table entries* function. You can edit these values.

## Using SETTAB and RESTAB



RESTAB \* is automatically called after the test or if an error occurs.

To prevent the SETTAB settings being reset, enter RESTAB%. This deletes all reset data for the previous SETTAB settings. Subsequent SETTABs are logged again and can be reset with RESTAB.

## Sort table fields

You can sort the function part (FKT) table fields for detail maintenance. Three sort criteria can be used consecutively with the same function:

1. Dictionary definition sequence
2. Alphabetic by field name (*Field name* column)
3. Alphabetic by keywords (*Description* column)

By default, the table fields are in their Dictionary definition sequence. The *Sort* function sorts the function part fields by the next sort criterion.

## See also:

[RESTAB - Reset table \[Page 63\]](#)

## Using SETTAB and RESTAB



If customizing standard settings are changed, they must be reset in the same test.

CATT requires a fully installed system (e.g. periods must be open). Customizing tables should therefore only be set in exceptional cases.

### Rules for using SETTAB:

- Only use the function when table setting differences, or errors caused by particular settings, are to be tested.
- CATT automatically resets a SETTAB at the end of a test. If you want to prevent this, call RESTAB% in the test case. The system then keeps all changes made using SETTAB, even after the test is finished.
- If a CATT test terminates because of the termination flag, RESTAB is automatically called for this test.
- Some errors cause the main CATT transaction to crash:
  - the user enters /N in the OK code when the transaction is running in the foreground.
  - a transaction contains the ABAP command LEAVE TO TRANSACTION...
  - a session terminates unexpectedly.

**RESTAB – reset table**

- more than 36 different function modules are called in one run by the CATT function FUN.
- a run-time error occurs in a function module which was called by FUN.

CATT cannot call RESTAB itself in these cases. Choose *Utilities* → *Call RESTAB* in the CATT initial screen.

## RESTAB – reset table

All changes made with SETTAB during a test are buffered. At the end of the test, they are automatically reset:

- If you want to reverse changes to a particular table, enter:  
RESTAB <table name>
- If you want to reverse all table changes made in a test case, enter:  
RESTAB \*

If a CATT function terminates a test, RESTAB\* is automatically called, even if the test does not call it

If you want to avoid resetting table changes, enter RESTAB% after the SETTAB statements. CATT can no longer reset changes buffered up to this point. You can reset subsequent SETTAB statements with RESTAB.

### See also:

[SETTAB - Set customizing table \[Page 61\]](#)

## DO n... (EXIT)... ENDDO loops

To repeat CATT functions, you can put them in a loop.

Enter the **DO** function in the *Funct.* field, and the desired number of loop iterations in the *Object* field in the function maintenance screen. You can enter a number between **1** and **999**.

Enter the CATT functions which are to be repeated.

End the loop with **ENDDO**.

The special variable &LPC contains the current loop counter.



```
DO 5
REF      K1100001
ENDDO
```

## EXIT – conditional termination

The test module K1100001 runs five times.



You can nest DO...ENDDO loops and [IF...ENDIF \[Page 64\]](#) blocks.

## EXIT

If a loop is to be conditionally ended, you can enter the EXIT <condition> function between **DO** and **ENDDO** (see [EXIT - Conditional termination \[Page 64\]](#)). The condition is checked in each loop iteration. If the condition is satisfied, the loop is immediately ended, and the statement after **ENDDO** is executed.

This makes loop constructs very flexible.

## EXIT – conditional termination

You can conditionally terminate a test case or a [DO n... \(EXIT\)... ENDDO loop \[Page 63\]](#) with the **EXIT <condition>** function.

Enter EXIT in the *Funct.* column, and the condition (see: [Use condition \[Page 65\]](#)) in the *Text* column. The *Object* column remains empty.

## IF ... ENDIF Conditions

You can make the execution of CATT functions depend on the contents of a variable by setting a condition for its execution.

Enter IF in the *Funct.* column, and the condition (see: [Use condition \[Page 65\]](#)) in the *Text* column. The *Object* column remains empty.

```

IF                <condition>
...               CATT Functions
ENDIF

```

The CATT functions specified between IF and ENDIF are only performed if the condition is satisfied.



You can nest IF...ENDIF blocks and [DO...ENDDO \[Page 63\]](#) loops.

You can also make the execution of functions depend on alternative variable contents:

```

IF                <condition>
...               CATT Functions
ELSEIF           <condition>
...               CATT Functions

```

ENDIF

The CATT functions specified after ELSEIF are only performed if the IF condition is not fulfilled and the ELSEIF condition is fulfilled.

You can also specify the execution of CATT functions for the case that the value of the variables do not satisfy any condition:

```

IF          <condition>
...        CATT Functions

ELSEIF     <condition>
...        CATT Functions

ELSE       CATT Functions

ENDIF
    
```

The CATT functions specified between **IF** and **ELSEIF** are only performed if the condition is satisfied. Otherwise the functions defined for ELSE are performed.

## Use of conditions

Conditions allow you to control the test flow logic. A condition has the following form:

<var1> <op> <var2>.

Valid compare operators <op>:

- EQ (=)
- NE (<>, ><)
- LT (<)
- LE (<=, =<)
- GT (>)
- GE (=>, >=)

All variables, SY fields, test module parameters, SET/GET parameters and constant values can be used as variables <var1> and <var2>.



Function	Object	Text	
IF		&V01 = &V02	compare two variables
IF		&JHR GT &D01(04)	year and date variable length 4
EXIT		&V01 <= '5'	variable and constant
CHEVAR		SY-SYSID = 'S11'	SY field and constant

## Schedule Execution Wait Time

# Schedule Execution Wait Time

## Use

You can define waiting times between CATT functions or transaction screens in a CATT test case.

## Waiting Times Between CATT Functions

At any place in the test script.

The current work process is released during the wait time. This can be useful in the following cases:

- If a transaction uses V2 update, it is also asynchronous in CATT, so the data may not be available for a following transaction.
- If a transaction updates in buffered tables, user function modules (function FUN) may access this data before they have been updated.



CATT functions avoid the buffered table access problem automatically.

## Procedure

1. Define the parameter `&LWT` as a local variable.
2. Assign a number or a parameter to the parameter at the points where you want to wait. The number is the wait time in seconds:

Function	Object	Text
SETVAR		&LWT = '5'

3. Run the test procedure. The test is interrupted for the specified time at each point at which a value is assigned to the variable. The log is as follows:

```

|  SETVAR &LWT = '4'
|  &LWT = WAIT 4 SECONDS

```

## Waiting Times Between Screens

The waiting time must be defined in the test script before the transaction call (TCD or function module test with dialogs). This waiting time remains active until you change the definition.

The current work process is released during the wait time. This can be useful in the following cases.

## Procedure

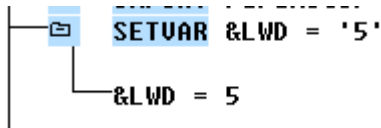
1. Define the parameter `&LWD` as a local variable.
2. Assign a number or a parameter to the parameter at the points where you want to wait. The number is the wait time in seconds:

Function	Object	Text
SETVAR		&LWD = '5'

- Run the test procedure. A waiting time is inserted from this position between transaction screens or function modules with dialogs. The waiting time in seconds is given by:

$$T = \left( \sum \text{Felder} + \sum \text{Eingegebene Zeichen} \right) * \&LWD / 10$$

- The log is as follows:



## Parameterization

### Use

You enter data in SAP input screens in all business processes which are modelled in test cases. They are used to create master data, documents, orders, etc. To make test cases flexible, it is better to parameterize these input fields rather than use constants, and to perform calculations with these parameters or variables where appropriate.

### Integration

Almost all CATT functions require input values and return results. Examples:

- Transaction call and function module call with dialog**  
 Screen field entries result in database changes. The result of the transaction can be, for example, the return of a success message with master record or document number.
- Function module call**  
 A function module performs a function (e.g. determine fiscal year), whose result is returned in an export parameter.
- Check customizing or database table contents**  
 The key for reading a table entry can be assembled from variables. Checking field contents against values which you provide returns a positive or negative result.
- Refer to a test case**  
 You pass data to a referred test case in constants, parameters or variables. They can be used, e.g. for conditions or as input values for the transaction to be tested. The referred test case usually returns a result which can be read in its export parameters by the referring test case.

### Functions

- Test case parameters, which are visible in the test case parameter maintenance screen:
  - Import parameters: Test case interface via which you can pass values when you call the test case.

## Parameters

- Export interface: Test case interface via which you can pass the test case result (e.g. master data ID). The calling test case can read the result.
- Local variables: variables for calculating and buffering values in the test case.
- Special variables:
  - variables which contain important CATT or system runtime information.
  - variables for test case texts which are to be translated (international tests)
  - date variables
  - SET/GET parameters

## Parameters

Parameters define the test case import and export interfaces. Local variables can also be used to buffer values in a test case. Parameters have length 132 internally.

- **Import parameters**

Import parameter values are passed when a test case is called, and are available **locally** in the test case. They can have default values which are used if no other value is passed in the call.

- **Export parameters**

You can assign the test case result to an export parameter, which can be passed to a variable in the calling test procedure when you return from a test module.

- **Local variables** are used in test cases for calculations or to receive export parameters from referred test cases or called function modules.

### Parameter and local variable naming convention:

You can use any twelve-character names, preceded with an &.



The following names are already used by CATT and cannot be used as parameter or variable names:

&Xnn, where nn is a two-digit number and X is 'T' or 'M'



You can still use the CATT parameters and internal variables which were supported up to Release 3.X. From 4.5A they have a length of 132.

## Define parameters

You can use parameters at the following places in a test case:

**Define parameters**

- screen simulation dialog parameters (TCD function detail screen or type F test cases)
- value calculations and checks (SETVAR and CHEVAR functions)
- function module interface parameters (FUN function detail screen)
- read and check tables (CHETAB and SETVAR <table> functions detail screen)
- define conditions (IF; EXIT)
- REF function variant and next screen parameters

By double-clicking on an existing parameter name, you go to the parameter detail screen, where you can enter:

- import parameter default values.
- reference ABAP Dictionary fields for parameters, in the *Data element* input field. The data element short texts are copied.
- a parameter text which differs from the data element short text with the *User short text* function. You should only use your own short texts if the ABAP Dictionary short text descriptions do not apply.

## Procedure

### Definition on the screen input field maintenance screen

1. Go to the CATT maintenance transaction function editor.
2. Position the cursor on the line which calls the transaction to be parameterized (TCD command) and choose *Choose*.  
You go to the test screen sequence definition.
3. *Choose* the screen on which the parameterized values are to be passed.  
You go to the screen definition containing all input fields and any constant input values.
4. Put the cursor on the input field for which you want to define a parameter.



To specify your own parameter name, enter & followed by the parameter name in the appropriate field. Otherwise the input field data element name is used as parameter name.

5. Choose one of the following functions, depending on which type of parameter you want to insert:
  - insert import parameter
  - insert export parameter
  - insert local variable

The parameter is created. The associated field is no longer ready for input. If the field had previously contained a constant, it is the default value when you define an import parameter.

### Definition on CATT function detail screens (e.g. REF, FUN)

1. Go to the CATT maintenance transaction function editor.

## Assign values to parameters

2. Position the cursor on the function for which you want to maintain detail information and choose *Choose*.

You go to the detail maintenance screen.

3. *Choose* the input field for which you want to define a parameter.

The parameter is created. If there was a default value in the import interface of a referred test module (REF function only), it becomes the default value.

## Definition in test case maintenance screens

1. Go to the maintenance screen (e.g. editor).
2. Enter a twelve-character parameter name beginning with &.
3. Confirm your entry with **ENTER**.
4. Confirm the dialog box which appears.
5. You can choose the parameter type on the *Maintain test module <test case name> import parameters* dialog box. You can also define a default value or your own short text, depending on the type. You can specify a data element via which ABAP Dictionary documentation is read in.

Choose *Continue* after defining the parameters.

You have now defined a new parameter or variable.



If your parameter refers to a date field, you can handle it like a [date variable \[Page 79\]](#).



If the data element documentation is not appropriate for the parameter, you can define your own short text. Double-click on the parameter. You go to the dialog box *Maintain test module import parameter <test case name>*. Choose *User Short text* and enter a short text for the parameter.

## Assign values to parameters

### Assigning values in a test case:

The system assigns values to the system variables and CATT special variables automatically. You can assign values to the remaining variables as follows:

- enter a variable in a table field (**see also:** [SETVAR <table> \[Page 58\]](#)).
- with the [SETVAR \[Page 58\]](#) function, with which you can also calculate and perform string operations.

Assign values to parameters

- via the FUN and REF function export interfaces.
- Choose *Goto* → *Text variables*, to set text variables (&Tnn).

### Refer to or run test case

You must ensure that the import and export interface has the correct values when you refer to or run test cases, as follows:

- [Function REF \[Page 49\]](#)  
Go to the detail maintenance screen by double-clicking on the line. You can maintain the import and export parameters used by the referred test case.  
Constants (not for export parameters) and parameters are allowed. The import interface values can also come from a variant of the referred test case.
- [Individual execution screen \[Page 81\]](#)  
You can assign one-off valid values to the test case import parameters.
- [Variants \[Page 72\]](#)  
You specify sets of test case import parameter runtime values in variants.

### Import parameter default values

The use of default values reduces the test case maintenance effort. The default values are overwritten by values passed during the test.

Choose *Goto* → *Parameters*. You go to the import parameter maintenance screen. You can enter import parameter default values.

Examples of the resolution of default and passed import parameter values:

Default value	Import parameter contents
'abc'	abc
space	not defined on the screen
"(2 inverted commas)	space
SY-xxxx (system variables)	SY-xxxx contents
&JHR (standard variables)	&JHR contents

Value passed	Import parameter contents
'abc'	abc
&Vnn, &Dnn (variables)	&Vnn contents, &Dnn contents
(Parameter)	&Inn contents, &Enn contents (from calling procedure or test module)
' (1 inverted comma)	not defined on the screen or initial in tables
"(2 inverted commas)	space
SY-xxxx (system variables)	SY-xxxx contents
%xxx	%xxx contents
&JHR	&JHR contents

The following uses of variables in default values can lead to errors:

Default	Import parameter contents
(internal variables)	not resolved

## Variants

(Parameter)	not resolved
' (1 inverted comma)	error
%xxx (SET/GET parameters)	not resolved

## Variants

### Use

You can pass values to a test case at runtime in import parameters which can, e.g. be put in transaction input fields. This makes use of test cases flexible.

You can store sets of values which you often want to pass to the import parameters at runtime, in variants. You then only need to specify the variant name at test case runtime.

### Prerequisites

Before you can create test case variants, you must have created test case import parameters.

### Functions

You can maintain variants in R/3 or [locally on the presentation system \[Page 74\]](#). You can choose between [individual entry \[Page 73\]](#) and fast entry in R/3, depending on the number of variants to be maintained.

You can decide which variant to use when you call a test case.

### Activities

When a variant runs, the system checks for each import parameter whether a value is defined for it in the variant. If so, this value is passed to the parameter at runtime. Otherwise, the parameter default value is used. If there is no default value, the parameter has the initial value.

You can use the following values to define variants:

normal entry	the parameter takes the entered value
no entry	the parameter default value is used.
''	the parameter is initialized
'	the parameter is not used. if the field for this parameter is filled by SET/GET parameters, these parameter values are used.
'!'	the field in which the parameter is used is initialized (e.g. to delete SET/GET parameters)

## Edit Variants in R/3

### Use


Only edit variants in R/3 if you only want to edit individual variants (individual entry) or a limited number of variants (quick entry). For a large number of variants, use [external maintenance \[Page 74\]](#).

### Procedure

#### Individual entry

1. Enter the test case name in the CATT maintenance transaction initial screen
2. Choose *Goto* → *Variants* → *Edit*. You go to the test case variants display screen.
3. You can now create a new variant or edit an existing one:

#### Create

1. Choose . You go to the new variant maintenance screen.
2. Copy the default number or enter any other eight-character name.
3. Enter a short description in the input line next to the number.
4. Continue from step 2 under *Edit*.


#### Edit

1. Choose a variant from the list with double-click.  
You go to the variant detail screen.
2. All import parameters which are used in the test case are listed in the *Import parameters* area. Enter values for the desired parameters.



Import parameters that are used in transaction input screens can be maintained in the input screens. Choose *Maintain in screens*. You go to the initial screen of the first transaction called by TCD, and can navigate among the screens with the scrolling functions. Only parameterized fields are ready for input. If the parameters have default values, they are displayed. If you change a value, it is the value for the current variant.

Go back to the current variant maintenance screen with *Back*.

3. Save your entries.
4.  returns to the variant maintenance initial screen.







#### Quick entry

1. Enter the test case name in the CATT maintenance transaction initial screen
2. Choose *Goto* → *Variants* → *Edit*. You go to the test case variants display screen.
3. Choose.  
All existing variants are displayed in a table in a dialog box.

## Edit Variants Externally

4. All existing variants are displayed in a table in a dialog box. The standard format of the table is:

Variant	Description	Input field 1	Other input fields
Variant ID	Short description of the variant	Value for input field 1	a cell for each additional input field

5. Edit the existing variants by entering new values in the input field columns.
6. Add new variants as follows:
-  copy existing variant
  -  insert new variant
5. Load the variants locally on your PC () and edit them with a desktop application.
6. Import the variants from an external text file () .
7. Delete one or more variants by selecting them and choosing  .
8. End fast entry with  .

## Edit Variants Externally

### Use

External variant editing allows you to create variants for the test case import parameters in a table calculation program such as Microsoft EXCEL.

### Prerequisites

The data is in a text file whose elements are separated by tabs.

### Procedure

1. Export your test case variants using *Goto → Variants → Export*.



If you have not yet created any variants for the test case, you can create a text file containing all test case parameters and their short texts and default values, with *Goto → Variants → Export defaults*. You can define your variants in this text file.

2. Enter the path and file name in which the file is to be saved in the *Copy to local file* dialog box.
3. Edit the file with an external program in the appropriate format.
4. Save the file as a text file with tab column separators.
5. Close the file in the external program.

## Create where-used list

6. Import the edited file with *Goto* → *Variants* → *Import*. You can only choose this function in test case maintenance change mode.
7. Enter the path and file name in which the file is to be saved in the *Read from local file* dialog box.
8. Save the test case.

## File format

The data file of the external variant is a text file whose elements are separated by tabs.

The file contains the following columns:

Column	Meaning
[Variant ID]	Variant ID
[Variant text]	Variant short text
&<parameter>	Test case import parameter. <parameter> must be the test case parameter name.

The first row contains the column headers (as in the table).

The second row contains the default parameter values.

The third row contains a note on file use.

From the fourth row you can define variants.

## Create where-used list

## Prerequisites

Save test case changes before creating a parameter where-used list.

## Procedure

1. Go to the parameter and variable maintenance screen.
2. Choose the parameter for which you want a where-used list.

3. Choose *Parameter use* ()

You get a list in which all uses of the parameter are highlighted in color.

4. Go back to the parameter and variable maintenance screen.



You can print the overview with *Test procedure* → *Print*. Uses are printed **bold**.

## Special variables

# Special variables

You can use the following special variables in test cases:

- [Message variables \[Page 76\]](#)  
After successful completion, transactions send a dialog message to the user, which you can analyze using the message variables.
- [Text variables \[Page 77\]](#)  
These variables can be used to make fields language-dependent.
- [System variables \[Page 78\]](#)  
The system variables provide system information.
- [CATT special variables \[Page 78\]](#)  
You can also use CATT's own special variables, as well as the general system variables, to access system data. These can be inserted directly in a character string (literal).
- [Date variables \[Page 79\]](#)  
For date calculations

## Message variables

Use message variables to read the variable contents of system messages. System messages contain up to four variables, so there are four message variables in test cases.

Message variables start with &M followed by a two-digit number (01 - 04). Variable &M01 refers to the first variable component of a system message, variable &M02 to the second, and so on.

If you want to use the contents of these variables for processing, you must pass them to local variables or export variables.

Other message variables, with which you can get further information about system messages, are in [CHEERR - check error message \[Page 52\]](#) under "Special test case system message variables".



The transaction "Create Vendor" sends a success message containing the vendor number. Get that number in your CATT test module.

1. Run a test case which tests the transaction **FK01**
2. In the log, branch to the note *TCD*.

SET/GET parameters

You get a transaction success message. It consists of the message ID followed by the message text [log \[Ext.\]](#).


3. Click on the message.

```
Vendor  was created in company code
Message no. F2 271
```

When comparing this message text with the one in the log, you will see that the account number and the company code number are replaced with blanks. These blanks are the place holders you can read using the message variables. The system puts the vendor number in the first message variable &M01 and the company code in the second message variable &M02.

## SET/GET parameters

You can get the names of the parameters by positioning the cursor on the application field and pressing F1. You can display the parameter with *Extras* → *Technical info*. SET/GET parameters begin with “%”, which is usually followed by three letters.



Function	Object	Text
SETVAR		&V01 = %BUK

%BUK: Company code



SET/GET parameters must be specified in upper-case letters. They should be used sparingly, because they make demands on the SET/GET memory and can lead to errors.

**See also:** [Notes and recommendations \[Page 93\]](#)

## Text variables

The system provides 99 text variables (&T01 - &T99) with a length of 50 characters for each test case. You use them to process language-independent fields. You can pass the contents of the variables to screens, function modules or tables. If the target field is shorter than 50 characters, the system truncates the string.

## System variables

The translation tool recognizes the text variables, so you can translate them into the required languages.

## System variables

The system variables contain system information. You can use all the system SY fields in CATT. The names of the **SY** fields are in the Dictionary structure **SYST**.



Function	Object	Text
SETVAR		&V01 = SY-LANGU

Examples of system fields:

- **SY-LANGU** (logon language)
- **SY-UZEIT** (current time)
- **SY-DATUM** (current date)
- **SY-UNAME** (user name)

## CATT special variables

As well as the CATT-independent system variables, you can also use special variables.

Special variables can be included directly in a character string (literal).

You cannot use variable offset and length as usual, e.g. &M01+02(08).

- **Client &MND**

This is the same value as the contents of the system variable SY-MANDT.

- **System date &DAT**

The date is in &DAT (see date variables) in external format, without format characters. You define which external format is effective, as a default value in the user master record.

In contrast to &DAT, SY-DATUM is always in the internal format: YYYYMMDD

- **User name &NAM**

This is the same value as the contents of the system variable SY-UNAME.

- **RFC target system &LDS**

The current RFC target system on which a remote test is running.

- **Years**

- &JHR      current year

- &JHV previous year
- &JHN next year

An immediately following entry +nn or -nn increases or decreases the year by nn.

- **Current loop counter: &LPC**

The variable &LPC contains the current value of the innermost loop count in DO/ENDDO loops.



The special variables must be specified in upper-case letters.

## Date variables

You can calculate (add and subtract days) with date variables ( &D01 to &D10) and parameters which are based on a date data element.

### Date formats

You can use the following date formats:

- Internal format: YYYYMMDD
- External formats with and without formatting characters:
  - DD.MM.YYYY DDMMYYYY
  - MM/DD/YYYY MMDDYYYY
  - MM-DD-YYYY MMDDYYYY
  - YYYY.MM.DD YYYYMMDD
  - YYYY.MM.DD YYYYMMDD



Define your external format in your user master record (default values).

### Setting date variables

Set date variables as follows:

- a constant value in any valid date format (the format is checked)
- from a SY-DATUM field
- from the CATT special variable &DAT
- from another date variable
- from any other CATT variable in any date format

## Execution

- from the export parameters of a referred test case.

## Passing date variables

The result of an assignment to a non-date variable is always in external format.

## Arithmetic operations with date variables

You can use the following arithmetical operations:

- Date calculation:**

If the first operand is of type DATE and the second is of a different numerical type, a date calculation is performed, i.e. the date in the first operand is incremented by the value of the second operand.

SETVAR	&D01 = SY-DATUM + '30'
	&D01 = 19931215 + 30
	&D01 = 19940114
SETVAR	&V01 = '5'
	&V01 = 5
SETVAR	&D02 = &D01 - &V01
	&D02 = 19940114 - 5
	&D02 = 19940109

- Day difference:**

If both operands in a subtraction are of type DATE and the result is not a date variable, the positive and negative day difference is calculated.

## Check and compare date variables

You must specify the execution conditions for the following CATT functions (IF ... ENDIF), for cancellation (EXIT) of a DO/ENDDO loop, and variable contents check conditions (CHEVAR) in the usual form (<variable1> <operand> <variable2>) for date variable checks and comparisons ([See also: Use of conditions \[Page 65\]](#)).

If at least one of the two variables is a date variable, a date comparison is made. The non-date variable or constant is automatically converted into date format.



SETVAR	&D01 = '31011994'
	&D01 = 19940131
CHEVAR	&D01 < SY-DATUM
	19940131 < 19940117

Condition ( &D01 < SY-DATUM ) is not satisfied

## Execution

You can choose between:

## Individual Execution

- [Individual execution \[Page 81\]](#)  
The test case is executed directly from the maintenance transaction via an execution screen in which you can enter execution information.
- [Collective execution \[Page 82\]](#)  
You can run several test cases consecutively.
- [Remote execution \[Page 84\]](#)  
Test cases can be executed remotely in another R/3 or R/2 System by entering an RFC destination.

If your test case contains [external applications \[Page 23\]](#), see [Running external applications \[Page 25\]](#)

## Individual Execution

You can run one test case as follows:

1. Enter the name of the test case in the initial screen and choose *Test procedure* → *Execute*. You go to the execution screen.
2. Specify:
  - [log \[Page 85\]](#) type.
  - test case **processing mode**  
The processing mode only affects the execution of transactions in the CATT function **TCO** and dialog function modules.
  - **foreground**  
The test case runs in dialog. You can correct field entries or influence the test by entering OK codes. Go to the next screen with **ENTER**.
  - **background**  
The transaction runs without dialog.
  - **only display errors**  
The transactions run without dialog until the first error or termination. They then switch to dialog. You can now change any incorrect entries. When you confirm your entries, the transaction continues without dialog until the next error.
  - **Variants**  
If the chosen test procedure has [variants \[Page 72\]](#), you can specify:
    - *'none'*: No variants are executed.
    - *'All'*: All variants are executed.
    - *'special, generic'*: Variants can also be selected generically.

### Collective execution

- 'external from file': a [file \[Page 74\]](#) specified on the presentation system is uploaded to the application system at runtime, and all variants defined in it are executed.
3. Enter any import parameter values that you want to change for this run.
  4. Choose *Test procedure* → *Execute*.

A log is produced at the end of the test procedure if you did not set the *no log* flag in the initial screen. The System displays the short log directly if no errors occurred. If errors occurred, a long log is displayed from the module in which the error occurred.




If you changed input values in processing modes "foreground" or "only display errors", they do not appear in the log.



You must confirm system messages of type 'I' with ENTER before you can continue. This also applies to messages that only appear in the status bar. Type I system messages are ignored in background processing mode.

## Collective execution

You can use the [CATT management \[Page 14\]](#) collective execution function to run several test procedures together.

1. Choose *Utilities* → *Collective execution* in the maintenance transaction initial screen. You go to the CATT management overview screen.  
The collective execution selection screen appears.
2. Choose the execution criteria. You can specify further criteria in addition to general selection criteria, such as application or creator of the procedure:
  - *Test status*  
You can select by the test status with the *Since* date. The *Since* date specifies the time at which a test status was last assigned.  
  
If you want to execute all test procedures, regardless of their test status, you must enter an asterisk (\*) in the test status field and delete the *Since* date.
  - *Variants*  
You can specify here whether you want to execute all existing variants as well, or only a particular variant (of each selected test procedure).
  - *Sequence control*  
A sequence can be defined for a test procedure in the *Context* field in the CATT attributes (see also the notes below).
  - *Execution*

You specify execution details here.

3. Enter any global parameter values. The parameters of all test procedures are given this value.
4. Save your entries.
5. Choose *Program* → *Execute* (or *Execute + Print*, or *Execute in background*).

## Test status

To be able to assign a test status, the test status assignment must be set in the user defaults (see also [Authorization \[Page 15\]](#)). The test status is assigned in the test case management data (in the attribute maintenance screen *Management data* function, *Test data* area, *Status* field), after the test run. You can see this test history via *Management data*. It takes one of the values "PASS", "FAIL" and "Untested".



The test status should only be set in final tests.

You can choose selection criteria in the F4 possible entries help.

Variant test statuses are taken into account if you set the option *with variant test status*. Otherwise, the assigned test status applies only to the default parameterization. The variant test status is evaluated as follows:

- If the default parameterization, or at least one variant has the test status "Untested", the selected test procedure has the status "Untested".
- If there is no "Untested" status, the test procedure is "FAIL". If the status is not "Untested", and either the default parameterization, or at least one variant has the test status "FAIL", then the test procedure has the test status "FAIL".
- The procedure only has the test status "PASS" when the default parameterization and all variants have the test status "PASS".

You can call up a list of all current test data for the procedure, and all variants in all languages, in the management data window by choosing *All test data*.

The three list elements provide the following functions via F2:

- *Procedure number*  
contains the complete test data history, including all variants and languages.
- *Variant number*  
contains the test data history of a variant in all languages.
- *Session number*  
allows you to go to the log.

## Sequence control

If you specify the sequence, all CATT procedures run in the order selected. If the selection contains CATT procedures which are flagged as leader or trailer, this is taken into account in the run sequence.

If you mark the field *Total leader/Total trailer*, all CATT procedures with the leader flag **1** or **2**, or the trailer flag **8** or **9** are included, independently of the selection.

## Remote execution

If you also want to choose CATT procedures with leader values **3** or **4**, they must satisfy the selection.

## Execution

If you mark the field *List processing required* in the *Prepare CATT collective test* screen, you receive a list of all selected test procedures the first time that you choose *Execute*. You can exclude procedures from the collective execution by removing them from this list by double-click.

By selecting *with runtime statistics*, a minimum, mean, and maximum run-time is output for each CATT procedure, provided you have also marked the field *List processing required*.

These data are the result of previous time measurements, which were made when this CATT procedure was executed in the current system. If the CATT procedure was either never executed in the current system, or only a long time ago, no run-times are output.

You can also specify the processing mode and the log type (see Individual execution). The default values are processing mode **n** (background) and log type **s** (short).

You can make entries in the *Session* and *Short description* fields so that you can later locate the log again via the log history.

You specify how many minutes run-time is to be available for the execution of the selected CATT procedures in the *Maximum run-time in minutes* field. If the limit is reached, the CATT procedure currently being executed is completed, and the collective test is then immediately ended. The CATT procedures planned to follow are no longer executed.

The default value **0** has no effect on the run-time available.

## Remote execution

Test cases can either run locally in the system which contains the test scripts, or remotely in another system. A remote system is accessed via the SAP Remote Function Call (RFC) interface. A remote system is identified by the RFC destination which contains the system name, the client and the user name which you use in this system. You can edit RFC destinations in the SAP system with the transaction SM59. The remote system must not necessarily be a different system, it could also be another client in the same system.

The SPA/GPA parameter RFC ID determines whether CATT is executed locally or in a remote system. If this parameter is empty, CATT is executed locally; if it contains a valid RFC destination, it is executed remotely. You can set the parameter globally, so that its value is always assigned at logon.

Choose *System* → *User profile* → *Own data*. Choose the *Parameter* tab. Enter the parameter RFC ID in the *Parameter* column and the remote system in the *Value* column.

You can also set the parameter individually at run time. The value specified remains valid until you logoff.

Enter a test case in the CATT main screen (transaction SCAT). Choose *Execute*. You go to the automatic test case execution screen. Enter a valid destination in the *R/3 RFC destination* field and choose *Execute*.

When running automatic test cases directly with the CAT\_START function module, the RFC destination for the current run can be passed in the RFC\_DEST parameter.

## Data and parameters in remote execution

CATT functions are defined in test cases in the management system (system 1). Function modules are called remotely in the execution system (system 2) and they perform the CATT functions locally. The values in the remote system are used for the system variables used in CATT functions, e.g.

SY-MANDT = &MND

SY-LANGU = &SPR

SY-UNAME = &NAM

SPA/GPA parameters are also processed with CATT functions in the remote system, i.e. each parameter value is read live from the remote system. When they are set, each value is stored in the remote system memory.

## Authorization

At run time the destination client in the remote system is first checked for CATT authorization. Set this flag when the client is created.

You need the usual authorizations for executing test procedures in the remote system target client, for the authorization object S\_DEVELOP. This authorization cannot be restricted to certain test cases. You must be authorized to execute all test procedures, as the value '\*' is checked. The authorizations required in transactions and function calls are also checked in the remote system, as required.

## Logging

All test cases are logged. The log is hierarchically structured according to the test cases used, and displayed as a structure with nodes.

The test case (or several in the case of collective execution) is displayed on the first level. The functions called, the parameters and variables, and test cases used, are listed below. You can also choose test case attributes. Errors are displayed with their messages.

There are two log types:

- **Long**

All CATT function data are logged. If an error occurs, a long log is automatically created, beginning from the module where the error occurred, even if you chose the option *no log* in the initial screen.

- **Short**

If no errors occur, the log only contains information about the functions called by the test case, and parameter contents.

You can specify the log type for a test when it is performed.

You can also display the run times.

## Display log

### Display log

At the end of each test, a detailed test result log is created. A log is displayed at the end of the test run if you specified in the execution screen that you wanted one (short or long).

To get an existing log:

1. Go to the selection screen for logs in the system via *Utilities* → *Logs*.
2. Enter selection criteria and search. A list of all test logs which satisfy the selection criteria is displayed.
3. Choose a test log by putting the cursor on its line and choosing *Choose* (or double-click). The log is displayed in a hierarchical structure in which you can navigate. The test case (or several in the case of collective execution) is displayed at this level.
4. Display the test data for the test case. Position the cursor on a test case. To see the log elements under this node, choose either *Expand nodes*, or double-click on the symbol in front of the node title. All procedures executed are displayed, with any error texts.
5. You can print out a view of your choice.



If the job RSCATDEL is scheduled, logs are deleted after 14 days. To keep a log longer in the System, change the expiry date manually. Choose *Goto* → *Procedure attributes* in the log. You go to a dialog box in which you can enter an expiry date.


### Analyze Log


#### Use

The log documents a test. Errors are highlighted in color.

#### Procedure

The log appears automatically after a test. To display logs of previous tests:

1. Go to the test case editor.
2. Choose *Utilities* → *Display log(s)*.
3. Complete the following selection screen and choose .  
A list of all test logs which satisfy the selection criteria is displayed. If there is only one log, the list is skipped. Continue at 5.

4. Choose a log by double-click.  
The log is displayed.
5. Analyze the log.
6. Leave the log with .

## Result

Tests are logged. The log is hierarchically structured according to the test case used, and displayed as a structure with nodes.

The test case is displayed on the first level of the log. The transactions called, the screens used and the entries, outputs and checks in these screens are listed below. Errors are displayed with their messages.

### Logs with no Errors

```

00626712 00:00:02 STA_P99_MY_T 06.12.1999 15:41:55 P99 006 EN EDINGERM 0001 0000 0000 0001
46B pwnd0011 Windows NT ORACLE
├── P99_MY_TEST 00:00:02 FK01 Create Vendor (Accounting) EDINGERM
│   ├── TCD FK01 Create Vendor (Accounting) [00:01 min N]
│   │   └── SF2271 Vendor 0000100007 was created in company code 0001
│   │   ├── SAPMF02K 0105 Vend.Mastr: Create RF View Request Screen
│   │   ├── SAPMF02K 0111 Customer Master: General Data, CAM Address, Communication
│   │   ├── SAPMF02K 0120 Vend.Mastr: General Control Data
│   │   ├── SAPMF02K 0130 Vend.Mastr: General Payment Transactions Data
│   │   ├── SAPMF02K 0210 Vendor Master: Company Code Data Accounting Information
│   │   └── SAPMSCEM 0001 Transaction message

```

### Logs with Transaction Errors

```

00626710 00:00:01 STA_P99_MY_T 06.12.1999 15:39:08 P99 006 EN EDINGERM 0001 0001 0000 0001
46B pwnd0011 Windows NT ORACLE
├── P99_MY_TEST 00:00:01 FK01 Create Vendor (Accounting) EDINGERM
│   └── Error EF2162 in application transaction FK01
│       └── Vendor 100006 already exists for company code 0001
│   ├── TCD FK01 Create Vendor (Accounting) [00:00 min N]
│   │   └── EF2162 Vendor 100006 already exists for company code 0001
│   │   ├── SAPMF02K 0105 Vend.Mastr: Create RF View Request Screen
│   │   ├── SAPMF02K 0111 Customer Master: General Data, CAM Address, Communication
│   │   ├── SAPMF02K 0120 Vend.Mastr: General Control Data
│   │   ├── SAPMF02K 0130 Vend.Mastr: General Payment Transactions Data
│   │   ├── SAPMF02K 0210 Vendor Master: Company Code Data Accounting Information
│   │   └── SAPMSCEM 0001 Transaction message

```

Errors are marked red. If the test case contains several transactions, only those with errors are marked red.

## Analyze Log


## Log with Positive Field Checks

```

00626713 00:00:02 STA_P99_MY_T 06.12.1999 15:43:55 P99 006 EN EDINGERM 0001 0000 0000 0001
46B pwndf0011 Windows NT ORACLE

P99_MY_TEST 00:00:02 FK01 Create Vendor (Accounting) EDINGERM
├── TCD FK01 Create Vendor (Accounting) [00:01 min N]
│   └── SF2271 Vendor 0000100008 was created in company code 0001
├── TCD FK03 Display Vendor (Accounting) [00:01 min N]
├── SAPMF02K 0106 Vndr mastr: Request screen Disp/Chng FI view
├── SAPMF02K 0111 Customer Master: General Data, CAM Address, Communication
│   ├── BDC_CURSOR = RF02K-LIFNR
│   ├── BDC_OKCODE = /00
│   ├── BDC_SUBSCR = SAPLSZA1 0300 ADDRESS
│   ├── BDC_SUBSCR = SAPLSZA1 0301 COUNTRY_SCREEN
│   └── ADDR1_DATA-COUNTRY ✓ = DE
├── SAPMF02K 0120 Vend.Mastr: General Control Data
├── SAPMF02K 0130 Vend.Mastr: General Payment Transactions Data
├── SAPMF02K 0210 Vendor Master: Company Code Data Accounting Information
├── SAPMF02K 0215 Vend.Mastr: Company Code Paymt Transctn Data
└── SAPMSCEM 0001 Transaction message

```

A successful check is indicated in the log by the color green and the  icon. The field value is displayed in the log.


## Logs with Target/Actual Comparison Errors

```

00626714 00:00:02 STA_P99_MY_T 06.12.1999 15:45:33 P99 006 EN EDINGERM 0001 0001 0000 0001
46B pwndf0011 Windows NT ORACLE

P99_MY_TEST 00:00:02 FK01 Create Vendor (Accounting) EDINGERM
Error ATT297 in application transaction FK03
Screen field ADDR1_DATA-COUNTRY Expected value: EN Actual value: DE
├── TCD FK01 Create Vendor (Accounting) [00:01 min N]
│   └── SF2271 Vendor 0000100009 was created in company code 0001
├── TCD FK03 Display Vendor (Accounting) [00:00 min N]
│   └── ATT297 Screen field ADDR1_DATA-COUNTRY Expected value: EN Actual value: D
├── SAPMF02K 0106 Vndr mastr: Request screen Disp/Chng FI view
├── SAPMF02K 0111 Customer Master: General Data, CAM Address, Communication
│   ├── BDC_CURSOR = RF02K-LIFNR
│   ├── BDC_OKCODE = /00
│   ├── BDC_SUBSCR = SAPLSZA1 0300 ADDRESS
│   ├── BDC_SUBSCR = SAPLSZA1 0301 COUNTRY_SCREEN
│   └── ADDR1_DATA-COUNTRY ✘ = EN
├── SAPMF02K 0120 Vend.Mastr: General Control Data
├── SAPMF02K 0130 Vend.Mastr: General Payment Transactions Data
├── SAPMF02K 0210 Vendor Master: Company Code Data Accounting Information
├── SAPMF02K 0215 Vend.Mastr: Company Code Paymt Transctn Data
└── SAPMSCEM 0001 Transaction message

```

The error is indicated by the color yellow and the  icon. The target and actual values are logged in that order.

### Logs with Actual Value Read Errors

```

00626716 00:00:02 STA_P99_MY_T 06.12.1999 15:53:49 P99 006 EN EDINGERM 0001 0001 0000 0001
46B pwdf0011 Windows NT ORACLE

P99_MY_TEST 00:00:02 FK01 Create Vendor (Accounting) EDINGERM
Error ATT296 in application transaction FK03
Screen field ADDR1_DATA-HOME_CITY not checked

TCD FK01 Create Vendor (Accounting) [00:01 min N]
SF2271 Vendor 0000100011 was created in company code 0001

TCD FK03 Display Vendor (Accounting) [00:00 min N]
ATT296 Screen field ADDR1_DATA-HOME_CITY not checked

SAPMF02K 0106 Vndr mastr: Request screen Disp/Chng FI view
SAPMF02K 0111 Customer Master: General Data, CAM Address, Communication
  BDC_CURSOR = RF02K-LIFNR
  BDC_OKCODE = /00
  BDC_SUBSCR = SAPLSZA1 0300 ADDRESS
  BDC_SUBSCR = SAPLSZA1 0301 COUNTRY_SCREEN
  ADDR1_DATA-COUNTRY ✓ = DE
  ADDR1_DATA-HOME_CITY ✗ = New York

SAPMF02K 0120 Vend.Mastr: General Control Data
SAPMF02K 0130 Vend.Mastr: General Payment Transactions Data
SAPMF02K 0210 Vendor Master: Company Code Data Accounting Information
SAPMF02K 0215 Vend.Mastr: Company Code Paymt Transctn Data
SAPMSCEM 0001 Transaction message
    
```

The error is indicated by the color red and the ✗ icon. Only the target value is logged.

### Marking processed screens

The screens which were processed during a test are shown in the CATT test plan log: They are displayed in black, screens not processed are in blue. This shows how far a test ran before an error occurred. You can also see any gaps in the screen sequence.

You go to a screen like:

```

00001501 00:00:00 STA_TEST_S01 08.02.2000 17:51:05 BIE 000 DE TESTUSER01 0001
46D ds0046 OSF1 ORACLE

TEST_S010_02 00:00:00 TEST_S010_02
Fehler S00344 in Anwendungstransaktion S010
Batchinput-Daten für Dynpro SAPMSSCE 1100 sind nicht vorhanden

TCD S010 SAPscript Standardtexte [00:00 min N]
S00344 Batchinput-Daten für Dynpro SAPMSSCE 1100 sind nicht vorh

SAPMSSCE 1100 SAPscript Standardtexte Einstiegsbild
SAPLSTXX 2101 Editor
SAPLSTXX 2101 Editor
SAPLSTXX 2101 Editor
SAPLSTXX 2101 Editor
SAPLSP01 0100 popup_to_confirm_step
SAPMSSCE 1100 SAPscript Standardtexte Einstiegsbild
SAPMSCEM 0001 Nachricht aus Transaktion
    
```

Blue/gray screens were not processed in the test.

This new function is particularly useful for error messages such as 'Data for screen XX not found'.

## Common Log Errors

## Common Log Errors

Errors are documented in the log by error messages. Common errors and their causes:

- *Leave to transaction is not allowed in Batch Input*  
The ABAP/4 command LEAVE TO ... before COMMIT WORK cancels the test.
- *No Batch input data for screen XXX*  
Possible causes:
  - test case screen sequence error.
  - test case screen missing.
  - unexpected dialog box during test.
- *Field <table field name> not ready for input*  
Values were assigned to an output field in the test case.
- *Fill all required fields*  
No value was assigned to a required field.

## Troubleshooting



[Processed screens are marked \[Page 86\]](#) in test plan logs which were created by running a test case in Release 4.6C.

Proceed as follows to find test errors:

1. Find the transaction or function in which the error occurred in the log.
2. Set this test case to "foreground processing" (in the individual execution execution screen, for a referred module, the REF function in the detail maintenance screen).
3. Rerun the test

You can also:

- Check the database contents during the test in another session.
- Run the transaction online in an alternative session.
- Navigate in the transaction during the test.  
The OK code field must first be cleared. Continue afterwards with the deleted OK code.  
This procedure is particularly useful in long lists (e.g. Report Writer lists), to check entries or for manual F1/F4 checks.
- Run the transaction in the debugger.

## Running test cases from the log

You can rerun either part or all of a test from its log. All parameters are reset to their original values. If you ran the original test case remotely, the same RFC destination is also used.

### Procedure

1. Display a test log.
2. Mark a test case.
3. Choose *Execute*.

The system creates a new session and runs the selected test case and creates a new log.



When you restart test modules from the log, parameter values from other test cases are no longer included because they do not run. The only functions executed are those in the restarted test module or procedure.

## Process Logs in Groups


### Use

You can now process several CATT test plan logs together.

### Prerequisites

You are in the Extended CATT mode.








### Procedure

1. Choose **Utilities** → **Logs**.
2. Enter selection criteria in the following screen and search with . A list of all test plan logs which satisfy the selection criteria is displayed.



You can use [wildcards \[Ext.\]](#) in the selection.

## Archiving logs

3. In the list of logs found you can:
  - Select logs by clicking in the first column. Marked logs are indicated by a tick in the first column *M*.
  - Display a selected log with .
  - Display the attributes of a log with .
    - You can maintain the expiry date in the attributes dialog box.
  - Change settings with .
  - Refresh the list with .
  - Select all test plan logs with .
  - Select all test plan logs with .
  - Process selected test plan logs together with . You can:
    - Change the expiry date.
    - Schedule archiving.
    - Exclude from archiving.
    - Delete immediately.



Deletion is irreversible.



The list of logs contains the column *B*. An *X* indicates that the test plan log in this row was created by running a test case in the background. They can be used for validation because they were not changed at runtime.

## Result

The changes made are made to all selected test plan logs.

## Archiving logs

You can archive data which you no longer need, but which must still be retained in an accessible form, with the archiving program. The data are written in archive files and then deleted from the system.

The following CATT data are archived:

- Procedure information
- Session information

You must set the archiving log flag when you run the test case from the Test Workbench or in the log attributes.

## Notes and Recommendations

Only sessions which have this flag, and whose expiry date is not later than the current system date, are archived. The system inserts the appropriate flag at run time.

### Create archive

Schedule the archiving session as a background job using the transaction SARA.

The archiving session puts the logs in an archive file and deletes the log data from the database. The report RSCATDEL, which deletes all data which are no longer required or whose expiry date has been reached, is executed.



Run the background job regularly.

#### See also:

*Application data archiving* documentation

### Display archive

You can display the archived sessions, just like any other sessions.

1. Choose *Utilities* → *Test plan* → *Logs*.
2. Choose *Read from archive* in the transaction selection screen.
3. After *Execute* the archive management offers you a selection of files, each of which corresponds to an archiving session.
4. Choose the desired archive files and *Continue*.

You get a selection list of all sessions in these files.

## Notes and Recommendations

This section contains additional information about the creation and use of test procedures.

### Test case creation tips

- Only create test modules for transactions which you know well.
- Only call transactions by reference to a test module.
- Only put one transaction in a test module.
- Always use parameters to pass values to transactions.
- Choose the parameters and screen sequence so as to make the test case as generally usable as possible.

## Rules for creating and maintaining test cases

- Avoid creating new test cases when existing ones can be modified.
- Always modify test cases so that they remain compatible.
- Document all test cases.
- Use existing test cases in other applications to use their transactions, requesting modifications if necessary.
- Use variants to broaden the range of tests.

## Rules for creating and maintaining test cases

### Using parameters in application-independent test cases

Note:

- Constants can only be assigned to standard delivery customizing objects (never to date fields or master record or document numbers).
- When test cases are changed, the parameter set can be extended, but not moved or reduced, to retain upwards-compatibility. The change is made by the person responsible.

### Language-independence

To minimize the number of test cases required, they must be created without language-dependent elements (Fast Path, language-dependent function codes). Text variables must be used for language-dependent fields (**See also:** [Text variables \[Page 77\]](#)).

The text variable table is connected to the translation tool.

To ensure the language-independence of measurement units, use constants, variables or a parameter as input values.



Do not use text variables &Txx **for measurement units.**

The value must always be specified in the test case original language. If a test case does not run in its original language, the conversion exit CUNIT translates language-dependent measurement units input in screens into a language-independent internal format and then back into the execution language. CUNIT uses standard SAP conversion function modules

### Test case documentation

Describe the test case (use of variables, data passing) in the function part, using the [TXT \[Page 52\]](#) function.

You can create a long text for more detailed documentation:

1. Go to the CATT maintenance transaction. Go to change mode. Choose *Goto* → *Long text* in the attribute, function and parameter maintenance screens.

## Clients for creating and running test cases

You go to the documentation editor with standard header variables. (They are replaced by logon language texts for display.)

2. Enter header texts. Get empty lines for input with *Insert* (F5). Leave insert mode with *End insert* or F5.
3. Save your entries.

## Clients for creating and running test cases

You can create client-independent test cases in any client, but you can only run them in one client. This must not be a productive client, as customizing settings are changed and test master data is created (e.g. documents), which can lead to errors in the productive system.

To allow test cases to run in a client, the client table **T000** must be maintained in System administration under the menu *Administration* → *Client management* → *Client maintenance*. The *CATT procedure execution allowed* flag must be set in the client detail view *Restrictions* group.



Correction popups can appear in the customizing transactions if the *Rec changes autom.* flag is set in the table **T000**. Do not set this flag when creating CATTs, otherwise the CATT procedure screen sequence for this customizing transaction may no longer be correct.

## External TCD data

If a test case is flagged as *External TCD data*, the field contents of the first transaction are obtained from external text files, not from CATT. The transaction is generally performed repeatedly under external control, up to a maximum of 999 times. The screen sequence in the external data must be the same as the CATT example screen sequence. The values for BDC\_OKCODE and BDC\_CURSOR are always supplied by CATT. The values for other screen input fields are generally read from the external files. You can specify that the CATT field contents are used in individual screens.

The path is taken from the SPA/GPA parameter CTP. If this is empty, the standard path of the computer is used.

### Text file structure

The control record WS\_UPLOAD in the file ABLAUFNR.000 is written back, possibly modified. The transaction records WS\_UPLOAD from the file ABLAUFNR.nnn are not written back

## External TCD data

### Control record

A control record is a sequence comprising a general control line followed by transaction control lines. The control record structure is as follows:

The first line contains general control information:

- Flag from offset 0 length 3
  - 000... general control (1st line in record)
- Flag from offset 3 length 1
  - 000X... productive execution of all TCD data desired, is changed to 000R... after execution
  - 000R... no further execution desired, already productively executed, log Rcode=10
  - 000T... test mode, external TCD data can be executed repeatedly, no change of the control record
- Text from offset 4 length 30
  - 000XText... general text, is not used internally
- result field for productive execution from offset 34 length 80
  - 000RText... procedure data (separated by SPACE):  
System ID (3-character)  
Session number (8-character)  
Date (YYYYMMDD) (8-character)  
Time (HHMMSS) (6-character)  
Release (3-character)  
Client (3-character)  
Language (1-character)  
User name (12-character)

The following lines contain transaction control information:

- Data ID from offset 0 length 3
  - nnn transaction control; a transaction record for the data record nnn exists under ABLAUFNR.nnn (screens and fields).  
Data ID in the range 001 .. 999
- Flag from offset 3 length 1
  - nnnX... execution of the external TCD data record nnn desired; changed to one of the following after productive execution:
  - nnnS... transaction variant ran successfully
  - nnnE... transaction variant ran with errors
  - nnnA... transaction variant cancelled
  - WS\_UPLOAD error in log Rcode=01..08

- screen error in log Rcode=11..99
  - screen sequence number from Rcode - 10
- Text from offset 4 length 30
  - nnnXText... Variant text is passed to the log in &LTX
- result field for productive execution from offset 34 length 80
  - nnnSText...Message: last transaction message

## Transaction records

Transaction records consist of a sequence of screen lines and field lines. The transaction record structure is as follows:

- Screen lines from offset 0 length 12 or 13
  - ProgramScrn:
    - Program = Module pool (8-character)
    - Scrn = Screen number (4-character)
    - X = CATF flag (1-character), i.e. take original CATF field contents
- Field lines begin with space, then from offset 1
  - LnLcNameContents:
    - Ln = Field name length (2-character)
    - Lc = Field contents length (2-character)
    - Name = Field name (length in Ln)
    - Contents = Field contents (length in Lc)
- Field lines always refer to the previous screen line

## Programming notes

The following program development notes can make the creation of test procedures easier.

### Transaction batch-input-compatibility

CATT can only currently model transactions which are not batch-input-compatible, with restrictions, so you should program transactions batch-input-compatible as far as possible.

- **COMMIT WORK:**

From Release 4.0 CATT can continue a transaction after a COMMIT WORK with the 'no COMMIT WORK end' mode. In other modes COMMIT WORK cancels the transaction for batch input and CATT. If an extra COMMIT WORK is necessary before the logical end of transaction, the function module DB\_COMMIT can be used (see function module

## Tips and tricks for creating test cases

DB\_COMMIT documentation). This does not apply to a COMMIT WORK at the end of the transaction.

- **LEAVE TO TRANSACTION before COMMIT WORK**

In many cases the transaction which has just ended is called again with LEAVE TO TRANSACTION, as a simple means of returning to the initial screen and ensuring that all work fields are cleared. To end the transaction for CATT, it is sufficient to enter the command COMMIT WORK immediately before the LEAVE TO TRANSACTION.

- There must be a logical transaction end.
- List screens must have the F15 function.
- The **ENTER** key must not be linked permanently to a function code.

## Transactions with different screen sequences in online and batch input modes

### CATT call in session 'with COMMIT WORK termination':

CATT uses the same interface as the batch input procedure to execute transactions. To ensure that the online version is tested when the screen sequence differs in online and batch input, the application transaction must contain the following MEMORY read call where SY-BINPT = "X" indicates batch input:

```
DATA: CATTACTIVE (1) TYPE C.
IMPORT CATTACTIVE FROM MEMORY ID 'CATT'.
IF SY_SUBRC = 0 AND CATTACTIVE = 'X'.
```

This causes the online alternative to be executed in test runs.

### CATT call in 'without COMMIT WORK termination' mode:

Both the system field SY-BINPT and the memory flag CATTACTIVE have their initial values. The system flag CATTNEU in the Memory ID 'CATTNEU' is set to 'X' to indicate that the transaction is running in a CATT procedure. The system flag CATTNEU is in the memory ID 'CATTNEU':

```
DATA: CATTNEW (1) TYPE C.
IMPORT CATTNEW FROM MEMORY ID 'CATTNEU'.
```

## Tips and tricks for creating test cases

- Do not use external number assignment.  
(Get numbers from the CATT number range with the function module CAT\_NUMBER\_GET\_NEXT)
- Do not specify constant dates.  
Use variables: &JHR, &DAT; date variables

---

**Tips and tricks for creating test cases**

- Use **variables and parameters** instead of **constants** in screens to make test modules flexible
- Use **text variables (&T01,...,&T99) for language-dependent fields.**  
(See also "Language-independence" in [Rules for creating and maintaining test procedures \[Page 94\]](#))
- **Define allowed error messages.**  
If this error message is sent, it is not logged.
- Only run parts of a test case in the foreground.  
Enter processing mode **A** in the REF function detail maintenance for a called test case.  
Enter processing mode **A** in the TCD function detail maintenance screen for a transaction.  
This part of the test case then runs in the foreground, even if the test started in "background processing".
- You can **deactivate individual lines in the CATT function editor.**  
If you want to exclude a function temporarily, you do not need to delete it, you can deactivate it to by flagging the / column at the end of the line in the function editor. This function does not then run in the test.
- Use the **Field list (function TCD) for changed transactions:**  
If dialog transaction changes have removed fields from screens, the field list helps you find the fields used in CATT again.  
Fields which were previously used by CATT are at the end of the list (after the BDC fields), with their previous value, and can so be easily identified.
- You can display all test check data with the SE38 report RSCATPRF.