Transactional Correctness in ABAP

Best Practice Document

and

End User Guide

Version: 2.1
Date: 27.11.2014
Content

1 Introduction ........................................................................................................ 5

2 Theoretical background .................................................................................. 6

2.1 ACID-Principles .......................................................................................... 6

2.2 Logical Unit of work (LUW) ....................................................................... 6

3 Concept and prerequisite .............................................................................. 8

3.1 Concept of the tool .................................................................................... 8

3.2 Technical prerequisites of the tool .............................................................. 8

4 How to use the “standalone” tool for ABAP .............................................. 9

4.1 Recording an ST05/ST12-Trace .................................................................. 9

4.2 Evaluating the ST05/ST12 Traces using the tool ......................................... 11

4.3 Evaluating the tool results ......................................................................... 12

5 How to execute an TC analysis from Solution Manager UI ...................... 14

5.1 Starting the TC analysis application in SAP Solution Manager ................ 14

5.2 Step 1: “Start / Stop Traces” ..................................................................... 14

5.3 Step 2: “Select Traces” ............................................................................ 15

5.4 Step 3: “Result” ....................................................................................... 17

6 Integration of TC Tool with E2E Traces .................................................... 19

6.1 Capturing E2E Traces .............................................................................. 19

6.1.1 Start SAP Client Plug-In to take a trace ............................................... 19

6.1.2 Execute transaction to be analyzed (Take the trace) ............................. 20

6.1.3 Evaluate the trace result ..................................................................... 22

6.1.4 Java-Analysis ..................................................................................... 24

6.1.5 ABAP-Analysis .................................................................................. 25

7 Recommendations on how to evaluate the tool results ........................... 26

7.1 Recommendations on how to evaluate the results .................................... 26

7.1.1 Rule 1: No COMMIT and ROLLBACK statements between changes in the main program 26

7.1.2 Rule 2: No COMMIT and ROLLBACK statements between changes within each tRFC/qRFC 27
7.1.3 Rule 3: No changes in sRFC/aRFC unless executed in an update task .......... 28
7.1.4 Rule 4: An explicit COMMIT statement is needed to trigger changes in an update task 29
7.1.5 Rule 5: If some changes in the main program are executed in an update task, the others should be as well ................................................................................................... 30
7.1.6 Rule E1: A Database Record might be modified without an active Enqueue... 30
7.1.7 Rule E2: A Dequeue might be set without an active Enqueue ...................... 31
7.1.8 Rule E3: A Dequeue is set without all parameters specified ......................... 31
7.1.9 Rule E4: Modifying DB-operation on a table that is not part of an active lock object 32
7.1.10 Rule E5: The changed table entry might only be locked by a Shared lock .... 33
7.1.11 Rule E6: Data might be committed without an active enqueue .................. 33
7.1.12 “Rule Ex”: No ENQ-, DEQ- or DEQ ALL-Statement found at all. Please check whether Enqueue-Trace was captured................................................................................. 34

8 Technical Information .................................................................................. 36
8.1 Release dependencies ........................................................................................................... 36

9 Tips / Misc. technical issues / pitfalls .......................................................... 37
9.1 How to prepare the inbound queues for TC analysis.......................................................... 37
9.2 Do and Don’ts for Transactional Correctness ........................................................................ 39

10 Further Examples ......................................................................................... 43
10.1 Examples for rule violations and recommendations .......................................................... 43
10.1.1 Example 1: Program ZDCM_TC_1 ..................................................................... 43
10.1.2 Example 2: Program ZDCM_TC_2 ..................................................................... 44
10.1.3 Example 3: Program ZDCM_TC_3 ..................................................................... 45
10.1.4 Example 4: Program ZDCM_TC_4 ..................................................................... 46
10.1.5 Example 5: Program ZDCM_ENQ_0 ................................................................. 46
10.1.6 Example 6: Program ZDCM_ENQ_1 ..................................................................... 47
10.1.7 Example 7: Program ZDCM_ENQ_2 ..................................................................... 48
10.1.8 Example 8: Program ZDCM_ENQ_4B ................................................................. 49
10.1.9 Example 9: Program ZDCM_ENQ_4C ................................................................. 50
10.1.10 Example 10: Program ZDCM_ENQ_4D ............................................................. 51
10.1.11 Example 11: Program ZDCM_ENQ_5 ............................................................... 53
10.1.12 Example 12 : Program ZDCM_ENQ_6C ........................................................... 53
10.2 Full listings of sample programs ............................................................................ 54
10.2.1 Program ZDCM_TC_1 ........................................................................................ 54
10.2.2 Program ZDCM_ENQ_0 ..................................................................................... 57
11 Glossary ............................................................................................................... 59
1 Introduction

In order to ensure consistency of data in the database, transactional correctness is an important element. If multiple programs are allowed to update the same information simultaneously, data correctness can be at risk. Transactions not only control the access of the same data by multiple programs, in the event of a system failure the usage of transactions also makes sure that the data will be in a consistent state after the recovery.

This End User Documentation should help to use the analysis tool provided for TC-Analysis for ABAP and also explains basic concepts of Transactional Correctness in ABAP programming.

Chapter 2 provides some insight into the ACID-principles that need to be kept in order to ensure Transactional Correctness and it also describes SAP’s concept of a “Logical Unit of Work” (LUW). Chapter 3 contains a basic description of the concept of the tool and the technical prerequisites that need to be met in order to use them. How to use the tool in a managed system is outlined in detail in chapter 4 of this document. Chapter 5 explains how it is used from within SAP Solution Manager. Chapter 6 explains the integration of the TC tool with E2E Traces. Chapter 7 explains the checked rules and corresponding warning messages in detail and also provides basic recommendations on how the warnings can be overcome. Chapter 8 briefly describes release dependencies, whereas chapter 9 provides additional information including recommendations on Do and Don’ts for ensuring Transactional Correctness. Further example programs are provided in chapter 10, whereas this document concludes with the glossary in chapter 11.
2 Theoretical background

2.1 ACID-Principles

In order to check transactional Correctness of applications coded in ABAP, we have to analyze whether the Transactional correctness concepts are fulfilled.

Transactional concepts/systems are usually rated by their fulfilment of the ACID-principles. These are:

- **Atomicity** refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them.

- **Consistency** property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not).

- **Isolation** refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction. This constraint is required to maintain the performance as well as the consistency between transactions in a DBMS system.

- **Durability** refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone.

2.2 Logical Unit of work (LUW)

A LUW describes a transaction:

![Database operations: insert, update, delete](image)

- **Consistent state 1**
- **Intermediate states**
- **Consistent state 2**

**rollback**

DB LUW (transaction) as offered by the database
Transactional correctness means that all data that is logically dependent on another has to be posted either all together or not at all. The amount of database changes required for this operation is called a Logical Unit of Work (LUW).

A database logical unit of work (LUW) is a non-separable sequence of database operations. Database LUWs allow you to encapsulate logically related actions from a business process.

At the beginning and end of the LUW, the database is in a consistent state. The database LUW is either fully carried out by the database system, or is not carried out at all. A database LUW is opened at the start of a transaction (when the connection with the database is made), and whenever a previous DB LUW is closed following a database commit. The database LUW is closed with a database commit. It is only in the commit that the data is written to the database (after which it can no longer be reversed). Before the database commit, you can undo the changes using a database rollback.

For example, when transferring sums of money in financial accounting, you must deduct an amount from one account and then add it to another account. Before and after the process, the data is consistent, but in between the two steps, it can be inconsistent.

Typically, a commit or rollback operation is initiated from an application program using one of the available programming languages that supports commitment control. These types of commit and rollback operations are known as explicit commit and rollback requests.

However, in some instances the system initiates a commit or rollback operation for a commitment definition. Commit and rollback operations initiated by the system are known as implicit commit and rollback requests.

An SAP LUW is a logical unit consisting of dialog steps, whose changes are written to the database in a single database LUW. Multiple SAP LUWs combine to form an SAP Transaction.
3 Concept and prerequisite

3.1 Concept of the tool

The tool is based on data captured in ST05/ST12 traces, but it only considers statements of type INSERT, UPDATE, DELETE, COMMIT and ROLLBACK as well as ENQUEUE, DEQUEUE and DEQUEUE ALL and brings them into a chronological order depending on their trace time stamp. The tool also memorizes if the statements are executed in a dialog work process or in an update work process. This list is then evaluated by the tool.

The TC tool evaluates multiple rules that should be obeyed to ensure transactional correctness. For the first and most basic rule, the tool counts the number of “change blocks” (all insert/update/delete statements between two commit/rollback statements). In case of more than one of these “change blocks” captured in the analyzed trace, transactional correctness might be at risk.

The tool also offers the possibility to enter the “RFC type” of the analyzed trace, or the “RFC type” is determined automatically by the end-to-end trace. In case of sRFC (synchronous RFC) or aRFC (asynchronous RFC), there should be no “change blocks” at all, or transactional correctness might be at risk.

Also, the TC tool recognizes if a change block contains changes that will be processed using an update task (tables VBMOD/VBDATA/VBHDR). If a change block contains such changes and at the same time also contains changes that will be processed outside of the update task, transactional correctness might also be at risk.

Furthermore the tool checks whether changes to entries in database tables are done while the corresponding records are correctly locked using a locked object and whether Enqueues and Dequeues are correctly set.

In all cases where transactional correctness is endangered, the tool colours the corresponding statements in the result screen and shows a description which rule is violated.

3.2 Technical prerequisites of the tool

The tool for the analysis of transactional correctness in ABAP that can be executed in a managed system (Report /SDF/DCC_TC) is available with ST-PI 2008_1. This ST-PI has to be installed in your system in order to have the tool available.

You can receive the latest version of this ST-PI in the SAP Service Marketplace:

http://service.sap.com/supporttools

The ability to evaluate enqueues and dequeues is available with ST-PI 2008_1 SP9.

To use the transactional correctness tool from within SAP Solution Manager in order to analyse a program or an interface in a managed system, it is necessary to have ST 7.10 SP12 installed in the SAP Solution Manager.
4 How to use the “standalone” tool for ABAP

4.1 Recording an ST05/ST12-Trace

The “standalone” tool for ABAP is Report /SDF/DCC_TC and can be started in the managed system. Since it evaluates ST05-traces, you first have to record an ST05-trace of the transaction you want to evaluate before using the tool.

Make sure that the following prerequisites are met:

- If possible, record only one single SAP LUW in one ST05-trace, i.e. steps that should be processed together to grant transactional correctness. If you record more than one single SAP LUW in one ST05-trace, this could falsify the result of the TC tool and show problems with transactional correctness although there are none.

- Before recording the ST05-trace, make sure that inbound queues are processed with a user different from your own user. To do so, fill the ‘destination with logon data’ field in the QIN scheduler (transaction SMQR) accordingly. If the ‘destination with logon data’ field is empty, your own user might be used to process inbound queue entries created by you. This could falsify the result of the TC tool and show problems with transactional correctness although there are none.

Please turn to chapter 9.1 in order to receive more detailed instructions on how to do so.

- While recording the trace, do not use update debugging as this could falsify the result of the TC tool and show problems with transactional correctness although there are none.

As a next step, in transaction ST05, select the SQL trace and activate it. Please note that you also might need to use the trace option “Activate Trace with Filter”. Using this option, you can explicitly include and exclude specific tables from the trace. We do not recommend using the inclusion-functionality here, since it might leave table updates out of focus that might hamper Transactional Correctness. In contrast, it might be required to exclude specific tables from the trace, since updates might be done on database tables that are not relevant for the Transactional Correctness of the specific application – e.g. tables that are required for ABAP-processing or similar tasks.

Please find here the list of tables that might need to be excluded from the SQL trace:

QIWKTab, QSSENDDEST, SNAP, SNAPY, REPOLOAD, REPOSRC, REPOTEXT, DYNPSOURCE, DYNLOAD, DYNPTXLD, DDLOG, EUF4VALUES
Now execute the transaction or step that you want to analyse and deactivate the ST05-trace.

Alternatively, you can use ST12 trace to capture the ABAP and SQL Trace. However, only the SQL trace would be useful in this case.
Once the transaction/program is completed, stop the trace and let the trace be captured. The advantage of using ST12 transaction to capture SQL trace is that the trace can be captured in any mode, i.e. either Transaction/Program could be traced. Similarly the trace could also be taken for a different user or a Work process.

4.2 Evaluating the ST05/ST12 Traces using the tool

You can start the TC tool by selecting and executing report "/SDF/DCC_TC" in transaction SE38. The selection-screen would be defaulted with the Date, Time and User-Id of the last trace captured by you. If there are multiple traces, entering the Transaction ID on the selection screen would uniquely identify the transaction for which Trace needs to be evaluated.

![Selection Screen for TC Tool](image)

Now the TC tool shows a list of all database changes in your SQL-trace and the description show where transactional correctness is at risk.

Additionally there is a field for RFC Type which can be used for evaluation of RFC records. This means that if during taking the trace an RFC-call has been captured, you can state one of the following RFC-types in that field:

- **NUNI** – no RFC or unknown RFC-Type
- **SRFC** – synchronous RFC
- **ARFC** – asynchronous RFC
- **TRFC** – transactional RFC
QRFC – queued RFC

It is also possible to provide the Transaction ID, the Extended Passport Root ID, the Connection ID or the Connection Counter in case it helps to identify the trace records that should be analyzed for Transactional Correctness.

4.3 Evaluating the tool results

The result list of the tool looks as follows:

```
<table>
<thead>
<tr>
<th>Record no.</th>
<th>Trace date</th>
<th>Trace time</th>
<th>Transaction</th>
<th>Work Proc. Type</th>
<th>Program</th>
<th>Table</th>
<th>Statement</th>
<th>Lock</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.06.2014</td>
<td>14:16:30.168</td>
<td>ZDC_TOD_DCC_TOD</td>
<td>INSERT</td>
<td>VALUES('2014-01-01' AS 'realdate')</td>
<td>INSERT</td>
<td>SDWB</td>
<td>INSERT</td>
<td>No lock/unlock</td>
</tr>
<tr>
<td>2</td>
<td>11.06.2014</td>
<td>14:16:30.168</td>
<td>ZDC_TOD_DCC_TOD</td>
<td>INSERT</td>
<td>VALUES('2014-01-01' AS 'realdate')</td>
<td>INSERT</td>
<td>SDWB</td>
<td>INSERT</td>
<td>No lock/unlock</td>
</tr>
</tbody>
</table>
```

Result Screen of /SDF/DCC_TC (No warning messages displayed)

The columns contain the following data:

- **Record no.** The number of the trace record in the result list
- **Trace date** Date of the captured trace time stamp
- **Trace time** Time stamp of the analyzed trace record
- **Transaction** Transaction code the analyzed trace record was executed from
- **Work Proc. Type** Type of Work Process. (Dialog, Background, etc.)
- **Program** Executing program
- **Table** Table the statement was performed on
- **Statement** Type of the analyzed statement. As mentioned above the tool only analyzes and lists INSERT-, UPDATE- and DELETE-Statements as well as COMMITs and ROLLBACKs. For the analysis of enqueues and dequeues, ENQUEUE, DEQUEUE and DEQUEUE ALL statements are also considered.
- **Lock** Are we locking/unlocking tables
- **Description** If Transactional Correctness might be at risk, the corresponding warning message will be listed here
In the column “Description” you will find comments which are evaluated based on one of the rules that are being used for analysis of the Statement. These are described in detail in chapter 7 of this document.

Since the tool output is an ALV-Grid, you can also use the standard features of ALV-Lists using the buttons displayed above.
5 How to execute an TC analysis from Solution Manager UI

5.1 Starting the TC analysis application in SAP Solution Manager

The Transactional Correctness of programs or interfaces in one of the managed system can also be evaluated from within SAP Solution Manager. This functionality is delivered with SAP Solution Manager 7.1 SP12.

You can start the application from Solution Manager Workcenters (Transaction code solman_workcenter) -> Work Center “Business Process Operations” -> “Data Consistency Management” -> “Transactional Correctness Check” in section “Analysis Tools”

5.2 Step 1: “Start / Stop Traces”

1. On tab “ST05”, enter the RFC destination of the managed system for which you want to check the transactional correctness.

2. Enter the user who is to execute the program you want to analyse in the managed system

3. In section “Tables”, you will find a list that contains tables that are to be excluded from the transactional correctness analysis. Please note that these tables will not be excluded from the trace that is captured, but only from the analysis that you will conduct in Step 3.

    The following tables are preselected: QIWKTAB, QSENDDEST, SNAP, SNAPIT, REPOLOAD, REPOSRC, REPOTEXT, DYNPSOURCE, DYNPLOAD, DYNPTXTLD, DDLOG, EUF4VALUES

    These tables are excluded, since they are not relevant for the analysis, and might jeopardize the analysis if they are included. Review this list before each analysis.

    You can choose “Insert Row” or “Append Row”, to add tables. Choose “Delete Row” to remove a table from the list.

4. Then choose “Start Trace” to start the SQL trace (“ST05-Trace”) in the managed system.

5. Choose “Stop Trace” to finish the trace.
5.3 Step 2: “Select Traces”

After you have taken the trace, navigate to Step 2 “Select Traces”.

1. In section “Trace Restrictions”, enter the RFC destination of the managed system containing the program you want to analyze. If you have just taken a trace in step 1, the RFC destination is already prefilled.

2. Enter the ID of the user who executed the program you want to analyze. This should be the user that has been traced. If you have just taken a trace in step 1, this field is already prefilled.

3. The Date and Time fields are pre-filled by the system, with the start date and time and end date and time the trace that you have just taken. If required, you can change the contents of these fields, if required.

4. If the trace records are to be distinguished by the extended passport root ID, the Extended passport connection ID”, the extended passport connection counter, or the TransGUID, you can enter the required data in the respective fields.

5. Section “Data for Evaluation”: if the program you traced used an RFC, you can either choose the RFC type or “Unknown RFC Type” if an un-identified RFC was used. You can also leave this field blank. (RFC-Types available for selection via the drop-down box: “synchronous RFC”, “asynchronous RFC”, “transactional RFC”, “queued RFC”.

6. Choose “Load Trace”.

The system displays the trace records based on the selection criteria specified.
The following information is carried in the result table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Name</td>
<td>Name of application server the database trace record was read from</td>
</tr>
<tr>
<td>Trace Rec. Number</td>
<td>Number of the evaluated database trace record</td>
</tr>
<tr>
<td>Client</td>
<td>Client of the managed system in which the database was traced</td>
</tr>
<tr>
<td>Date</td>
<td>Time stamp (date) of the recorded trace</td>
</tr>
<tr>
<td>Time</td>
<td>Time stamp (time) of the recorded trace</td>
</tr>
<tr>
<td>Object Name</td>
<td>Table the statement was performed on</td>
</tr>
<tr>
<td>Statement</td>
<td>Analyzed statement</td>
</tr>
<tr>
<td></td>
<td>Note: The tool only analyzes and lists INSERT, UPDATE, UPSERT, and DELETE</td>
</tr>
<tr>
<td></td>
<td>statements, and COMMITS and ROLLBACKs.</td>
</tr>
<tr>
<td>Work Process no.</td>
<td>The number of the work process in which the SQL statement was executed in</td>
</tr>
<tr>
<td></td>
<td>the managed system.</td>
</tr>
<tr>
<td>Executed Operation</td>
<td>The SQL operation related to the trace record.</td>
</tr>
<tr>
<td></td>
<td>Note: The tool only analyzes and lists REEXEC, EXECSTA, and EXCUSED</td>
</tr>
<tr>
<td></td>
<td>statements. For the analysis of enqueues and dequeues, ENQUEUE, DEQUEUE,</td>
</tr>
<tr>
<td></td>
<td>and DEQUEUE ALL statements are also considered.</td>
</tr>
<tr>
<td>Program Name</td>
<td>Program that was executed and traced in the managed system</td>
</tr>
<tr>
<td>Process Type</td>
<td>Work process type in which the action was executed in the managed system</td>
</tr>
<tr>
<td>Executed Transaction</td>
<td>Transaction the trace was executed from (in the managed system)</td>
</tr>
<tr>
<td>User Name</td>
<td>User ID that was traced in the managed system</td>
</tr>
<tr>
<td>TransGUID</td>
<td>Transaction GUID of the captured trace record</td>
</tr>
<tr>
<td>Extended Passport Root ID</td>
<td>Extended Passport Root ID of the captured trace record</td>
</tr>
<tr>
<td>Extended Passport Connection ID</td>
<td>Extended Passport Connection ID of the captured trace record</td>
</tr>
<tr>
<td>Extended Passport Connection Counter</td>
<td>Extended Passport Connection Counter of the captured trace record</td>
</tr>
</tbody>
</table>

7. Choose “Upload/Download” to download the results. You can also upload trace results from your local hard drive – but please note that only traces can be uploaded that have been captured by the TC Application in SAP Solution Manager. A specific XML-file format is used for this.

8. Select all trace records that are to be analyzed. Ideally, select all trace records that belong to the same SAP logical unit of work (SAP LUW), so that affected data belongs together logically, and should be posted to the database together or not at all. Please note that it is only possible to navigate to the next step (“Step 3 – Result”) if only trace records have been selected that were captured on the same server.
5.4 Step 3: “Result”

The result of the trace evaluation for Transactional Correctness is displayed when you select Step 3 “Result”, after you have selected trace records for analysis in Step 2.

1. Choose tab “Commit Structure” in order to display the check results of the commit structure of the analyzed program, based on the selected trace records.

The system evaluates how many change blocks are contained in the trace (a “change block” is ended by a COMMIT statement) and whether there is a risk to the transactional correctness of the program. Furthermore it is checked whether database updates are executed in a dialog work process or an update work process.

The column “Description for TC check result line” contains a warning message, if transactional correctness is at risk. These warning messages are described in detail in chapter 7 of this document.
2. Choose tab “Enqueues” in order to display the result of the Enqueue/Dequeue structure check of the analyzed program, based on the selected records.

They system evaluates whether correct and complete Enqueues and Dequeues are set, and whether they match the captured database changes.

The column “Description for TC check result line” contains a warning message, if transactional correctness is at risk. These warning messages are described in detail in chapter 7 of this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Execution time for SQL trace</th>
<th>Description for TC check result line</th>
<th>Object name</th>
<th>Statement</th>
<th>Program Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.07.2014</td>
<td>13:41:23:711</td>
<td>Rule 1: There should be no COMMIT and ROLLBACK statements...</td>
<td>ZDDC_TC2 DR</td>
<td>INSERT VALUES[200...</td>
<td>ZDDC_TC2</td>
</tr>
<tr>
<td>12.07.2014</td>
<td>13:41:23:711</td>
<td>Rule 1: There should be no COMMIT and ROLLBACK statements...</td>
<td>ZDDC_TC2 DR</td>
<td>INSERT VALUES[200...</td>
<td>ZDDC_TC2</td>
</tr>
<tr>
<td>12.07.2014</td>
<td>13:41:23:711</td>
<td>Rule 1: There should be no COMMIT and ROLLBACK statements...</td>
<td>ZDDC_TC2 DR</td>
<td>INSERT VALUES[200...</td>
<td>ZDDC_TC2</td>
</tr>
<tr>
<td>12.07.2014</td>
<td>13:41:23:711</td>
<td>Rule 1: There should be no COMMIT and ROLLBACK statements...</td>
<td>ZDDC_TC2 DR</td>
<td>INSERT VALUES[200...</td>
<td>ZDDC_TC2</td>
</tr>
</tbody>
</table>

Result Screen of the “Enqueue Check” of the TC-Tool in SAP Solution Manager
6 Integration of TC Tool with E2E Traces

This section briefly describes the integration of the TC Tool with End-To-End (E2E) Trace Tool.

You can download the latest version of the SAP Client Plug-In to take the trace from SAP Note 1435190. Alternatively you can also use the SAP HTTP Plugin in the E2E Trace Application that is available with SAP Solution Manager (One navigation possibility to execute the E2E Trace Application in SAP Solution Manager: Solution Manager Workcenters – via transaction code solman_workcenter - > Work Center “Business Process Operations” -> “Data Consistency Management” -> “End to End Trace”). (Also see section “Options” in the E2E Trace Application.)

6.1 Capturing E2E Traces

6.1.1 Start SAP Client Plug-In to take a trace

If you use the SAP Client plugin to capture an E2E trace, execute “plugin-starter-gui.exe” and click “Launch”. Now a blank Internet Explorer Session is opened.

Then enter a „Business Transaction Name“ in order to find your analysis later on. You can also enter an „Executed Step Name“. Additionally enter the Host name and the Port name (which can be derived from the URL of the application you are executing) in the Client Plug-In.

Example URL: http://lu0199:60000/AtsServlet/rfc -> Host: lu1099, Port: 60000
6.1.2 Execute transaction to be analyzed (Take the trace)

Click „Start Transaction“ in the SAP Client Plug-In and execute the transaction you want to analyze in the Browser Window that has been opened by the Client Plug-In. The trace is then being taken.

Click „Stop Transaction“ in order to stop recording of the traces and click „Exit“.

In the E2E-Trace-Application, click “Refresh”
E2E Trace Analysis

Start trace collection by marking the line with the trace you want to analyse and check the entries in PopUp “Trace collection confirmation dialog” as shown below.

And Click “Yes”.

Note that “Execute transactional correctness check” entry must be marked.
6.1.3 Evaluate the trace result

Once the trace is captured, display the results by selecting each step of the transaction and click the 'Display' button. The Result displayed also shows the distribution of the response time into different sub-components.
Display E2E Trace Results

Select a record from the HTTP Messages Table and Click the button „Auto-Analysis“. 
E2E Trace: Selecting Auto-Analysis

Depending on the record, click on either JAVA or ABAP Analysis (Icon in the right column of table “Requests tree”). This would bring you to the Transactional Consistency screen. The rule that was violated is displayed in the description column.

E2E Trace: Select detailed analysis

6.1.4 Java-Analysis

E2E Trace: TC Check results for JAVA

The concept of Transactional Correctness for Java is not described in this document. Please refer to SAP Note 1298475 for details on Transactional Correctness for Java. This note also contains a standalone analysis tool that can be used to analyze SQL traces taken from a Java application for Transactional Correctness.
### 6.1.5 ABAP-Analysis

**E2E Trace: TC Check results for ABAP**

![E2E Trace: TC Check results for ABAP](image)
7 Recommendations on how to evaluate the tool results

7.1 Recommendations on how to evaluate the results

The Rules have been defined for evaluation of the trace record by the TC-Tool. The below rules are explained in detail here.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Rule Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1</td>
<td>There should be no COMMIT and ROLLBACK statements between changes in the main program.</td>
</tr>
<tr>
<td>Rule 2</td>
<td>There should be no COMMIT and ROLLBACK statements between changes within each tRFC/qRFC.</td>
</tr>
<tr>
<td>Rule 3</td>
<td>There should be no changes in sRFC/aRFC unless executed in an update task.</td>
</tr>
<tr>
<td>Rule 4</td>
<td>An explicit COMMIT statement is needed to trigger changes in an update task.</td>
</tr>
<tr>
<td>Rule 5</td>
<td>If some changes in the main program are executed in an update task, the others should be as well.</td>
</tr>
<tr>
<td>Rule E1</td>
<td>A Database Record might be modified without an active Enqueue.</td>
</tr>
<tr>
<td>Rule E2</td>
<td>A Dequeue might be set without an active Enqueue.</td>
</tr>
<tr>
<td>Rule E3</td>
<td>A Dequeue is set without all parameters specified.</td>
</tr>
<tr>
<td>Rule E4</td>
<td>Modifying DB-Operation on a table that is not part of an active lock object.</td>
</tr>
<tr>
<td>Rule E5</td>
<td>The changed table entry might only be locked by a Shared lock.</td>
</tr>
<tr>
<td>Rule E6</td>
<td>Data might be committed without an active Enqueue.</td>
</tr>
<tr>
<td>“Rule Ex”</td>
<td>No ENQ-, DEQ- or DEQ ALL-Statement found at all. Please check whether Enqueue-Trace was captured.</td>
</tr>
</tbody>
</table>

Rules used by the TC Tool for evaluation

7.1.1 Rule 1: No COMMIT and ROLLBACK statements between changes in the main program

Rule Text:

“There should be no COMMIT and ROLLBACK statements between changes in the main program.”
Relevant for:

RFC-Type „Others“ (NUNI)
RFC-Type „Not defined“ (NUNI)
No RFC involved (NUNI)

Explanation:

More than one change block is captured in the trace, which means there is more than one COMMIT statement in a single logical unit of work (LUW).

The tool counts the number of change blocks (that is, all INSERT, UPDATE and DELETE statements) between two COMMIT statements. If more than one change block is captured in the trace, transactional correctness might be at risk for the statements after the first change block.

Recommendation:

Review the underlying ABAP code. Check whether the data in the separate change blocks is logically related (for example, sales order header data and sales order item data). If it is logically related, inconsistencies in the database can occur if the processing of the data aborts between the two COMMIT statements – for example – sales order header data is stored in the database, whereas the related sales order item data are not.

7.1.2 Rule 2: No COMMIT and ROLLBACK statements between changes within each tRFC/qRFC

Rule Text:

"There should be no COMMIT and ROLLBACK statements between changes within each tRFC/qRFC"

Relevant for:

- RFC-Type „tRFC“
- RFC-Type „qRFC“
Explanation:

More than one change block is captured in the trace, which means there is more than one COMMIT statement in a single logical unit of work (LUW).

The tool counts the number of change blocks (that is, all INSERT, UPDATE and DELETE statements) between two COMMIT statements. If more than one change block is captured in the trace, transactional correctness might be at risk for the statements after the first change block.

Basically this is the same check as executed for „Rule 1“, but checks the above separately for each tRFC / qRFC-call.

Recommendation:

Change the ABAP code so that all data that belongs together logically is committed to the database at the same time. There should only be one COMMIT statement at the end of the LUW.

If the data in the different change blocks in not related logically (for example – data of two logically independent sales orders is posted to the database in the two different change blocks), transactional correctness of the data is not at risk.

7.1.3 Rule 3: No changes in sRFC/aRFC unless executed in an update task

Rule Text:

“There should be no changes in sRFC/aRFC unless executed in an update task.”

Relevant for:

- RFC-Type „sRFC“
- RFC-Type „aRFC“

Explanation:

The analysed trace contains changes that were performed in an update task, and changes that were performed outside an update task.

The tool recognizes if a change block contains changes that are made by an update task (tables VBMOD, VBDATA or VBHDR). If a change block contains such a change and also contains changes that are processed outside the update task, check if all changes can be processed in the update task. Changes processed outside the update task have this warning.
A COMMIT statement is required, since updates in an update task are only triggered by an explicit COMMIT statement.

**Recommendation:**

Transactional correctness might be at risk if data processing aborts when the changes outside the update task have already been committed to the database, but those performed inside the update task have not yet been committed, so check whether the changes performed outside the update task can also be performed in the same update tasks as the other changes.

If there is no COMMIT statement to trigger the changes in the update task, add it.

Also follow Rules 1 and 2.

**7.1.4. Rule 4: An explicit COMMIT statement is needed to trigger changes in an update task**

**Rule Text:**

“An explicit COMMIT statement is needed to trigger changes in an update task.”

**Relevant for:**

- All RFC-Types (NUNI)
- No RFC involved (NUNI)

**Explanation:**

The analysed trace contains changes that were performed in an update task, and changes that were performed outside an update task.

The tool recognizes if a change block contains changes that are made by an update task (tables VBMOD, VBDATA or VBHDR). If a change block contains such a change and also contains changes that are processed outside the update task, check if all changes can be processed in the update task. Changes processed outside the update task have this warning.

A COMMIT statement is required, since updates in an update task are only triggered by an explicit COMMIT statement.

**Recommendation:**

See recommendations above for Rule 3.
7.1.5 Rule 5: If some changes in the main program are executed in an update task, the others should be as well

Rule Text:

“If some changes in the main program are executed in an update task, the others should be as well.”

Relevant for:

- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)

Explanation:

The analysed trace contains changes that were performed in an update task, and changes that were performed outside an update task.

The tool recognizes if a change block contains changes that are made by an update task (tables VBMOD, VBDATA or VBHDR). If a change block contains such a change and also contains changes that are processed outside the update task, check if all changes can be processed in the update task. Changes processed outside the update task have this warning.

A COMMIT statement is required, since updates in an update task are only triggered by an explicit COMMIT statement.

Recommendation:

See recommendations above for Rule 3.

7.1.6 Rule E1: A Database Record might be modified without an active Enqueue

Rule Text:

“A Database Record might be modified without an active Enqueue.”

Relevant for:

- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)
Explanation:
The trace contains an INSERT, UPDATE, UPSERT or DELETE statement, but there is no enqueue active when the modifying statement is executed. So, the same table record can be modified by more than one process at the same time, which can lead to inconsistencies, since isolation is not guaranteed.

Recommendation:
Ensure that during the database modification, a correct enqueue is set by calling the enqueue function module for the lock object.

7.1.7 Rule E2: A Dequeue might be set without an active Enqueue

Rule Text:
“A Dequeue might be set without an active Enqueue.”

Relevant for:
- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)

Explanation:
The trace contains a DEQUEUE statement when there is no active ENQUEUE statement. An enqueue may not have been set, so the same table record may be modified by more than one process at the same time, which can lead to inconsistencies, since isolation is not guaranteed.

Recommendation:
See recommendations above for Rule E1.

7.1.8 Rule E3: A Dequeue is set without all parameters specified

Rule Text:
“A Dequeue is set without all parameters specified.”
Relevant for:
- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)

Explanation:

The trace contains a DEQUEUE statement, but the corresponding dequeue function module is not called with all parameters, so the corresponding enqueue will not be deleted. This impacts performance, since the enqueue will be held longer than required and other work processes will not be able to modify the locked object.

The corresponding enqueue will only be deleted with the next Dequeue All, which can either be triggered explicitly (via an ABAP command) or automatically at the end of the SAP LUW, or at the end of the SAP transaction.

Recommendation:

Ensure that all the parameters of the call of the dequeue function module are the same as in the call of the corresponding enqueue function module.

7.1.9 Rule E4: Modifying DB-operation on a table that is not part of an active lock object

Rule Text:

“Modifying DB-operation on a table that is not part of an active lock object.”

Relevant for:
- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)

Explanation:

The trace contains an INSERT, UPDATE, UPSERT or DELETE statement on a table that is not part of an active lock object.

A lock object is active when there is an active enqueue. For the table with no active lock, the same table record can be modified by more than one process at the same time, which can lead to inconsistencies, since isolation is not guaranteed.
Recommendation:

For the table being modified, check whether there is a corresponding lock object and modify the code so that this lock object is called via the corresponding enqueue function module. If there is no lock object for this table, create one or assign this table to a lock object that is active during program execution. Lock objects can be created in transaction SE11.

### 7.1.10 Rule E5: The changed table entry might only be locked by a Shared lock

#### Rule Text:

“The changed table entry might only be locked by a Shared lock.”

#### Relevant for:

- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)

#### Explanation:

The trace contains an INSERT, UPDATE, UPSERT or DELETE statement on a table for which only a shared lock is active at the time of the modification.

The most commonly used lock modes are “Shared Lock”(S), “Write Lock”(E), “Exclusive, not cumulative”(X). If only a shared lock is set on a data record, it can be modified by a work process other than the lock holder, so isolation is not guaranteed and inconsistencies might occur.

#### Recommendation:

Check the lock arguments in the call of the enqueue function module that locks the table to be modified. Change the lock mode to a “Write Lock” (E) or “Exclusive, not cumulative” (X). Adapt the call of the dequeue function module, since the enqueue will only be deleted by the dequeue function modules if enqueue and dequeue function modules are called with the same parameters. If the lock mode is not set explicitly when the enqueue function module is called, the lock mode will be taken from the lock object’s master data (in transaction SE11, choose Lock Object -> Tables -> Lock mode).

### 7.1.11 Rule E6: Data might be committed without an active enqueue

#### Rule Text:

“Data might be committed without an active enqueue.”
Relevant for:

- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)

Explanation:

The trace contains a COMMIT WORK statement when there is no active enqueue. Changes to the database will only be consistent after they have been committed or rolled back. Ensure that the enqueue is still active at the time of the commit. If not, isolation might not be guaranteed and the data to be modified can be changed by another work process before the current changes are posted to the database. This can lead to inconsistencies.

Recommendation:

Ensure that the correct enqueue function module is called before the changes to the database table, and the correct dequeue function module is called only after the data is committed to the database.

7.1.12 “Rule Ex”: No ENQ-, DEQ- or DEQ ALL-Statement found at all. Please check whether Enqueue-Trace was captured.

Rule Text:

“No ENQ-, DEQ- or DEQ ALL-Statement found at all. Please check whether Enqueue-Trace was captured.”

Relevant for:

- RFC-Type „Not defined“ (NUNI)
- No RFC involved (NUNI)

Explanation:

This means that in the analyzed trace no ENQUEUE-, DEQUEUE- or DEQUEUE ALL-Statements have been found at all.

Recommendation:

Check whether in the traced program there are indeed no ENQUEUE-, DEQUEUE- or DEQUEUE ALL-statements – or whether the analyzed SQL-trace has been started also with the option “Enqueue Trace” selected. If e.g. an ST05-Trace is taken without this option being selected (See chapter 4.1 for
details), none of the statements mentioned above will be captured in the trace and thus the Enqueue/Dequeue-Analysis cannot be properly executed by the TC-Tool.
8 Technical Information

8.1 Release dependencies

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Availability</td>
<td>ST_PI_2008_1</td>
</tr>
<tr>
<td>Enqueue &amp; Dequeue functionality</td>
<td>ST_PI_2008_1 SP9</td>
</tr>
<tr>
<td>Availability in SAP Solution Manager</td>
<td>ST 7.10 SP12</td>
</tr>
</tbody>
</table>

Release dependencies
9 Tips / Misc. technical issues / pitfalls

9.1 How to prepare the inbound queues for TC analysis

The first step is to make sure that inbound queues are processed with a user different from your own user for *CSA and *CRI

1. Create destination in transaction SM59 with Connection Type L “Logical Destination”
   a. Reference Entry “NONE”

![](image1.png)

Creating an RFC Destination in transaction SM59

2. The provision of a client is not required, since this destination is a reference to the NONE reference.

Then use transaction SMQR and enter the RFC Destination you have just created as USERDEST for registration entry CRI* and CSA*.

Mark the queue and push the button “Registration”.

![image2.png]
Enter the destination you have just created as USERDEST.
9.2 Do and Don’ts for Transactional Correctness

1. No COMMITs and ROLLBACKs between changes in one SAP LUW.
   - Avoid COMMITs or ROLLBACKs (explicit and implicit!) between database changes in one SAP LUW.
   - Bundle all of the database changes for the SAP LUW using bundling concepts to execute database changes in a single database LUW.
   - In general, if customer specific database updates should be executed together with database updates in standard code, customers should not use their own COMMIT WORK or ROLLBACK WORK statements in BAdIs or other enhancements but bundle customer specific changes with standard changes.

2. If some changes in the main program are executed in an update task, the others should be as well.
   - Update Bundling techniques should be used – e.g. you can execute changes to database tables in update function modules that are registered for execution via CALL FUNCTION ... IN UPDATE TASK. Only with the next COMMIT WORK statement the execution of these function modules is triggered - and they are executed in an update task.

3. An explicit COMMIT WORK statement is needed to trigger changes in an update task.
   - If there is a COMMIT WORK statement in the SAP LUW in the standard code, do not add another customer specific COMMIT WORK statement because all database changes of the SAP LUW should be bundled in one database LUW.
   - However, if the SAP LUW is completely customer specific, a customer specific COMMIT WORK statement is needed to trigger the update task.

4. Use of “SET UPDATE TASK LOCAL”.
   - Priority update function modules - registered during the current SAP LUW using CALL FUNCTION ... IN UPDATE TASK - are registered in the ABAP memory instead of the VBLOG database table.
   - The current work process and not the update work process run these modules, when the COMMIT WORK statement is executed.
5. There should be no changes in sRFC/aRFC unless executed in an update task.
   - Avoid database changes in RFC function modules that are called using sRFC (CALL FUNCTION ... DESTINATION) or aRFC (CALL FUNCTION ... STARTING NEW TASK).
   - If database changes using sRFC or aRFC cannot be avoided, use update function modules that are called in the RFC function modules to bundle all changes.
   - Do not forget to include a COMMIT WORK statement in an RFC function module after the last database change to trigger the update task. However, this kind of bundling should only be used for database changes in one single system.
   - Use tRFC or qRFC for database changes in multiple systems since they provide logging and restartability features.

6. Implicit Commit Statements - End database LUW implicitly after
   - sRFC Calls (except with option “keeping logical unit of work”)
   - Dialog step
   - Dialog message Pop-up
   - “Call Transaction” statements
   - Submit Report
   - Wait up to x seconds
   - Leave to screen

7. Explicit Commit Statement Ends DB LUW with
   - EXEC SQL statement “COMMIT”

8. Explicit Commit Statement Ends SAP LUW with
   - “COMMIT WORK” in Customer ABAP Coding
   - “COMMIT WORK” Statement in Standard SAP Function Modules or Programs

9. An Explicit Commit Statement triggers the execution of an update task – an Implicit Commit Statement does not, but also note that explicit as well as implicit commit statements end the DB LUW – and thus trigger changes (e.g. Direct updates) to be written to the database.
10. The „Wait“-Statement triggers an implicit commit, which ends the DB LUW and triggers the changes to be written to the database – no matter whether the SAP LUW ends or not.

- If you require including a statement into your coding that triggers an implicit commit (e.g. a pop-up message), you should e.g. include a statement CALL FUNCTION ... IN UPDATE TASK or prepare update bundling in any other way before the implicit commit is triggered. This way the implicit commit does not commit any data to the database and the SAP LUW is not destroyed.

11. Programs that you call using the statements SUBMIT <program>, LEAVE TO TRANSACTION <t_code>, SUBMIT <program> AND RETURN or CALL TRANSACTION <t_code> run in their own SAP LUW, and update requests receive their own update key.

12. When you use SUBMIT <program> or LEAVE TO TRANSACTION <t_code>, the SAP LUW of the calling program ends.

- If no COMMIT WORK statement occurred before the program call, the update requests in the log table remain incomplete and cannot be processed. The same applies to inline changes that you make using PERFORM … ON COMMIT. Data that you have written to the database using inline changes is committed the next time a new screen is displayed.

13. If you use SUBMIT <program> AND RETURN or CALL TRANSACTION <t_code> to insert a program and then return to the calling program, the SAP LUW of the calling program is resumed when the called program ends.

- The LUW processing of the calling program and the called program is independent, that is, inline changes are committed when a new screen is displayed, and calls using PERFORM … ON COMMIT need a COMMIT WORK statement in the SAP LUW in which they are running.

14. If you receive a return code other than zero from the database interface in response to an Open SQL statement for changing data in the database, make sure the database is reset to the status it had before the change attempt was made.

You can do this by means of a database rollback. The database rollback undoes any changes made to the current database LUW (see the next unit).

- For return codes from DB change statements (Open SQL), the most suitable means of triggering a database rollback is to send a termination dialog message (A message or X message). This triggers a database rollback and terminates the associated program.
- All other message types (E, W, I) also involve a dialog but do not trigger a database rollback.

- You can also trigger a database rollback using the ABAP statement ROLLBACK WORK (without terminating the program at the same time). You should not use the ROLLBACK WORK statement directly, unless you do not want to reset the program context (unlike a termination dialog message).

- Additionally, in case of a database error, appropriate error messages with all the possible information should be sent to the user which would help later in error analysis.

15. Mass changes cannot be committed in one step because the rollback segments have a limited size.

16. It is always necessary to Set Locks on tables while accessing the tables to change them.

   Similarly, it is important to release the lock as soon as possible, so as not to hinder other users unnecessarily.

   - Enqueue and Dequeue function modules could be used to perform locking of the table records.

   - In case of large number of locks, instead of directly accessing the lock table, you can first collect the locks in a local lock container and then send all the locks collected in the local lock container to the lock table at one time. Communications with the lock server are minimized by sending locks in bundles.
10 Further Examples

10.1 Examples for rule violations and recommendations

In the following chapters, the mentioned rules are explained with examples.

The full listings of sample programs that do not violate any rule can be found in chapter 10.2 of this document. Details about the rules being violated and detailed explanations and recommendations for the various cases can be found in the sub-chapters of chapter 7.1 of this document.

The below examples of programs violating certain rules all are derived from the two programs called “ZDCM_TC_1” and “ZDCM_ENQ_0”, which are the ones listed in chapter 10.2 – so this document contains examples of breached rules as well as best practices.

10.1.1 Example 1: Program ZDCM_TC_1

... 

SELECT * FROM zyz_dcc1 INTO w_zdcc_tc2
WHERE obj_id > 30 AND obj_id < 40.
INSERT INTO zdcc_tc2_dr values w_zdcc_tc2.
CLEAR w_zdcc_tc2.
ENDSELECT.
COMMIT WORK.

Using Rules: Result screen of “/SDF/DCC_TC” (1)

From the trace, we see the following:

We lock and unlock the tables with the statements ENQUEUE and DEQUEUE.

We also see there is only one Change Block and one Commit.
You can see the full listing of this program in Chapter 10.2.1.

There is an insert to table followed ZDCC_TC1_DR followed by another insert to table ZDCC_TC2_DR. Lastly, there is a Commit statement.

There are no comments in the Description column of the TC tool result screen. There should be no issues with transactional correctness in this program.

10.1.2 Example 2: Program ZDCM_TC_2

* Modify the database

```sql
SELECT * FROM zyz_dcc1 INTO w_zdcc_tc1
WHERE obj_id > 70 AND obj_id < 80.
INSERT into zdcc_tc1_dr values w_zdcc_tc1.
CLEAR w_zdcc_tc1.
ENDSELECT.
COMMIT WORK.
```

```sql
SELECT * FROM zyz_dcc1 INTO w_zdcc_tc2
WHERE obj_id > 90 AND obj_id < 100.
INSERT into zdcc_tc2_dr values w_zdcc_tc2.
CLEAR w_zdcc_tc2.
ENDSELECT.
```

ROLLBACK WORK.

Using Rules: Result screen of ”/SDF/DCC_TC” (2)

Rule 1 has been violated.

The tool detected more than one „Change-Block“. This can lead to inconsistencies in case processing aborts after the first change block (which has already been committed), but before the COMMIT/ROLLBACK for the second Change-Block - i.e. Data to the table ZDCC_TC1_DR is inserted to the table, but before the ROLLBACK, the data to the table is committed. This could lead to an...
inconsistent state (assuming that data from table ZDCC_TR1_DR should also be rolled back with the ROLL BACK statement).

10.1.3 Example 3: Program ZDCM_TC_3

CALL FUNCTION 'ZDCM_TC_UPD_MODULE' IN UPDATE TASK.

* Modify data
SELECT * FROM zyz_dcc1 INTO w_zdcc_tc2
WHERE obj_id > 120 AND obj_id < 130.
INSERT INTO zdcc_tc2_dr VALUES w_zdcc_tc2.
APPEND w_zdcc_tc2 TO l_zdcc_tc2.
CLEAR w_zdcc_tc2.
ENDSELECT.

COMMIT WORK.

Using Rules: Result screen of “/SDF/DCC_TC” (3)

First of all, the tool has detected that there are change-statements to tables VBMOD, VBDATA and VBHDR.

This is based on the calling of Z_DCM_TC_UPD_MODULE in an update task. (See code snippet above)

Furthermore, additional change-statements to other tables have been detected, which shows that there are also changes done outside the update task (Insert statements on table ZDCC_TR_DR2). They should also be done within the update task, since otherwise in case of an error one set of changes might be posted to the database, while the other set is not – which can lead to inconsistencies. This is indicated by the description from Rule 5, which states “If some changes in the main program are executed in an update task, the others should be as well”.

In the result screen Rule E4 (“Modifying DB-operation on a table that is not part of an active lock object”) also is described as being violated (for the statements changing data in tables VBMOD, VBDATA and VBHDR), but in this case this is a tool error, which will be fixed shortly. The Enqueue-
Checks are also applied to statements to update tables, though these tables should not be relevant for the Enqueue-Check.

10.1.4 Example 4: Program ZDCM_TC_4

* Execute the update function module (that also modifies data)
CALL FUNCTION 'ZDCM_TC_UPD_MODULE' IN UPDATE TASK.

* COMMIT WORK.

Using Rules: Result screen of "/SDF/DCC_TC" (4)

The program has detected that Rule 4 – “An explicit COMMIT statement is needed to trigger changes in an update task” - has been violated.

Even though it is fine to call a function in an update task and we see the references to tables VBMOD and VBDATA in the output, there will be no update as the COMMIT WORK statement has been commented out and therefore will not be executed. In the TC tool result screen we also see that the INSERT-Statement to update table VBHDR is missing, which is the indication that the COMMIT WORK-statement is missing that should trigger the execution of the update task.

Again we also see that the Enqueue-Checks are applied to the update tables VBHDR, VBMOD and VBDATA, which has its root cause in the tool error that is described in section 10.1.3.

10.1.5 Example 5: Program ZDCM_ENQ_0

CALL FUNCTION 'ENQUEUE_EZ_ZDCM_ENQ1'
     EXPORTING
     *          mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclus  
ive not cumulative
     MANDT     = sy-mandt
     OBJ_ID    = w_zdcm_enq1-obj_id
     VALUE     = w_zdcm_enq1-value

..
CALL FUNCTION 'DEQUEUE_EZ_ZDCM_ENQ1' 
EXPORTING 
  *          mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative 
  MANDT = sy-mandt 
  OBJ_ID = w_zdcm_enq1-obj_id 
  VALUE = w_zdcm_enq1-value.

Checking the results of the TC Tool, we can see no Rules are being violated. 
You can see the full listing of this program in Chapter 10.2.2.

10.1.6 Example 6: Program ZDCM_ENQ_1

... 
MODIFY zdcm_enq1 FROM w_zdcm_enq1. 
CLEAR w_zdcm_enq1. 
COMMIT WORK.

Checking the results of the TC Tool, we see that Rule E1 is being violated. 
Rule E1 states “A Database Record might be modified without an active Enqueue” 
Checking the source code, we find no active enqueues. 
Rule E6 also shows up in the trace expands on this. 
Rule E6 states that “Data might be committed without an active enqueue”.

www.sap.com
Furthermore we see – in the last line of the result screen of the TC tool result – the statement that no Enqueue-, Dequeue- or Dequeue All-Statements have been detected at all in the recorded trace.

### 10.1.7 Example 7: Program ZDCM_ENQ_2

```plaintext
... 

MODIFY zdcm_enq1 FROM w_zdcm_enq1.

COMMIT WORK.

CALL FUNCTION 'DEQUEUE_EZ_ZDCM_ENQ1'

EXPORTING

  mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative"
  MANDT = sy-mandt
  OBJ_ID = w_zdcm_enq1-obj_id
  VALUE = w_zdcm_enq1-value.
```

Checking the results of the TC Tool, we see that Rule E2 is being violated.

Rule E2 checks for “A dequeue might be set without an active Enqueue”

The trace highlights a DEQUEUE statement being set when there is no active ENQUEUE statement.

Rules E1 & E6 are also highlighted in the trace.
10.1.8 Example 8: Program ZDCM_ENQ_4B

CALL FUNCTION 'ENQUEUE_EZ_ZDCM_ENQ1'
  EXPORTING
    mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative
    MANDT = sy-mandt
    OBJ_ID = w_zdcm_enq1-obj_id
    VALUE = w_zdcm_enq1-value
  *
MODIFY zdcm_enq1 FROM w_zdcm_enq1.
COMMIT WORK.

CALL FUNCTION 'DEQUEUE_EZ_ZDCM_ENQ1'
  EXPORTING
    mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative
    MANDT = sy-mandt
    OBJ_ID = w_zdcm_enq1-obj_id
    VALUE = w_zdcm_enq1-value.

Checking the results of the TC Tool, we can see that Rule E3 is being violated. Rule E3 states “A Dequeue is set without all parameters specified”.

The trace contains a DEQUEUE statement, but the corresponding dequeue function is not called with all parameters, so the corresponding enqueue will not be deleted. This impacts performance, since the enqueue will be held longer than required and other work processes will not be able to modify the locked object.

The corresponding enqueue will only be deleted with the next Dequeue All, which can either be triggered explicitly (via an ABAP command) or automatically at the end of the SAP LUW, or at the end of the SAP transaction.
10.1.9 Example 9: Program ZDCM_ENQ_4C

CALL FUNCTION 'ENQUEUE_EZ_ZDCM_ENQ1' 
  EXPORTING 
    *          mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative 
    MANDT = sy-mandt 
...

MODIFY zyz_dcc_enq_1 FROM w_zdcm_enq1.

COMMIT WORK.

CALL FUNCTION 'DEQUEUE_EZ_ZDCM_ENQ1' 
  EXPORTING 
    *          mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative 
    MANDT = sy-mandt 

Checking the TC Tool, we see that Rule E4 is being violated.

Rule E4 states “Modifying DB-Operation on a table that is not part of an active lock object”.

An enqueue is set and a dequeue is performed, but a modification is performed on a table that has not been locked.

In order to find out, which tables are part of a lock object, you should refer to the lock object via transaction SE11. Here in tab “tables” you will find the tables that are locked if the corresponding lock object is called.
10.1.10 Example 10: Program ZDCM_ENQ_4D

CALL FUNCTION 'ENQUEUE_EZ_ZYZ_DCC_ENQ_1'
EXPORTING
*     mode_ZYZ_DCC_ENQ_1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative
     MANDT = sy-mandt
     OBJ_ID = w_zdcm_enq1-obj_id
     VALUE = w_zdcm_enq1-value
...

CALL FUNCTION 'DEQUEUE_EZ_ZYZ_DCC_ENQ_1'
EXPORTING
*     mode_ZYZ_DCC_ENQ_1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative
     MANDT = sy-mandt
     OBJ_ID = w_zdcm_enq1-obj_id
     VALUE = w_zdcm_enq1-value.

Checking the results of the TC Tool, we can see that Rule E5 is being violated.
Rule E5 states “The changed table entry might only be locked a Shared lock”.

The enqueue and dequeues match but the issue here is that the lock is set on table ZYZ_DCC_ENQ_1 using lock object EZ_ZYZ_DCC_ENQ1.

A shared lock does not prevent other processes from modifying the table.

You can find the lock mode used by a specific lock object by checking transaction SE11 for the lock object. Field “Lock mode” on tab “Tables” contains the default lock mode used by the lock object.

The default lock mode of a lock object can be overwritten by parameter “mode_<table name>” when calling the Enqueue Function Module of a lock object.

Example:

```plaintext
CALL FUNCTION 'ENQUEUE_EZ_ZYZ_DCC_ENQ_1'
EXPORTING
  mode_ZYZ_DCC_ENQ_1 = 'X' "E = Write Lock, S = Read Lock, X = Exclusive not cumulative"
  MANDT = sy-mandt
  OBJ_ID = w_zdcm_enq1-obj_id
  VALUE = w_zdcm_enq1-value
```
10.1.11 Example 11: Program ZDCM_ENQ_5

CALL FUNCTION 'ENQUEUE_EZ_ZDCM_ENQ1'
EXPORTING
mode_ZDCM_ENQ1 = 'S' "E = Write Lock, S = Read Lock, X = Exclusive not cumulative"
MANDT = sy-mandt
OBJ_ID = w_zdcm_enq1-obj_id
VALUE = w_zdcm_enq1-value

CALL FUNCTION 'DEQUEUE_EZ_ZDCM_ENQ1'
EXPORTING
mode_ZDCM_ENQ1 = 'E' "E = Write Lock, S = Read Lock, X = Exclusive not cumulative"
MANDT = sy-mandt
OBJ_ID = w_zdcm_enq1-obj_id
VALUE = w_zdcm_enq1-value.

Checking the results of the TC Tool, we can see that Rule E5 is being violated.

Rule E5 states “The changed table entry might only be locked a Shared lock”.

Looking at the code, we see a shared lock / read lock being set when the Enqueue Function Module is called (mode_ZDCM_ENQ1 = ‘S’). A Write Lock (‘E’) or an exclusive lock (‘X’) should have been used instead.

10.1.12 Example 12 : Program ZDCM_ENQ_6C

CALL FUNCTION 'ENQUEUE_EZ_ZDCM_ENQ1'
EXPORTING
mode_ZDCM_ENQ1 = 'E' "E = Write Lock, S = Read Lock, X = Exclusive not cumulative"
MANDT = sy-mandt
OBJ_ID = w_zdcm_enq1-obj_id
VALUE = w_zdcm_enq1-value
CALL FUNCTION 'DEQUEUE_EZ_ZDCM_ENQ1' 
EXPORTING 
  mode_ZDCM_ENQ1 = 'E' "E = Write Lock, S = Read Lock, X = Exclusive not cumulative 
  MANDT = sy-mandt
  OBJ_ID = w_zdcm_enq1-obj_id
  VALUE = w_zdcm_enq1-value.

COMMIT WORK.

Checking the results of the TC Tool, we can see that Rule E6 is being violated.

Rule E6 states “Data might be committed without an active Enqueue”.

Looking at the code, the enqueue and dequeue are set correctly, but the dequeue is set before the COMMIT WORK is performed. This means that in the time between the execution of the dequeue and the COMMIT WORK parallel processes can modify the uncommitted data.

10.2 Full listings of sample programs

10.2.1 Program ZDCM_TC_1

REPORT ZDCM_TC_1.

TYPES: BEGIN OF t_zdcc_tc1,
  mandt TYPE zdcc_tcl_dr-mandt,
  obj_id TYPE zdcc_tcl_dr-obj_id,
  value TYPE zdcc_tcl_dr-value,
END OF t_zdcc_tc1.

TYPES: BEGIN OF t_zdcc_tc2,
  mandt TYPE zdcc_tc2_dr-mandt,
  obj_id TYPE zdcc_tc2_dr-obj_id,
  value TYPE zdcc_tc2_dr-value,
END OF t_zdcc_tc2.

DATA: l_zdcc_tc2 TYPE STANDARD TABLE OF t_zdcc_tc2,
  l_zdcc_tc1 TYPE STANDARD TABLE OF t_zdcc_tc1.

DATA: w_zdcc_tc1 TYPE t_zdcc_tc1,
  w_zdcc_tc2 TYPE t_zdcc_tc2.

START-OF-SELECTION.

* Set the enqueue.
* Though we know that it is not ideal to do this here.
* It would be better to set the enqueue only directly before
* the data is modified in order to minimize lock times
* For demo reasons we set OBJ_ID in the enqueue to a value that does not
  have any specific meaning.
* Lock object EZ_ZDCC_TC1_DR locks both tables that will be changed later on
w_zdcc_tc1-obj_id = 1.

CALL FUNCTION 'ENQUEUE_EZ_ZDCC_TC1_DR'
  EXPORTING
    mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclus
ive not cumulative
    MANDT = sy-mandt
    OBJ_ID = w_zdcc_tc1-obj_id
    _KEYFIELD2 = "Value"
    _SCOPE = '2'
    _WAIT = ' ',
    _COLLECT = ' ',

  EXCEPTIONS
    FOREIGN_LOCK = 1
    SYSTEM_FAILURE = 2
    OTHERS = 3.

IF sy-subrc <> 0.
  * Retrieve message displayed within Function Module
  * After confirmation of message pop-up, program is stopped.
  * Note: Message still is empty.
message id sy-msgid
  type 'I'
  number sy-msgno
  with sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
EXIT.

ENDIF.

** Modify the database

SELECT * FROM zyz_dcc1 INTO w_zdcc_tc1
    WHERE obj_id < 10.
    INSERT into zdcc_tc1_dr values w_zdcc_tc1.
    CLEAR w_zdcc_tc1.
ENDSELECT.

SELECT * FROM zyz_dcc1 INTO w_zdcc_tc2
    WHERE obj_id > 30 AND obj_id < 40.
    INSERT into zdcc_tc2_dr values w_zdcc_tc2.
    CLEAR w_zdcc_tc2.
ENDSELECT.

COMMIT WORK.

CLEAR w_zdcc_tc1.

* Set the dequeue
* w_zdcc_tc1-objid is set to 1 to match dequeue and enqueue ...
* ... remember: We only do this for demo purpose here. This program should
* demo the COMMIT-Check, not the Enqueue-Check.

w_zdcc_tc1-obj_id = 1.

CALL FUNCTION 'DEQUEUE_EZ_ZDCC_TC1_DR'
    EXPORTING
        mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclus
        MANDT = sy-mandt
        OBJ_ID = w_zdcc_tc1-obj_id.

sy-msgv1 = 'Program'.
sy-msgv2 = sy-cprog.
sy-msgv3 = 'has been successfully executed.'.

message id sy-msgid
    type 'I'
    number sy-msgno
    with sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
### 10.2.2 Program ZDCM_ENQ_0

```
REPORT ZDCM_ENQ_0.

TYPES: BEGIN OF t_zdcm_enq1,
  mandt  TYPE zdcm_enq1-mandt,
  obj_id TYPE zdcm_enq1-obj_id,
  value  TYPE zdcm_enq1-value,
  zdate  TYPE zdcm_enq1-zdate,
  time   TYPE zdcm_enq1-time,
  process TYPE zdcm_enq1-process,
  zuser  TYPE zdcm_enq1-zuser,
  progname TYPE zdcm_enq1-progname,
END OF t_zdcm_enq1.

DATA: l_zdcm_enq1 TYPE STANDARD TABLE OF t_zdcm_enq1.

DATA: w_zdcm_enq1 TYPE t_zdcm_enq1.

START-OF-SELECTION.

** Demo 1
** Modify one single entry in table ZDCM_ENQ1
** and document date and time and user and
** program that did the change
** In this case "process" is set to 1
w_zdcm_enq1-mandt = sy-mandt.
w_zdcm_enq1-obj_id = 2.
w_zdcm_enq1-value = 'DD734E088253F3F1B27A001185B78845'.
w_zdcm_enq1-zdate = sy-datum.
w_zdcm_enq1-time = sy-uzeit.
w_zdcm_enq1-process = 1.
w_zdcm_enq1-zuser = sy-uname.
w_zdcm_enq1-progname = sy-cprog.

CALL FUNCTION 'ENQUEUE_EZ_ZDCM_ENQ1'
  EXPORTING
    mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive not cumulative
    MANDT         = sy-mandt
    OBJ_ID        = w_zdcm_enq1-obj_id
```
VALUE = w_zdcm_enq1-value
*KEYFIELD2 = "Value"
*_SCOPE = '2'
*_WAIT = '',
*_COLLECT = '',

EXCEPTIONS
FOREIGN_LOCK = 1
SYSTEM_FAILURE = 2
OTHERS = 3.

IF sy-subrc <> 0.
  * Retrieve message displayed within Function Module
  * After confirmation of message pop-up, program is stopped.
  * Note: Message still is empty.
    message id sy-msgid
    type 'I'
    number sy-msgno
    with sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
  EXIT.
ENDIF.

MODIFY zdcm_enq1 FROM w_zdcm_enq1.

COMMIT WORK.

CALL FUNCTION 'DEQUEUE_EZ_ZDCM_ENQ1'
  EXPORTING
  *mode_ZDCM_ENQ1 = 'E'  "E = Write Lock, S = Read Lock, X = Exclusive
  MANDT = sy-mandt
  OBJ_ID = w_zdcm_enq1-obj_id
  VALUE = w_zdcm_enq1-value.

CLEAR w_zdcm_enq1.

sy-msgv1 = 'Program'.
sy-msgv2 = sy-cprog.
sy-msgv3 = 'has been successfully executed.'.
sy-msgv4 = 'Have a nice day! :-)'.

message id sy-msgid
  type 'I'
  number sy-msgno
  with sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
11 Glossary

ABAP

Advanced Business Application Programming - The SAP programming language.

ALV

SAP List Viewer. Tool for displaying and formatting lists in the SAP system based on the SAP GUI user-interface technology.

ACID

Referring to the ACID-principles (Atomicity, Consistency, Isolation and Durability) that are defined to guarantee transactional correctness when they are fulfilled

BAdI

Business Add-In. Business Add-Ins (BAdIs) are the basis for enhancements where BAdI methods in object plug-ins can be called from ABAP programs. The calling program controls which BAdI implementations are used by specifying filter values. A BAdI consists of a BAdI interface, a set of filters and some settings.

CRM

Customer Relationship Management. CRM is a customer management approach that comprises all processes involving direct customer contact throughout the entire customer relationship life cycle - from market segmentation, sales lead generation and opportunities to post-sales and customer service. In this document's context the term CRM refers to the SAP solution supporting companies to deploy their customer relationship management processes.

DBMS

A database management system (DBMS) is computer software that manages databases
Extended Passport Root ID

Extended Passport Connection ID

Extended Passport Connection Counter

In certain applications the concept of the SAP Extended Passport (EPP) is used to log user actions. In this context information about the user who originally triggered an action (even across system boundaries) can be logged. Two IDs with the following properties can be generated:

- The Root Context ID is generated for each new action. If additional steps (for example, over RFC) are performed due to this action, this ID remains constant, so that you can identify the original action (As of Basis 7.02, this is possible even across system boundaries).
- The Connection ID is always generated if a calls to a particular RFC destination are made within an action. As long as the RFC destination remains open within this action, the Connection ID remains constant for additional RFC calls using this connection.

LUW

Logical unit of work. A database logical unit of work (LUW) is a non-separable sequence of database operations (like UPDATE, DELETE or INSERT). An SAP LUW is a logical unit consisting of dialog steps, whose changes are written to the database in a single database LUW. Before and after the execution of the LUW the database is in a consistent state, no matter if the LUW has been successfully executed or not.

RFC

Remote Function Call. An interface for communication between an SAP System and other SAP- or third-party compatible system via TCP/IP or CPI-C connections. There are different types of RFC-Calls: synchronous RFC (sRFC), asynchronous RFC (aRFC), transactional RFC (tRFC), queued RFC (qRFC) and background RFC (bgRFC).

ST-PI


TC

Transactional Correctness
TransGUID / Transaction GUID / Transaction ID

A new Transaction ID is always assigned when a user starts a transaction. While the user is using this transaction, this Transaction ID remains constant for all of his or her actions within the transaction, until the user ends the transaction again. (Please note that under certain circumstances, a new Transaction ID might be generated for RFC-Calls executed within the same transaction.)
© 2014 SAP SE or an SAP affiliate company. 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 SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Please see http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark for additional trademark information and notices. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP SE or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.