

SIMATIC IT Unilab 6.7

Archiving

Concepts and User Manual

Preface

Table of Contents

Overview

Implementation

Starting the Archiving Cycle

Archiving to File

Reference Performance data

Appendix A: Unilab tables
archived for Type 3 configuration
data

1

2

3

4

5

6

Guidelines

This manual contains notices intended to protect the products and connected equipment against damage. These notices are graded according to severity by the following texts:

Caution

Indicates that if the proper precautions are not taken, this can result into property damage.

Notice

Draws your attention to particularly important information on handling the product, the product itself or to a particular part of the documentation.

Trademarks

All names identified by ® are registered trademarks of the Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Preface

Where is this manual valid?

This manual is valid for release 6.7 of SIMATIC IT Unilab.

Basic knowledge required

This guide is intended for SIMATIC IT Unilab users who are responsible for system configuration, such as application managers and system integrators (consultants). To be able to understand the concepts and examples discussed in this guide, the reader should at least have taken the SIMATIC IT Unilab Basic Training.

Purpose

This Concepts and User Manual explains how archiving in Unilab is done.

Related documentation

The Unilab Concepts Guides (part 1, 2 and 3) contain information related to the contents of this Concepts and User Manual.

These documents are all available online from the SIMATIC IT Unilab Documentation Library.

Conventions

The table below describes the specific typographic conventions that are used throughout this manual:

Symbol/Convention	Indicates...
E.g.	Where examples are given.
Text in bold	The names of menus, commands, dialog boxes and toolbar buttons and, in general, all strings (e.g. File menu; Save command).
KEY1+KEY2	Shortcut keys, which permit rapid access to commands (e.g. CTRL+C).
UPPERCASE	The names of keyboard keys (e.g. RETURN key).
<i>Italics</i>	Noun with special importance or significance for which emphasis is needed. The names of parameters that must be replaced with a specific name or value.
>	A succession of commands in which the command preceding the symbol must be selected before the command following it.

Symbol/Convention	Indicates...
	Code example.
Code example	

SIMATIC IT Documentation Library

The SIMATIC IT Documentation Library provides you with a comprehensive and user-friendly interface to access the overall product documentation where manuals and helps online can be browsed by functionality or by component.

Readme

The installation includes a readme file, which contains information on upgrade procedures and compatibility with previous releases. This file is supplied both in standard text (**Readme.wri**) and in Acrobat PDF (**Readme.pdf**) format.

This file is available in folder \ReleaseNotes of the setup DVD and is available from the SIMATIC IT Unilab Documentation Library.

SIMATIC IT Training Center

Siemens IA AS MES offers a number of training courses to familiarize you with the SIMATIC IT product suite. To successfully achieve this goal, training consists of lessons in both theory and practice.

Courses are held year-round, according to a program that is published well in advance of the first scheduled session.

The material on the basis of which our courses are conducted reflects the result of years of experience in process, LIMS, quality control and production management.

All courses are held by expert personnel that are aware of the developments and innovations in the Siemens IA AS MES product suite.

Courses are held in English at the Siemens IA AS MES Training Centers.

Upon request, training courses can also be organized on the customer's premises.

For more information on the training course calendar, please visit our technical web site (<http://www.siemens.com/simatic-it/training>).

SIMATIC IT Service & Support

A comprehensive Software Maintenance program is available with SIMATIC IT products. Software Maintenance includes the following services:

- **Software Update Service (SUS):** automatic distribution of upgrades and service packs
- **Technical Support Service (TSS):** support on technical problems with SIMATIC IT software (standard support and other optional services)
- **Online Support:** a technical web site providing information such as Frequently Asked Questions and technical documentation on SIMATIC IT products

Software Update Service (SUS)

This service provides automatic shipment of new versions and service packs when released. When a new version / service pack is available for shipping, it is typically shipped within one month.

One copy of the installation CDs is shipped for each Server covered by Software Maintenance.

Hot fixes (officially tested and released) are not shipped and must be downloaded from the Technical Support Service web site.

Technical Support Service (TSS)

Siemens provides a dedicated technical support team for SIMATIC IT products.

The following options are available:

Bronze support: 9 hours/day, 5 days/week

Silver support: 24 hours/day, 5 days/week

Gold support: 24 hours/day, 7 days/week

The principal language of the SIMATIC IT hotline is English.

SIMATIC IT partners and customers covered by the Software Maintenance program are entitled to direct access to the TSS.

Access to TSS

To be able to access TSS, the customer needs to register as a user on the Technical Support web site. Connect to <http://www.siemens.com/mes-simaticit/> and follow the **Technical Support Service** link.

The registration form must be completed with:

- Personal data
- The required company and plant information
- The Contract Number provided by Siemens Back Office when the contract is agreed.

Online Support

A customer who is a registered TSS user, can access the Technical Support web site (<http://www.siemens.com/mes-simaticit/tss>), which contains technical information such as:

- Service conditions (Phone numbers, Working hours, Reaction times,...)
- SIMATIC IT knowledge base: a technical support database that includes practical service solutions from Technical Support or the SIMATIC IT community
- SIMATIC IT software (e.g. hot fixes, software examples) and release notes that can be downloaded

- SIMATIC IT cross-industry libraries that can be downloaded (limited access to SIMATIC IT certified partners)
- SIMATIC IT product documentation that can be downloaded
- Frequently Asked Questions and useful tips.

Table of Contents

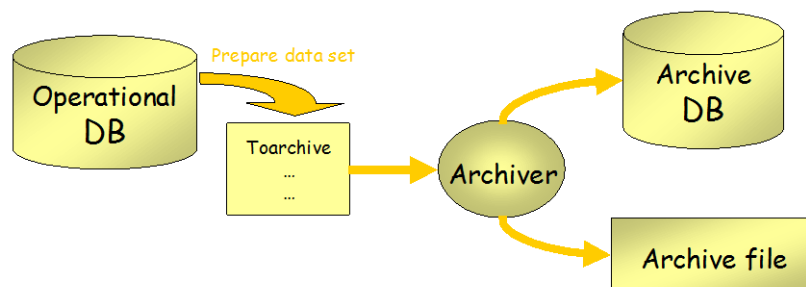
1 Overview.....	1-1
1.1 Assumptions and Archive Schedule	1-2
1.2 Archive and Restore Possibilities	1-3
2 Implementation	2-1
2.1 Specific Custom Code for the Scenario.....	2-1
2.2 Creating the Databases	2-1
2.3 Configuring Archiving.....	2-2
3 Starting the Archiving Cycle	3-1
3.1 Archiving Operational Data	3-1
3.1.1 Archive Plan	3-1
3.1.2 Data Lifecycle + Data Location after Archiving	3-3
3.1.3 Starting the Archiver	3-5
3.2 Archiving Configuration Data	3-6
4 Archiving to File	4-1
4.1 Requirements.....	4-1
4.2 Archive to File	4-2
4.2.1 Fill In ToBeArchived List	4-2
4.2.2 Archive the ToBeArchived List.....	4-3
4.3 Restore From File	4-4
4.4 Archiving Objects Recursively	4-4
4.5 Archive and Restore operations involving custom tables	4-6
4.5.1 Filling the uttoarchive table	4-7
5 Reference Performance data.....	5-1
5.1 Test System Configuration	5-1
5.2 Timing Results	5-2
6 Appendix A: Unilab tables archived for Type 3 configuration data	6-1

1 Overview

Maintaining an operational database involves handling conflicting interests: on the one hand it is best to keep the amount of data in an operational system to a manageable size for best performance, so regular archiving AND deleting of data is necessary after some time. On the other hand, it might be necessary to keep historical data on-line for statistical analysis and reporting. There are also legal issues with the keeping of the data: some data must be kept for a number of years, and must be ready for retrieval on demand. The management of this archiving problem is the primary goal of the SIMATIC IT Unilab Archiving Procedure.

The archiving procedures can also be used as a backup- or roll-out-mechanism. These are secondary uses of the procedure, which are not considered to be 'real' archiving and will therefore not be considered in the scenario.

The picture below illustrates the archiving mechanism in Unilab



The scenario requires a specific life cycle to “prepare” the set of objects that need to be archived. It will show the possibilities and uses of 'To File' and 'To DB' archiving. The scenario will also include the use of archiving pre- and post-processing (with customized **PrepareToArchive** and **AfterArchive** functions). Note that the scenario is based on the configuration EASYQC data, as a result, it can also be used for demo-purposes.

Although the criteria to archive specific data will be different in real-life situations, we anticipate a real-world use that will be in accordance to the given scenario. The more you use the procedure following this scenario, the more certain that the existing functionality will be convenient for your wishes. If you attempt to use archiving in situations that differ significantly from this scenario, you will probably run into the limitations of the procedure sooner or later.

Apart from operational archiving on a very regular base, we will also archive the configuration data every now and then. Note that the most care must be taken when restoring old configuration data because the current configuration is overwritten when restoring the old data. Although this might seem very crude at first, there is really no alternative: there can only be one valid definition of a sample type or a request type at any given moment.

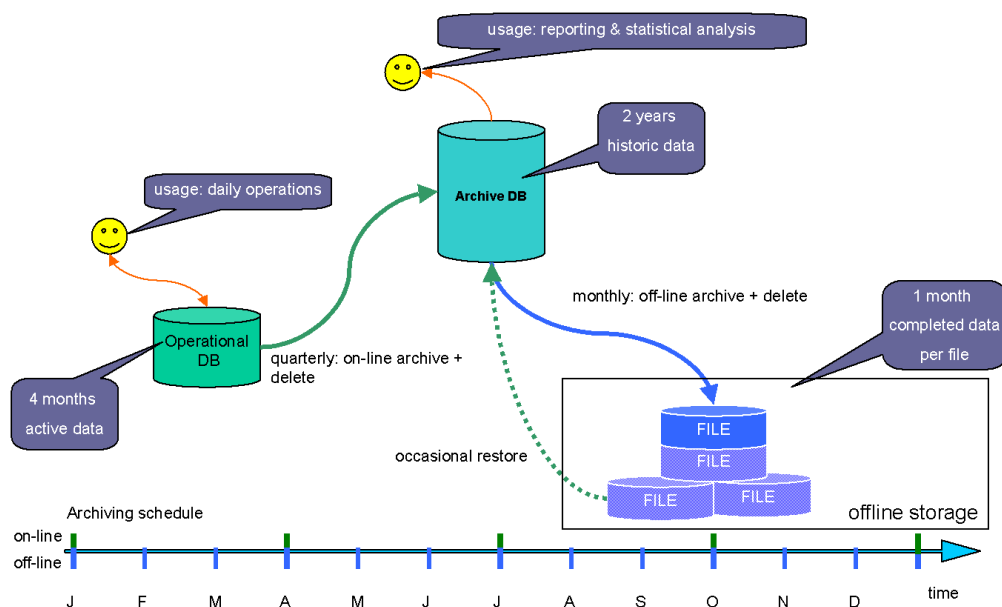
1.1 Assumptions and Archive Schedule

We assume to be running a production lab, with frequent sampling based on a fixed schedule. We assume that the number of finished samples remains more or less the same in the course of the year (this will not be the case in a season-dependent sector). These assumptions allow us to make a tight archiving schedule. We will make a quarterly archive of all finished samples that have a status **ToBeArchived (OnLine)**. This status will be set 3 months after a samples **Completion Date**. The operational database will thus contain all currently open work + 3 months of finished samples.

A number of samples are used for shelf life testing. We will keep these samples separate from the others. We also assume that no sample other than the shelf-life samples remains open longer than 3 months.

We will create an Archive database that will contain all historic data over a period of 2 years. After 2 years, we will archive this data to file and delete the data from the historic database. This archive will be based on the sample completion date.

The illustration below explains the overview scenario for archiving in Unliab.



Note that this archiving strategy is strongly based on the **completion date** of samples. The login-date could also be used, but this is only possible if the samples are being completed within a relatively small timeframe (e.g. within the first week after entry in the lab). Other approaches could be to archive across products, batches or projects instead of purely time-based. This is interesting in sectors with short product lifetimes or limit-time projects.

The basic idea is to have an easy way to know in which file the archived data will land so that we can easily trace it back in case we have to restore specific data.

1.2 Archive and Restore Possibilities

To be able to archive to or restore from a database, the archive and the on-line databases must have the same Unilab version.

The policy for restoring files is quite clear: the user must be able to restore all data that has ever been archived in a file, no matter which object type or from which Unilab version the data is.

Illustration: Configuration data on a 21 CFR part 11 database

The illustration below shows how configuration data is restored on a 21 CFR part 11 database.

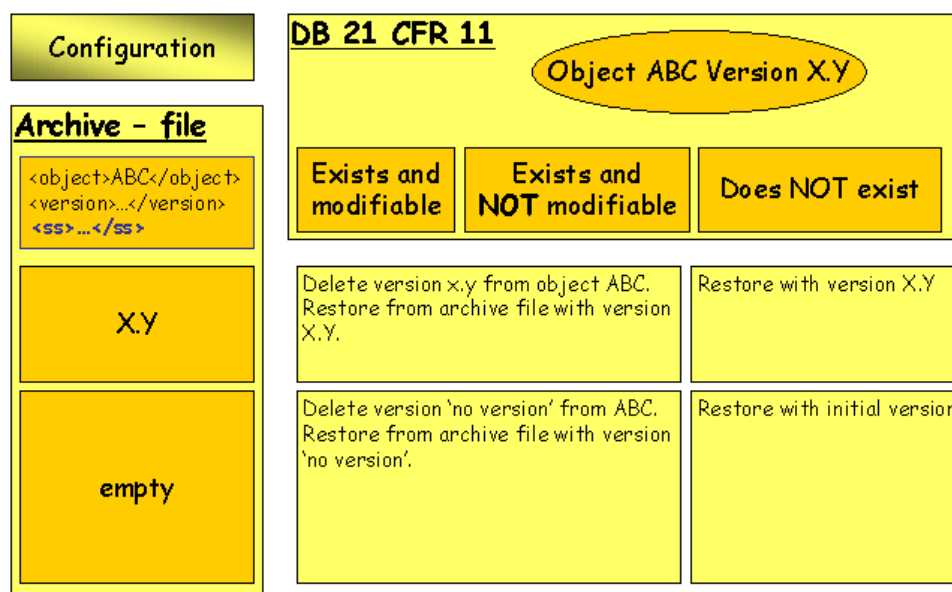
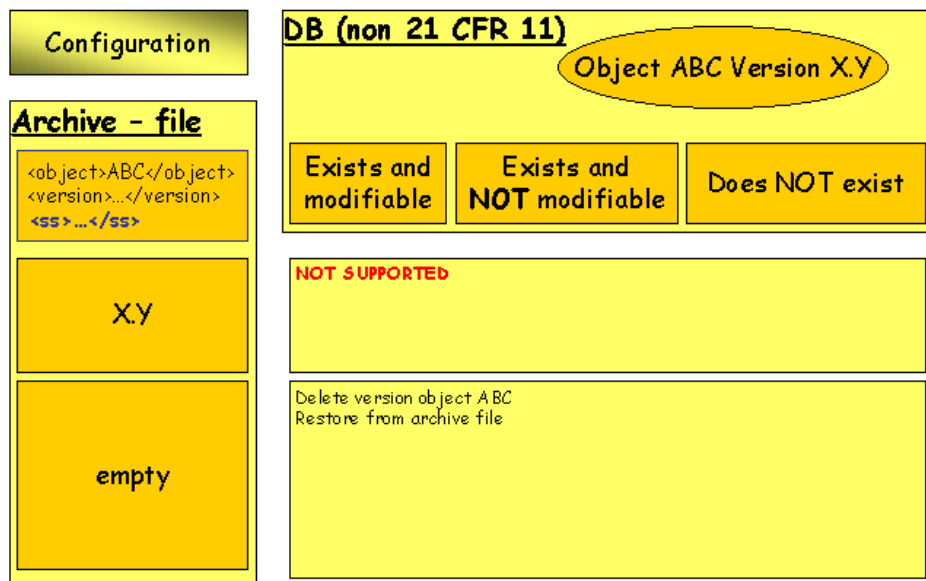
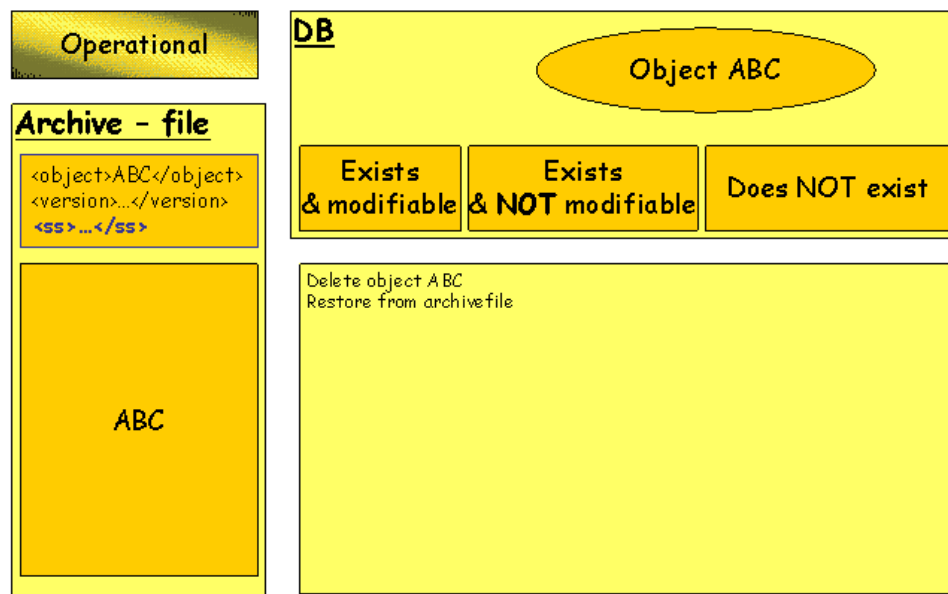


Illustration: Configuration data on a non-21 CFR part 11 database

The illustration below shows how configuration data is restored on a non-21 CFR part 11 database.


**Illustration: Operational data on a database**

The illustration below shows how operational data is restored on a database.



2 Implementation

2.1 Specific Custom Code for the Scenario

A number of specific custom code functions have been written. Some of the logic (such as archive naming and planning-rules) can be customized. The places where this has been done are marked with the  sign in this document.

2.2 Creating the Databases

Preinstallation and Configuration

The creation of the operational database will be done as explained in the SIMATIC IT Unilab Server Installation Guide. When Unilab 4.2.1 or higher is installed, a pre-installation of the archiving procedures is done automatically.

This pre-installation is not enough to actually use archiving with an archiving database (the pre-installation is sufficient for off-line archiving only). When archiving to an archive database is going to be used, the archive database must be installed, prior to configuring the operational database for archiving purposes.

We will use an operational database with ORACLE_SID=OPER

Note

It is strongly recommended to keep the DBA-name of both archiving and operational database the same. We will use 'UNILAB' for both databases. (This is specifically important for the system configuration. Overwriting **utsystem** with data from a database with a different DBA-name will cause serious problems: you will no longer be able to login as DBA on that database).

Planning and Creating the Archive Database

The archive database can be created using the standard Unilab installation scripts: the structure of the archive DB is exactly the same as the operational DB structure! The required customizations can be made as explained below within the frame of the Unilab Server Installation procedure.

However, the archive database will contain more data than the operational database. We know that the database will be 'read-only' most of the time. Only after 2 years a deletion of records will occur, updates will only occur in the status-fields (fixed length field, so this has no consequences for the record-size).

Considering such, it is advisable to create the archive database with other storage parameters than the operational database. The tables can be larger, but the **PCTFREE/PCTUSED** parameters can be chosen so that the database blocks are filled to their maximum (set **PCTFREE** to a very low value - 5 would be a good choice. The higher you set **PCTUSED (<95)**, the more efficient the storage will be reused after deletion, but this might affect performance during Delete and Restore operations). If you change the **PCTFREE/PCTUSED** parameters, you must keep an eye on the row-chaining statistics of your tables. If excessive row chaining occurs, you will have to recreate your table for optimal performance.

It is also possible to create different indices on this database that are tuned for specific long-running reporting queries.

Note

The database will only be used after 4 months of operational data have been collected. Most projects need some time to come up to speed, so the first year of operational use is not always a good reference to estimate the required data-volume.

It is strongly recommended to put the archiving database on another system, both for data-availability and performance reasons. Note that the archive database must be the same version as the on-line database, and the databases must be operated with the same NLS settings (such as date-format, character set and decimal separator). This must be taken into account when upgrading the operational database. The two computers need not be the same: it is valid to put the operational database on a Unix system and the archive on an NT or vice versa.

The archiving database will also be used for archiving to file. Make sure the Unilab archiving directory is created, if you want to use 'radef.exe', the **unilink extproc** must be properly installed and running and the **UTL_FILE** entries in the **initarch.ora** file are correctly set. If you take care when filling in the crdbnew.xls spreadsheet, the necessary files will automatically be put in place. You just need to configure an extra listener for the external procedures. Refer to the SIMATIC IT Unilab Server Installation Guide for a detailed explanation of this.

We will use an archiving database with **ORACLE_SID=ARCH**.

Note

It is strongly recommended to keep the DBA-name of both archiving and operational database the same. We will use 'UNILAB' for both databases. (This is especially important for the system configuration. Overwriting **utssystem** with data from a database with a different DBA-name will cause serious problems).

2.3 Configuring Archiving

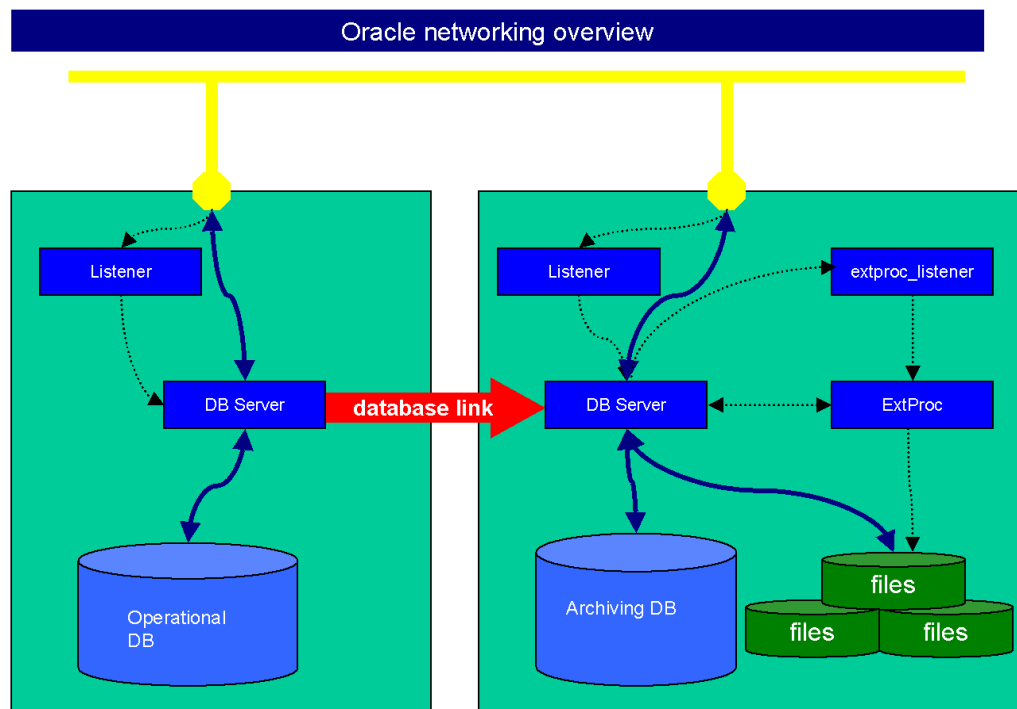
Sql*Net Middleware

First of all, it is necessary to have proper SQL*Net connectivity between the two databases. In the Unilab installation procedure, you will find a paragraph describing the required set-up.

The logical connection between the database servers is called a 'database link'. A database link uses the Oracle networking software to make a transparent connection to another database.

The archiving database will be considered 'remote' and the operational database will be considered 'local' as far as the database link is concerned.

The illustration below shows the setup for networking.



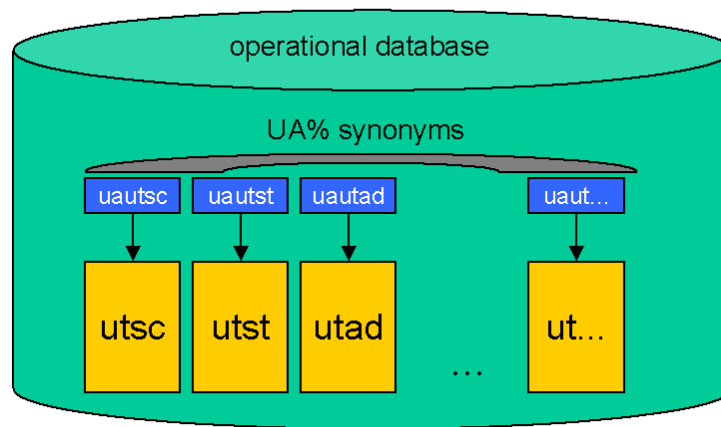
The previous picture shows the dataflow between the operational and archive databases (thick blue lines). The dotted lines show which 'overhead' must be installed to be able to organize this dataflow: (i) a listener on the operational side to establish the connections with operational Unilab Clients, (ii) a listener on the server side to establish the server-to-server connections for archiving, (iii) **Extproc**, incl. its listener, to establish the file-connections.

The database link used for archiving is called **UNIARCH**. Transactions that are performed between a local database and tables in a remote database are called 'distributed transactions': they require specific handling of commit and rollback for guaranteed consistency (so-called 2-phase commit). As of Oracle 8, this capability is included in the standard Oracle installation; no special options must be installed.

Operational Views for Archiving

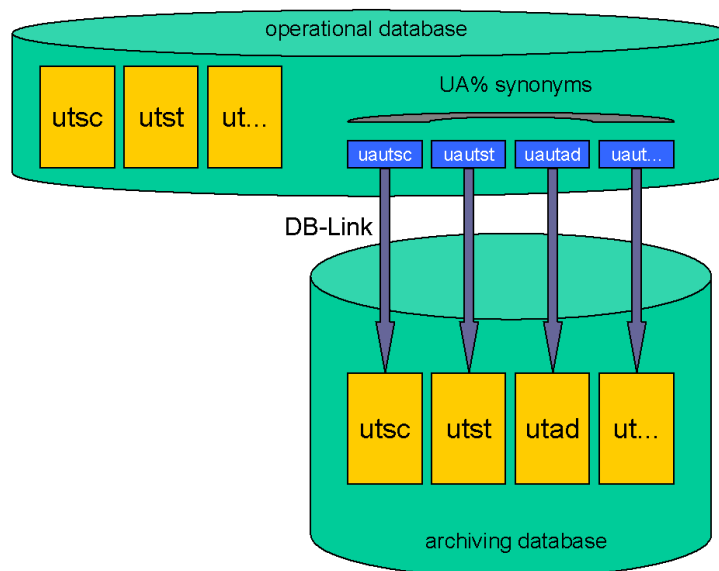
A number of synonyms (names starting with **UA**) have been created on the operational database during standard database creation. These synonyms point to the tables in the local database. This set-up can only be used for archiving 'to file'.

The illustration below shows the synonyms in the operational database.



If you want to archive to another (archiving) database, these synonyms must be redefined in the operational database so that they point to tables in the archiving database through the database link.

The illustration below shows the synonyms in the archiving database.



As explained below (see also: the SIMATIC IT Unilab Server installation guide), this redefinition can be done with the sql script **crsynoop.sql**. This script must be run on the operational database.

The redefinition can only occur after the creation of both databases, and the creation of the **UNIARCH** database link.

If you try to archive to a remote database on a database that has not been properly prepared, you will get an error message indicating the self-reference of the archive synonyms.

Steps to Activate the Remote Archiving

1. Statement to Create the Database Link

In **Server Manager**, connect as Unilab DBA (to the operational DB) and issue the following statements (*replace italics with correct values*):

```
CREATE DATABASE LINK UNIARCH  
CONNECT TO unilab_dba  
IDENTIFIED BY dba_password  
USING 'sql_net_connect_string'
```

After this, it should be possible to select from the remote database:

```
SELECT * FROM global_name@UNIARCH
```

This should return the name of the remote database (ARCH in our example).

2. Create Operational Archiving Synonyms

In Server Manager, connect as Unilab DBA (to the operational DB) and issue the following statements:

```
set serveroutput on  
@crsnoop.sql
```

Creating Project-specific Tables

Groupkey tables and non-standard Unilab tables are not automatically created when an attempt is made to restore data from such a table. If you plan to copy the existing groupkey data, you will need to create the group-key structures before archiving. You can create group keys in the archive database using the Define Group Key client application. You will need to connect the application to the archiving database then.

Note

It is not required to copy over your groupkey data, since the retrieval of historic data might very well be based on groupkey sets other than the operational group keys. Also note that, although it is possible to run the Unilab applications on top of the archive database, this is not the intention of this database. The database should primarily be used as a historical reporting data repository.

The same remarks hold for all add-on tables (these typically have a name starting with **AO**). The creation of add-on tables must be done in the same way it was done on your operational database. Your SI or VAR will probably supply you with the necessary information on how to do this.

If an attempt to restore a non-existing project-specific table is made, this data will be skipped during restore. Standard Unilab data will be restored even if the restore of project-specific tables fails.

This has to be taken into account when a new project is started, especially if new tables are created dynamically during the use of the Unilab software (as can happen with group keys). At present, no special checks are made by the archiver to make sure the necessary tables are created. It is however possible to customize **PrepareToArchive** and **AfterArchive** procedures, which are executed before and after the archiving takes place (for Restore, the **PrepareToRestore** and **AfterRestore** have a similar function).

Extra Steps to Execute the Scenario

The scenario is based on the EASYQC configuration data. It is recommended to create at least the EASYQC configuration data (life-cycle, statuses, sample-types, etc...). Make sure a number of samples are created with lifecycle = **SC Archive**, (i.e. with lc= **LA**).

3 Starting the Archiving Cycle

3.1 Archiving Operational Data

3.1.1 Archive Plan

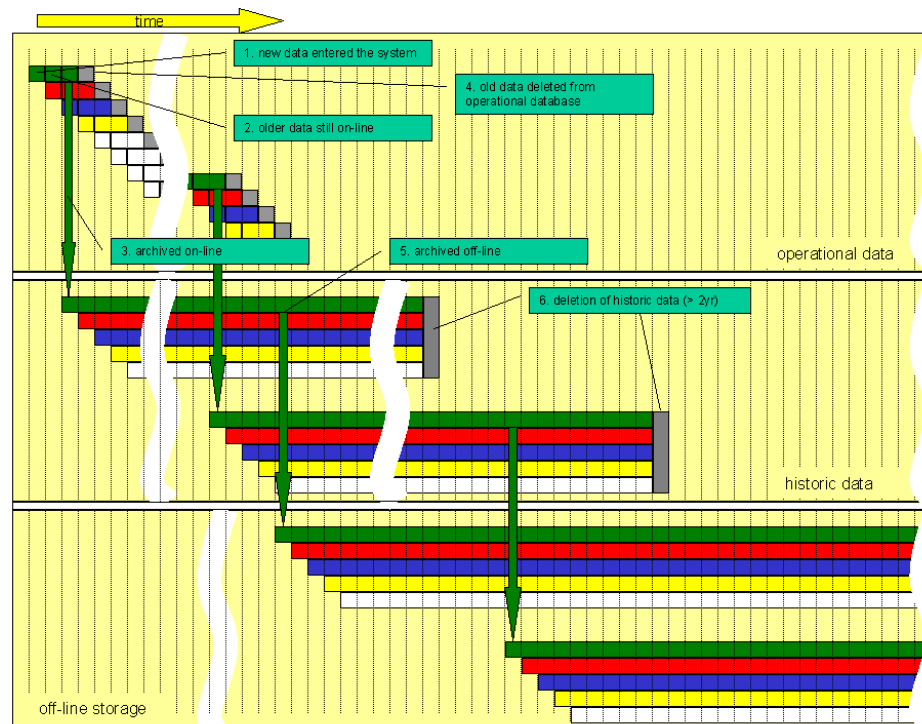
The scenario assumes that the on-line archiving (copy from operational DB to archive DB), is executed once every month. The archive can be triggered manually through the client-application **radef**, or alternatively, it is possible to automatically schedule the archiver through a database job. The use of these two mechanisms is explained further on.

The **PrepareToArchive** procedure searches for all samples/requests/worksheets that have been completed since the last archive. Only completion dates longer than 3 months ago are taken into consideration. The status of these samples is changed to **ToBeArchived**.

When archiving is started, all samples/requests/worksheets that have status **ToBeArchived** will be copied to the archive database. Subsequently, the **AfterArchive** procedure will change the status to **Archived (On-line)**.

A second run will delete all data with status **Archived (On-Line)** that is older than 4 months.

The illustration below shows an overview of the archiving schedule:



Each colored square indicates a month worth of data. The rectangles indicate the life cycle and location of this data. Arrows indicate the archival of data (=copy of data):

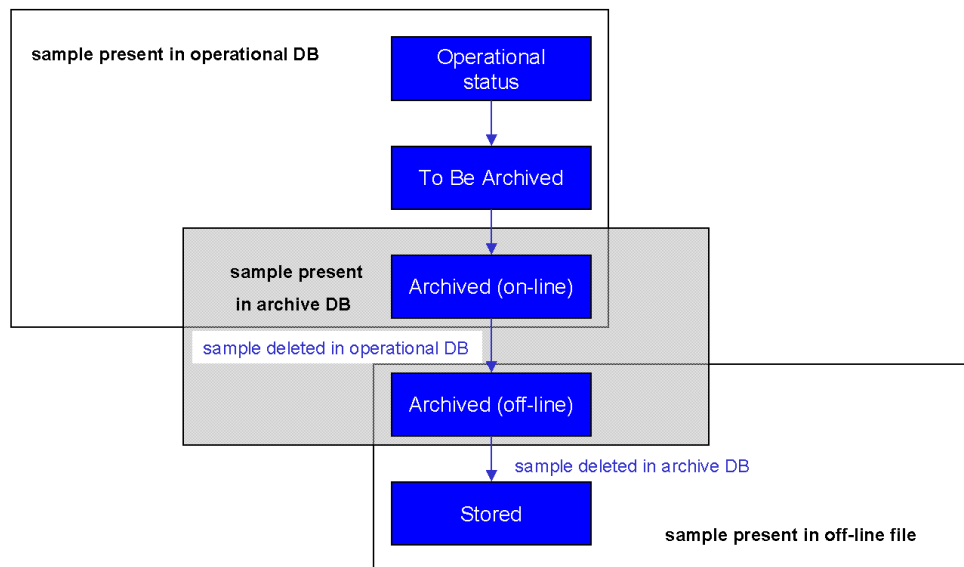
1. New data enters the system (Created, In Execution, Measured, Complete, ...).
2. After some time, the work on the data is finished (**Validated**, **Cancelled**,...). The status of the data can become **ToBeArchived** after 3 months.
3. Every month, the data that is older than 3 months (and got status **ToBeArchived**) will be copied over to the on-line archive. The status of the samples will be set to **Archived On-Line**. Note that these data will exist in both the operational and the archive DB simultaneously, for a one-month's period.
4. Every month, the data that is **Archived On-Line** for a month, will be deleted from the operational system. From now on, the data only exist in the archiving database.
5. Every month, we copy the one-year-old data to a file.
6. Every year, we delete data that is older than 2 years from the "archive" DB. From now on, this data can only be found in the file. This file can be copied back to the archiving database when needed.

The picture shows clearly that data in the different systems have different lifetimes. In our scenario, the data will be available in an on-line format for a minimum of two years to a maximum of three years. It is clear that these periods can be freely chosen and adapted to any specific situation. In the ideal world, one should be able to find a situation in which it would almost never occur that data have to be copied from the archive database back to the on-line database. Note also that, because the status of the data is used to differentiate between data that must be archived and data that are archived, all data will be archived exactly once.

Some sites will also want to 'compact' their archive data: this is done by deleting data that has no significance for statistical analysis (e.g. history logging). The difference between significant data and superfluous data is very site-dependent. Some sites only need to retain the end-results of an analysis, others will want to keep as much detail as possible. Compacting data is not implemented in this scenario. This is something that is typically done in the **AfterArchive** procedure!

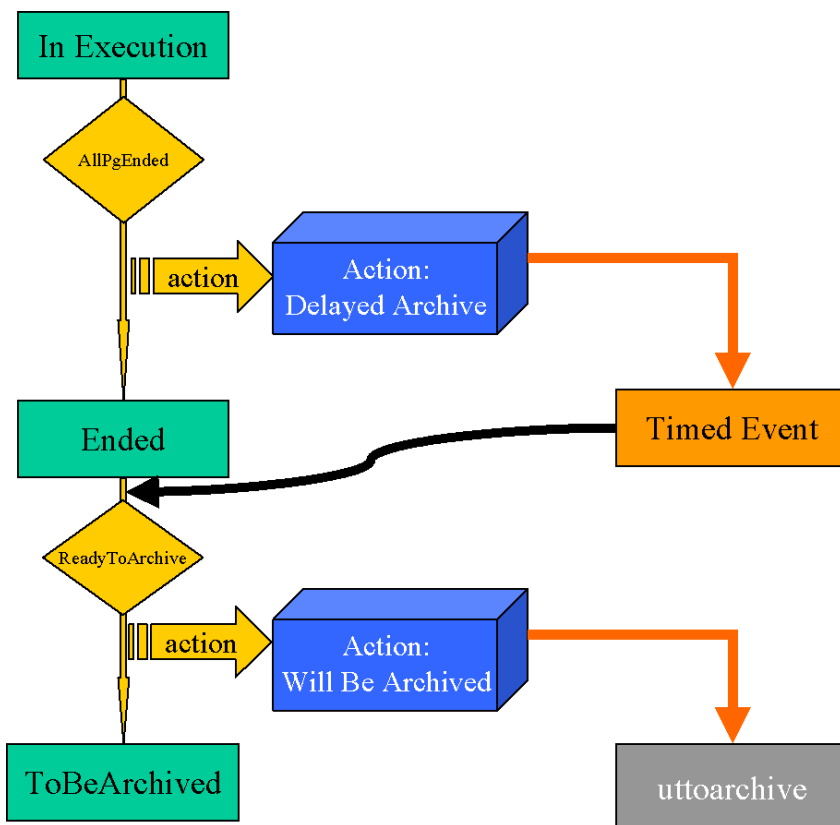
3.1.2 Data Lifecycle + Data Location after Archiving

The illustration below shows the data lifecycle and data location after archiving.



To make this scenario work in a fairly automatic way, the SC lifecycle (**LA**) was extended.

The illustration below shows the sample life cycle for archiving.



As soon as a sample is finished (all parameter results are filled in), the status of the sample is set to **Ended**. At the same time, the action **DelayedArchive** will create a 'timed event' that will move the sample-status to **ToBeArchived**. This will only happen after some time (3 months in our example). So after 3 months, the sample-status is changed, and the action **WillBeArchived** inserts the sample in the **uttoarchive** table.

Note (📌)

The '3 month' period is hard-coded in the **UNACTION.DelayedArchive** action. If you want another period, you can change **UNACTION** to reflect your own rules.

In the operational applications, it is now clearly marked that the sample will be archived on the next archiver run. All samples that make the transition to **ToBeArchived** in the same month will get the same 'archive-id' (**u4arMMYYYY**), where MM=month and YYYY is the current year. The name of the archive is dynamically created in the **UNACTION.WillBeArchived** action.

Since archived data has an overlapping time between its residence in the archive and its residence in the operational database, we will indicate this in the status of the data.

Apart from changing the status from the sample from **ToBeArchived** to **Archived - Online**, we also want the sample to be scheduled for deletion after a month (we want to keep the archived sample on-line for one month). This can best be done with the use of the archiver-custom function **AfterArchive**.

Archived (on-line) is a state in the operational database that indicates that the same sample is duplicated in the archive. After some time, we will delete these duplicates and only the copy in the archive database will remain. The same approach will be used when archiving to file. This allows us to separate the deletion process from the copy process. This improves the reliability of the archiving procedure since we only delete data from which we have already made a successful copy.

Note (📌)

This scenario needs the additional states **To Be Archived TA**, **Archived On-line AN**, **Archived Off-line AF**. The handling of the state-transitions is partly done in the life cycle (see previous page) and in the archiver custom-functions **PrepareToArchive** and **AfterArchive**. These are two stored procedures that can be customized for the specific behavior you want during archiving. The default of these procedures is to do nothing at all.

PrepareToArchive

The **PrepareToArchive** procedure is automatically called before the actual archiving is started. It is typically used to select the data you want to archive or delete. It can also be used for specific state transitions. In our scenario, the samples are automatically inserted in the archive during their normal state-transitions, so we do not need any extra code in **PrepareToArchive**.

The **AfterArchive** procedure is automatically triggered after a successful archiving cycle. **AfterArchive** is typically used to set the status of data in the database.

We will use the **AfterArchive** in this scenario to reschedule the archiver to delete the same samples we just archived. This is possible because after the archiving cycle, the **uttoarchive** table still contains the records that were used as a reference for the archiver. They are marked with **handled_ok=1**. Normally, these records will be deleted after the **AfterArchive** procedure. Now we will update **uttoarchive** as follows (🔗):

```
UPDATE uttoarchive
SET copy_flag = 0,
    delete_flag = 1,
    archive_id = 'DELETE',
    archive_to = 'FILE',
    archive_on = ADD_MONTHS (SYSDATE, 1)
    handled_ok = 0
WHERE handled_ok=1
```

This statement does the following: reset the records for those samples that were just archived (**handled_ok=1**) so that they will be deleted (**delete_flag=1**) and not copied (**copy_flag=0**). This deletion can only occur after one month (**ADD_MONTHS(SYSDATE,1)**). Be aware that, with **copy_flag=0** and **delete_flag=1**, samples will be deleted without being copied. To avoid confusion, we set the **archive_id = 'DELETE'** (as a reminder that this is a delete-only archive). In the scenario, it is assumed that the copy was made first! The 'archive_to = FILE' is set to avoid confusion with existing DB-archives.

PrepareToRestore

Likewise, two procedures, **PrepareToRestore** and **AfterRestore** exist to allow for specific functionality during the Restore cycle.

Typical use of **PrepareToRestore** is to create specific custom objects (e.g. tables or sequences) in the target database. Note that the default behavior of the restore procedures is to NOT create objects (e.g. tables, users, views, sequences,...) in the database. The reason for this behavior has to do with the intrinsic complexity of this task: when and how to create these objects is often project-dependent.

Note

The source-code for the implementation of the scenario can be found in the standard **UNARCHIVE.SQL** file. You will need to remove the comments ('-- arscen --' in front of each line) from the code, and recompile the file.

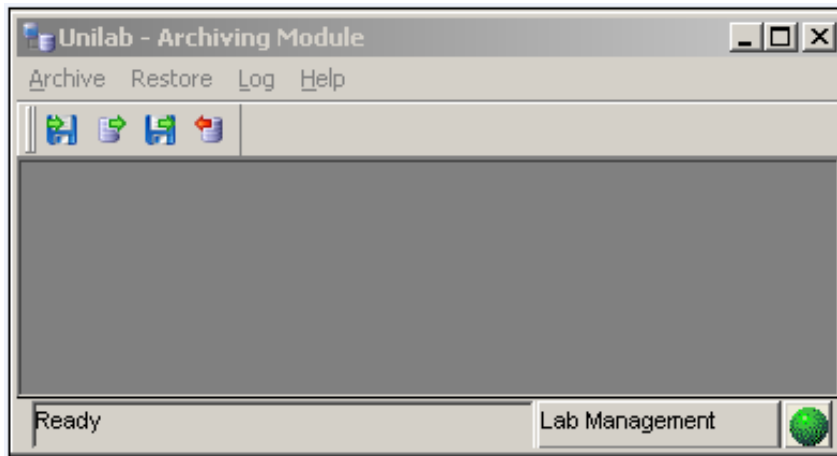
The transitions in the life cycle can be made manually or automatically. The choice between these two options is directly related to the specific working procedure of a lab: some labs can define a simple rule that can be carried out by the system automatically; sometimes the selection of old data is a manual task. Both strategies can be implemented using the Unilab lifecycle mechanism. In this example, we use the automatic state transition (**In Execution -> Ended**: action **DelayedArchive**) to trigger the archiving code.

3.1.3 Starting the Archiver

After some time, the **uttoarchive** table will be filled with samples that need to be archived. The archiver itself is actually nothing more than a PL/SQL routine (**UNAPIRA.Archive**). As soon as this PL/SQL procedure is executed, the archiving will start. There are basically 2 ways to start this procedure.

Interactive Archive

An interactive program (**radef**) can be used to trigger the archive procedure: you start this program and you choose the **Archive To Database** menu.



You can then select the current archives that can be made (e.g **archive-id= ARCH082000**). Subsequently, all samples belonging to this archive-id, with an archive-on date before 'now' (**SYSDATE**), will be copied to the archive database. If there are many samples, this can take quite a while (refer to the chapter about performance in this document to have an idea of the expected duration of this action).

The **AfterArchive** procedure will then reinsert the samples in the **uttoarchive**-table, but now they will be marked for deletion. This can only be done one month after the archive has been made (archive-on date has been set). Also, the archive-id is changed to **DELETE**, to clearly indicate the meaning of these entries.

Scheduled Archive (job)

It is also possible to start the archiver automatically using Oracle job-queues or other scheduling mechanisms (such as Unix cron or Windows Task Scheduler). In the case of the scenario, we could schedule the archiver to run once a month for the actual archiving, whereas a second run would see to deleting previously-archived samples. These actions can then be carried out during off-peak periods (e.g. during the night or in the weekends).

Example of a job that schedules the archiver every month for the DELETE archive (as explained above.) Note that the 'FILE' parameter is in fact not really used (but must be present), deletion has no real target (🗑️):

```
CREATE OR REPLACE PROCEDURE ArchiveJob IS
l ret code INTEGER;
BEGIN
    l ret code := UNAPIRA.Archive ('DELETE', 'FILE',
    'DELETE');
END;
```

3.2 Archiving Configuration Data

For configuration data, we cannot use the same concept as for operational data, because there is no equivalent for the 'completion date' (except if you use a **ToBeArchived** flag for obsolete configuration data).

The most important difference between configuration data and operational data is that configuration data always reflect the “current” situation, whereas samples and requests are created at a specific moment on the time line.

We could archive (without deletion) as soon as a specific configuration becomes active (or obsolete). In real life, it will often happen that configuration data changes on a daily basis (e.g. new users are added to the system when required, new sample-types or request-types are defined, new parameters, and infocards are defined, etc,...).

Apart from the initial system set-up and occasional lab reorganization, there are no real milestones in the configuration data where it is obvious to make a consistent archive of the complete configuration set.

It seems to make sense to archive your configuration data along with your operational data (since this operational data is based on the configuration). However, it is next to impossible to create a fully-consistent configuration archive that is always up-to date with regard to the samples you archive. The first sample of a specific sample-type might well be based on a different version of the sample-type definition than the definition that is now active in your system.

Note

Since operational data contains a copy of the most important configuration data on which it is based, the consistency between the configuration archive and the operational archive need not be a concern. E.g. a sample can exist as an object on its own, without the need to reference the original configuration data on which it is based.

Alternatively, you might think about having your configuration archived automatically after each and every change, but you can only have one copy of your data at once (so two changes on the same day, would destroy the first - and probably most valuable - archive).

So the only feasible archive seems to be an archive made on a regular basis (for example, quarterly or yearly), where we only keep the most recent changes.

The configuration archive can also be used as a means to transport a configuration between different databases. There are a number of restrictions that must be taken into account when this is what you want to do. Both databases must be the same Oracle version, must have the same NLS-settings, and some care must be taken with respect to the mutual compatibility of the Unilab versions (these restrictions are more or less the same as those imposed between an operational and an archive-database).

There are 3 types of configuration data (which must be clearly distinguished from one another).

Type 1

All data related to the configuration of sample-types and request-types. This data can be archived on a per-object basis. Note that, unlike operational archiving, the archiving of a 'parental' object will not automatically archive all underlying structures (e.g. **utst** will include **utstpp**, but not **utpp** and so on).

object type
worksheet type
protocol
request type
sample type

parameter profile
parameter definition
method definition
info profile
info field definition

Type 2

Is a specific category of data that can also be archived on a per-object basis.

object type
attribute
equipment
lifecycle
layout
groupkey definition
journal
chart type

Type 3

Is all other configuration data (includes user-related data such as **utus**, **utupus**, **utdd**, **utad**, **utsystem**, ...).

See Appendix A: Unilab tables archived for Type 3 configuration data for the complete list of Unilab tables that are archived for Type 3 configuration data.

4 Archiving to File

This chapter describes how you can export existing Unilab objects to a flat file and how you can import this file into another database using the standard **Archiving Module**.

This is an easy way to extract data from the Unilab operational database. When you send this file to TSS, they can reproduce your case with your operational data.

The functionality is available in the standard Unilab **Archiving Module**. Using specific settings, it is possible to archive to file without deletion.

Important

You can only use this procedure to restore data from a database into a test or support database. If you want to restore archived data into an operational database, you have to use the **Unilab Archiving Module**.

4.1 Requirements

Archive Destination

During the database creation (**crdbnew.xls**), you can specify an archiving folder, where the archive files will be located.

The following settings are required:

- A database parameter, which specifies the folders where Oracle has access to the files.

```
SELECT value
FROM v$parameter
WHERE name = 'utl_file_dir'
```

- The system setting that specifies the location of the **ArchiveFiles**.

```
SELECT setting_value
FROM utssystem
WHERE setting_name = 'ARCHIVE_DIR'
```

Make sure **utl_file_dir** can access the specified setting.

Important

Remember to consult topic **Checklist for file manipulation in Unilab** in the **Unilink Online help** for details on which operations are required prior to archiving to file.

Remark: Update **setting_value** in **utssystem**, if the archive directory is different in the executed queries. Make sure you have the required JAVA permissions.

Archive Module

The archive module is a standard module in Unilab.

Client Archiving Application

The **Archiving Module**, which can be used to restore and archive as flat file, is a standard Unilab application.

Remark

Select this additional component during the installation of the client.

SqlPlus

A **SqlPlus** session where you can execute and compile PL/SQL code.

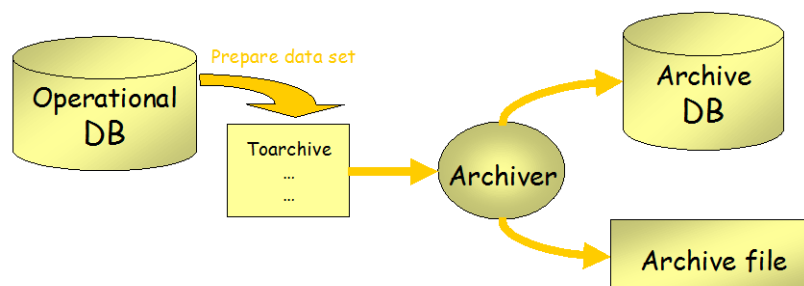
4.2 Archive to File

You need to execute the following steps to archive to a file:

- Fill in the **ToBeArchived** list, which defines the object to be archived.
(API **Unapira.AddToToBeArchivedlist**)
- Archive the **ToBeArchived** list to a file.
(API **Unapira.Archive**)

Remark

The example available in the DB API help combines both APIs.



4.2.1 Fill In ToBeArchived List

First, you have to make a list of the objects you want to archive. The API you can use is the **Unapira.AddToToBeArchivedlist** API. See the DB API online help for more details.

This API will fill the **UTTOARCHIVE** table, which is used by the Archive API function.

```
FUNCTION AddToToBeArchivedList
(a_object_tp      IN VARCHAR2,
 a_object_id      IN VARCHAR2,
 a_object_version IN VARCHAR2,
 a_object_details IN VARCHAR2,
 a_copy_flag      IN CHAR,
 a_delete_flag    IN CHAR,
 a_archive_id     IN VARCHAR2,
 a_archive_to     IN VARCHAR2)
RETURN NUMBER;
```

When you want to make a copy to a file, you need to use the following settings:

```
a_copy_flag := '1'
a_delete_flag := '0'
a_archive_to := 'FILE'
```

The argument 'a_archive_id' is a unique identifier, specifying the name of the archive you want to make. A single archive (identified by a unique archive_id) can contain several different objects.

Example:

```
l_ret_code := Unapira.AddToToBeArchivedList
(a_object_tp => 'st',
 a_object_id => 'ChickenMeat',
 a_object_version => '0001.00',
 a_object_details => '',
 a_copy_flag => '1',
 a_delete_flag => '0',
 a_archive_id => 'ArchiveStChicken',
 a_archive_to => 'FILE');
```

Remark that every SQL session requires a **UNAPIGEN.SetConnection**.

4.2.2 Archive the ToBeArchived List

There are two ways to start **Archiving**.

Archive using API Unapira.Archive

See also the DB API Help for more details

Example:

```
l_ret_code := Unapira.Archive
(a_archive_id => 'ArchiveStChicken',
 a_archive_to => 'FILE',
 a_archfile => 'StChickenFile');
```

Archive using Client Archiving Module (radef.exe)

From the **Archive > To File** menu, you can select the archive ID you have specified:

Both procedures will create a file named **ArchiveStChicken** in the folder specified as archive directory (see **UTSYSTEM**).

The file is a text file and looks like this:

```
[Information]
archive_id      ArchiveStChicken
archive_date    03/08/2004 14:58:56
unilab_version Version 5.0 Service Pack 3
archiver_version 1.0
userid         UNILAB
DBMS_version    9.2.0.4.0
LANGUAGE        AMERICAN_AMERICA.WE8ISO8859P1
DECIMAL_CHARACTER .
date_format     DD/MM/YYYY HH24:MI:SS

[Index]
[object_tp      object_id object_version object_details
  copy_flag delete_flag  archive_id archive_to
  archive_on]
st  ChickenMeat    0001.00          1          0      ArchiveStChicken
FILE
```

4.3 Restore From File

The generated file can now be reused on another database.

Copy the file to the **ARCHIVE_DIR** of the new database.

Restore using Client Archiving Module

Start the **Archiving Module** and connect to the database where you want to restore the sample. Select the **Restore > From File** menu.

The application will display all files available in the specified folder. Select the file you want to restore.

It is important that your Client application uses the same Oracle decimal separator as the separator used during archiving. The header of the **ArchivedFile** shows the decimal separator used. If necessary, you must synchronize your Oracle **NLS_NUMERIC_CHARACTERS** setting. Otherwise, this will result in numeric conversion errors during restore.

4.4 Archiving Objects Recursively

The standard archiving functionality does not archive recursively for configuration objects. When archiving a sample type, the used parameter profiles are not in the archive.

We have provided a custom solution for this problem in the procedure **ArchiveObjectToFile**. This procedure is available on our TSS Website and in the **Server Setup\contrib\tools** folder of the product CD, and can be compiled in your database.

Create the archive

The procedure has the following arguments:

```
ArchiveObjectToFile
(a_object_tp      VARCHAR2,
 a_object_id      VARCHAR2,
 a_version        VARCHAR2,
 a_object_details VARCHAR2);
```

Where

- **a_object_tp**
 - for configuration objects: 'rt', 'st', 'wt', 'pp', 'pr', 'mt', 'ip', 'ie', 'eq', 'au', 'cy', 'pt', 'ly', 'gk', 'jo' and 'lc'.
 - for operational objects: 'rq', 'sc', 'ws', 'sd', and 'ch'.
- **a_object_id**: the object you want to archive
- **a_version** [optional]: the object version you want to archive.
 - If the value for configuration objects is left empty, all versions will be archived.
 - Operational objects do not have a version, so the argument is not used for them.
- **a_object_details** [optional]: contains the lab for an equipment, and the pp_keys for a parameter profile.

If the value for an equipment is left empty, all labs of that equipment will be archived. The same for parameter profiles.

 - E.g. for an equipment: 'lab=Paris'
 - E.g. for a parameter profile: 'pp_key1=Sangria#pp_key2= #pp_key3= #pp_key4= #pp_key5= '

Example

```
Connect to the database as dba using sqlnet /nolog
SQL>BEGIN ArchiveObjectToFile('st', 'Sangria', NULL, NULL);
END;
```

This will generate an archive of sample type **Sangria**.

The archive ID will be **ARCH<object_tp>_YYMMDD-HH24MISS**.

The archive file will be generated automatically, so you no longer need to run the Archive procedure.

The file can now be restored using the procedure described above.

Some examples:

PL/SQL call	Comment
ArchiveObjectToFile('sc', '20040810-012', NULL, NULL);	Makes an archive of the entire sample.
ArchiveObjectToFile('ws', ' WS-01230', NULL, NULL);	Makes an archive of the worksheet, including the samples assigned.
ArchiveObjectToFile('eq', 'GC-	Makes an archive of all versions of the

120', NULL, 'lab=Paris');	equipment 'GC-120' in lab 'Paris'.
ArchiveObjectToFile('st', 'ChickenMeat', '0001.00', NULL);	Makes an archive of sample type 'ChickenMeat' having version '0001.00'. Also the parameter profile, the parameter definition and the method definition are archived.
ArchiveObjectToFile('pr', 'Antibiotics', '0100.00', NULL);	Makes an archive of parameter definition 'Antibiotics' having version '0100.00', including the method definitions assigned.
ArchiveObjectToFile('lysccreate', 'Basic Layout', NULL, NULL);	Makes an archive of the sample creation layout 'Basic Layout'.

4.5 Archive and Restore operations involving custom tables

When performing Archive/Restore operations with the Unilab Archiving module, custom tables are also taken into account, provided that they satisfy the following conditions:

- The name of the table to be archived must start with **at** (as foreseen when projects require the creation of custom tables in the Unilab database);
- The table in question must contain a column with a name that corresponds to the abbreviation used in Unilab for the object to be archived. See below:

object	abbreviation
sample	sc
request	rq
worksheet	ws
study	sd
chart	ch
journal	jo
all other objects	Not applicable. For details, see *4 in the table provided in Filling the uttoarchive table.

It is also possible to archive other add-on tables (which must necessarily begin with **at**) by using the object type 'at' in table **uttoarchive**. For more information, see below.

4.5.1 Filling the uttoarchive table

Table **uttoarchive** can be filled by using api, scripts, triggers, database links.

A list of possible valid values for each object type is provided below:

object_tp	object_id	version	object_details	Notes
sc	<<sc>>			
rq	<<rq>>			
ws	<<ws>>			
sd	<<sd>>			
ch	<<ch>>			
jo	<<jo>>			
st	<<st>>	<<st version>>		
pp	<<pp>>	<<pp version>>	pp_key1=...#pp_key2=...#pp_key3=...#pp_key4=...#pp_key5=...	
wt	<<wt>>	<<wt version>>		
ip	<<ip>>	<<ip version>>		
ie	<<ie>>	<<ie version>>		
pr	<<pr>>	<<pr version>>		
mt	<<mt>>	<<mt version>>		
eq	<<eq>>	<<eq version>>	lab=...	
rt	<<rt>>	<<rt version>>		
pt	<<pt>>	<<pt version>>		
cy	<<cy>>	<<cy version>>		
au	<<au>>	<<au version>>		
lc	<<lc>>	<<lc version>>		
ly<<ly_tp>>	<<ly>>	<<ly version>>		*1
gk<<gk_tp>>	<<gk>>	<<gk version>>		*2
co	<<co>>			*3
at	<<at>>			*4

Notes:

- *1: The key of a layout is made up of 2 columns in utly: **ly_tp** and **ly**. The layout type must be appended to the string 'ly' in the **object_tp** column.
- *2: The different group key type definitions are stored in different tables, all of which start with utgk. The groupkey type must be appended to the string 'gk' in the **object_tp** column (e.g: gksc, gkst, gkrt, gkrq,...).
- *3: This corresponds to Type 3 archiving. No id is necessary in this case, as all table contents are archived/restored.
- *4: This specific case makes it possible to archive all add-on tables that are not related to one of the standard Unilab objects (as listed in the table at the beginning of this paragraph).

5 Reference Performance data

5.1 Test System Configuration

Operational Database:

The test system configuration of the operational database should be as follows:

- HP 9000/800
- Model L1000-36
- HP PA RISC 64 Bit Processor
- 1GB RAM
- 4x10GB Fast/Wide SCSI Hard-disks.
- HP UX 11.00 (64 bit)
- Oracle 8.0.6.0.0 (32 bit)
- Unilab 4.2.1 Beta6 - OOBE config data + 999 samples created (no cell details).

Archive Database:

The test system configuration of the archive database should be as follows:

- HP Vectra VL600-PIII 533
- Intel Pentium III 533 Mhz Single Processor
- 256 MB RAM
- 1x 6GB UltraATA Quantum Hard Disk [**]
- Windows NT 4 Workstation SP6
- Oracle 8.0.5.2.2
- Unilab 4.2.1 Beta6 - OOBE config data + 999 samples created (no cell details).

Network

Average throughput: 1MByte/s (10 MBps Ethernet - TCP/IP).

Notes on the Used Systems

It can be that some of the used systems have become obsolete, and even not supported anymore. Nevertheless these tests still give a realistic impression of the system performance.

The systems were not used for other activities during the measurements. Archiving is not CPU-intensive: the systems showed between 90% and 99% idle time during the tests (except for 'restore' operations), so performance is strongly I/O-bound.

It is NOT recommended to have only one disk in your server. Furthermore, SCSI (especially the newer Fast/Wide SCSI and UltraSCSI) tends to have a better performance than ATA (a SCSI controller contains specific electronics that releases the CPU from disk-operations, whereas ATA requires extra CPU-power for disk operations). The newer UltraAta/66 is comparable to SCSI for performance, and comes at a considerable lesser cost.

5.2 Timing Results

Notes on the Performance Data

These results were measured on a fairly small database. It is not the intention to provide guaranteed minimum times, in fact, real live data is most probably more complex than the data in these databases, and will thus take more time. We rather choose to give performance data for a well-known set of data that can easily be replayed on another system. If you try this scenario on your machine, you can compare if your system performance is comparable to what we expect (For best comparison, you best use our reference archive file as the data source).

During testing, we found that performance can be severely hampered when database structures such as indices become imbalanced due to repeated complete 'delete/insert' operations. If you perform the delete + restore test a number of times on the same database without rebuilding your indices, you will see that after some time, the performance starts to drop (even on an 'empty' database). This supports the theory that it is good practice to periodically maintain your databases (that is: rebuild indices and even rebuild tables).

Windows NT

The table below lists the timing results for Windows NT.

Operation	Speed in sc/hour	Archive Log in MB/minute
Copy 999 samples (no delete) from Archive database to file. The resulting file size was approximately 17 MB.	4000	N/A
Delete 999 samples (no copy) from Archive database.	3600	10
Restore 999 (non-existent) samples from file to Archive database.	1400	1
Restore 999 (existent) samples from file to Archive database.	1400	2.5
Copy and delete 999 samples from Archive database to file.	1000	1

Unix

The table below lists the timing results for Unix.

Operation	Speed in sc/hour	Archive Log in MB/minute
Copy 999 samples (no delete) from Operational database to file. The resulting file size was approximately 13 MB.	6600	N/A
Copy and delete 999 samples from Operational database to file.	3000	1
Restore 999 (non-existent) samples from file to Operational database.	1500	1

Unix <-> Windows NT

The table below lists the timing results for Unix/Windows NT.

Operation	Speed in sc/hour	Archive Log in MB/minute
Copy 999 samples (no delete) from Operational to Archive database.	1500	800 k/min on archive DB
Copy and delete 999 samples from Operational to Archive database.	900	5
Restore 999 (non-existing) samples from Archive to Operational database.	1500	1
Restore 999 (existing) samples from Archive to Operational database.	1500	2

6 Appendix A: Unilab tables archived for Type 3 configuration data

The following Unilab tables are archived for Type 3 configuration data:

table	description
utad	address
utadau	address attribute
utadhs	address history
utadhsdetails	address history details
utdd	data domain
utup	user profile
utupau	user profile attribute
utupfa	user profile functional access rights
utuphs	user profile history
utuptk	user profile task
utuppref	user profile preferences
utuphsdetails	user profile history details
utuptkdetails	user profile task details
utupus	user profile users
utupusel	user profile user experience level
utupusfa	user profile user functional access rights
utupustk	user profile user task
utupuspref	user profile user preference
utupustkdetails	user profile user task details
utupusoutlookpages	user profile user outlook pages
utupusoutlooktasks	user profile user outlook tasks
utfa	functional access rights
utel	experience level
uttk	task
uttkhs	task history
uttksql	task sql statement
uttkpref	task preference
uttkhsdetails	task history details
utss	status
utsswl	Worklist group key assignement rules
utuc	unique code mask
utucau	unique code mask attribute
utuchs	unique code mask history
utuchsdetails	unique code mask history details
utucaudittrail	unique code mask audit trail for unused values
utucobjectcounter	object counter used in unique code mask
utcomment	standard comment

Timing Results

table	description
utcounter	counter
utapplic	application title
utcd	communication driver
utobjects	object defaults
utotdetails	object details for layouts and tasks
utpref	preferences
utpreflist	list of possible values for preferences
utprintcmds	Standard reports (print commands)
utqualification	qualification
utsystem	system setting
utsystemcost	system setting for cost calculation
utsystemlist	list of possible values for system settings
uttitlefmt	windows titles
utuicomponent	user interface components for tasks and layouts
utweeknr	calendar table1/2
utcatalogueid	cost calculation catalog
utcataloguedetails	cost calculation catalog details
utcurrency	cost calculation currency
utdba	dba settings
utdecode	translation for crypic settings in reports
utedtbl	list of tables for "edit table"
utedtblhs	"edit table" history
utedtblhsdetails	"edit table" history details
utevrules	event rules
utgksupportedev	for future extension
utkeypp	Parameter profile additional keys
utlab	Laboratory
utlabel	Print lable
utlu	Lookup
utunit	units of measure
utvformat	Variable format
utyearnr	calendar table2/2
utfi	file
utfihs	file history
utfihsdetails	file history details
utlo	location
utloau	location attribute
utlocs	location condition set
utcs	condition set
utcsau	condition set attributes
utscn	condition setconditions
utxslt	xslt transformation
utcstyle	chart type style
utsapcode	sap/qm interface table
utsapunit	sap/qm interface table

table	description
utulpeers	database link tables for xml interface
utsapplant	sap/qm interface table
utsapmethod	sap/qm interface table
utcstylelist	chart type style list of possible values
utsaplocation	sap/qm interface table
utsapcodegroup	sap/qm interface table
utsapoperation	sap/qm interface table
utshortcut	keyboard shortcuts (when not related to attributes)