Transaction Handling in SAP EWM (Extended Warehouse Management)

November 18, 2009
Table of Contents:

1 SAP – Important Disclaimers and Legal Information ............................................... 3

2 Transaction Handling in SAP EWM (Extended Warehouse Management) ............. 6

2.1 Structure of a Typical Program (Transaction) in EWM ........................................... 6

2.2 A Sample LUW to Change a Customer Field ........................................................... 8

2.3 Saving Data Changes to End an LUW ................................................................. 12

2.4 Ending an LUW in the Case of Errors or Undo Changes .......................................... 13

2.5 Typical Situations that May Lead to Errors/Inconsistencies ................................ 14

2.5.1 Example 1 ........................................................................................................ 14

2.5.2 Example 2 ........................................................................................................ 14

2.5.3 Example 3 ........................................................................................................ 14

2.5.4 Example 4 ........................................................................................................ 14

2.6 Further Things to Keep in Mind .............................................................................. 15

2.6.1 Context of Program ........................................................................................... 15

2.6.2 Use Proper Functions to Save and/or Clean Up Data ....................................... 15

2.6.3 Types of Messages/Error Handling ................................................................. 16

2.7 Further Information ............................................................................................... 18
1 SAP – Important Disclaimers and Legal Information

General Disclaimer

SAP does not represent or endorse the accuracy or reliability of any of the information, content, or advertisements (collectively, the "Materials") contained on, distributed through, or linked, downloaded, or accessed from any of the services contained on this Web site (the "Service"), nor the quality of any products, information, or other materials displayed, purchased, or obtained by you as a result of an advertisement or any other information or offer in or in connection with the Service (the "Products"). You hereby acknowledge that any reliance upon any Materials shall be at your sole risk. SAP reserves the right, in its sole discretion and without any obligation, to make improvements to, or correct any error or omissions in any portion of the Service or the Materials.

THE SERVICE AND THE MATERIALS ARE PROVIDED BY SAP ON AN "AS IS" BASIS, AND SAP EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SERVICE OR ANY MATERIALS AND PRODUCTS. IN NO EVENT SHALL SAP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES OF ANY KIND WHATSOEVER WITH RESPECT TO THE SERVICE, THE MATERIALS, AND THE PRODUCTS.

SAP encourages you to exercise discretion while browsing the Internet using this site.

SAP makes no representations concerning any endeavor to review the content of sites linked to this site or any of the Materials, and so SAP isn't responsible for the accuracy, copyright compliance, legality, or decency of material contained in sites listed in the directory or in the Materials.

SAP respects the rights (including the intellectual property rights) of others, and we ask our users to do the same. SAP may, in appropriate circumstances and in its sole discretion, terminate the accounts of users that infringe or otherwise violate such rights of others.

If you believe that your work has been copied in a way that constitutes copyright infringement, please follow the instructions at the top of this page.

SAP – Guidelines for Using SAP Trademarks

Throughout the world, thousands of customers recognize and select SAP solutions on the basis of the company's trademarks and service marks, which signify high-quality computer software products and services. Without these trademarks and service marks (collectively referred to as "trademarks"), consumers would not be able to distinguish SAP solutions from those of other companies, nor would they be able to readily identify the superior quality that SAP trademarks represent. Therefore, it is critically important that trademarks of SAP and its subsidiaries are protected.

For guidelines on trademark usage, please see the following information:

- SAP trademarks and SAP offering names
- Proper use of trademarks
- Additional trademark usage information for third parties
- Trademark usage information for all print and online media

If after reviewing the Guidelines for Using SAP Trademarks you still have a question about use, please direct your inquiry to trademarks@sap.com. SAP will make reasonable efforts to respond to your request. However, based on volume of requests you should allow several weeks for a response. In the absence of a response, the Guidelines for Using SAP Trademarks shall govern any and all uses.

Designing Marketing Material

If you are a developer, partner, customer, or other third party, always use your own proprietary design style when creating and producing a marketing piece. You must not copy the SAP design style or borrow any SAP design elements.

Guidelines for Using SAP-Copyrighted Material

The use of original texts, graphics, images, screen shots, and other materials from SAP sources must be approved by SAP. In addition, when using such materials, you must include a copyright notice – either in an adjacent area or as a footnote – to indicate SAP's copyright. The copyright should read as follows:

"© SAP AG <year>. All rights reserved."

Please note that SAP photography is only licensed for SAP use and must not be used by third parties in their communications.

SAP – Copyrights and Trademarks

ENGLISH – German version below

© Copyright 2009 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, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

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

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group 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.

GERMAN

© Copyright 2009 SAP AG. Alle Rechte vorbehalten.

Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Softwareprodukte können Softwarekomponenten auch anderer Softwarehersteller enthalten.

Microsoft, Windows, Excel, Outlook und PowerPoint sind eingetragene Marken der Microsoft Corporation.


Linux ist eine eingetragene Marke von Linus Torvalds in den USA und anderen Ländern.

Adobe, das Adobe Logo, Acrobat, PostScript und Reader sind Marken oder eingetragene Marken von Adobe Systems Inc. in den USA und/oder anderen Ländern.

Oracle ist eine eingetragene Marke der Oracle Corporation.

UNIX, X/Open, OSF/1 und Motif sind eingetragene Marken der Open Group.
2 Transaction Handling in SAP EWM (Extended Warehouse Management)

This document describes what has to be considered when you build your own transactions in which EWM objects (such as deliveries, warehouse tasks, HUs, and so on) are changed, created, or deleted in SAP EWM. This document is designed for technical consultants/developers who do programming in an EWM system. Most of the concepts (for example, locking of objects before updating, calling posting functions, and so on) should be nothing new for developers who have worked in an SAP environment before. Nevertheless, it may happen that parts are not considered in a program. Therefore this document illustrates some “typical” situations which can be critical and also shows concepts that should be taken care of. The purpose of the document is not to be a complete programming guideline but just to point out some important parts.

⚠️ Failure to consider these concepts may result in data inconsistencies!

2.1 Structure of a Typical Program (Transaction) in EWM

An application program (transaction) may consist of several SAP logical units of work (LUWs). The case in which one transaction corresponds to exactly one SAP LUW may exist, but is rather an exception. Usually (for example, in RF-UI), a user can execute multiple LUWs one after the other (for example, confirm one warehouse task, then unload an HU, then post goods issue).

An SAP LUW in EWM should have the following pattern:

- **Set Warehouse**
  The system needs to know which warehouse is used. Check whether a warehouse has been set yet in `/SCWM/CL_TM=>SV_LGNUM`. If not, set the warehouse you are working in. To do this, use `/SCWM/CL_TM=>SET_LGNUM(...)`. Do not change the warehouse inside a process! Also, do not use any other parameter of this method.

- **Lock**
  Ensure that the objects (such as deliveries) you want to change will be locked by the methods you are using. Establish whether you need to do explicit locks. When you use locks, ensure that you use the same locking level as in the SAP standard. For example, if SAP locks deliveries for similar processes on item level and you now lock on header level, this may cause locking errors in other (parallel) processes.

- **Update**
  Now updates on objects (for example, deliveries, warehouse tasks, and so on) can take place. Ensure that you are using the appropriate functions. Do not directly access the database to change data. Direct database changes have the side effect that some automated processes (such as PPF action, status updates, quantity calculations, and so on) that may run after such changes are not started, for example. Direct database updates can also cause severe data inconsistencies.

- **Check Errors**
  Evaluate the exceptions and messages that are returned by these functions/methods. It is essential to evaluate this information to establish whether or not a data change was successful. Not using this information may lead to data inconsistencies if data is later still saved, for example. Some more information about how messages/errors are returned is provided in the chapter
  - **No errors occurred**
    Usually, a success message or no message/exception indicates that the data change was successful. Now you can save the data and end the LUW. For details, see the following chapter “2.3 Saving Data Changes to End an LUW”
  - **Errors occurred**
    Usually an E or A message or an exception indicates that the data change was not successful. Because, typically, you do not know which data the system changed and whether everything is consistent you must not continue the process. It is first necessary to end the LUW and return to a consistent state. For details, see the following chapter “2.4 Ending an LUW in the Case of Errors or Undo Changes.”.
Note:

- The description of functions and methods in this document does not mean that these functions/methods have been released by SAP or that SAP guarantees that they will be kept stable. SAP may change/remove them without notice. Also, do not use any other methods or parameters from the classes mentioned. For example, /SCWM/CL_TM has multiple other methods. Use only the methods mentioned in this document.
- The methods, functions, and classes mentioned in this document are not official programming interfaces and have not been released for customers/partners. They can be changed or deleted by SAP at any time without prior/further notice. Any use is at your own risk.
- The above only applies to EWM objects. These can be recognized by the fact that they usually use the namespaces /SCWM/ and /SCDL/. For other objects (for example, /SCMB/ or SAP basis objects), other transaction rules may exist which are not described here.
2.2 A Sample LUW to Change a Customer Field

The following sample program illustrates a possible LUW. It describes the changing of customer data in a delivery. It contains basic error handling and transaction control. You will find there the above-mentioned steps regarding what an LUW should look like. These steps (setting warehouse, lock, update, and so on) are highlighted in yellow in the code.

Note: The program does not contain any checks (such as status checks) if, for example, a change of the delivery is allowed. For example, a delivery for which a GI has been posted should usually never be changed.

In the example, a customer-specific field is changed. It is very dangerous to change any other SAP fields because you usually do not know whether a field change is allowed and, if so, when/how, or the consequences of such a change. For example, if execution has started, changing may lead to problems in the process. The same applies to actions or other methods. For example, the service provider will allow you to change the product or quantity, or delete items. But without knowing the exact effects and consequences of these changes/actions it is very dangerous to use them. Also note that the example only applies to a delivery change. If warehouse tasks or any other objects are changed, the code (especially methods for locking and saving) may look different.

REPORT ZUPDATE_HEADER_EEW_DATA.

* This sample program shows how one an outbound delivery order (ODO)  
* a customer specific field (Z_ZUSATZ) is filled/changed.  
* The program does a locking and reading of the data  
* it then changes the EEW field  
* the program also contains error handling  
* It also considerses validation errors  
* based on if errors occurred or not it saves or rejects (ROLLBACK) the changes.  
* The program uses the delivery service provider (SP).  
* The program is ment to be used as a separate program, so not to be used inside a BADI or  
* other already running programs (as the setting of the warehouse/save/rollback will destroy a running LUW/transaction)

* Note: The program is only for demo purpose. It is not ment for any  
* productive usage.

DATA:

lo_sp TYPE REF TO /scdl/cl_sp_prd_out,
lo_message_box TYPE REF TO /scdl/cl_sp_message_box,
l_t_a_head TYPE /scdl/t_sp_a_head,
l_t_sp_k_head TYPE /scdl/t_sp_k_head,
l_s_sp_k_head TYPE /scdl/s_sp_k_head,
l_t_a_head_eew TYPE /scdl/t_sp_a_head_eew_prd,
l_t_a_head_eew_out TYPE /scdl/t_sp_a_head_eew_prd,
l_s_sp_action TYPE /scdl/s_sp_action,
l_v_rejected TYPE boole_d,
l_v_error_occurred TYPE boole_d,
l_v_validation_error_occurred TYPE boole_d,
l_t_return_codes TYPE /scdl/t_sp_return_code,
l_t_validation_messages TYPE /scdl/dm_message_tab,
l_t_messages TYPE /scdl/dm_message_tab.

FIELD-SYMBOLS:

<ls_a_head_eew> TYPE /scdl/s_sp_a_head_eew_prd,
<ls_messages> TYPE /scdl/dm_message_str.

* create service provider for processing delivery and and message box  
* the service provider is not used here for a UI (so no attribute handler is used)
TRY.
  CREATE OBJECT lo_message_box.

  CREATE OBJECT lo_sp EXPORTING
    lo_message_box = lo_message_box
    iv_doccat = /scdl/if_di_doc_c=>sc_doccat_out_prd
    iv_mode = /scdl/cl_sp=>sc_mode_classic.
ENDTRY.

* set warehouse that is used
  /scwm/cl_tm=>set_lgnum('EWMZ')

* fill GUID of delivery header
  CLEAR ls_sp_k_head.
  ls_sp_k_head->docid = '00000000000100442833000000000000'.
  APPEND ls_sp_k_head TO lt_sp_k_head.

* try to lock (also creates the delivery instance immediately)
  clear lt_return_codes.
  clear lv_rejected.
  lo_sp->lock( EXPORTING
    inkeys = lt_sp_k_head
    aspect = /scdl/if_sp_c=>sc_asp_head
    lockmode = /scdl/if_sp1_locking=>sc_exclusive_lock
    IMPORTING
      rejected = lv_rejected
      return_codes = lt_return_codes ).

* check if any error occurred
  READ TABLE lt_return_codes TRANSPORTING NO FIELDS WITH KEY failed = abap_true.
  IF sy-subrc = 0 OR lv_rejected = abap_true.
    lv_error_occured = abap_true.
  ENDIF.

* if no error so far...
  if lv_error_occured = abap_false.
    * select customer fields EEW for the delivery
      clear lt_return_codes.
      clear lv_rejected.
      lo_sp->select( EXPORTING
        inkeys = lt_sp_k_head
        aspect = /scdl/if_sp_c=>SC_ASP_HEAD_EEW_PRD
        OPTIONS IMPORTING
          outrecords = lt_a_head_eew
          rejected = lv_rejected
          return_codes = lt_return_codes ).

* check if any error occurred
  READ TABLE lt_return_codes TRANSPORTING NO FIELDS WITH KEY failed = abap_true.
  IF sy-subrc = 0 OR lv_rejected = abap_true.
    lv_error_occured = abap_true.
  ENDIF.

    loop at lt_a_head_eew ASSIGNING <ls_a_head_eew>.
    * now fill the customer specific field Z_ZUSATZ
      <ls_a_head_eew->Z_ZUSATZ = '1'.
    endloop.
  endif.

* if no error so far...
  if lv_error_occured = abap_false.
    * update customer fields EEW for the delivery
      clear lt_return_codes.
      clear lv_rejected.
      lo_sp->update( EXPORTING
        inrecords = lt_a_head_eew
        aspect = /scdl/if_sp_c=>SC_ASP_HEAD_EEW_PRD
        OPTIONS IMPORTING
          outrecords = lt_a_head_eew_out
          rejected = lv_rejected
          return_codes = lt_return_codes ).
* check if any error occurred
  READ TABLE lt_return_codes TRANSPORTING NO FIELDS WITH KEY failed = abap_true.
  IF sy-subrc = 0 OR lv_rejected = abap_true.
    lv_error_occured = abap_true.
  ENDF.
endif.

* if no error so far...
if lv_error_occured = abap_false.

* validate the delivery (also triggers determinations)
  this is an optional step. It is assumed in this example that if validation errors occur
  the delivery should not get saved.
  * If also deliveries with validation errors (blocked status) should get saved,
  * the error handling has to distinguish between validation errors and other errors
  * validation error messages are in the message box and are not returned as REJECTED or RETURN_CODES

  ls_sp_action-action_code = /scdl/if_bo_action_c=>sc_validate.
  clear'lt_return_codes.
  clear lv_rejected.
  lo_sp->execute( EXPORTING
      aspect = /scdl/if_sp_c=>sc_asp_head
      inkeys = lt_sp_k_head
      inparam = ls_sp_action
      action = /scdl/if_sp_c=>sc_act_execute_action
      IMPORTING
        outrecords = lt_a_head
        rejected = lv_rejected
        return_codes = lt_return_codes ).

* check if any error occurred
  READ TABLE lt_return_codes TRANSPORTING NO FIELDS WITH KEY failed = abap_true.
  IF sy-subrc = 0 OR lv_rejected = abap_true.
    lv_error_occured = abap_true.
  ENDF.
endif.

* get all messages that occurred. Get the always as validation messages
  * are also of interest
  lt_messages = lo_message_box->get_messages( ).

* build two tables, one with validation messages and one with "real" errors
  loop at lt_messages ASSIGNING <ls_messages> where consistency_message = abap_true.
    append <ls_messages> to lt_validation_messages.
  delete lt_messages.
endloop.

  loop at lt_messages TRANSPORTING no fields where msgty ca 'EAX'.
    lv_error_occured = abap_true.
    exit.
  endloop.

  loop at lt_validation_messages TRANSPORTING no fields where msgty ca 'EAX'.
    lv_validation_error_occured = abap_true.
    exit.
  endloop.

* now save delivery dependant on if error occurred or not.
  * here validation errors are also considered. This depends on the business logic.
  if lv_error_occured = abap_false and lv_validation_error_occured = abap_false.
    clear lt_return_codes.
    clear lv_rejected.
  lo_sp->save( IMPORTING rejected = lv_rejected ).
  endif.

* check if during save serious errors occurred.
  IF lv_rejected = abap_true.
    lv_error_occured = abap_true.
  endif.

* if errors occurred then get the messages again
  lt_messages = lo_message_box->get_messages( ).
  ENDF.
endif.
* now do a commit (here with wait) or rollback dependant on if errors occurred or not
if lv_error_occurred = abap_false and lv_validation_error_occurred = abap_false.
  commit work and wait.
  /scwm/cl_tm=>cleanup( ). "Clear buffers and release locks" else.
  rollback work.
  /scwm/cl_tm=>cleanup( ). "Clear buffers and release locks" endif.

* now for example, messages could be displayed
2.3 Saving Data Changes to End an LUW

This chapter describes the handling in the event that application objects have been changed and should be saved and no errors/exceptions have occurred.

- First the posting methods/functions of the objects you changed need to be called. In the case of the delivery and when a service provider is used, this is usually a SAVE method. If, for example, you changed/confirmed warehouse tasks/deliveries, call the SAVE method of the corresponding service provider. You should only call one SAVE method. Do not call multiple ones together. Only call multiple ones if you are absolutely sure that this is required (because, internally, one SAVE method usually calls further posting functions automatically, for example). If you now call multiple ones, this could cause a duplicate calling in the event that one SAVE method has already called the other posting functions. If you are unsure which one to use, contact your SAP consultant). On the other hand, ensure that you are calling the right SAVE methods. This means that if you have changed a delivery and you call a method that saves S&R (shipping and receiving) or WT (warehouse task) data, ensure that they also save delivery data automatically. If they do not, you need to call a method that saves the delivery data separately. In short, ensure that you are calling the right SAVE method for the changed objects.
  - If the SAVE method did not return errors/exceptions (evaluate the return parameters), then you can do a COMMIT WORK (or COMMIT WORK AND WAIT should you want to continue working on these objects). If the SAVE method returned an error/exception, proceed as mentioned in “2.4 Ending an LUW in the Case of Errors or Undo Changes.”

- After a successful SAVE, you need to call the CLEANUP method. (If you are using a service provider, use this, otherwise use /SCWM/CL_TM.) It is mandatory to call this method. Otherwise the system is not yet consistent. For example, locks have not all been released and the buffered data may still contain changes and not reflect the database state.
  - Use the parameter IV_REASON = /SCMB/IF_SP_TRANSACTION=>SC_CLEANUP_END if you do not want to keep the deliveries you have just changed in the memory. (For example, because you want to continue processing other deliveries.) This is the default parameter and also the recommended value.
  - Use the parameter IV_REASON = /SCMB/IF_SP_TRANSACTION=>SC_CLEANUP_COMMIT if you want to keep the deliveries you have just changed in the memory (for example, if you want to read them right afterwards). This value is not recommended to be used.

- Now you can continue with the next LUW.

It is important that the following sequence is adhered to. Never call the CLEANUP and then the COMMIT WORK, for example.

The sequence is also important regarding enqueue locks. In most cases, the enqueue SCOPE = 2 is used. However, because the delivery has an internal buffer that retains the information about locks, for example, direct ENQUEUE or DEQUEUE functions are not allowed for locking or unlocking deliveries. After the above CLEANUPS, the locks of the delivery are released. This also means that you need to lock the objects again if you then start a new LUW.
2.4 Ending an LUW in the Case of Errors or Undo Changes.

This chapter describes the handling if application objects have been changed and errors/exceptions have occurred or the changes should not be saved.

- First call a ROLLBACK WORK to get rid of UPDATE TASK functions and/or any other updates which may be registered. This is also necessary to undo changes to persistent classes.
- After this, you need to call the CLEANUP method (if you are using a service provider, use this, otherwise use /SCWM/CL_TM). It is mandatory to call this method, otherwise the system is not yet consistent.
  - Use the parameter IV_REASON = /SCMB/IF_SP_TRANSACTION=>SC_CLEANUP_END. This is mandatory. Do not use a different value.
- Now you can continue with the next LUW.

The sequence is also important regarding enqueue locks. In most cases, the enqueue SCOPE = 2 is used. However, because the delivery has an internal buffer which retains the information about locks, for example, direct ENQUEUE or DEQUEUE functions are not allowed for locking or unlocking deliveries. After the above CLEANUPS, the locks of the delivery are released. This also means that you need to lock the objects again if you then start a new LUW.
2.5 Typical Situations that May Lead to Errors/Inconsistencies

This chapter provides some examples of transactions. The programs contain some typical problematic parts that may lead to inconsistencies. The examples are intended to demonstrate where to be especially careful.

2.5.1 Example 1

You have built your own transaction that allows the user to change data. Inside the code, you want to change data of objects (for example, deliveries) D1 and D2. You first change D1. Then your program recognizes that D2 cannot be changed because the user has entered a wrong input value. You issue an error message and the control is returned to the UI.

Now the user continues in the UI.

The problem here is that after an error is recognized control is given back to the UI without return to a consistent state. Here the handling is described in the chapter “2.4 Ending an LUW in the Case of Errors or Undo Changes.” Otherwise it may be that the user performs a different step (=LUW) in the UI. If this new step changes data D3 and saves it, then changes D1 and D3 are saved (because D1 was not undone).

Comment:
- Issuing messages to the user (at least at lower programming levels) using a MESSAGE command without using the INTO statement is a potential source of inconsistencies. The reason is that this will immediately end the current program flow and there will be a return to the UI. In this case, the program control has no chance to react to such an error (and it cannot be returned to a consistent state by using ROLLBACK/CLEANUP for example). Therefore, this is also not used in the EWM standard. In the EWM standard, messages are usually collected and given back to the top level (using message containers or instances of message classes, for example). On the top level, they can then be displayed.

2.5.2 Example 2

You have built your own transaction which allows the user to change data. Inside the code you want to change data of D1 and D2. You change D1 and D2. Then you call a COMMIT WORK (but no SAVE method).

The result is that it was possible to post some parts of the data but not others, for example. This can be the case, for example, if persistent classes are used inside, because they are posted directly on a COMMIT WORK and do not require specific SAVE or other update functions:

2.5.3 Example 3

You have built your own transaction that allows the user to change data. Inside the code you want to change data of D1 and D2. You change D1 and D2. Then you call a SAVE and a COMMIT WORK (but no CLEANUP). If you now continue with the next LUW, it may be that some data in the buffer has not been reset. This can cause data inconsistencies the next time a SAVE is called.

2.5.4 Example 4

You call a function/method that updates data D1 and already itself calls a SAVE method. If you now (again) call a SAVE method, this may cause inconsistencies. Therefore never call a SAVE method multiple times without ending the LUW properly. Also, be sure that you know the function/method you are using to update data. Some functions/methods offer parameters to perform SAVE automatically. Especially in such cases, be sure that duplicate SAVEs are avoided.
2.6 Further Things to Keep in Mind

2.6.1 Context of Program

If you want to build your own application/code/LUW, first check the context in which your code is executed.

- Was an other LUW executed before your code? If so, are you sure that it ended in a consistent state?
  - For example, the previous LUW ended with an error and did not clear the changed data properly. If you were now to continue with your LUW, you may also save the previous (inconsistent) data.
  - Therefore you need to ensure that previous LUWs ended properly. If you are unsure, do a CLEANUP (TV_REASON = END) and a ROLLBACK WORK to clear any previous data fragments.
- You need to be sure about what happened before your program is called. Otherwise typical situations can be:
  - Data was changed before your code but the previous LUW has not yet ended. If you now create a new LUW inside a previous LUW, you may disrupt the program control. Example: inside a BADI implementation or modification you call a CLEANUP/SAVE or COMMIT WORK/ROLLBACK WORK).

2.6.2 Use Proper Functions to Save and/or Clean Up Data

This is often a reason for inconsistencies when an object (for example, delivery D1) is changed. A SAVE or CLEANUP method that was not designed to save or clean up delivery data is then called in the program.

This is especially the case if you build transactions that change different types of objects (for example, deliveries, HUs, warehouse tasks, waves, and so on).
2.6.3 Types of Messages/Error Handling

Information about whether the call was successful or subject to errors is returned in different ways, depending on the methods/functions you are using. As mentioned before, it is essential that messages (especially error messages) are handled correctly. In the following, some examples of how error situations may be notified are set out. It is not complete, because different message structures, message classes and so on may be used, for example. The important thing is that it is shown that there are different ways in which errors can be returned and that it is very dangerous to ignore them (if they contain error messages).

Note that if you get an instance of LO_MESSAGE or a table with messages, for example, this does not automatically mean that an error has occurred. It first only means that messages have been issued. You need to evaluate the message type (for example, success, information, error, and so on) to see which type is involved. (The yellow text line shows what should be handled and/or how.)

- **Exceptions in function calls:**

```plaintext
CALL FUNCTION 'XYZ'
  EXPORTING
  ....
  IMPORTING
  ...
  EXCEPTIONS
  not_found = 1
  wrong_input = 2.

=> Handle error situation in case SY_SUBRC <> 0.
```

- **Exceptions in method calls**

```plaintext
TRY.
  CALL METHOD 'XYZ'
  EXPORTING
  ...
  RECEIVING
  ...
  CATCH /scwm/cx_core .
  => Handle exception situation here.
ENDTRY.
```

- **Service provider calls**

```plaintext
  lo_serviceprovider->update{
    EXPORTING
    ...
    IMPORTING
    Outrecords  = lt_outrecords
    Rejected    = lv_rejected
    return_codes = lt_return_codes }.

=> Handle error in case REJECTED = 'X' or if RETURN_CODES contains an entry where FAILED = 'X'. Both cases need to be considered independent of each other.

Details about the error reason can be retrieved from the so called message box. E.g.

```plaintext
  lt_messages = lo_message_box->get_messages{ }
```
● **Method calls with message instance**

```plaintext
CALL METHOD lo_myclass->update
  EXPORTING...
  IMPORTING
    eo_message = lo_message
=> Handle errors. LO_MESSAGE has to be checked if it contains E or A messages
```

● **Combination of exceptions and messages**

```plaintext
TRY.
  /scwm/cl_xyz=>update(
    EXPORTING...
    IMPORTING...
    eo_message = lo_message).
  CATCH /scwm/cx_error error INTO lx_error.
=> Handle exception situation here.
ENDTRY.
=> Handle errors. LO_MESSAGE has to be checked if it contains E or A messages
```

Note that here the two situations need to be handled separately! It may be that the exception is raised without EO_MESSAGE populated, but also that EO_MESSAGE is populated without an exception being raised. So both cases need to be considered.
2.7 Further Information

If you are not familiar with terms such as SAP LUW, take a look at http://help.sap.com. Use the “Search Documentation” function to search SAP Netweaver for “Transactions and Logical Units of Work”. There, see SAP LUW, for example.

Note that this only gives you a general overview. As mentioned in the foregoing, to work with SAP LUWs in SAP EWM, you need to consider the topics set out above..