Updates in the R/3 System (BC-CST-UP)

HELP.BCCSTUP

Release 4.6C

Copyright

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Contents

Updates in the R/3 System (BC-CST-UP)

Updates in the R/3 System (BC-CST-UP)

Purpose

In the R/3 System, a business process is mapped by means of an SAP transaction, which can include several screen changes (for example, creating an order). The changes to the data caused by this business process should be written to the database in full or not at all. If the operation is canceled while the transaction is being executed, or if an error occurs, the transaction should not make any changes to the database. These activities are handled by the SAP Update System, which is described below.

The Update System also enhances reliability and performance, and makes it easier to restore the data when changes are made to the database.

Integration

Updates are closely related to the R/3 lock concept, which is described in the documentation on The R/3 Lock Concept [Ext.].

You can find further documentation on updates in Update Techniques [Ext.] in the ABAP documentation. This documentation explains how an application programmer can control the update logic using ABAP statements.

Features

The Update System is used to lighten the workload of the SAP transactions when time-consuming changes are made to the database. These are carried out asynchronously in special update work processes. The Update System also avoids rollback problems that occur as a result of the differences between the logical units of work in an SAP transaction and in the database.

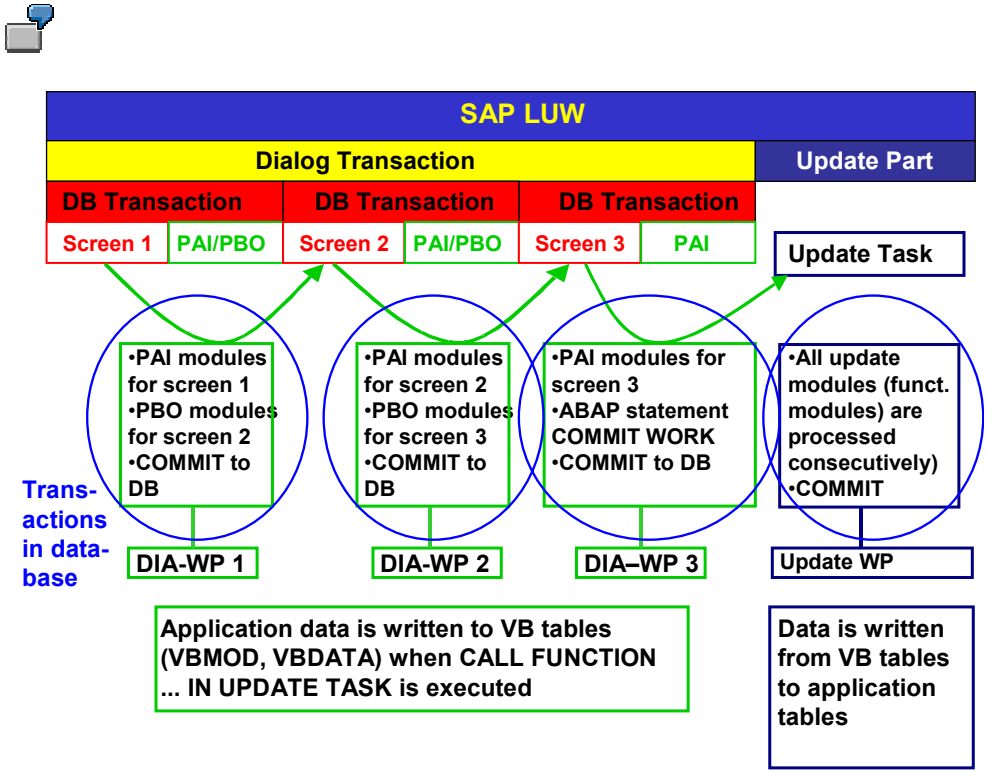
Exactly how an update is made in the R/3 System is described in the section entitled Functional Description of Updates [Page 7].

This documentation also contains sections on

- Update Management [Page 31] (transaction sm13)
- Monitoring Updates [Page 56]
- Analyzing and correcting update errors [Page 60]

Functional Description of Updates

The following graphic shows you how the R/3 System and the database interact when an SAP transaction is being executed, and illustrates the difference between a database LUW (**L**ogical **U**nit of **W**ork) and an SAP LUW.



The graphic shows an example of an **SAP LUW**. The dialog transaction extends over 3 screens. Each screen can be processed by a different (dialog) work process because while the system is waiting for the user to make an entry, the work process can be assigned a different task by the dispatcher.

The dialog transaction is completed with the statement COMMIT WORK; the update area of the LUW is then initiated: the update server (update task in the graphic) transfers the update request [Page 9] to an update work process. The time interval between two screen changes in the dialog transaction corresponds to a **database LUW** (completed with a COMMIT to DB); the update area is performed in a database LUW. The update area is described in greater detail in the section entitled The Update Process [Page 12].

Database LUW vs. SAP LUW

In the case of the **database**, an LUW is a **sequence of data operations that cannot be divided up**. The operations are either carried out in full or not at all. Database LUWs are modules which go to make up the database procedures for consistent data processing. An R/3 transaction can include several database LUWs (see example above), each of which can be terminated with a database COMMIT, which is generated automatically.

Functional Description of Updates

By way of contrast, an LUW for the **R/3 System** is a **business process, which cannot be divided up**. The process is either executed in full or not at all. **The SAP LUW of an R/3 transaction usually has to contain several database LUWs.** Under normal circumstances, it contains a dialog transaction (which maps a business process), and the command for writing the data to the database (update).

In each database LUW, data is written to the database to special **update tables** (and not to the application tables). Once the dialog area has been completed, all of the data in a **database LUW** is copied to the application tables: the update request [Page 9] is then performed. This is described in greater detail in the section entitled The Update Process [Page 12].

Advantages of the SAP Update System

The update system offers a number of advantages over database changes that are made directly in a transaction. These include:

- Enhanced performance for dialog transactions: time-consuming database changes are carried out asynchronously at the end of the LUW after the user has already moved onto the next LUW.
- Problems that arise as a result of different types of logical units of work (for database systems and for R/3 transactions) are avoided. By bundling updates [Page 15], the Update System supports critical rollback and restore functions for transactions.

To enhance performance even further, the application programmer can configure different types of updates (see V1 and V2 Update Modules [Page 21]):

- Time-critical V1 updates
- Non-time-critical V2 updates, which depend on V1 updates
- Non-time-critical V2 updates, which are gathered together and processed later (collective runs [Page 11])

See also:

Further details on the update process can be found in the following sections:

The Update Process [Page 12]

Distributed Processing of Updates [Page 20]

Main System Profile Parameters for Updates [Page 25]

Reporting Update Problems [Page 29]

Update Request

Definition

An update request or *update record* describes the data changes defined in an SAP LUW, which are carried out either in full or not at all (in a database LUW). (This only applies to V1 updates. V2 updates are triggered once the V1 update has been completed, and therefore take place in a separate database LUW.) See also The Update Process [Page 12].

An update request is identified by means of its **update key**.

Use

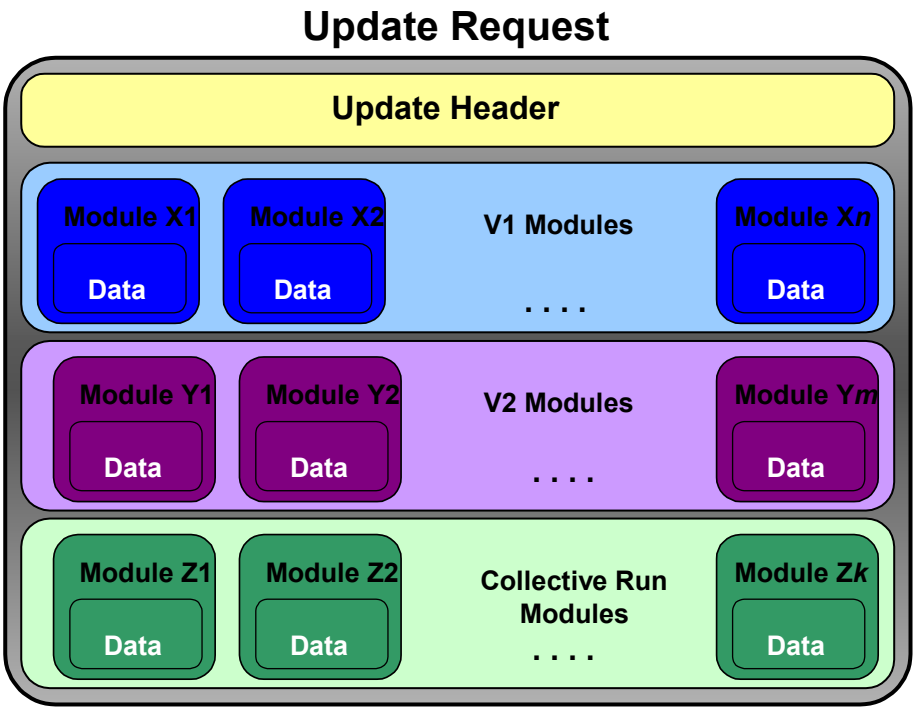
Once the dialog transaction has been completed (`COMMIT WORK`), and the update task has been called up, the update header is created. The update request is then created. The update data is stored in the update modules (function modules), which were created with the ABAP statement `CALL FUNCTION '…' IN UPDATE TASK`. The function module type is defined in the transaction for maintaining function modules (transaction `se37`, Function Builder [Ext.]). See below for further details.

Structure

An update request comprises an update header, V1 modules (or components), V2 modules and a collective run [Page 11].

An update module corresponds to a function module, and contains the update data and, in certain cases, error information, which is generated if the update is canceled.

The following graphic illustrates the structure of an update request.

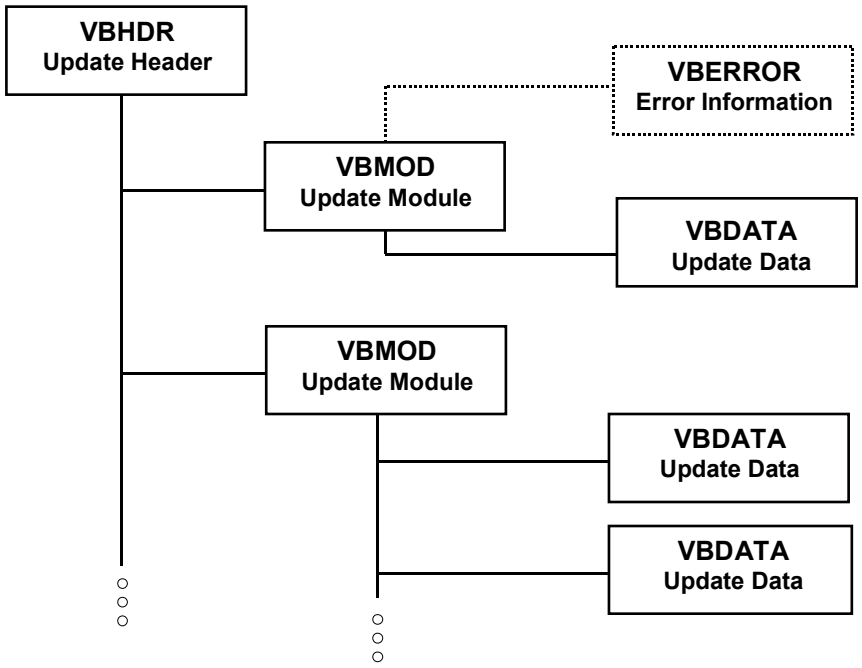


Integration

The following update tables in the database contain the following information:

Update Table	Contents/Description
VBHDR	Update headers (one per update request), see also Displaying the Update Header [Page 35]
VBMOD	Update modules (one per function module), *n* V1 modules and *m* V2 modules per update request (see also Displaying Update Modules [Page 37])
VBDATA	Data which is transferred to the modules (variables, structures, internal tables)
VBERROR	Error information which is generated if an update is canceled

The update tables are structured as follows:



You can also call this table up in the R/3 System with the table maintenance transaction (se16). Detailed information on maintaining tables can be found in the relevant documentation.

Collective Runs

Definition

In addition to V1 and V2 function modules, the system also supports collective run function modules. In contrast to the other modules, these are not updated until a special report (report `RSM13005`) starts the update (in background mode); in other words, they are not updated automatically. All function module calls are then collected, aggregated (see example) and updated together. They are handled in the same way as V2 update modules.

Use

When you program a transaction, you use collective runs if a function module is called up on a regular basis, and you want to avoid a situation whereby each individual call causes the database to be updated.

Time-critical updates must not be carried out using collective runs. In this case, the collective run is not carried out until (long after) the time-critical V1 updates have been completed.

One of the function modules increments a statistical entry by one. This is called up 10 times during the course of the transaction. If you implement this as a V2 function module, the function module is updated 10 times after the V1 has been completed; this means that the database is updated 10 times.

If you use classify this as a collective run, you can perform the update at any time in one single operation: the program establishes that the statistical entry has to incremented 10 units and carries this out in a database operation.

Integration

The collective run modules can be viewed in update management [Page 31] until the report that triggers processing is started. Update records whose V1 and V2 modules have been processed correctly, but which still contain collective runs, are set to status `V2` (see The Most Important Update Statuses [Page 17]).

The Update Process

Purpose

The Update System is used, for example, to lighten the workload of the R/3 transactions when time-consuming changes are made to the database. The changes are carried out asynchronously - usually with short delays in between - by special update work processes.

This is why the Update System is widely used in R/3 transactions (by almost every transaction that changes business data), although transactions can also change the data directly in the database.

Prerequisites

The application programmer decides whether, and if so, how the Update System is used for developing transactions. The different options open to the programmer are described in detail in the ABAP manual in the section entitled Update Techniques [Ext.].

When you are working with the SAP Update System, it is important to make sure that only **`insert`**s, **`update`**s and **`delete`**s are performed in the update. The necessary data (with **`select`** etc.) should, of course, be collected beforehand. Programming that has not be carefully thought through can result in poor system performance, and can, in certain cases, cause serious problems.

Process Flow

As illustrated in the graphic in Functional Description of Updates [Page 7], **`COMMIT WORK`** and the update task are called up at the end of a transaction; the dialog transaction comes to an end, and the update area of the SAP LUW is started. The following graphic illustrates the necessary actions and the sequence in which they carry out the different work processes [Ext.].

Update Procedure

- **Complete transaction (COMMIT WORK)**
- **Call update task**
  - **Complete VBHDR entry**
  - **Search for update server for V1 update (update dispatching)**

  Dialog Process
  - **Update server processes V1 modules**
  - **COMMIT to database**
  - **Release locks**
  - **Search for update server for V2 update (update dispatching)**

  V1 Update Process (UPD)
  - **Second update server processes V2 modules**
  - **COMMIT to database**

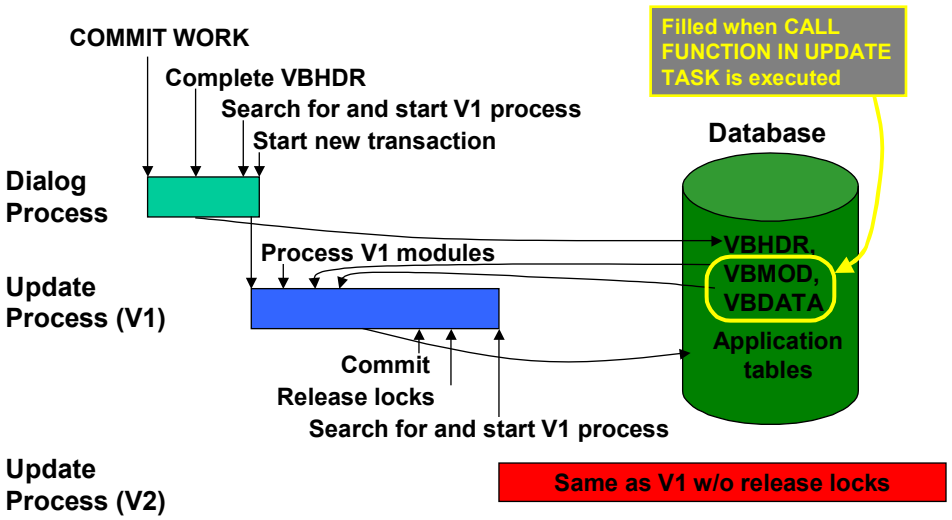
  V2 Update Process (UP2)
- **Start next transaction**

After the transaction has been processed, the dialog process completes the VBHDR entry (the update header of the update request [Page 9]) and searches for an update server for the V1 update. This process is described in greater detail in the section entitled Update Dispatching with Load Balancing [Page 22].

The update server distributes the tasks to an update work process. This processes the V1 modules of the update request, triggers a COMMIT to the database, and releases the R/3 locks on the update request (see The R/3 Lock Concept [Ext.]). The work process then searches for an update server for the V2 update, providing V2 update modules exist.

A V2 update server then passes this onto a V2 work process, which processes the V2 modules and triggers a COMMIT to the database.

The following graphic illustrates this from the point of view of the different work processes. The graphic also shows the times at which changes are made in the database.



Processing the V1 modules involves transferring the contents of the update tables VBMOD and VBDATA to the application tables of the database. The changes are not actually made to the tables in the database until the database LUW in which this takes place is completed. The R/3 locks are released and, if V2 update modules exist, the V2 update is started.

Result

If the update is completed without any errors, the update record is no longer listed in update management [Page 31] and the data is successfully processed.

Further information on the update process can be found in the following sections.

Bundling Updates [Page 15]

Synchronous and Asynchronous Updates [Page 16]

The Most Important Update Statuses [Page 17]

Error Handling and Data Consistency [Page 19]

Bundling Updates

An R/3 transaction can, depending on how it is programmed, carry out database changes directly without using the SAP Update System; the database changes are made at the end of a database LUW.

The transaction can, however, also use the SAP Update System and bundle the updates of several database LUWs (see Functional Description of Updates [Page 7]).

Bundling is an ABAP programming technique which collects database changes and performs these together at the end of a transaction. (See also Update Techniques [Ext.].)

In the examples described so far, the updates have always been bundled.

Bundling updates involves combining important changes made by an R/3 transaction in a database LUW. Only changes that are performed in a single database LUW can be reversed. **If it is important to be able to reverse all of the changes made with an R/3 transaction for the sake of maintaining data consistency, all of the changes must be bundled in one database LUW.**

In order to bundle the updates, the special ABAP form routines and function modules for the database changes are not processed until the ABAP keyword `COMMIT WORK` is output.

The R/3 System also features its own locking mechanism for restricting data access. This mechanism is used, in certain cases, to hold R/3 locks within an R/3 transaction beyond the boundaries of database LUWs. (At the start of a new database LUW, the database locks of the previous LUW are released. The R/3 locks, however, remain active). Further information on R/3 locks can be found in the documentation on The R/3 Lock Concept [Ext.].

Update bundling and the R/3 lock system maintain data integrity in processes that cover several database LUWs, and fulfill the requirements regarding rollback. This means that all of the data changes can be reversed if a runtime error occurs during the update.

Further information on LUWs and bundling can be found in the ABAP manual.

Synchronous and Asynchronous Updates

Database changes that are made via the SAP Update System and transferred to an **update work process**, can be carried out synchronously or asynchronously. The mode is specified in the ABAP source code of the R/3 transactions and cannot be changed dynamically by the user.

The R/3 transaction creates an update request [Page 9] with `CALL FUNCTION … IN UPDATE TASK`, and transfers this to an **update work process**. Data is then written to the update tables at the end of a database LUW.

With **synchronous updates**, the program that outputs the statement `COMMIT WORK AND WAIT` waits until the update work process outputs the status of the update.

Updates generated by batch input sessions are always carried out synchronously. Batch input with `CALL TRANSACTION USING` can be updated synchronously and asynchronously. In addition to this, every 100th update in a background job is carried out synchronously in order to avoid delays in the Update System.

With **asynchronous updates**, the program that outputs the statement `COMMIT WORK` passes the update onto the Update System and does not wait for the update process to respond.

Most of the updates in R/3 application transactions are programmed for the asynchronous mode. **These asynchronous updates are the objects that appear in update management.**

See also: Update Techniques [Ext.] in the ABAP manual.

The Most Important Update Statuses

Update management supports different statuses for update requests; these statuses are displayed in the update record overview (transaction **sm13**) in the `Status` field group.

The status indicates the phase of the update process [Page 12] that the request has reached, or in which the request has become "stuck".

The most important statuses are described below.

As already discussed in the section entitled The Update Process [Page 12], the dialog work process passes the update request onto an update work process after the dialog area has been completed. This then processes the V1 update modules. When the ABAP statement **COMMIT WORK** is received, the data is written to the database and the V2 update is output to a V2 work process (providing V2 modules exist in the update request).

The following statuses are possible during this phase:

Status	Phase
`init`	From the point at which the dialog work process passes the update request onto the update work process up until the COMMIT in the update work process.
`err`	If an error occurs in the `init` phase which prevents the update from being carried out.
`V1`	If the `init` phase is completed successfully, and the V2 modules are passed on for further processing. If no V2 modules exist, this update request no longer appears in the overview.
`V2`	If the V2 modules are also processed correctly, but there is still a collective run (can be regarded as V3) to be carried out. If there is no collective run [Page 11] to be carried out, this update request no longer appears in the overview.
`ok`	If the parameter rdisp/vb_delete_after_execution [Page 25] is set to 2 - in other words, automatic deletion is deactivated - an update that has been completed successfully has the status `ok`. If automatic deletion is activated (default), the update record no longer appears in the overview.

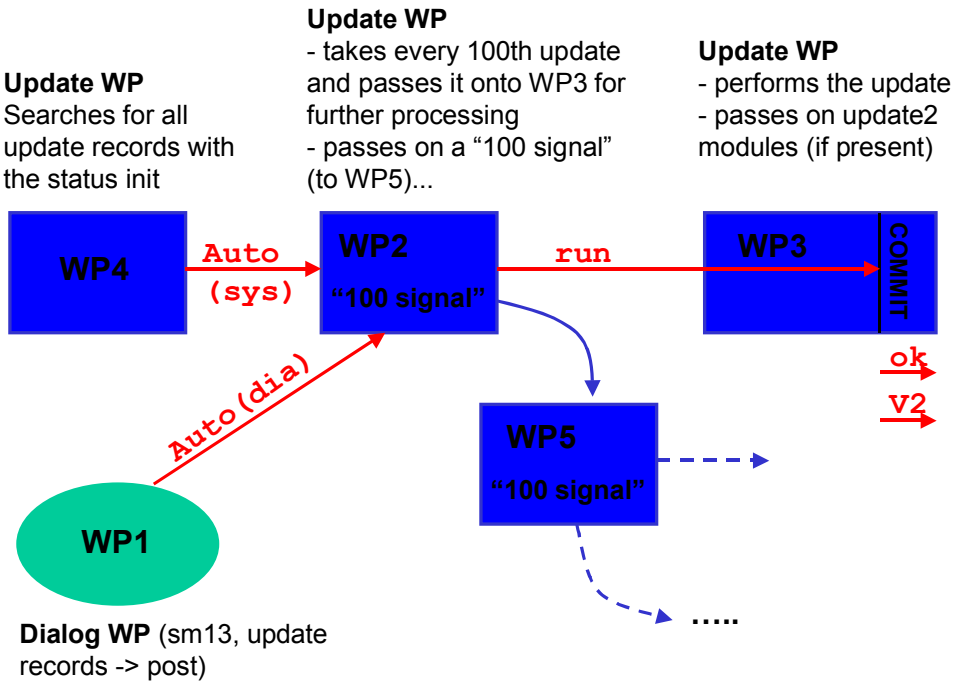
A situation may arise, however, in which an update record becomes "stuck" in the status `init` without switching to the error status `err`. If the record remains set to the status `init` for a prolonged period of time, the record can be updated subsequently in the following ways; the following statuses are then active (see also graphic).

Status	Phase
`auto(dia)`	The update record has been processed manually [Page 40] by the system administrator with transaction **sm13** (*Update records* → *Update*). The dialog work process (WP1) transfers all of these update requests to an update work process (WP2) during which the update record is set to the status `auto(dia)`.

The Most Important Update Statuses

`run`	The work process WP2 collects the requests and passes them on in batches to a further update work process (WP3), which then performs the actual update. The record is set to the status `run` up until COMMIT in WP3.
`auto(sys)`	Each time an update server is restarted, this checks to determine whether the update requests are set to `init`. In this case, it initiates automatic processing of the requests by means of update work processes. This takes place in the same way as when the update is started manually, except that an update work process (WP4) starts everything and not just a dialog work process. The update record is then set to the status `auto(sys)`.

This is illustrated in the following graphic.



If, under exceptional circumstances, the update is not successful the first time, the status `run` corresponds to the status `init` when the update is repeated. If the update becomes "stuck" in the status `auto` or `run`, the status must be reset [Page 41] as only records with the status `init` can be entered with the methods described.

See also:

Monitoring Updates [Page 56]

Analyzing and Correcting Update Errors [Page 60]

Error Handling and Data Consistency

In order to ensure the consistency of the data, an update must be carried out in full or not at all. This is referred to as a **rollback requirement**. If a runtime error occurs in part of the update, all of the critical database changes made as a result of the update must be reversed. Less critical changes cannot be made.

With update bundling [Page 15] and the R/3 lock system, the R/3 Update System ensures that the data remains consistent (see also The Update Process [Page 12]).

The following section describes how the Update System responds if part of the update (V1 or V2) is canceled due to a runtime error. Various scenarios are possible here.

Cancelation in a function module of the update task V1 (IN UPDATE TASK **requested**)

- Updates already made for V1 functions are reversed.
- All other update task requests (V1, V2 or collective run) are discarded.
- The user receives an express mail indicating that the update was unsuccessful (see parameter rdisp/vbmail [Page 25])

Cancelation in a Function Module of the Update Task V2 (IN UPDATE TASK **requested**)

- Updates already made for V2 functions are reversed.
- Updates already made for V1 functions are not reversed.
- An error message appears on the screen if the system has been configured for this.

Cancelation in a Collective Run Function Module

- Collective runs are handled in the same way as V2 update modules.

Distributed Processing of Updates

The task of distributing the update requests [Page 9] (uniformly) among the different update work processes is performed by an **update server**.

The update server, therefore, acts as an intermediary which receives the update request and distributes it to the update work processes.

In R/3 System in which the code pages of the R/3 application servers have different character sets), the update work processes change their codes pages dynamically in line with the code page of the update request to be processed.

Update distribution can be deactivate and configured with parameters of the R/3 System profile. Further information can be found in the section entitled Main System Profile Parameters for Updates [Page 25] or in the update profile parameters in transaction rz11 (use the search string rdisp/vb* in rz11 to search for the parameters).

Types of Update Work Process

There are two types of update work process, one for V1 updates (listed as *Update* processes in transaction sm50/sm51 display) and one for V2 updates (listed as *Upd2* in sm50/sm51 display).

There must be at least one V1 update process in the R/3 System; there can, however, be more. If there is more than one process, update processing is distributed among the work processes on the basis of a load balancing algorithm (see also Update Dispatching with Load Balancing [Page 22]).

V2 processes are optional. These are used to avoid performance problems with time-critical V1 updates by splitting V2 update component processing. (V2 update components usually contain statistical data and are, therefore, less time critical). Further information can be found in the section entitled The Update Process [Page 12] and V1 and V2 Update Modules [Page 21]. If there is no V2 work process, both types of update component are processed in the V1 work process. V2 work processes are also used if collective runs [Page 11] are carried out.

V1 and V2 Update Modules

An update is divided into different modules (see also Update Request [Page 9]). Each module corresponds to an update function module.

There are two types of module.

The R/3 System makes a distinction between **primary**, time-critical (**V1**) and **secondary**, non-time-critical (**V2**) update modules. The system also supports Collective Runs [Page 11] for function modules that are used on a regular basis.

This distinction allows the system to process critical database changes before less critical changes.

- **V1 modules** describe critical or **primary changes**; these affect objects that have a **controlling function** in the R/3 System, for example order creation or changes to material stock.
- **V2 modules** describe less critical **secondary changes.** These are pure statistical updates, for example, such as result calculations.

The V1 modules are processed consecutively in a single update work process on the same application server. This means that they belong to the same database LUW and can be reversed. Furthermore, V1 updates are carried out under the R/3 locks of the transaction that creates the update (see The R/3 Lock Concept [Ext.]). This ensures that the data remains consistent; simultaneous changes to the objects to be updated are not possible.

V2 updates are carried out in a separate LUW and *not under the locks of the transaction that creates them*. If your R/3 System contains a work process for V2 updates, these are only carried out in this system. If this is not the case, the V2 components are processed by a V1 update process.

All V1 modules of an update must be processed before the V2 modules.

Let us assume that a transaction makes planning changes to a material and balance sheet, and updates two sets of statistics.

Each of these changes is represented by means of an update module (call update function module) in the update request [Page 9] – the two planning changes by a V1 update module (time critical), and the statistical changes by a V2 update module (less critical). (The V1 modules have priority, although the V2 modules are usually also processed straight away).

This is described in greater detail in the section entitled The Update Process [Page 12].

Update Dispatching with Load Balancing

Several **update servers** can offer update services in an R/3 System. In this case, the R/3 System automatically distributes the update requests [Page 9] among the available update servers using a process referred to as **update dispatching**.

Update processing is triggered by a request from the dialog server on which the update was generated. This request is sent to a specific update server. The dialog server thus decides which update server is to be used for a particular update.

Load balancing is based on a list of the available update servers. This list is stored on each application server and is automatically updated by the message server when a server is started or stopped, or when its configuration changes as a result of a new operation mode being set. You can display this list by choosing *Goto* → *Server*.

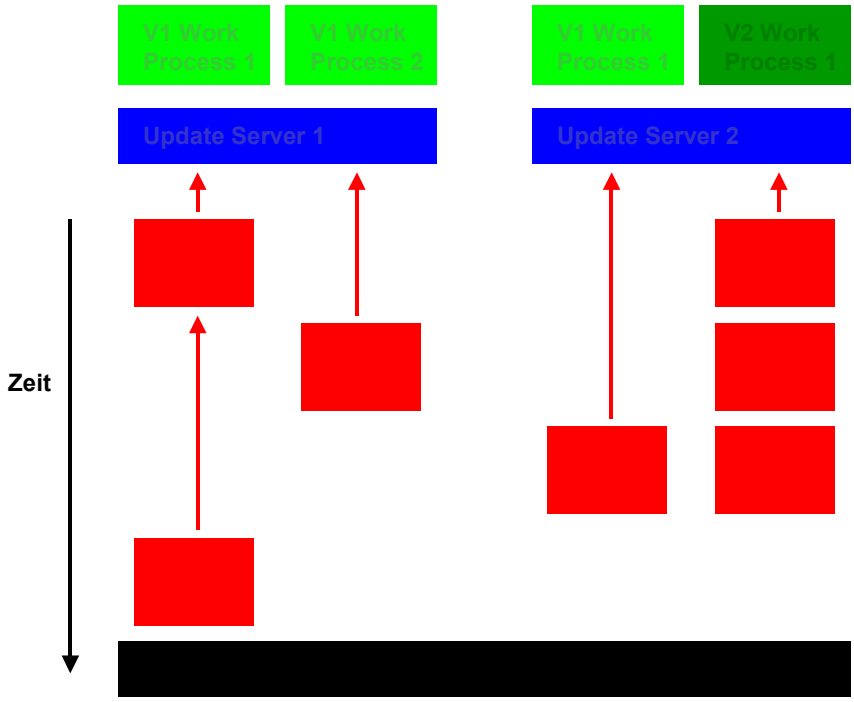
If there is more than one possible update server for processing an update record, the dialog server selects the server to be used on the basis of a load balancing algorithm.

This load balancing algorithm assigns update records to the update servers cyclically. A server sends update requests consecutively to each update server. If the number of requests sent is the same as the number of update work processes on an update server, the server is removed from the list until the start of the next cycle. Once all of the update work processes have been assigned update requests in this way in the R/3 System, the cycle starts again and with all of the update servers.

If a V2 work process *Upd2* has been defined, V2 update modules can only be processed by this work process.

If there are no V2 work processes, V2 updates are processed in V1 work processes. A V2 component must, however, wait until all of the V1 update requests have been processed.

The following example illustrates the load balancing algorithm.



Two work processes run on update server 1 for V1 update modules. Update server 2 has one V1 and one V2 update work process. A load balancing cycle for V1 updates, therefore, comprises three update requests. These are distributed as follows:

1. The first and second update request are sent to update server 1 (number of requests = number of V1 update work processes * rdisp/vb_factor parameter (default value 1; see also Additional System Profile Parameters [Page 27]). Setting the parameter to a value other than 1 is advantageous if an update server is significantly faster and is, therefore, to be assigned more update requests.
2. The Update System switches to update server 2 for the third request. The load balancing cycle is then complete (number of assigned updates = number of update work processes).
3. V1 update request 4 is sent to update server 1 again and the cycle is repeated.
4. V2 updates are processed consecutively by the V2 work process.

Reassignment in the Event of an Update Server Failure

If, for some reason, an update server fails, the other update servers in the system perform any updates that the defective server was not able to handle.

If an update server is shut down, for example, the R/3 message server establishes that the server is not available on the basis of an error message regarding the missing server connection or on account of the connection checks, which are carried out at regular intervals.

Once the message server has been informed of this, it updates the list of application servers and associated services, and distributes the list to the other application servers.

The first update server on the list automatically assumes responsibility for distributing the update requests of the defective server on the basis of the dynamic update assignment criteria described above.

Main System Profile Parameters for Updates

The following table contains an overview of the most important update parameters.

Note that you normally never have to set these parameters directly. Generally speaking, the most appropriate value is the default parameter value; otherwise the parameters are set for you by the profile management function (for example, if you set the number of update work processes in an instance profile).

Parameter	Description
rdisp/vb_dispatching	Activates (1) or deactivates (0) update dispatching with load balancing [Page 22]. Load balancing is normally active. Do not change this value unless instructed to do so by SAP.
rdisp/vbname	Specifies the name of the update server that is to process the updates if load balancing is deactivated (rdisp/vb_dispatching = 0). In the standard system, this parameter specifies the name of an update server (set when the update server is created). If rdisp/vb_dispatching is set to 0, the updates are only processed by the server in rdisp/vbname. If the parameter has not been set correctly (not to an update server), this is reported when the installation check is performed (transaction **sick**).
rdisp/vb_delete_after_execution	Determines whether update records are deleted automatically after they have been processed successfully. In the standard system, this is set to 1 (automatic deletion activated). If set to 2, automatic deletion is deactivated. This value can be used to set the update and database performance. In this case, the report rsm13002 with the parameter DELETE = X should run in the background at least once a day to prevent the update tables from becoming excessively large. See also the section entitled Background Processing [Ext.] in the CCMS documentation.
rdisp/max_vb_server	Maximum number of update servers permitted in the R/3 System. Default = 50 servers.
rdisp/vb_included_server	List of the R/3 update servers, which are to be used to process updates in accordance with the load balancing principle. No updates are assigned to update servers that do not appear in the list. This parameter is empty in the standard system. This means that all active update servers are taken into consideration for the load balancing mechanism. This is generally speaking the optimum value.

Main System Profile Parameters for Updates

rdisp/vbdelete	Specifies the number of days after which the update records are deleted. The parameter is set to 50 days in the standard system. Once this interval has expired, an update record is deleted irrespective of its status (processed, not processed, error etc.). If set to 0, automatic deletion is deactivated. This value should only be set temporarily, and only if an incorrect update record is to be kept for further analysis.
rdisp/vbmail	Specifies whether a user is to be informed via express mail that his update has been canceled. The parameter is activated (value = 1) in the standard system. If an update is terminated prematurely, the system sends an express mail via R/3 Mail to the user that created the update. If set to 0, the express mail is not sent automatically.
rdisp/vbreorg	Specifies whether an update server is to search for incomplete update records and delete these. Update records can be incomplete - if a transaction registers one or more update components - as a result of a screen change in which the transaction that creates the update generates a database commit; the transaction, however, is terminated prematurely and a rollback is triggered. Incomplete update records are of no significance, but take up space in the database. Reorganization is, therefore, active (value = 1) in the standard system. You can deactivate reorganization by setting the value to 0. In this case, you should make sure that the report rsm13002 is run in the background at regular intervals to delete the incomplete update records (see also Background Processing [Ext.]).
rdisp/vbstart	Specifies whether update requests that have not been processed or not fully processed (status = *run*, V2 modules not processed, etc.) are processed automatically when an update server is started. In the standard system (value = 1), update records such as these are processed (status usually = *auto*) (see The Most Important Update Statuses [Page 17]). If set to 0, automatic processing is deactivated. The value of this parameter can normally only be changed by SAP.

To obtain further information on the update parameters, enter the transaction code /nrz11 in the R/3 System and then search for the parameters whose names start with rdisp/vb.

Additional System Profile Parameters

The most important system profile parameters for updates are described in the section entitled Main System Profile Parameters for Updates [Page 25]. The system also supports additional parameters, which can be useful for correcting errors and optimizing update performance. These parameter are described below.

You can access further detailed information if you call up transaction **`rz11`**, search for the parameters and then display the documentation.

Parameter	Description
rdisp/vb_context	You can use this parameter to specify how V1 update dispatching [Page 22] takes place. The parameter is interpreted as a bit mask: Bit 0: the DB group is to be retained during dispatching (see also Configuring Update Groups [Page 44]) Bit 1: an update server on which the current language is installed must be selected for dispatching. The default value 3 (both bits set) specifies, for example, that an update server is selected for update dispatching, which is in the same DB group, and on which the current language is installed. Changes should only be made in order to correct errors. Value range: 0-3
rdisp/vb2_context	This parameter controls update dispatching for V2 updates (see rdisp/vb_context).
rdisp/vb_factor	Update request dispatching takes place in accordance with a simple procedure, which is described in Update Dispatching with Load Balancing [Page 22]. The parameter rdisp/vb_factor can be used to influence the number of update work processes relevant for dispatching in order to assign a greater number of update requests to servers that are particularly fast. The number of requests relevant for dispatching is the product of the current number of update processes and the value of rdisp/vb_factor. After the V1 part of an update request has been processed the V2 part is dispatched in the same way. The parameter value must have the following syntax: rdisp/vb_factor = S=<instance_name>,F=<factor>; … S=<instance_name>,F=<factor>;  rdisp/vb_factor = S=host1_C11_00,F=2.0; S=host2_C11_00, F=2.5;

Additional System Profile Parameters

rdisp/vb_key_comp	You can use the parameter rdisp/vb_key_comp to change the structure of the update key (see Displaying the Update Header [Page 35]). The key usually comprises the date, time, microseconds, host name, system number and work process number. In order to structure the index more favorably, you can move variable components to the start of the key by setting the parameter accordingly.
rdisp/vb_refresh_info	This parameter defines the time interval after which the context of an update server is redetermined. The context includes the number of server update work processes as well as the code page and DB node of the server. The context of an update server can change, for example, as a result of a new operation mode being set. This is why the context is reread every hour.
rdisp/vb_v2_start	This parameter specifies whether the V2 phase of an update request is started automatically at the end of the V1 phase or by a background job after a delay. Value range: 1: The V2 phase is started immediately after the V1 phase 2: The V2 phase is not started automatically

Reporting Update Problems

The Update System draws your attention to update problems as follows:

- The graphical Alert Monitor [Ext.] reports all update problems automatically.
- The Update System sends an express mail if an update is terminated prematurely. The user is informed that an express mail has been received by a dialog popup.
- The user whose update was terminated prematurely receives an error message.
- A system-wide message is displayed if the update is deactivated automatically or manually.
- All terminated updates are maintained in update management [Page 31]. They are assigned the status *err*. See also The Most Important Update Statuses [Page 17]

Express Mails and Error Messages

All users whose updates have been terminated prematurely are notified by the R/3 System of the update error not only by alerts but also by means of an express mail. The mail is sent in the R/3 System in which the problem occurred.

In the standard system, warnings per mail are activated; you can, however, deactivate them with the parameter *rdisp/vbmail* in the system profile (further information on this parameter can be found in the section entitled Main System Profile Parameters for Updates [Page 25] and in transaction rz11).

The R/3 System also displays a message pointing out the update error on the terminal of the user. Exceptions to this include situations in which a remote update is made by means of an RFC, or if the user logged off before the update was processed.

Note on Deactivating Updates

In the event of serious database errors, the update is stopped automatically (further information on this can be found in the section entitled Automatic Update Stop in the event of Database Problems [Page 30] and Deactivating and Reactivating Updates [Page 59]).

In these cases, you and all other users receive a warning message in the status bar or the active mode. All active transactions in the system are interrupted if updates are deactivated.

See also:

Monitoring Updates [Page 56]

Analyzing and Correcting Update Errors [Page 60]

Automatic Update Stop in the event of Database Problems

If a serious error occurs in the database of the R/3 System, the update is stopped automatically. Each database error message which requires the intervention of a database administrator is also passed onto the Update System, which then interrupts the update.

The automatic stop mechanism is valid for all databases supported by the R/3 System.

Let us assume that your Oracle database has just output the message "Table space overflow". The R/3 database interface recognizes the message as a serious error message and outputs a signal to the Update System, which then stops the update. The active transactions are interrupted (an appropriate message is output here) until the update is reactivated.

Stopping the update in the event of database problems makes it easier to restore normal operating conditions after the error has been corrected. Updates are not canceled but are assigned the status *init* or *auto*, which indicates that they have not yet been completed. The updates are then processed automatically when the update is reactivated after the error has been corrected.

The update is not stopped if a local error in an update function module causes an update to be terminated prematurely (see also Error Handling and Data Consistency [Page 19]).

Further information can be found in the section entitled Deactivating and Reactivating Updates [Page 59].

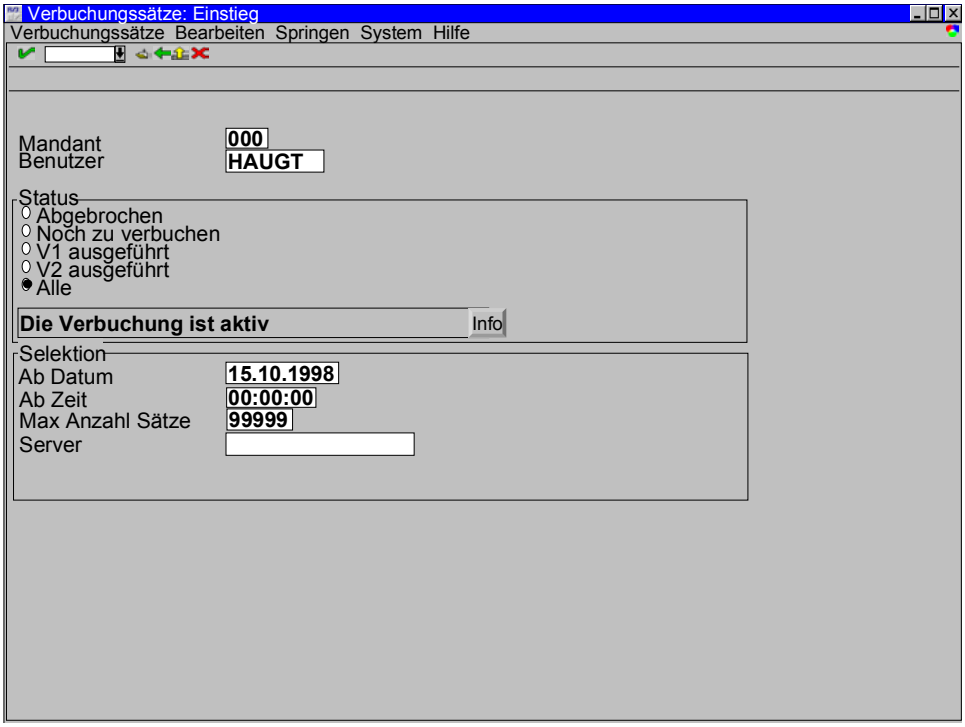
Update Management

Use

Update management is used to

- display updates
- test and debug canceled updates
- reset updates
- delete updates
- display statistics on updates

The update management initial screen (transaction sm13) can be seen below:



Integration

The sections entitled Monitoring Updates [Page 56] and Analyzing and Correcting Update Errors [Page 60] describe the procedures for monitoring updates and handling update errors.

Features

Update management supports the following functions:

Selecting and displaying updates [Page 33]

Analyzing Canceled Updates [Page 38]

Testing Canceled Updates [Page 39]

Update Management

Many of these functions can be carried out more easily in Update Program Administration (Transaction sm14) [Page 50].

Activities

You can call up Update Management from the initial R/3 screen either by choosing *Tools* → *Administration* → *Monitor* → *Update* or - from any R/3 screen - by entering **`/nsm13`** in the command field.

Selecting and Displaying Updates

You can use the following criteria for selecting updates:

Selection options for update records

Option	Description	Default setting
Client	Client in which the update was created	Current client
User	User who created the update	Current user
Type of update record	The following status selection options are available: • *Terminated*: only display canceled updates of the user • *To be updated*: display updates of the user that have not yet been processed (the reason for this is indicated in the display; see below • *V1 executed*: display executed V1 updates of the user • *V2 executed*: display executed V2 updates of the user • *All*: display all update records of the user	All
Date and time	Date and time as of which the updates are to be displayed	Today, 0.00 hours
Max. no. records	Maximum number of update records that can be displayed	99999
Server	Update server on which the update was carried out	

If you leave a field blank or enter *, a generic search is carried out for all possible entries.

Displaying Update Records

This function allows you to display all of the update records that not yet been fully processed.

The selected data records display contains the following statuses (see also The Most Important Update Statuses [Page 17]):

Update records that have not yet been processed are assigned the status ***init*** or ***auto***. Both of these are normal statuses, in other words, you do not need to do anything here as long as the list is not too long, and the updates are not processed.

init indicates that an update record is complete (header, function module calls and data elements exist), but has not yet been processed.

auto indicates that a record is complete and will be processed after the update has been started. Records that could not be processed when they were created usually have the status *auto*

Selecting and Displaying Updates

(update server not active, update deactivated). The updates are processed automatically when the update is reactivated.

run indicates that an update record is being processed. Run is only assigned to automatic update records and records that are processed via *Repeat*.

err identifies update records that caused an update to be terminated prematurely because an error occurred while the update request was being executed. The messages ***err(ext. commit)*** and ***err(no retry)*** can also be output here. See also Analyzing and Correcting Update Errors [Page 60]

ok means that no errors occurred while the update request was being executed (if rdisp/vb_delete_after_execution is set to 2, in other words, automatic deletion after execution of update is deactivated)

V1 means that V1 modules of the update request were executed without any errors occurring; this does not, however, refer to the entire update request (that is, there are still V2 updates and/or collective runs to be processed).

V2 means that V2 modules of the update request were executed without any errors occurring; this does not, however, refer to the entire update request (that is, there are still collective runs to be processed).

del means that the update request is being deleted.

If you do not want to use time intervals or users as selection options, you can also use Update Program Administration (Transaction sm14) [Page 50].

Displaying the Update Header

Use

The **update header** is used to manage the update records (update requests [Page 9]).

Prerequisites

You have called up the `Update Records: Main Menu` screen and have positioned the cursor on an update record.

Procedure

You can display the update header by choosing *Goto* → *Update header*.

The update header contains the following information.

Field	Description
Update key	Key under which an update request is stored as a data record in the database. The key writes appropriate messages to the system log if update processing is canceled. The express mail that informs you of the canceled update also contains this key. Compare the key in an express mail or in the system log with the value in this field to make sure that you have found the correct update record.
`Client`	Client in which the user is logged on.
`User`	User who triggered the update
`Language`	Language in which the user is logged on
`Account`	The *Account* field is currently not used
`Report`	ABAP report with which the update was created
`TCode`	Transaction code that called up the ABAP program
`ENQ key`	Key of the lock entries created by the transaction that created the update. The update task holds the locks until the V1 update has been successfully completed. The locks are canceled if an update error occurs. This is described in greater detail in the documentation on The R/3 Lock Concept [Ext.].
`Context`	Language that must be supported by the update server for this update request. Update group specified for the update. The context is only relevant if multiplexing for tables is active (Oracle Parallel Server database system). You can display the update server(s) in your system by choosing *Goto* → *Server* (`Language` column).
`Info`	Bit mask that contains further information on the update request. Default value = 1 (standard update request). 2 means that this request is from the batch input, and must **not be restarted manually**.

Displaying the Update Header

Ud.ret.cde	Update return code (for error analysis)
Status	Detailed description of status
Update server	Update server on which the update was carried out
Error text	Error message (if present) indicating that the update was terminated prematurely.

Displaying Update Modules

Prerequisites

You have called up the `Update Records: Main Menu` screen and have positioned the cursor on an update record (update request [Page 9]).

Procedure

You can display the update modules of the selected update record by choosing *Goto* → *Update modules* or by double-clicking the record.

A list of the update modules is then displayed. This contains:

- the module ID, that is the number of the module
- the module name
- the module type (`V1 NORMAL`, `V2 NORMAL`, `V1 NICHT SUB.PST.`, `ARFC` and `COLL.RUN`)
- the status of the module (*init, run, err* etc.), see also The Most Important Update Statuses [Page 17].

From this screen, you can display more detailed status information (position the cursor on the required module + `Update status` pushbutton), refresh modules (Refresh pushbutton), and display the update header (hat icon) and update data (glasses icon).

Analyzing Canceled Updates

Use

There can be many reasons why an update is canceled (`err` status in the overview). The problem often exists for a very short time only; in this case, the update records can be processed manually [Page 40]. If the problem is of a more serious nature, you have to analyze it further.

See also Analyzing and Correcting Update Errors [Page 60]

Procedure

In the update management transaction, choose *Update records* → *Debugging* to analyze a canceled update in the debugger. Further information can be found in the Debugger [Ext.] documentation.

You can also correct the errors in an update record via the debugging function. To activate and deactivate the update debugging function choose *Settings* → *Update debugging on/off*.

In both cases, the debugger stops at the start of the update module called up by the transaction.

Authorizations: in order to use the debugging function, you require the update authorization for the authorization object *System authorizations* (S_ADMI_FCD) as well as the ABAP authorization for debugging *ABAP Development Workbench* (S_DEVELOP).

Testing Canceled Updates

Use

Performing a test is recommended in the following cases:

- You suspect that a temporary problem has occurred, or you have corrected the update problem and now want to test the update to determine whether it functions correctly. If the test is completed successfully, the problem is has been corrected.
- You want to attempt to retrigger the error caused by the update problem. If the update is canceled during the test, the system outputs the error message with which the update was canceled. Click the message to call up online help.

An error message that is displayed during the test is not necessarily the original message because the error message always depends on the current system conditions.

See also Analyzing and Correcting Update Errors [Page 60]

Procedure

In the `Update records` screen, position the cursor on the record you want to test and choose *Update records* → *Test* to test the canceled update record.

Do not test any updates that are waiting to be tested or are in the process of being tested (status = `init`, `auto` or `run`) because processing could be canceled by the test. The different statuses are described in the section entitled The Most Important Update Statuses [Page 17].

Result

Test processes the selected function module and then outputs a message (in the status bar) indicating that the update was successful or indicating an error is the update was canceled.

Test does not make any changes to the database, or any changes made are reversed automatically after the update has been completed or canceled.

If the test was successful, you can run the canceled updates again (Processing Updates Manually [Page 40]).

Processing Updates Manually

Use

The function for processing update records manually is used to reprocess records, which for some reason did not successfully complete one of the phases in the update process.

This function can be used in most cases, and means that you do not have to delete the update records and enter the data again.

Prerequisites

The update records that have to be reprocessed are set to `err` on account of an update problem that occurred earlier, or `init`.

Procedure

The procedure used will depend on the status of the record at the time the problem occurred (see also The Most Important Update Statuses [Page 17]).

Update Record Set to `init`

You can instruct the Update System to process all updates with the status `init` (all records that have not yet been updated) by choosing *Update records* → *Update*. (If the update is active, the records are processed automatically; in other words, you do not need to do anything here). This is described in greater detail in the section entitled The Most Important Update Statuses [Page 17].

Update Record Set to `err`

You can instruct the Update System to reprocess all updates with the status `err` (canceled prematurely) by choosing *Update records* → *Repeat update*. To make sure that this functions correctly, you can test the canceled updates [Page 39] beforehand.

Updates for records that contain an update module of the type `V1 NICHT SUB.PST.` **cannot** be repeated.

Note that the repeat update for a V2 module cannot be carried out before the repeat updates for V1 modules. Processing a V1 module can also trigger processing of the associated V2 modules. A V2 component is no longer displayed after it has been successfully processed. (Detailed information can be found in the section entitled The Update Process [Page 12]).

Result

The update records are processed correctly and no longer appear in the overview.

Further Information

Repeating Canceled Updates [Page 61]

Resetting Update Statuses

Use

From time to time, an update that was set to `err` is not completed successfully when it is reprocessed, in other words, the update is always active (status = `auto` or `run`) and is neither fully processed nor terminated prematurely. (This problem only occurs when updates are repeated manually or when an update server is restarted if updates that have not been processed (status = *auto*) are processed automatically. Further details can be found in the section entitled The Most Important Update Statuses [Page 17].)

Two causes are currently known for this problem: a database problem of the type “dirty-read/dirty cluster” (because the update does not hold any locks for the data), or the target update server is not active when the update is repeated.

Under normal circumstances, the problem is only a temporary one (in both cases). Using transaction **`sm51`** and **`sm50`**, check whether the update is not processed. If it is not processed, reset it, and try to process it again (see also Processing Updates Manually [Page 40]).

Procedure

To reset the status of an update record, choose *Update records* → *Reset* → *Update status.*

Result

The update records are reset to the status `init` and have to be processed manually [Page 40].

Deleting Update Records

Use

Update records with the status `err` can be deleted.

Prerequisites

Do not delete the canceled updates (status = `err`)

- until you have completed the update error analysis. Remember that when update servers are started, they always search for updates that are older than 30 days and delete these. Once you have completed your analysis, you can stop the updates from being deleted automatically by increasing the value of the system profile parameter rdisp/vbdelete (see also Main System Profile Parameters for Updates [Page 25]).
- unless you are sure that the data in the update can be restored, if the update originates from a batch input session, for example, and can be repeated there, or if you have retrieved data from the canceled update so that you can enter it again.

Procedure

Choose *Update records* → *Delete* → *All records* or *Single.*

Single refers to the update record at which the cursor is currently positioned.

Never delete updates that have not yet been processed (status = `init`, `auto` or `run`) as this can result in data that is to entered in the R/3 System being lost.

 Deleting Updates in the Background

The function for deleting updates automatically after they have been successfully completed can be deactivated in order to prevent the system constantly deleting small numbers of update records (extremely high number of database accesses).

Deactivating this automatic delete function can, however, result in very long update tables in the database. To remove the updates, you have to run the report RSM13002 at least once a day (the option *DELETE* must to be set to X). Only updates that have been completed successfully are deleted with the report RSM13002.

Further information can be found in the description of the parameter *rdisp/vb_delete_after_execution* in the section entitled Main System Profile Parameters for Updates [Page 25] or in transaction rz11 in theR/3 System.

Result

The update record(s) is/are deleted, the data is lost and has to be reentered. All of the locks on database objects held for the update are also deleted (see also The R/3 Lock Concept [Ext.]).

Reorganizing Update Records

Use

You can use this function to instruct the update server to search for incomplete update records and delete these. Update records can be incomplete - if a transaction registers one or more update modules - as a result of a screen change in which the transaction that creates the update generates a database commit; the transaction, however, is terminated prematurely and a rollback is triggered.

Incomplete update records are of no significance, but take up space in the database.

If the parameter rdisp/vbreorg [Page 25] is set to 1, the update records are reorganized each time the update server is started.

Reorganization is, therefore, active (value = 1) in the standard system.

This function allows you to reorganize the update records manually.

Procedure

Choose *Update records* → *Reorganize*.

Result

Incomplete update records are deleted and do not occupy valuable space in the database.

Configuring Update Groups

Use

Automatic load distribution takes place in an R/3 System even with asynchronous updates. Load distribution (update dispatching with load balancing [Page 22]) usually takes place across all of the available update work processes.

This global dispatching is, however, not always desired.

Update work processes can, therefore, be combined by R/3 application servers to form groups within which load distribution (dispatching) takes place. **Dispatching does not take place outside these groups.**

You can activate dispatching by setting the profile parameter rdisp/vb_dispatching to 1 (see also Main System Profile Parameters for Updates [Page 25]). If this parameter is set to 0, all of the update requests are sent to the server specified in the profile parameter rdisp/vbname.

The procedure for creating the groups is described below.

As an alternative to the procedure described below, you can also configure server groups with the much more user-friendly Update Program Administration (Transaction sm14) [Page 50].

Procedure

1. To create groups, call up transaction **se16** (table maintenance) and make the following entries in the SERVERGRP and DBINSTANCE fields of the table ASGRP:

SERVERGRP	DBINSTANCE
GROUP1	<SID>
GROUP2	<SID>
…	…

<SID> represents the SAP system name. "GROUP1" and "GROUP2" are freely selectable unique names.

Let us assume that you want to create one update group for financial accounting documents and one for production planning. With the SAP system name <SID> = C11, the table ASGRP could contain the following entries:

SERVERGRP	DBINSTANCE
FI_GROUP	C11
PP_GROUP	C11
…	…

2. The following entries are made in the SERVERNAME and SERVERGRP fields of the table APSRV:

SERVERNAME	SERVERGRP
<app_server_name>	GROUP1
<app_server_name>	GROUP1
...	...
<app_server_name>	GROUP2
<app_server_name>	GROUP2
<app_server_name>	GROUP2
...	...

The entries should reflect the configuration of your system. The name of the application server is structures as follows: <host>_<SID>_<Num>.

Make sure that the server name is written correctly (call up transaction **sm51**, which displays the names of the servers).

The table APSRV could then be as follows:

SERVERNAME	SERVERGRP
app1_C11_00	FI_GROUP
app2_C11_00	PP_GROUP
app3_C11_00	PP_GROUP

Servers that are not assigned to any group in the table APSRV, are assigned implicitly to the group "DEFAULT". If this group does not contain an update server, no updates can be performed by these servers (the message "No update server active" is output when you save your data).

3. Stop and then restart all R/3 instances.

Configuring Update Groups

Modified configuration: if the application server assignments are changed, or if a new application server is defined in the system, you only have to carry out steps 2 and 3.

Diagnosis:

Start the test updates on the application servers of the individual groups: in transaction **`sm12`**, *Extras* → *Diagnosis in update*, you can check to see whether the groups were created successfully.

Detail on monitoring updates can be found in the section entitled Monitoring Updates [Page 56]; details on transaction **`sm12`** can be found in the section entitled Managing Lock Entries [Ext.].

Displaying and Resetting Update Statistics

In the update management initial screen (`Update Records: Main Menu`),choose *Goto* → *Statistics* to display the update activities on the server you are currently logged onto. The statistics contain all of the data collected as of the last reset or server restart.

To reset the statistics, choose *Update records* → *Reset* → *Statistics.*

You can now specify whether the statistics are to be reset only on the application server you are currently logged onto (*local*), or on all servers (*global*).

The statistics are divided up into the following sections:

Update requests

Update requests			
generated	1	executed (V1)	15
started (V2)	2	executed (V2)	2
canceled	0	deleted	16
Update is active			

The client is displayed on the left and the server on the right. The update request [Page 9] is sent from a dialog or batch work process (client) on the application server you are working on (see status bar) to the update work process (server), which may be running on a different application server.

The **1st line** specifies how many update requests were `generated` by the client (in other words, sent to the server), and how many were processed by the update work process (`executed (V1)`). This number is higher because you are on the application server on which the update server is running - this also processes update requests from other application servers.

If you call up the statistics on a server on which no update work process is running, `0` always appears alongside `executed (V1)`, `started (V2)` and `executed (V2)`.

As described in the section entitled The Update Process [Page 12], the update work process passes the V2 update modules onto a V2 work process – in other words, the **2nd line** contains the update work process of the clients and the V2 work process of the servers. Furthermore, both V2 updates were carried out.

In the **3rd line** you can see how many update requests were canceled by the client, and how many were deleted by the server.

The **4th line** indicates whether the update is active (see also Deactivating and Reactivating Updates [Page 59]).

Database Accesses

DB I/O	bytes written	bytes read

in total	456	119949
min	28	28
avg	152.000000	325.065041
Max	356	1172

This section logs the database accesses made as a result of the update: total number of bytes written and read, the smallest and largest packet, as well as the average size. This is, therefore, the data that is written to the update tables VBHDR, VBMOD, VBDATA and VBERROR.

Times

Times	Execution (V1)	Execution (V2)	Read	Write
Number	17	2	375	3
total (sec)	9.734802	6.046222	2.349286	0.113074
min (msec)	46.373000	3006.741000	0.125000	32.659000
avg (msec)	572.635412	3023.111000	6.264763	37.691333
max (msec)	3613.746000	3039.481000	355.894000	44.082000
kb/sec			50.306175	3.938240

This section contains information on the time required to execute the update requests. **Column 1:** number of V1 update requests executed, the total time required for these in seconds, as well as the shortest, average and longest time required in milliseconds.

Column 2: the same information as column 1 but, in this case, for V2 updates.

Column 3: number of read operations (updates tables in the database), the total time required for these in seconds, as well as the shortest, average and longest time required for a read operation in milliseconds.

Column 4: the same information as column 3 but, in this case, for write operations

Times	Update	Commit times	Delete
Number	5	42	54
total (sec)	0.183918	1.248347	0.283877
min (msec)	3.770000	0.217000	2.474000
avg (msec)	36.783600	29.722548	5.256981
max (msec)	122.488000	188.723000	66.108000

This section contains the time intervals required for the different steps (minimum, maximum, and average values). Update table updates (column 1), commits and delete operations in the update work processes.

Displaying Servers

Use

This display (initial screen: `Dispatching info`) contains the following information:

- the update groups created (see also. Configuring Update Groups [Page 44])
- the active update servers and associated work processes (`upd1` for V1 updates, `upd2` for V2 updates), group affiliation, languages supported, and update requests that have just been allocated
- the other application servers that are active in theR/3 System, the proposed work process type (D=Dialog, B=Batch(background), S=Spool, E=Enqueue) and group affiliation

Procedure

In the update management initial screen, choose *Goto* → *Server*.

You can also display the servers in Update Program Administration (Transaction sm14) [Page 50] (see also Server tab page Server [Page 53]).

Update Program Administration (Transaction sm14)

Use

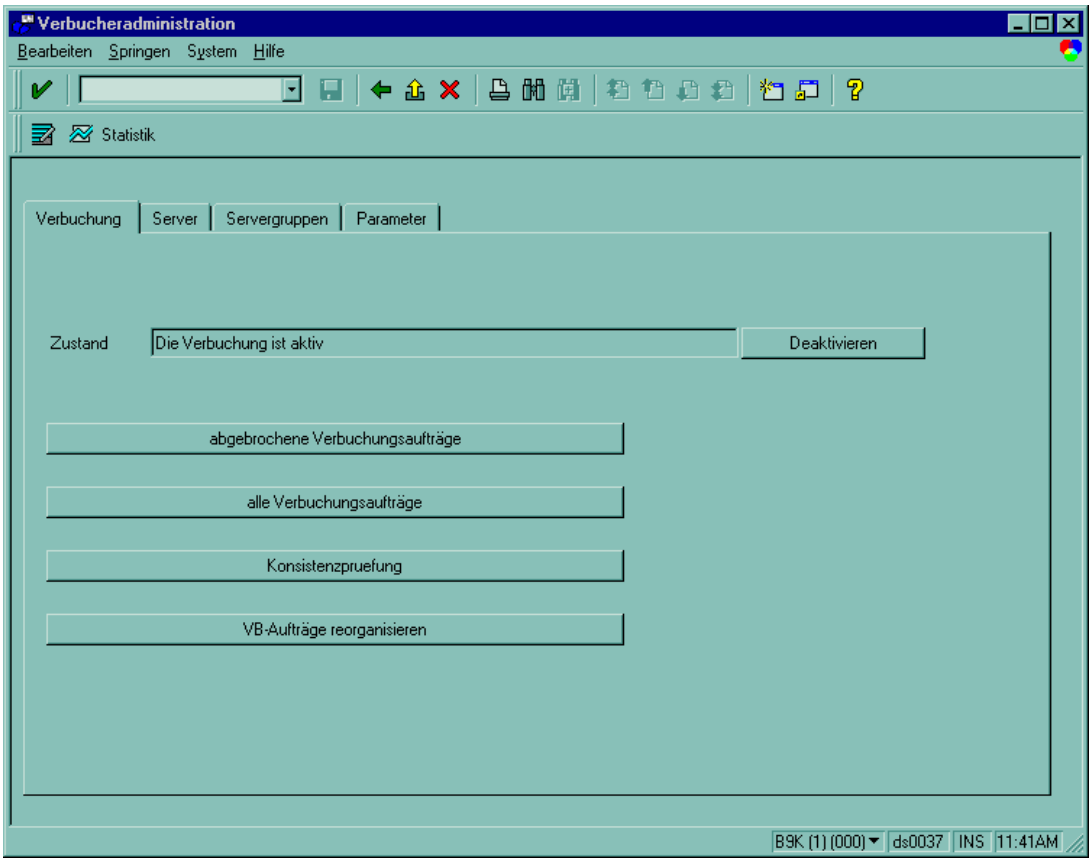
Compared with transaction **sm13**, **sm14** provides you with a different view of an update (Update Management [Page 31]) and contains additional functions for administering updates (for example, creating servers and server groups, checking parameters relevant for updates).

Integration

You can access all of the functions supported by **sm14** via the appropriate menu in the initial screen of transaction **sm13** or via other transactions. Transaction **sm14** merely groups together related functions so that you can use them more easily.

Features

The initial screen is shown below (update is active):



The initial screen contains four tab pages:

- Update tab page [Page 52] (shown here)
- Server tab page [Page 53]
- Server groups tab page [Page 54]

- Parameter tab page [Page 55]

Activities

You can call up Update Program Administration by entering transaction code **sm14** (or **/nsm14** if you want to call it up directly from a different transaction).

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C

Documentation not available for Release 4.6C