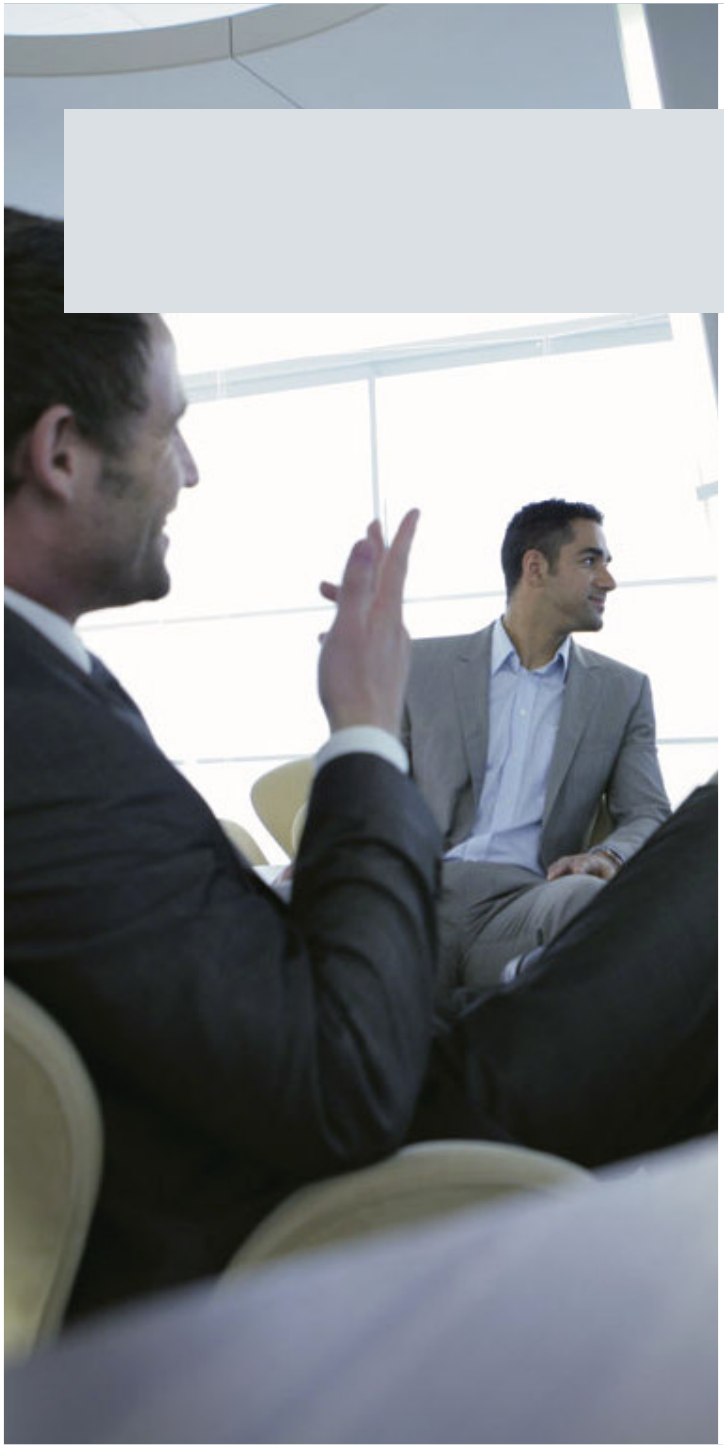
SIEMENS

Welcome



SIEMENS

Unilab Developer Course

Training Agenda

Topics (1)

Unilab database training

- Conventions used
- Structure of the Unilab database
- Detailed discussion of the Unilab tables
- Querying the database

Unilab Architecture

- 2 tier / 3 tier architecture
- Concept of event manager
- Concept of distributed event manager

Topics (2)

Introduction to the DB APIs

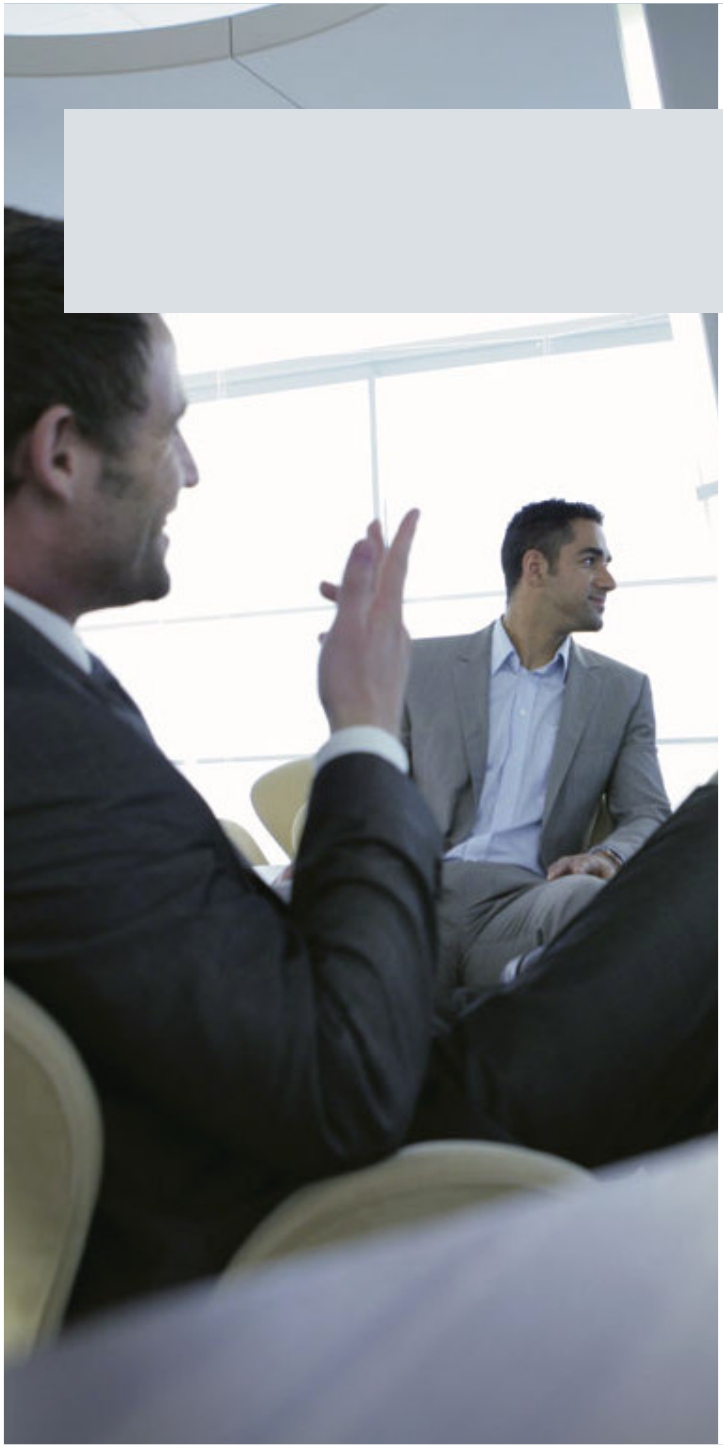
- Definition and use of APIs
- Structure of the APIs
- Functional reference

Customizing Unilab

- Unilab custom functions
- Introduction to DB custom functions
- Introduction to client custom functions
- How to use custom functions

Equipment

- Uniconnect
- Unilink



SIEMENS

Database

Overview

Database

Naming Conventions

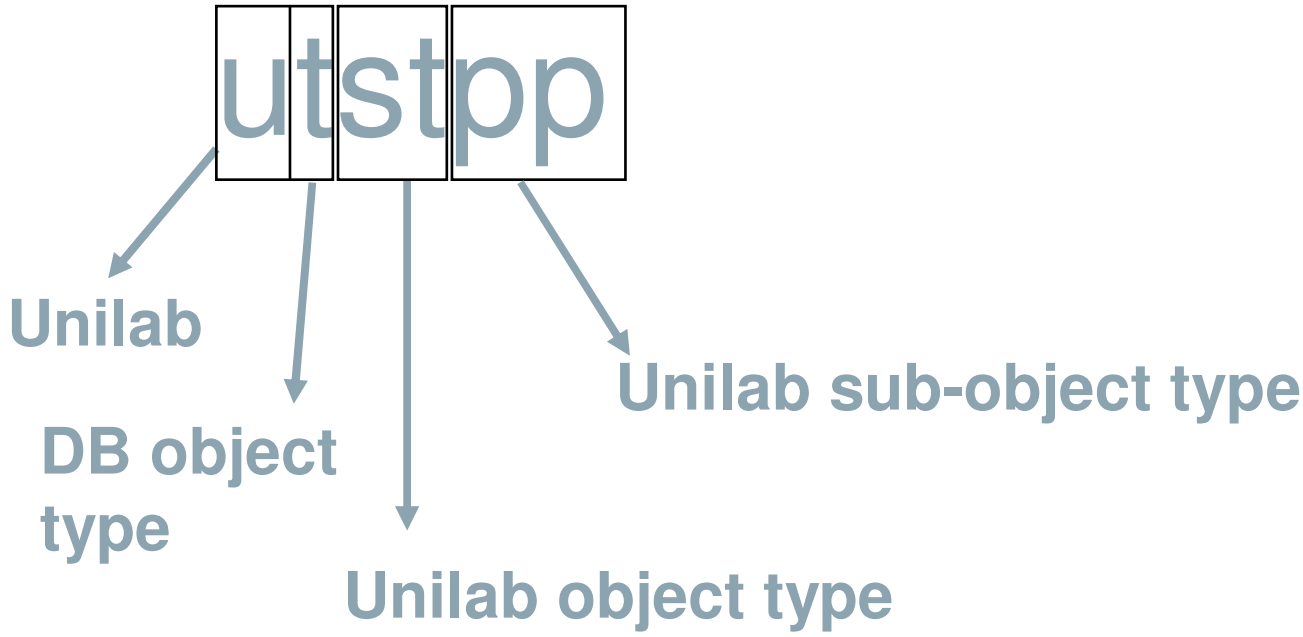
Structure of the Database

Detailed description of the Tables

Querying the Database

DB Conventions

Object naming convention



DB Conventions (2)

DB object types

- i index
- k primary key
- r role
- t table
- v view
- d data-domain related views

Packages

- unapi + object type
- un + custom function type

DB Conventions (3)

List of abbreviations

- Unilab objects (e.g. st, pp, up)
- Object properties (e.g. hs, ar)
- Config. versus operat. (e.g. pr - pa)

Table naming examples

- Main object table : utst
- History : utsths
- Relation between objects: utstpp
- Attributes on relation: utstppau

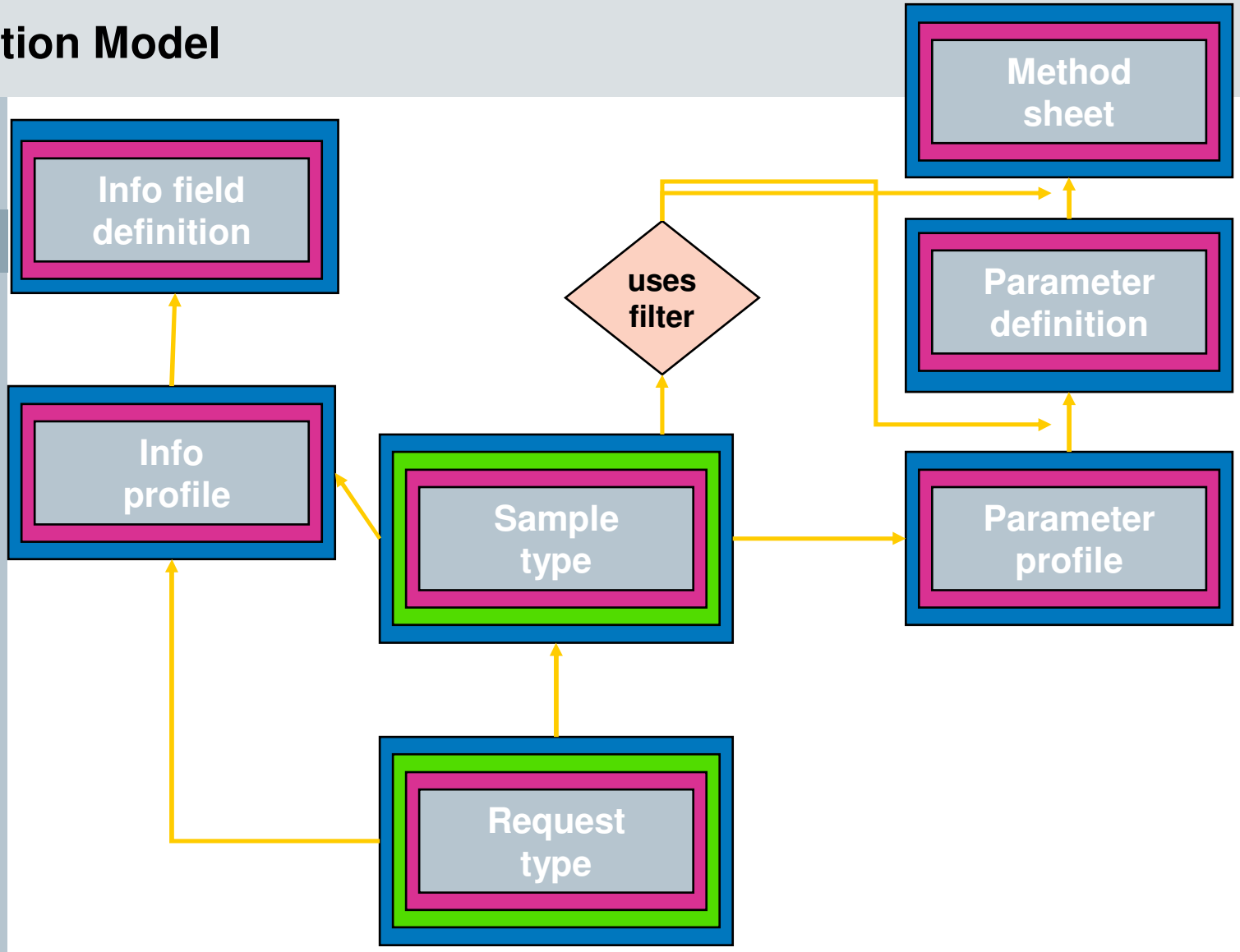
Database

DB Structure

DB Model

- Object structure => DB structure
- Configuration versus Operational
  - Functional differences
  - Technical differences
- Fixed table structure

Configuration Model



Object structure

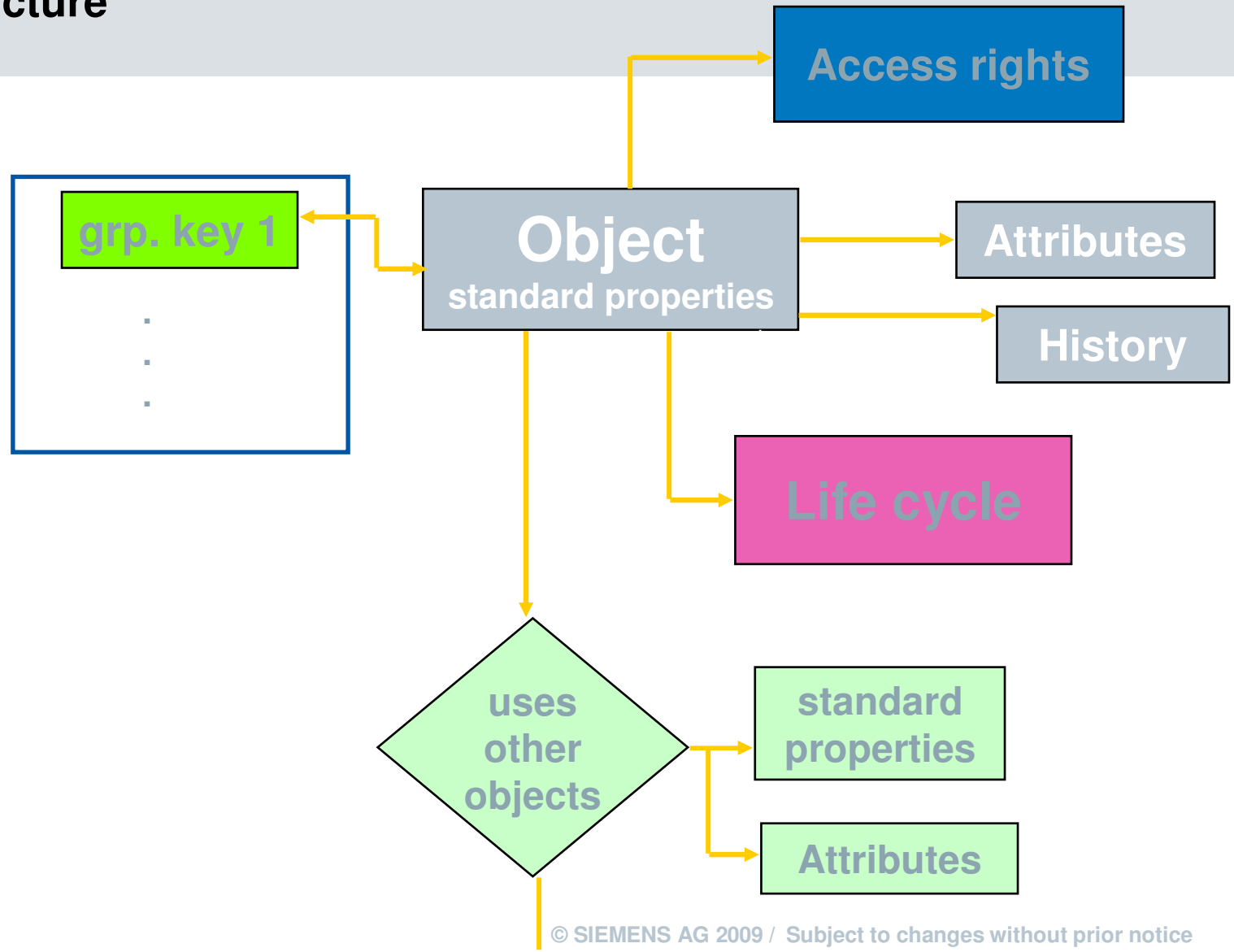
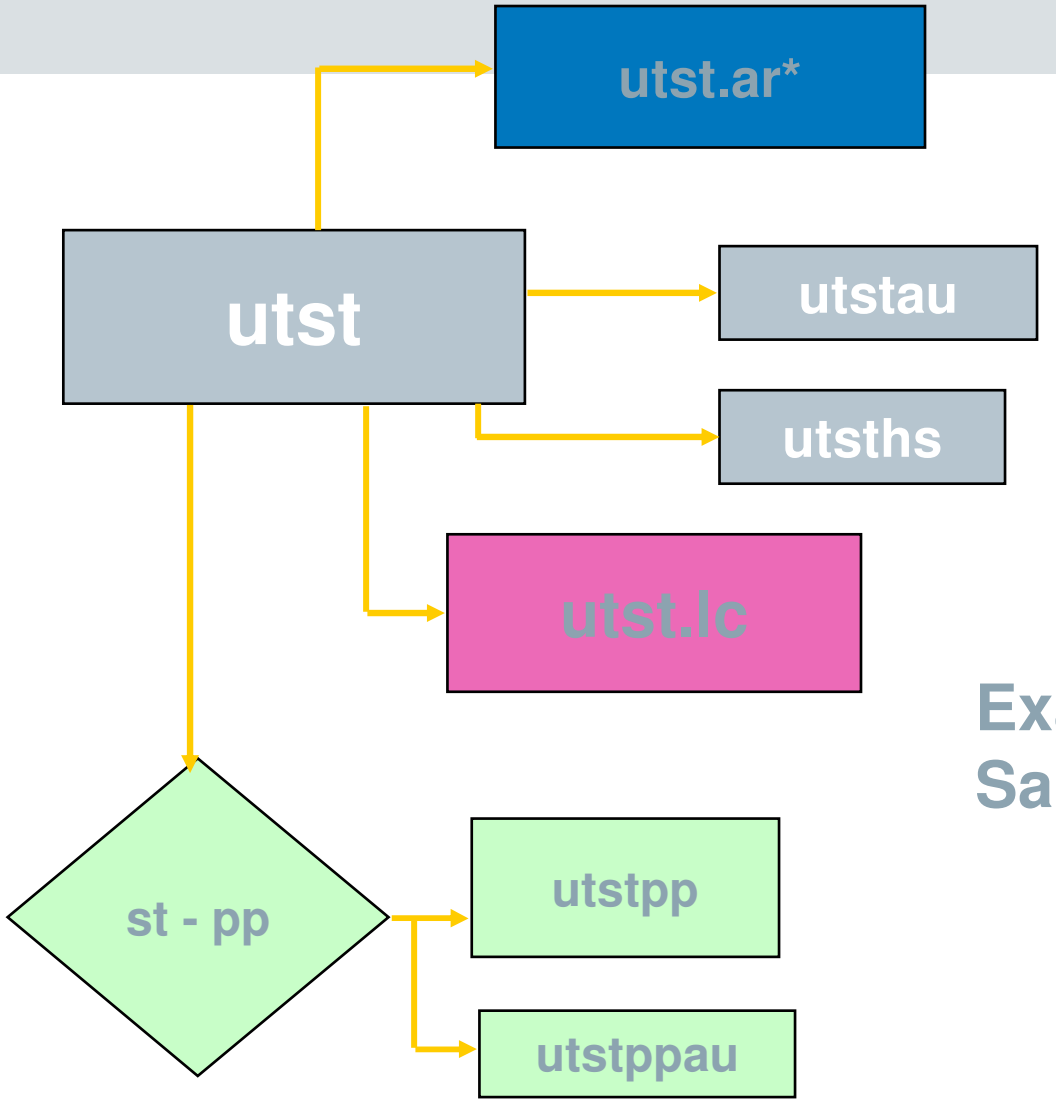
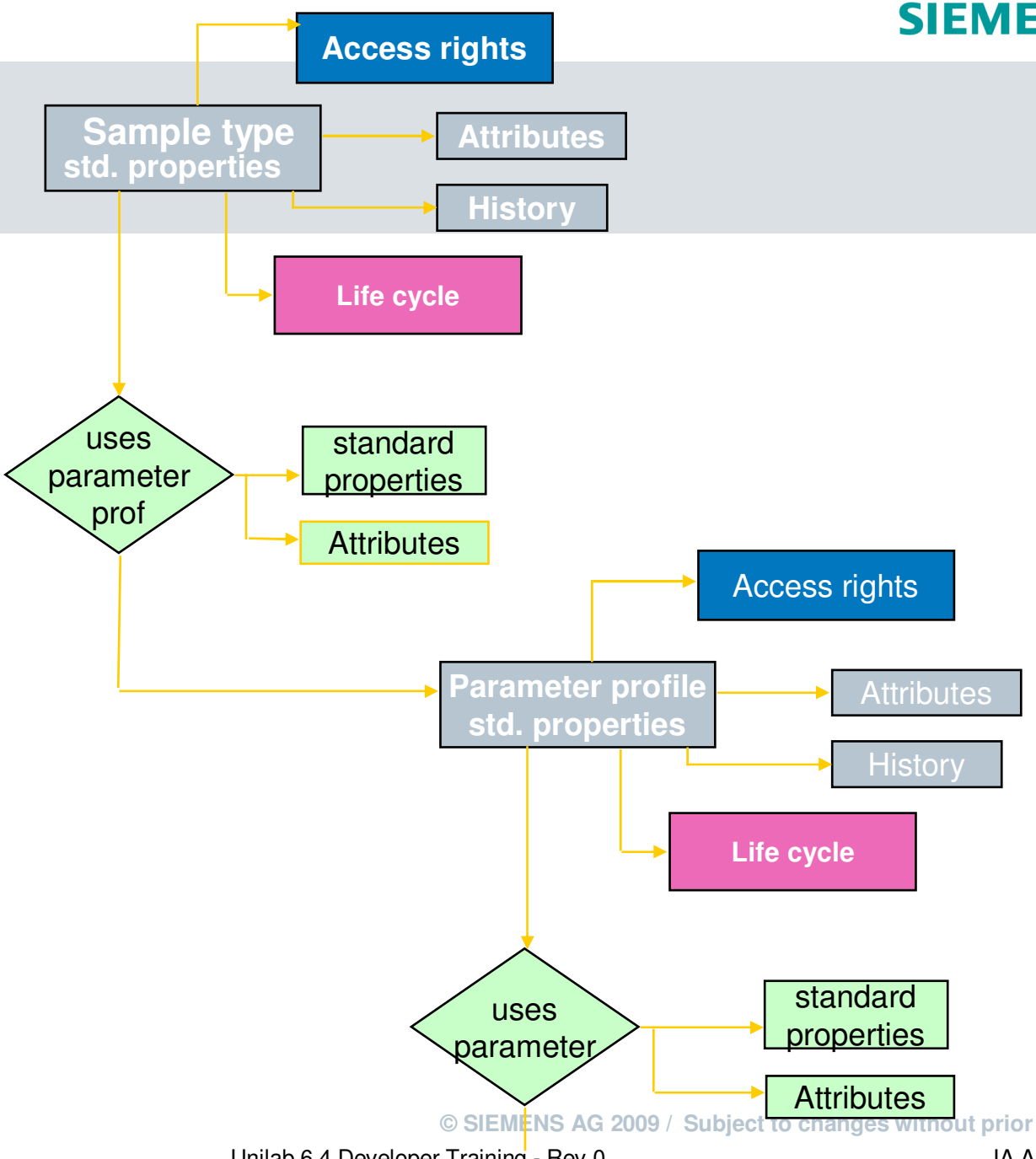


Table Model

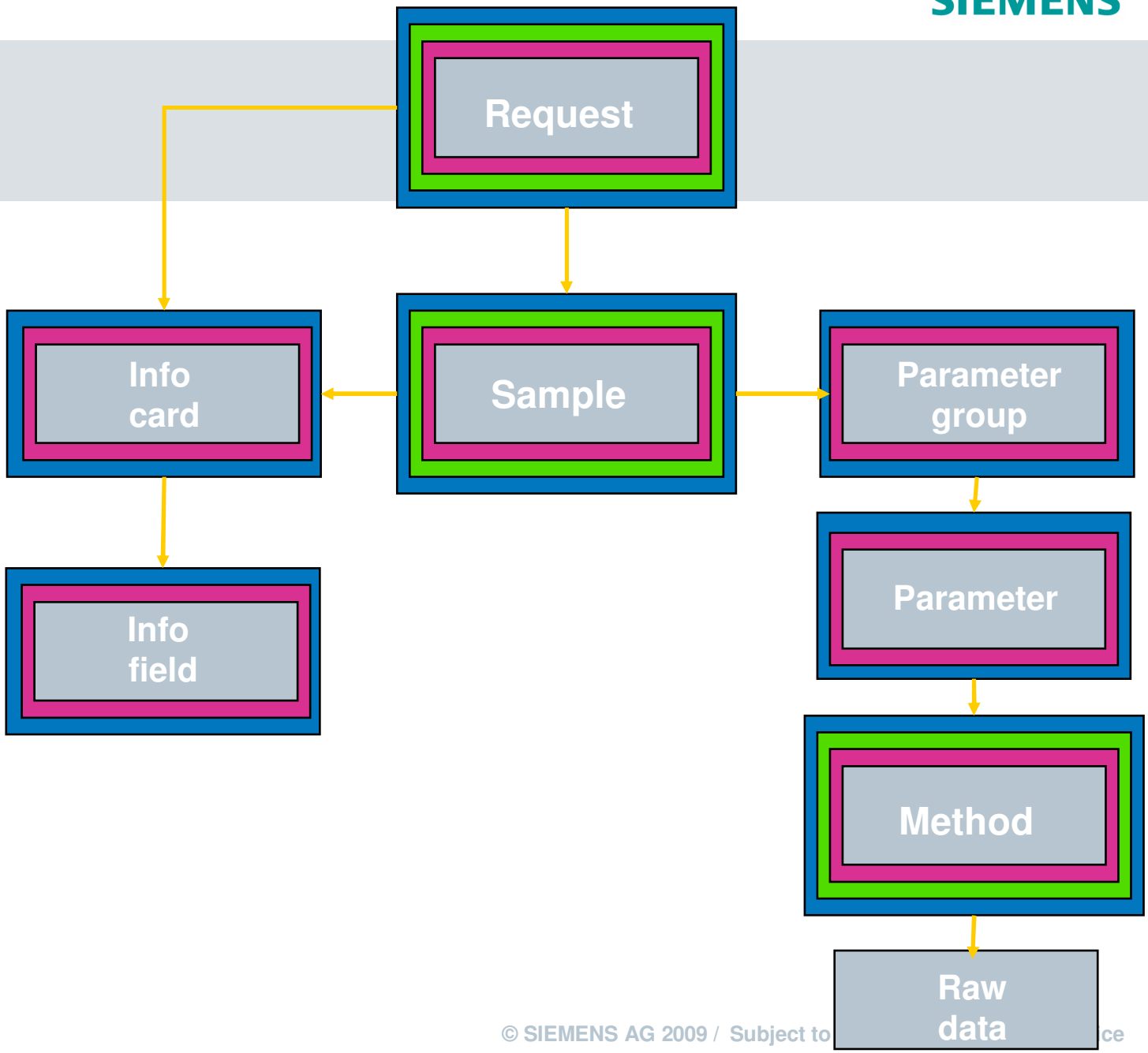


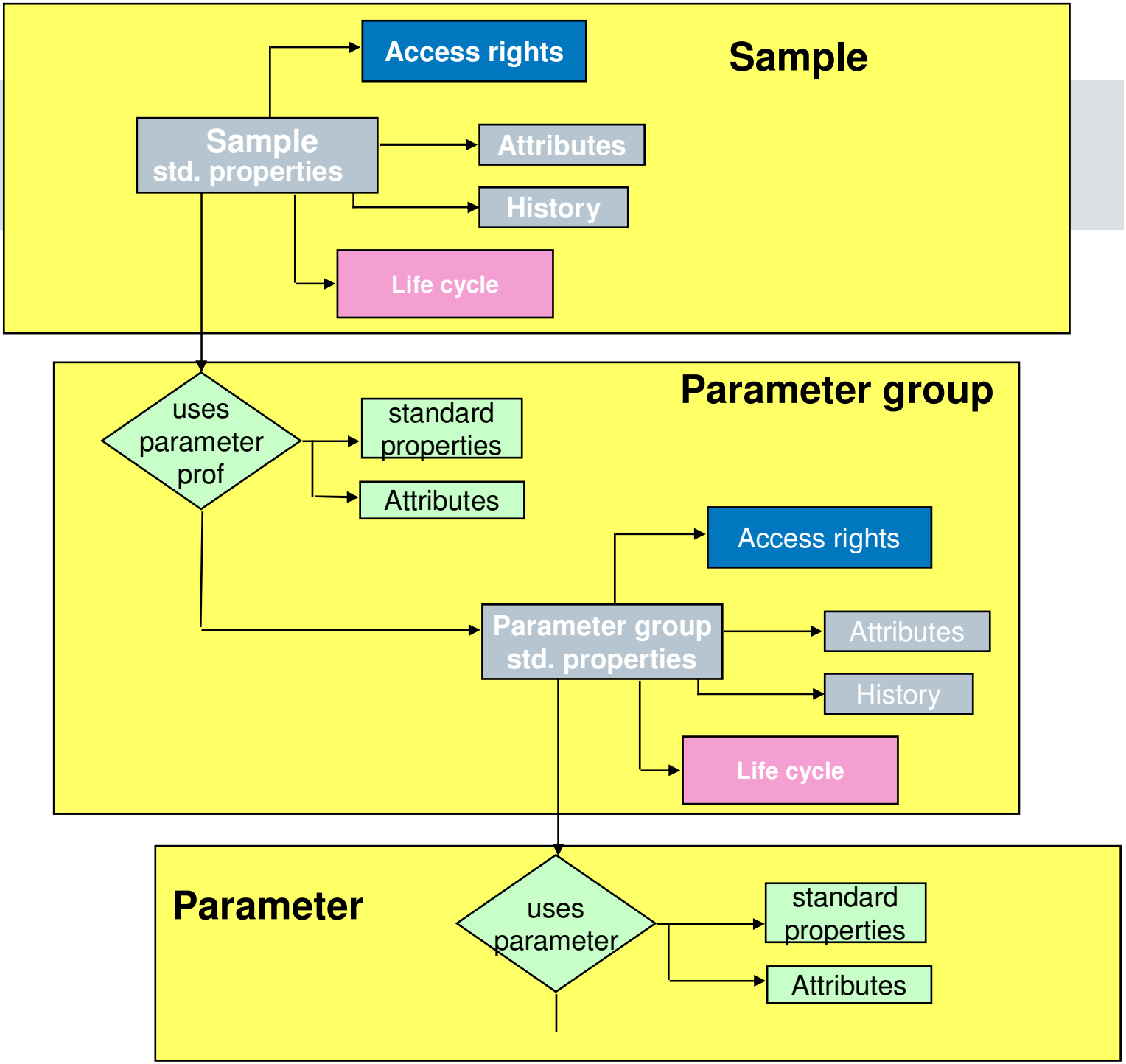
Example:
Sample Type

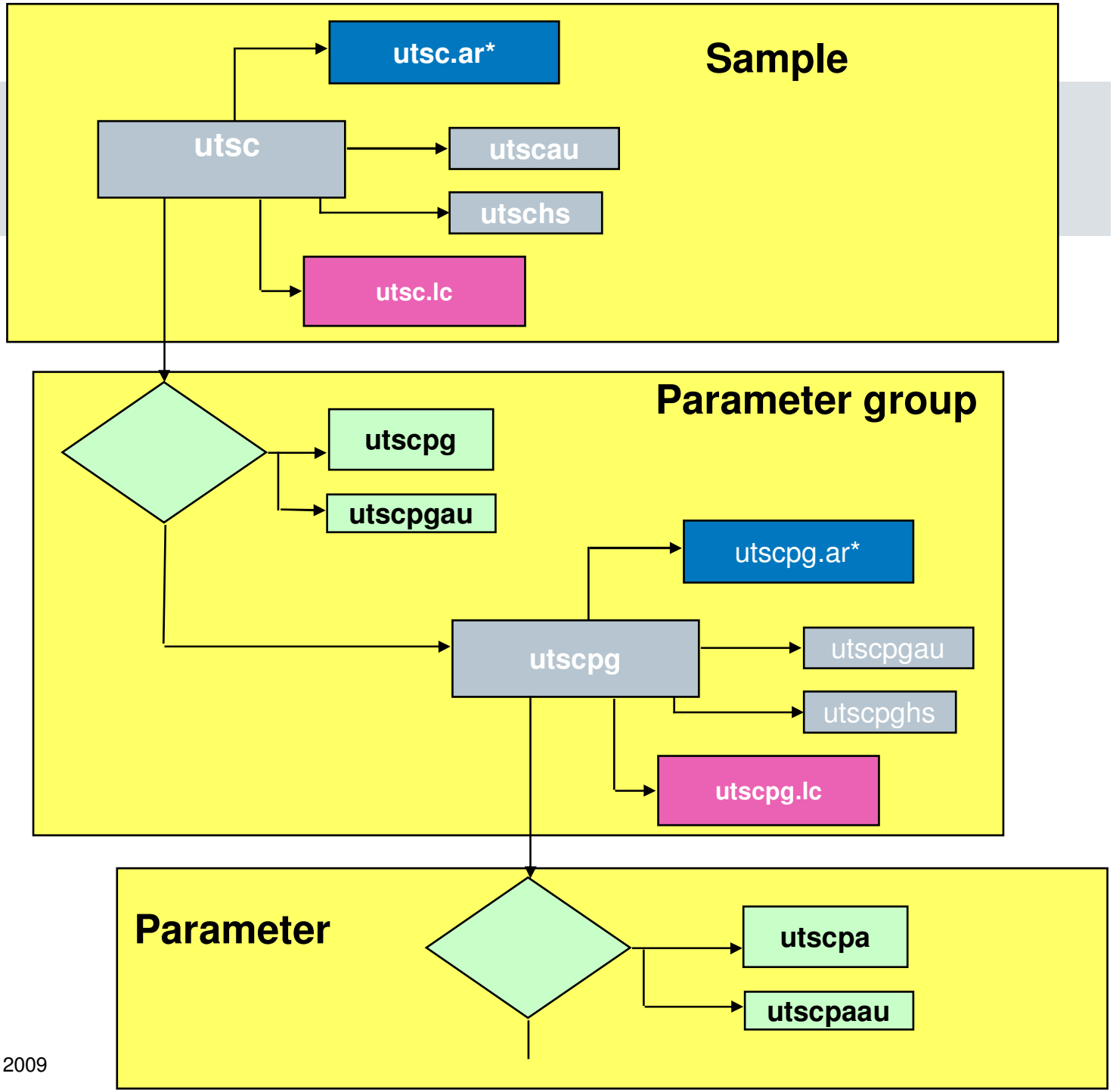
Configuration



Operation

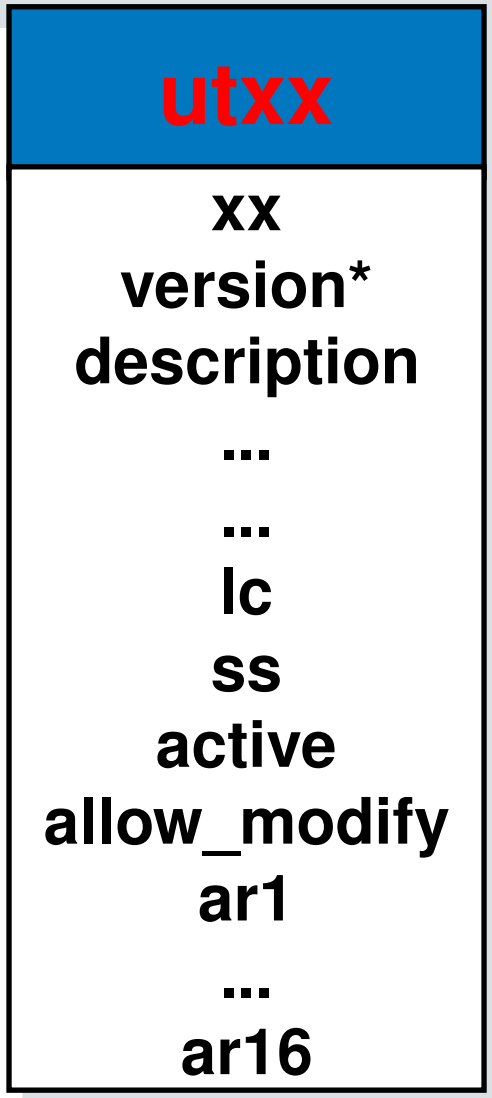






Database

Table structure



Main object table

primary key

all std. attributes

only if object has access rights

* Not in utsc, utrq

Table structure (2)

Used Object (Configuration)

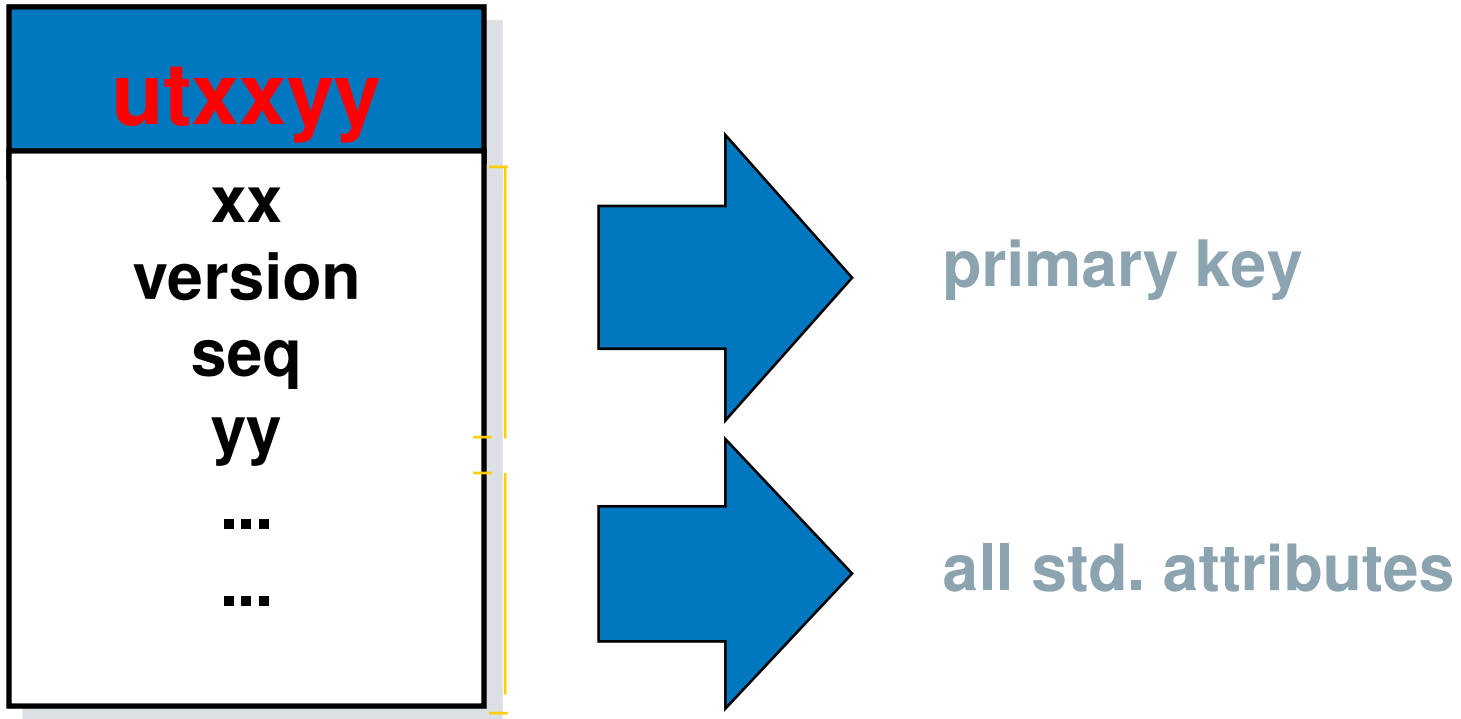
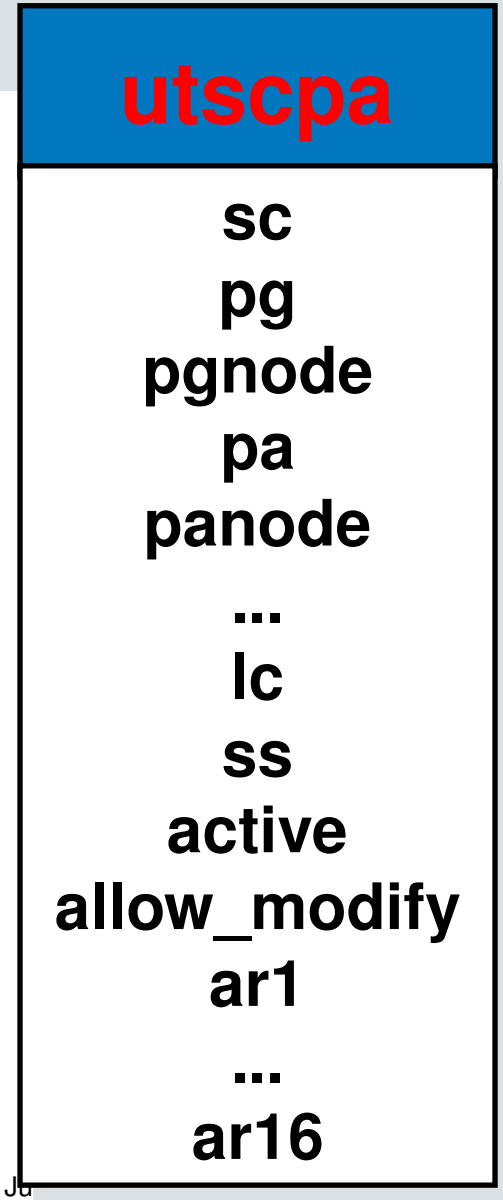


Table structure (3)



Operational object

primary key

all std. attributes

Node numbering

Nodes on all levels

- pgnode, panode, menode, icnode, iinode, rdnode

Each node only relates to one level

- pgnode: sc - pg relation

Nodes add up downward

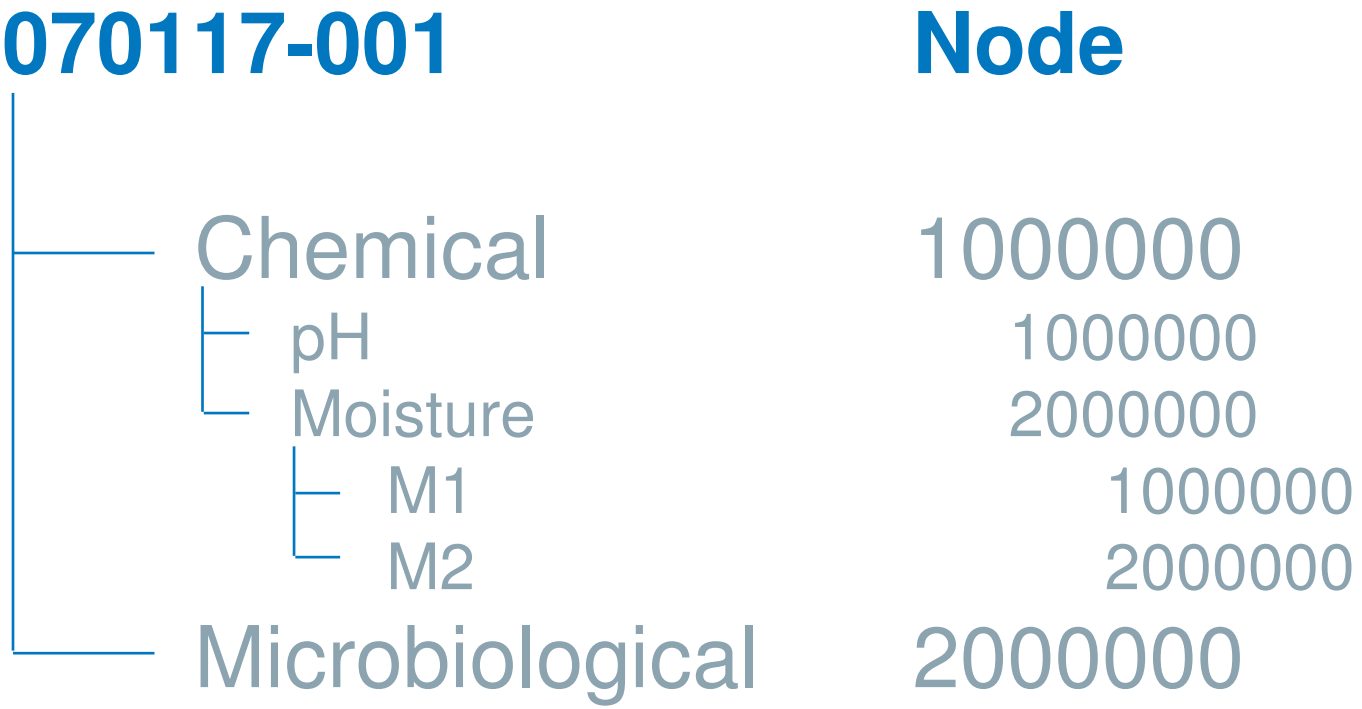
- sc - pg - pa: pgnode AND panode

Node numbering allows

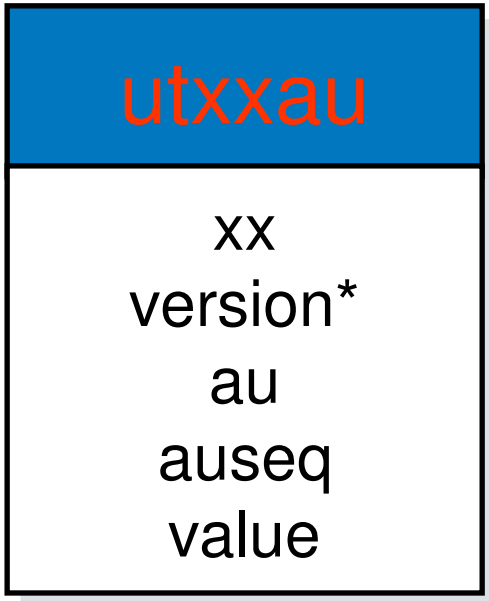
- insert before
- insert between
- append

Node Numbering(2)

Example



Attribute tables



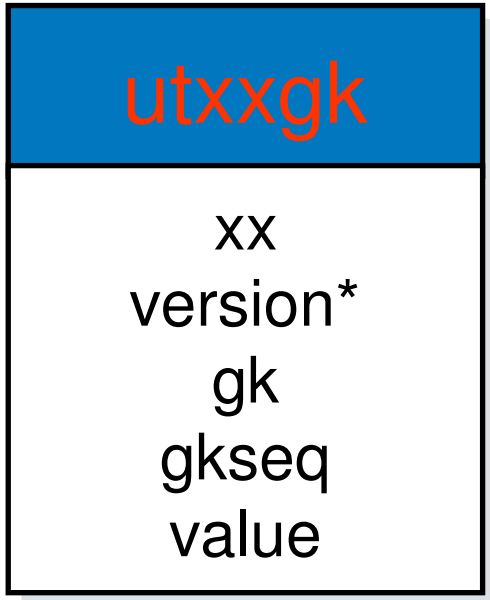
Au configuration tables

- Utau
- Utauhs
- Value lists
  - Utaulist
  - utausql

* Not in operational tables

Group key tables

Table structure identical to attributes



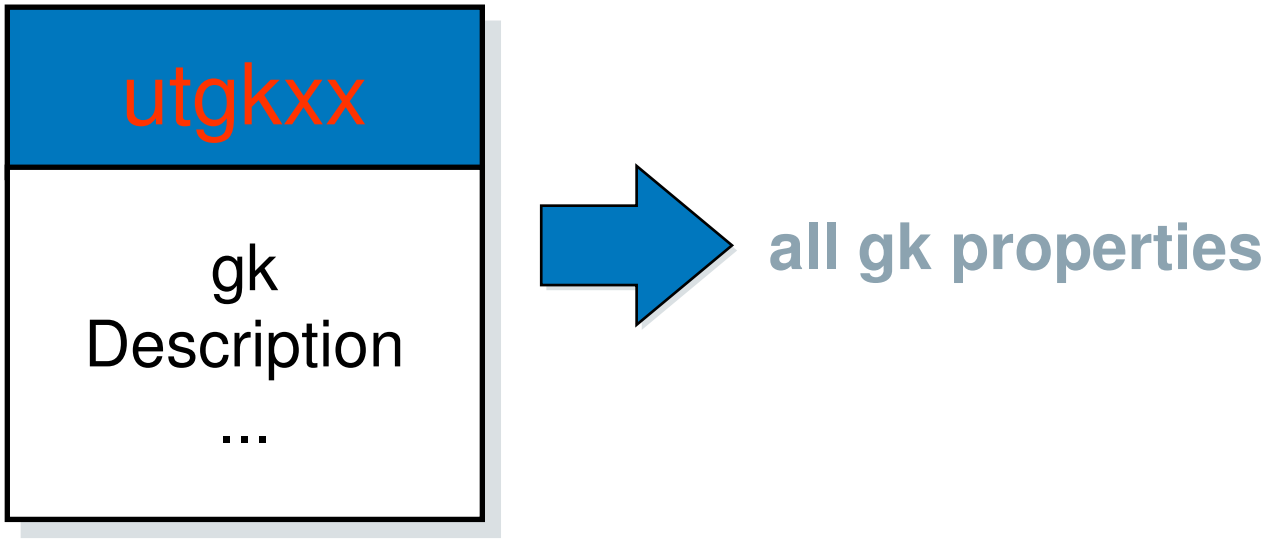
- Not all objects have group keys
  - Only st, sc, rt, rq and me

* Not in operational tables

Group key tables - Group key definition

utgkxx

- Example: utgkst, utgksc



- One table per object type
- utgkxxlist, utgkxxsql

Group key tables (2)

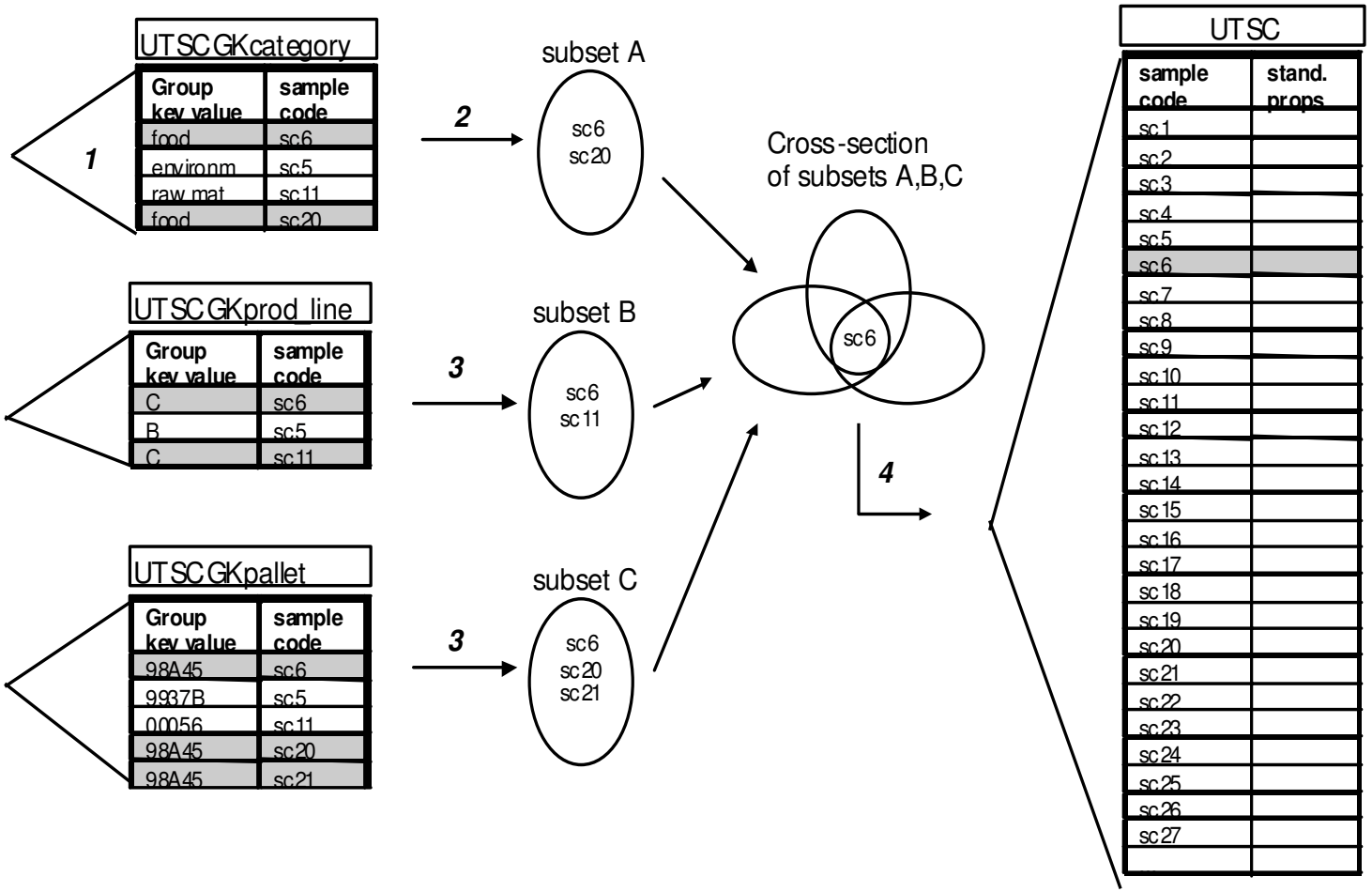
Special tables per group key

- E.g. group key ‘supplier’ on st
  - Special table utstgksupplier
  - Columns: st, supplier

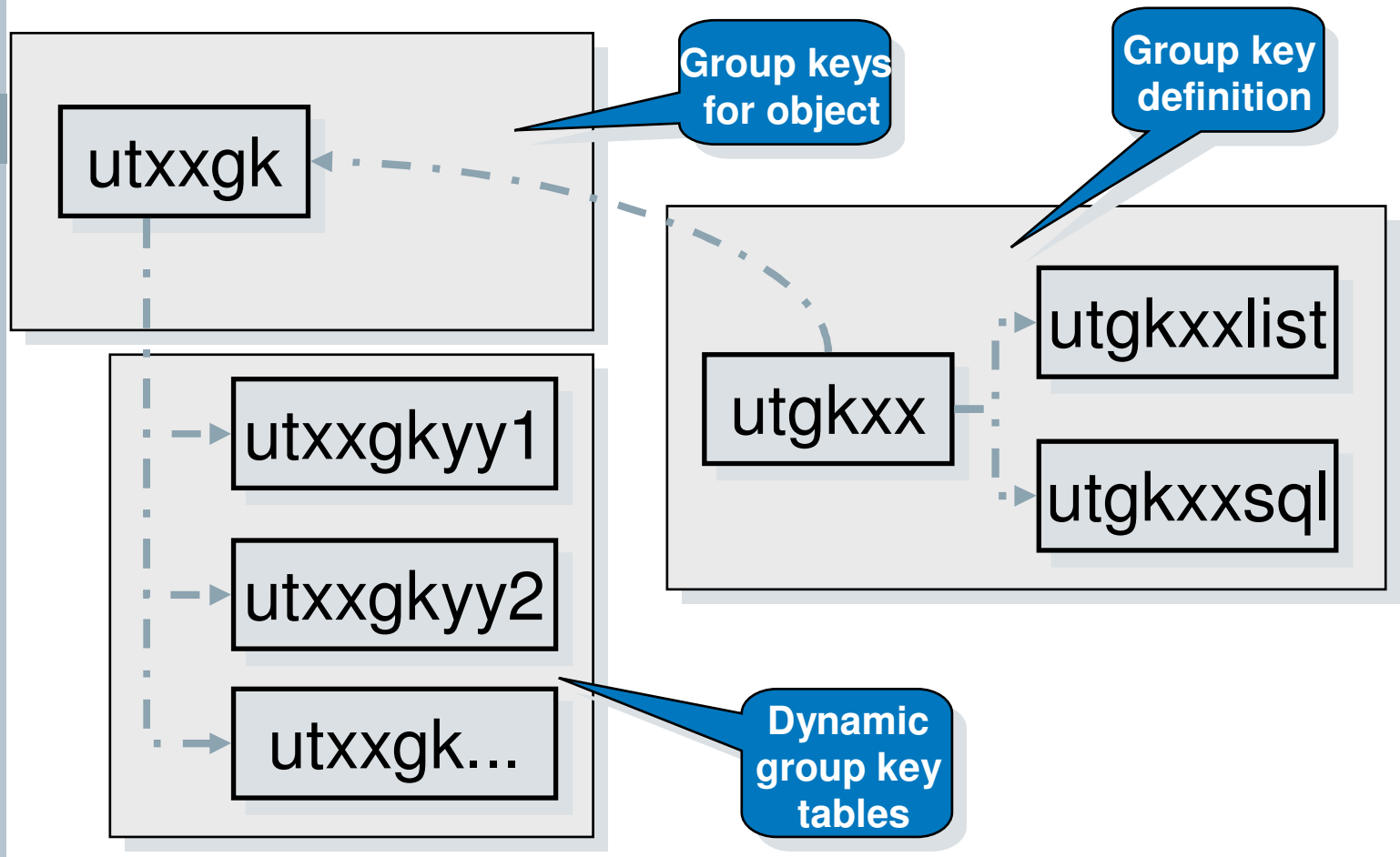
Use

- Safety: dynamic tables
- Recreation of structures
- In GUI: overview of all group keys

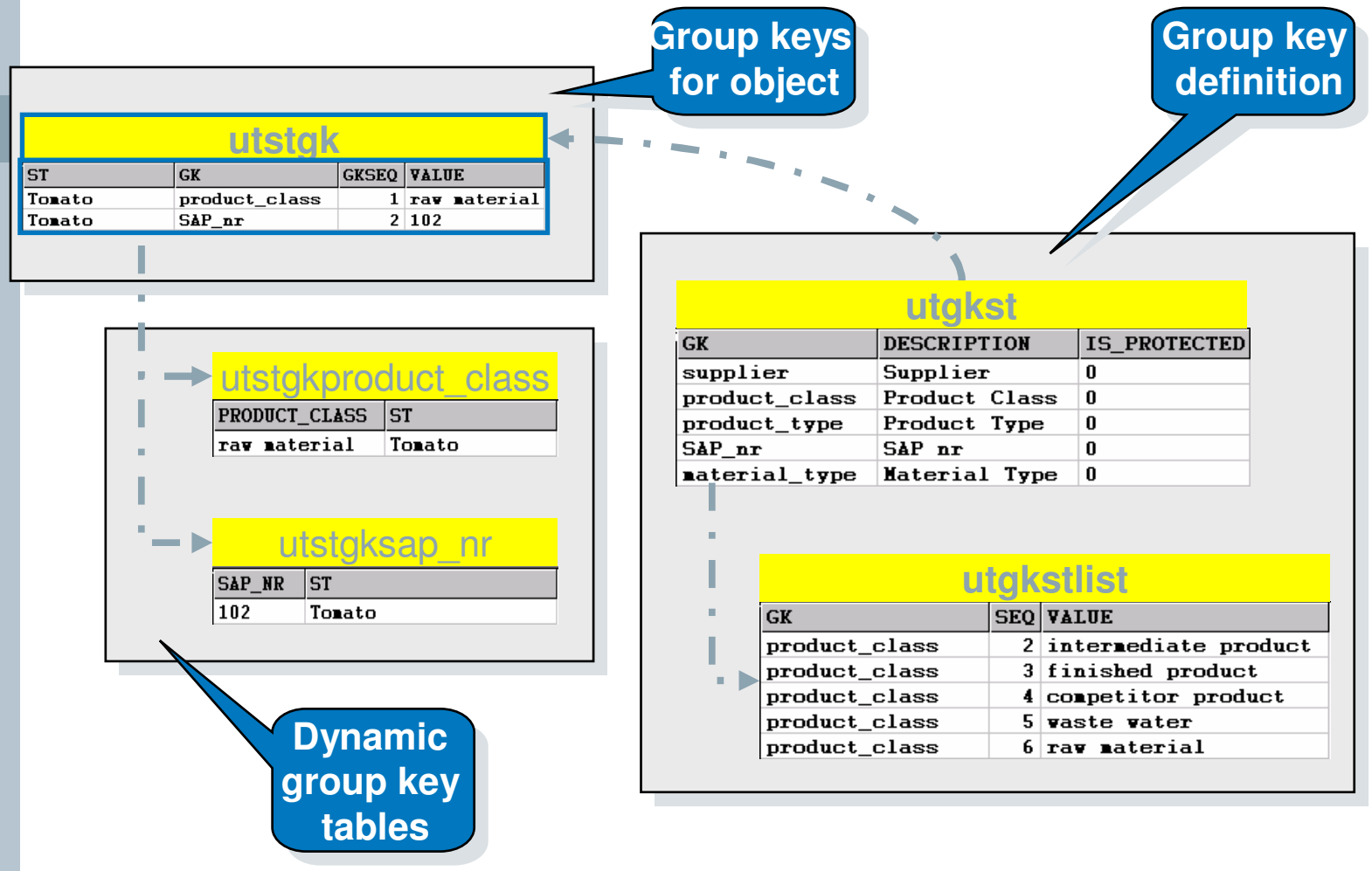
Group key mechanism



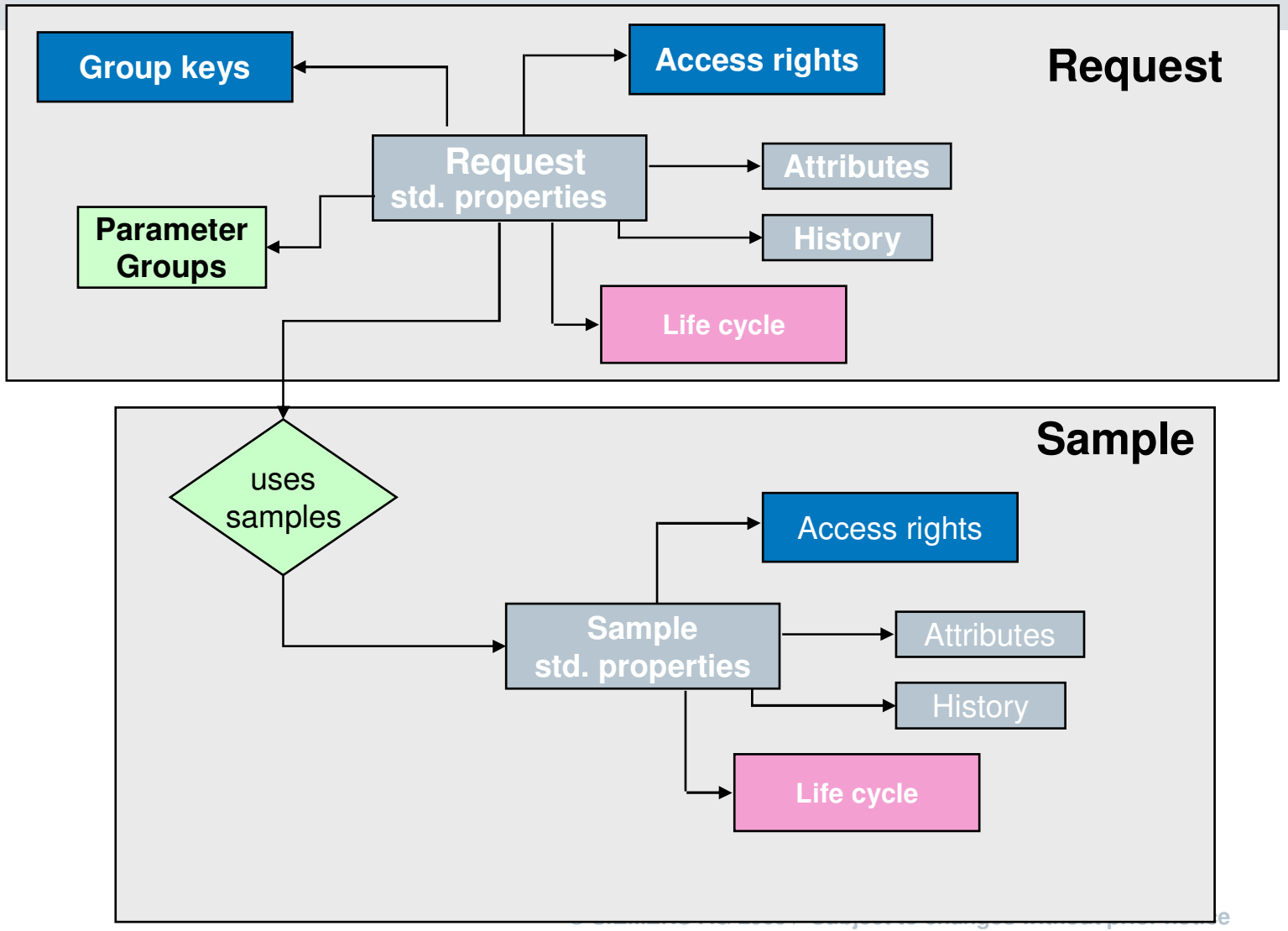
Group key tables - Overview



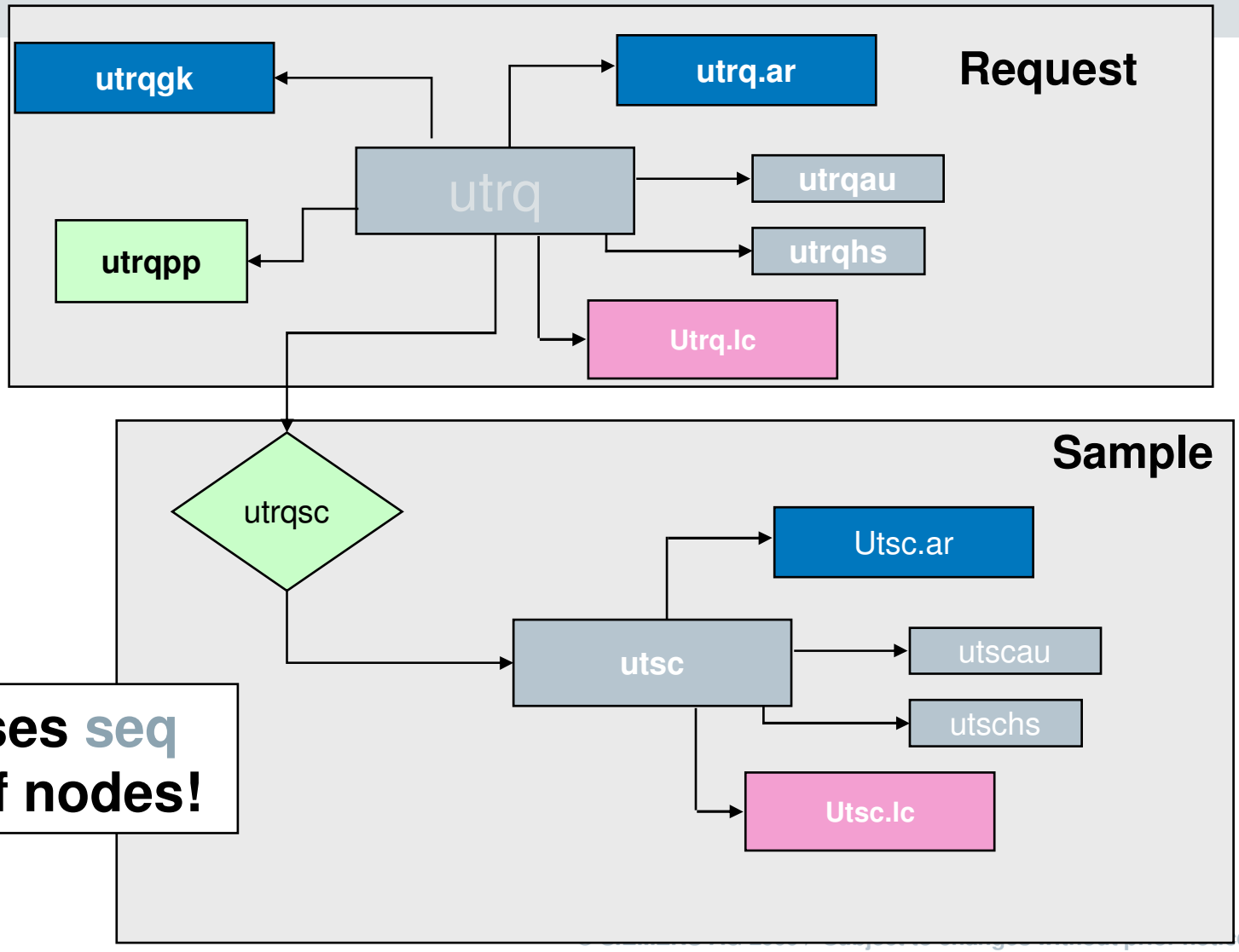
Group key tables - Example



DB Model - Requests

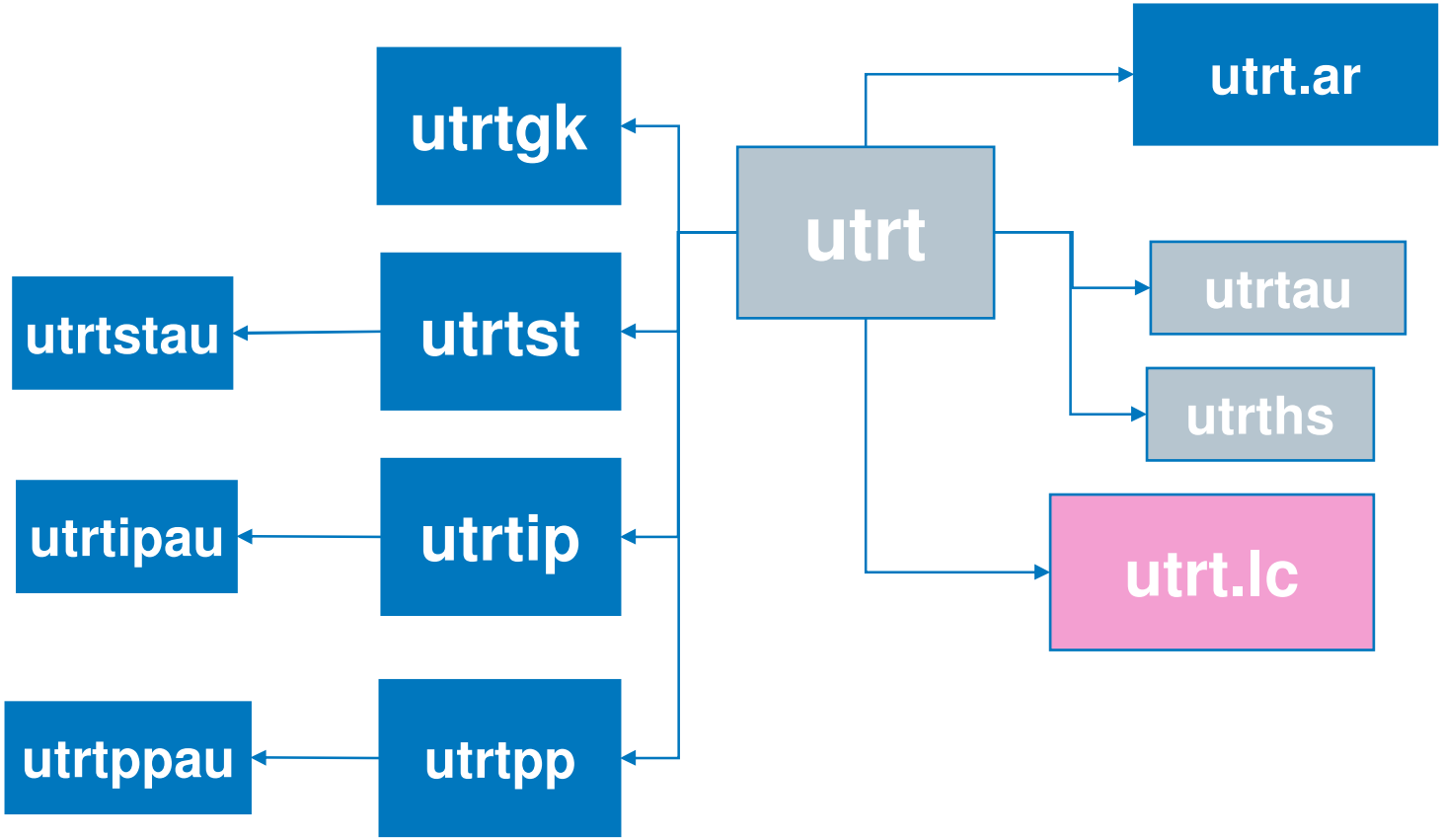


DB Model - Requests

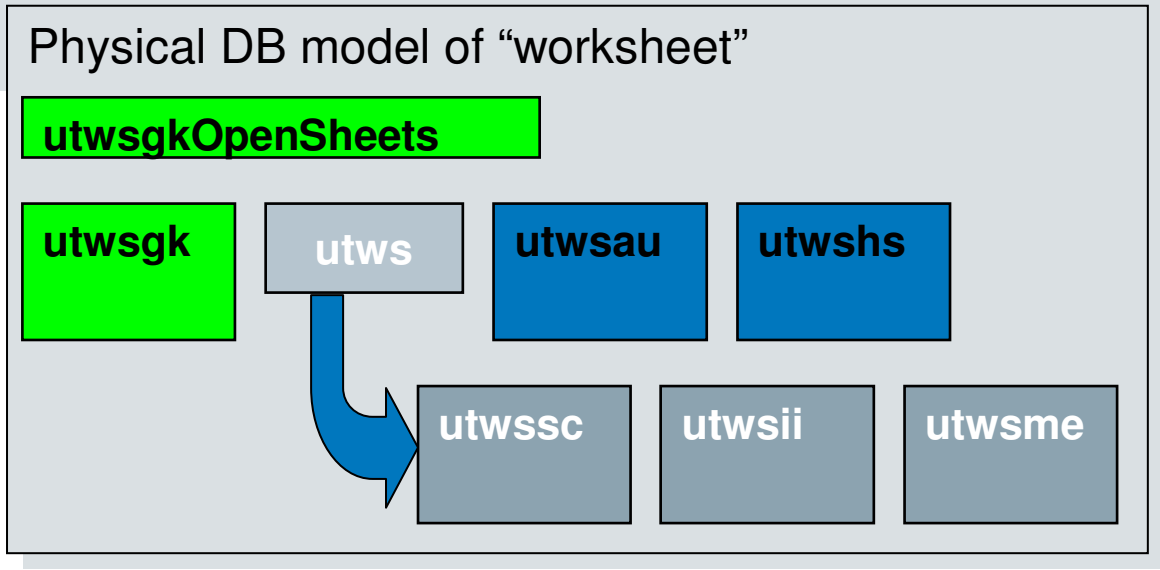


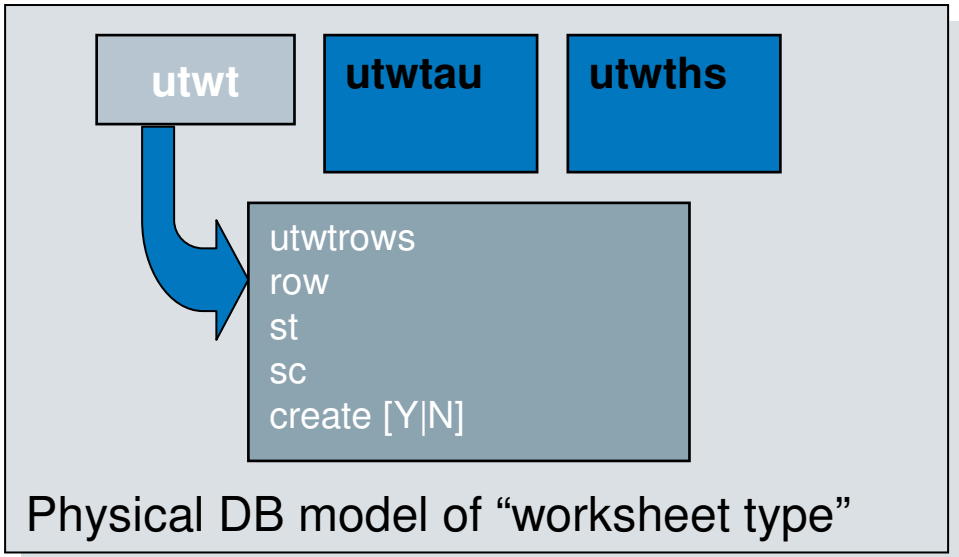
Utrqpp uses seq instead of nodes!

Request Types

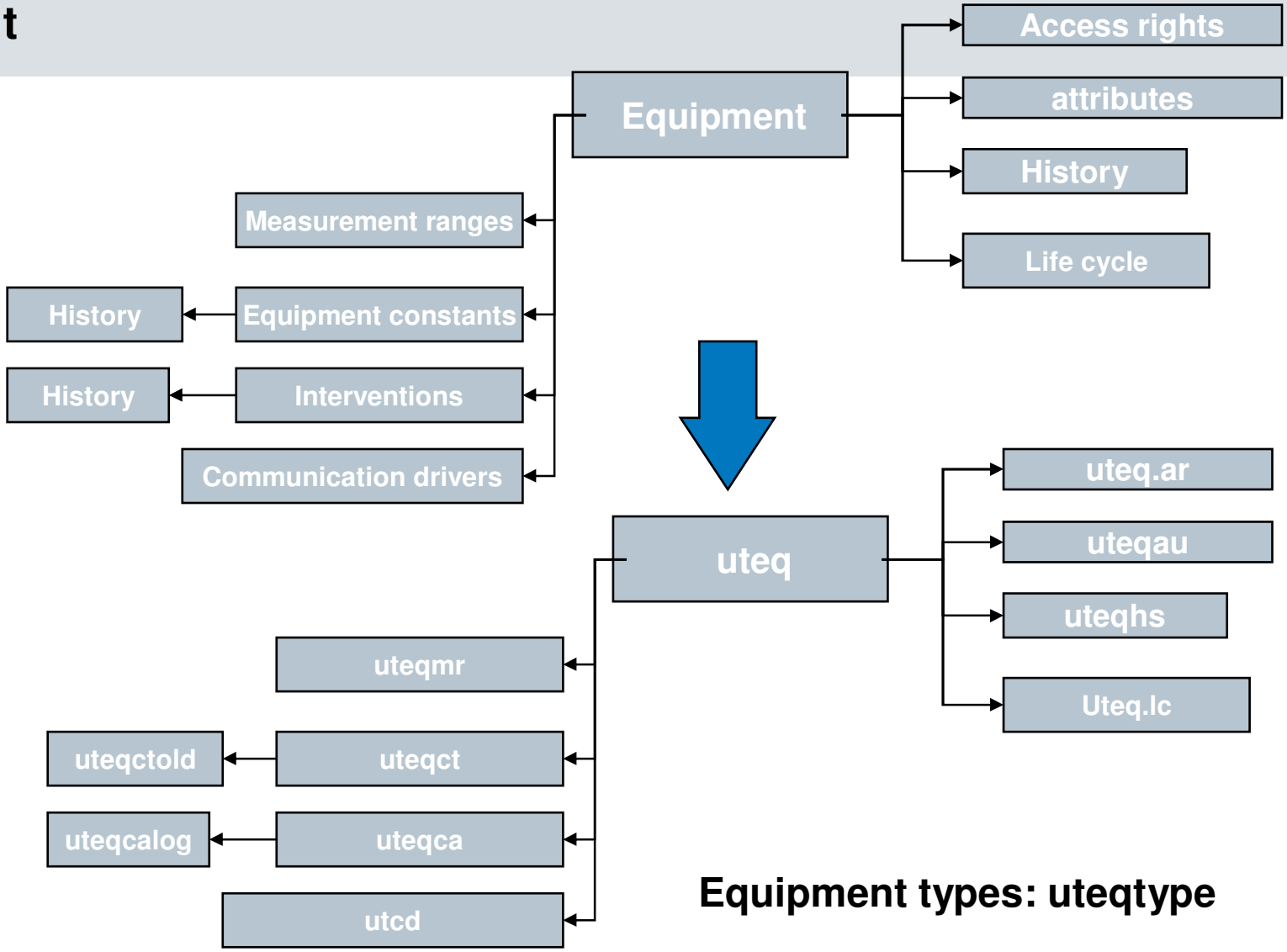


DB Model Worksheets



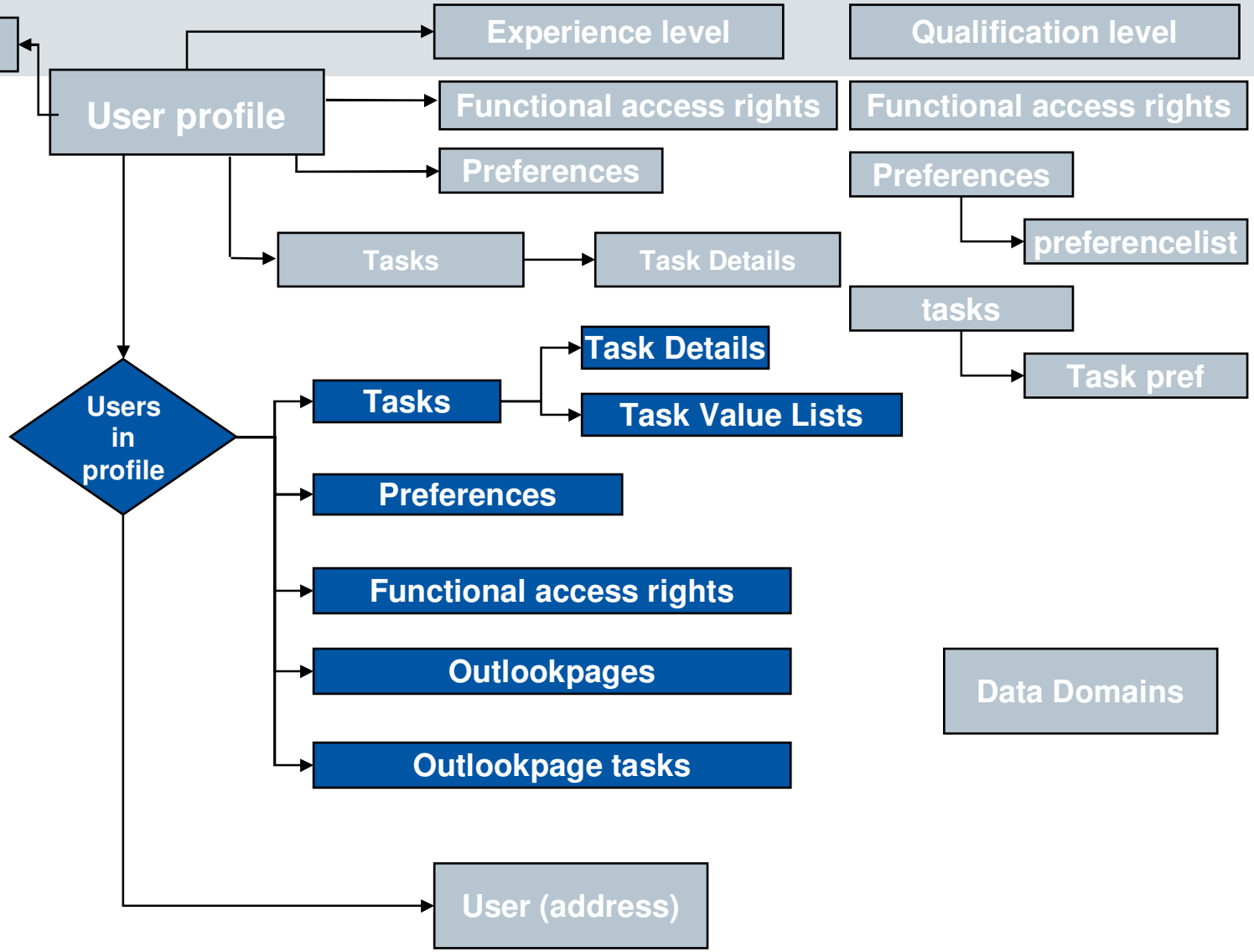


Equipment

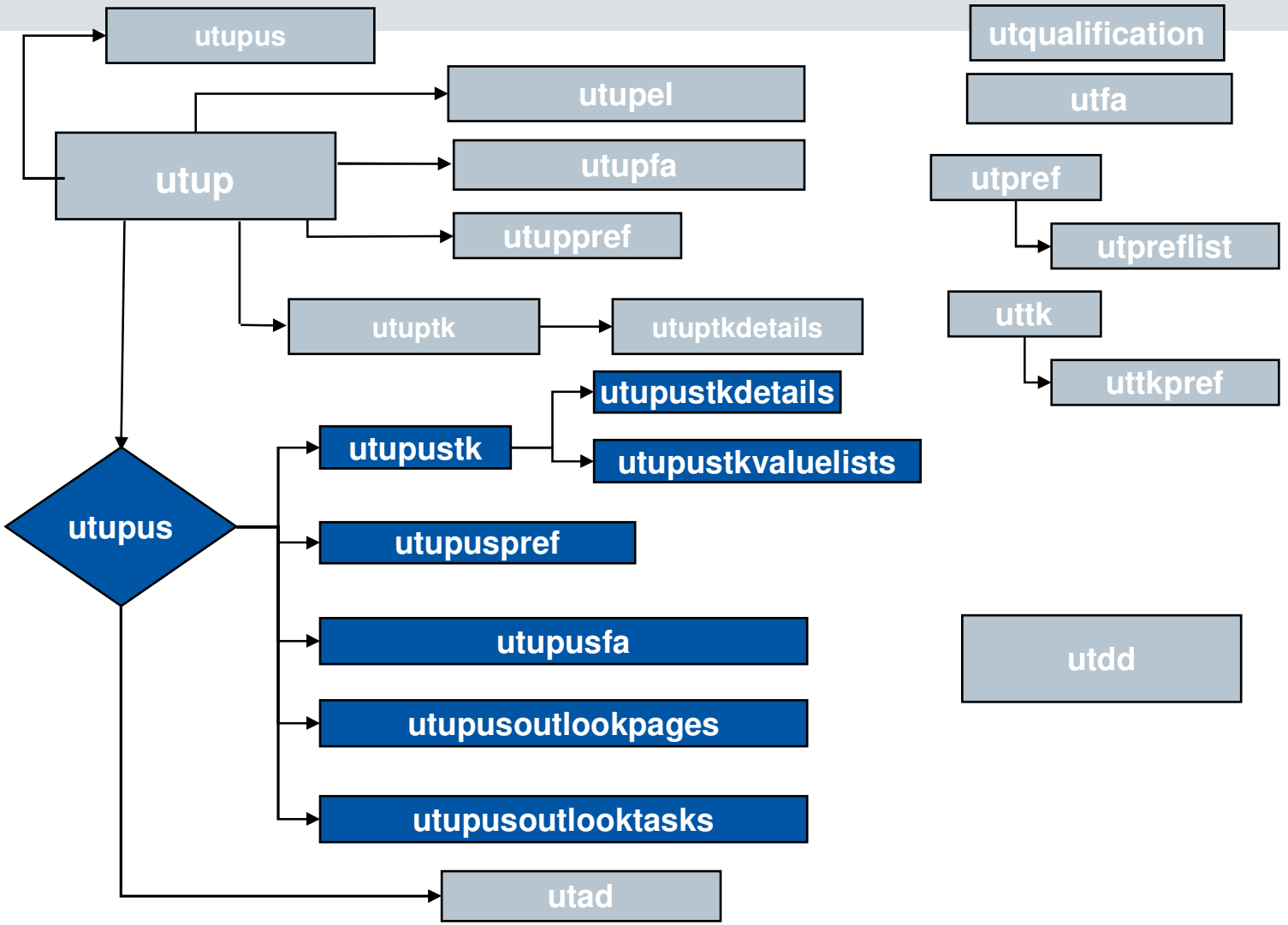


Equipment types: uteqtype

User profiles / users - Model

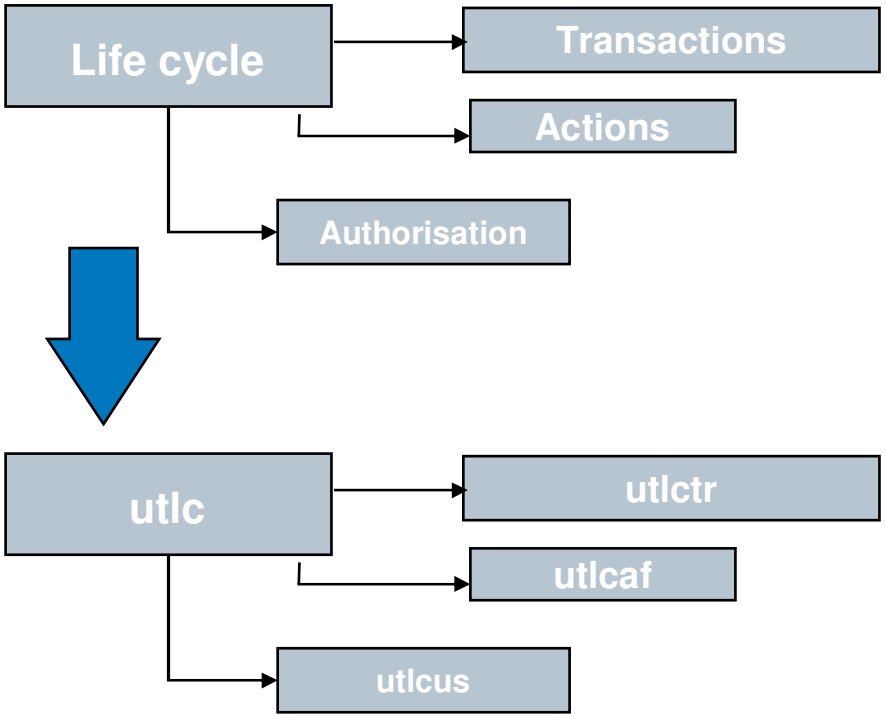


User profiles / users - Model



Life cycles - States

Life cycles

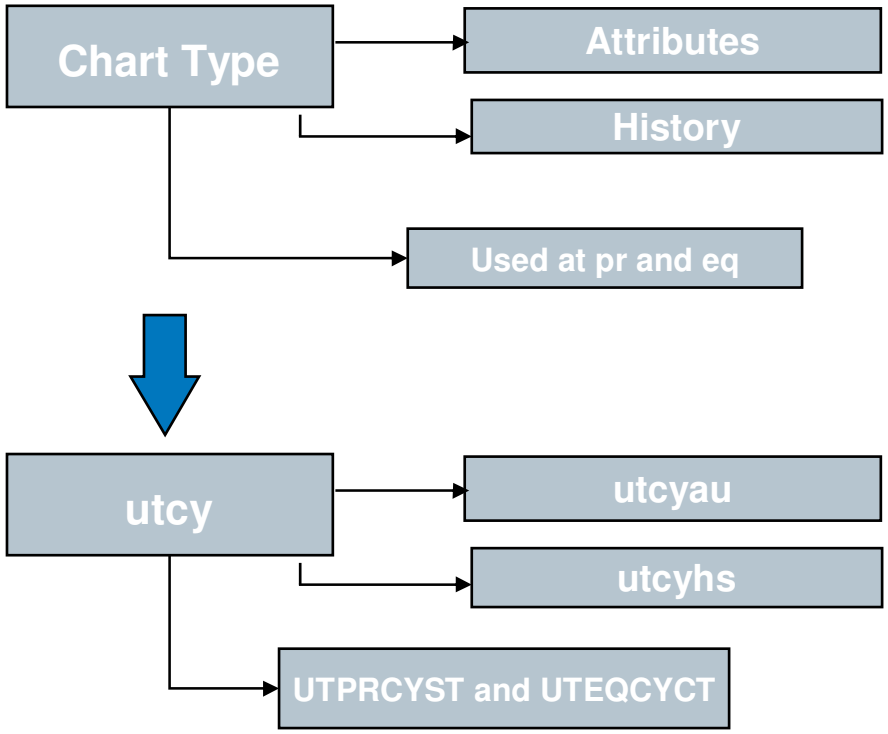


States

- utss
- utsswl: worklist assignment rules

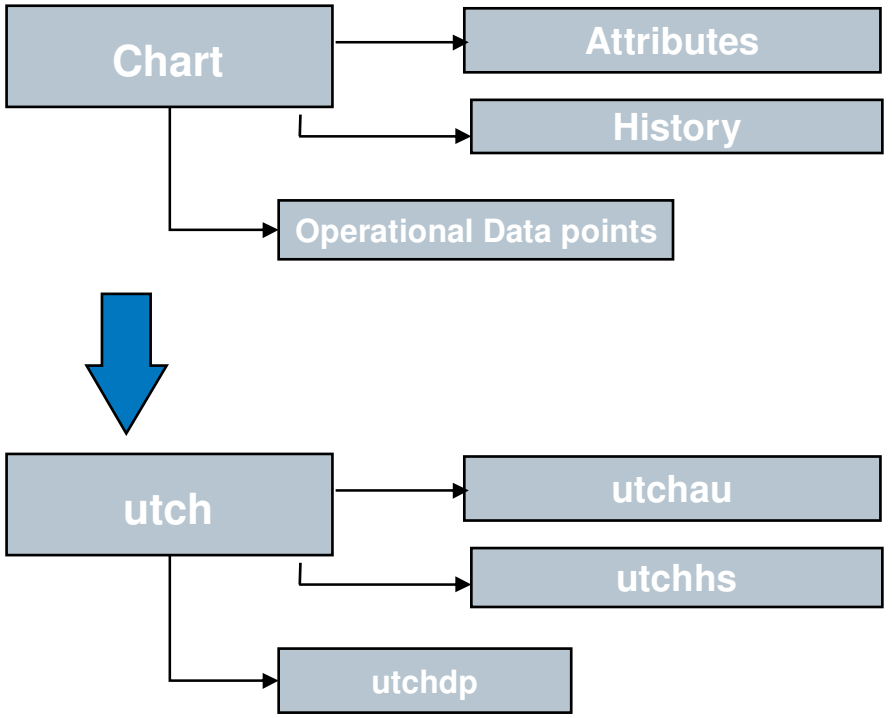
Tables for Charts (configuration)

Charts



Tables for Charts (operational)

Charts



- Alarm exceptions
  - utresultexception

Special Tables (1)

Specs

- utppspa, utppspb, utppspc
- utscpaspa, utscpaspb, utscpaspc

Reanalysis tables

- utrscxx (e.g. utrscpa)
- utrscpaspa: specs valid for reanalyses
- utrscrd: raw data for reanalyses

Unilink

- utul…

Layouts

- utly, utuicomponent

Unique code masks / Counters

- utuc,…/ utcounter

Special Tables (2)

Sample type based freq	→	utstprfreq, utstmtfreq
Lookup table	→	utlu
Shortcuts	→	utshortcut
Delays	→	utdelay
Week numbers	→	utweeknr
year numbers	→	utyearnr
Archiving	→	utarchindex, uttoarchive
Predefined comment	→	utcomment
Decode for reporting	→	utdecode
Editable tables	→	utedtbl
Long text table	→	utlongtext
Print command	→	utprintcmds
Variable Format	→	utvformat
Title format	→	uttitlefmt

Special Tables (3)

General tables

- utsystem: system setting
- utobjects: general object properties
- utapplic: Application names
- utotdetails: Object type details
- utdba: for restricted use by DBA (DBA settings in DB)

Error logging

- uterror
- Logging of errors on DB side
- Only general failures are logged

Column Data Types

CHAR

- Only CHAR(1) or CHAR(2)
  - Boolean flags:
    - '0' = FALSE / '1' = TRUE
  - Access rights
    - 'N' = No Access / 'R' = Read Only / 'W' = Write

VARCHAR2

- All strings
- Length conventions:
  - Object IDs 20
  - Descriptions, titles 40
  - Values (ii, me, pa,...) 40
  - lc, ss 2
  - Comments (why in hs) 255

Column Data Types (2)

TIMESTAMP (with local time zone, with time zone)

- All relevant dates (e.g. sample dates)
  - creation date / sampling date / exec_start_date / date1...date5

RAW

- Color coding / Shortcut keys

NUMBER

- Sequencing (3)
- Numeric values (3 or 4)
  - E.g. priority, delay, freq_val
- Nodes (9)

FLOAT

- Numeric result values
  - value_f
- Specs

Physical Structure

Used table spaces

- uni_datac : configuration data
- uni_datao : operational data
- uni_indexc : configuration indexes
- uni_indexo: operational indexes
- uni_lob: large objects
- uni_temp: temporary table space
- uni_undo: undo table space

All tables, indexes, constraints have storage clauses

DB Installation Issues

Installation sizes

- 3 pre-configured DB sizes:
  - Small (5000 samples)
  - Medium (15000 samples)
  - Large (50000 samples)

Notice that:

- Full audit trail
- Copying attributes and group keys from configuration

are ‘space-sensitive’ issues.

Database

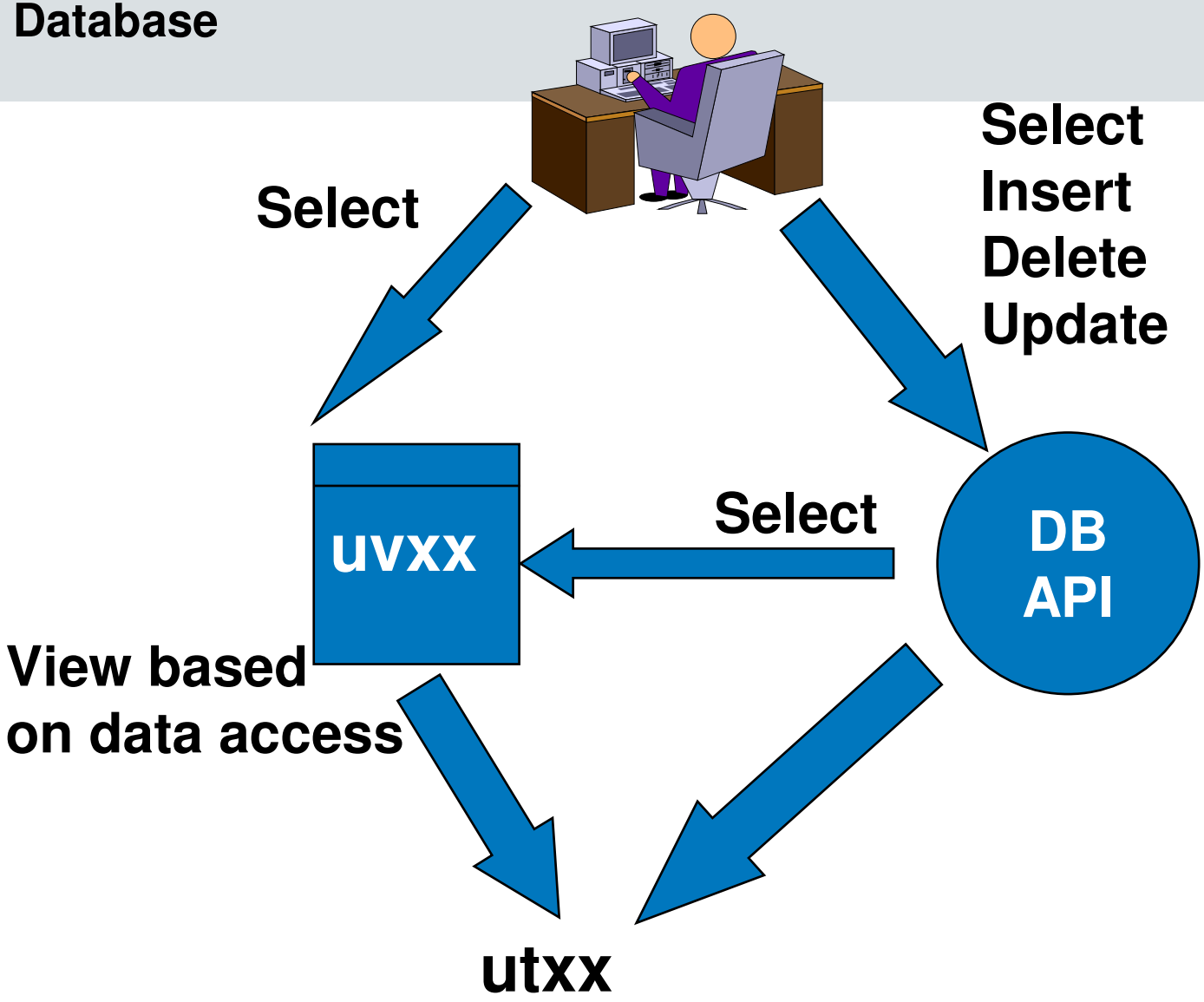
Querying the DB

Views vs Tables

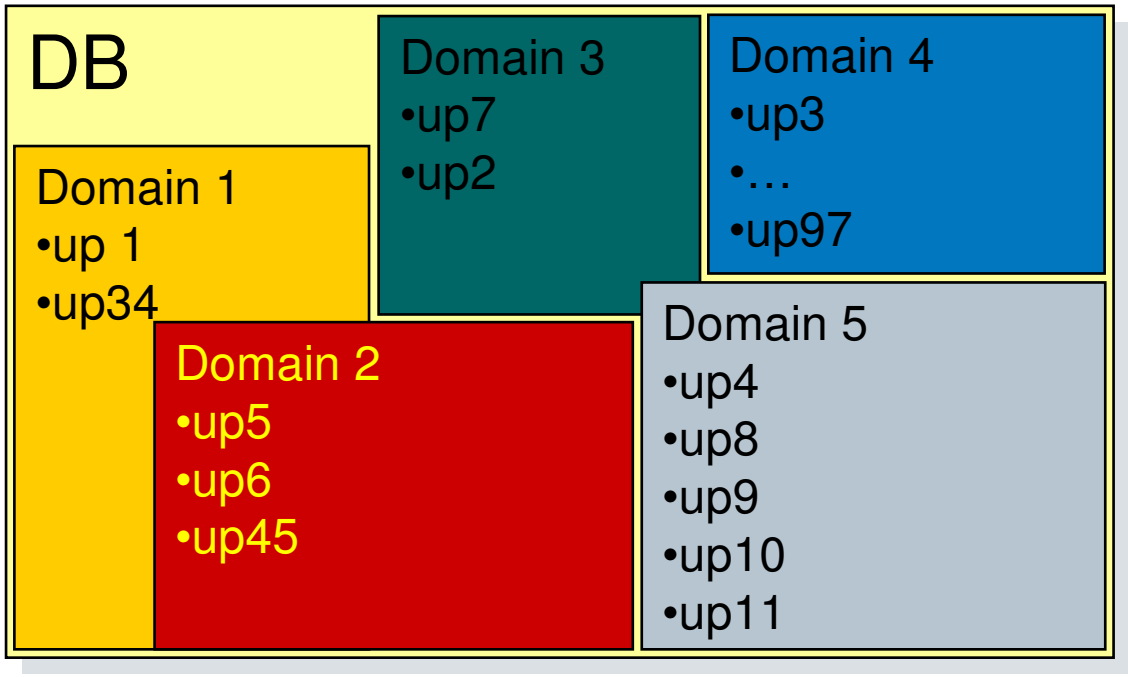
- Always use the views to query ALL tables
- On user creation, correct synonyms are created
- Name of view = name of table (except 2nd char)

  Examples:
  - uvst instead of utst
  - uvuppref instead of utuppref
- Use DB-API in applications
  - Ensures Authorisations
  - Performs required joins

Querying the Database



Data Access for User Profiles



Each User Profile (up) belongs to one specific data domain

View uvxx for dd2

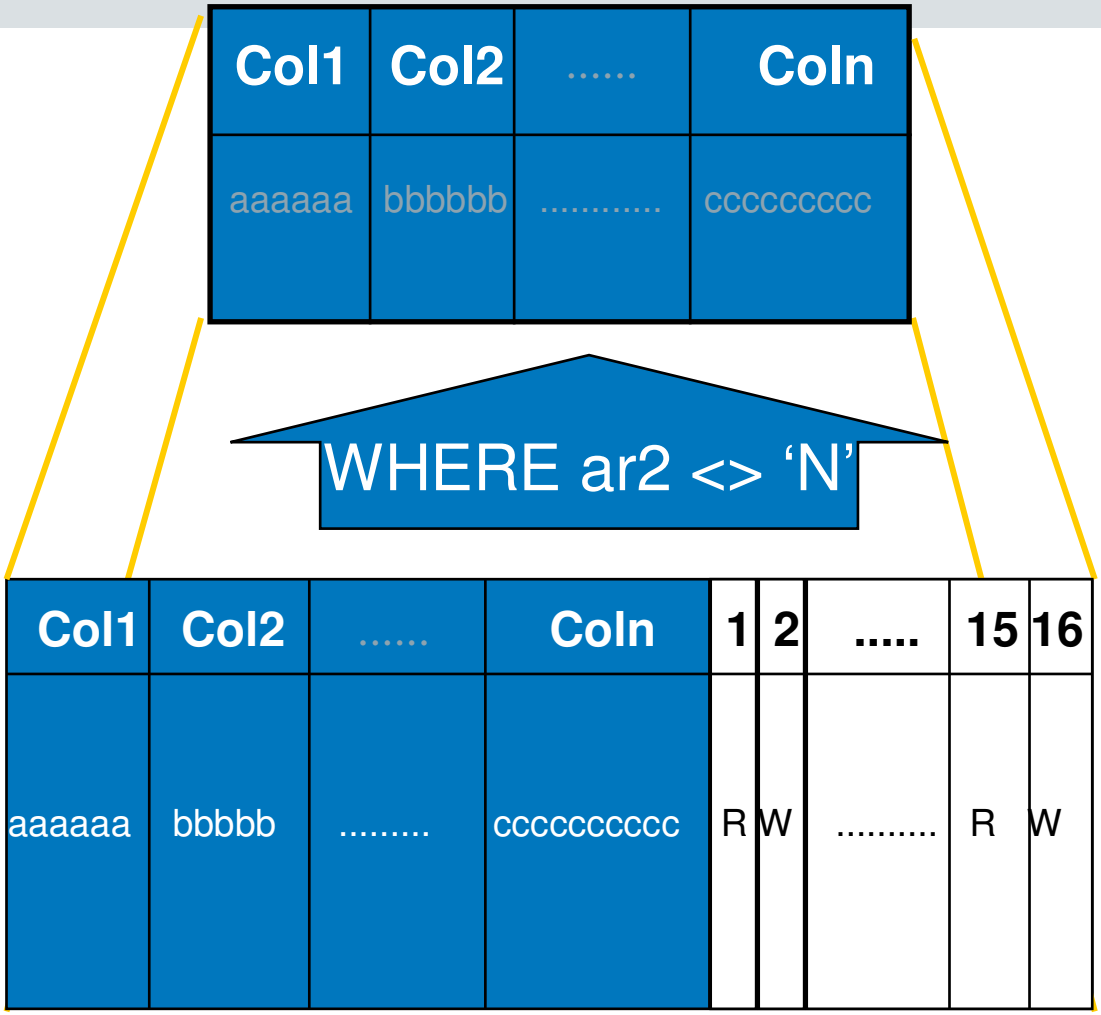
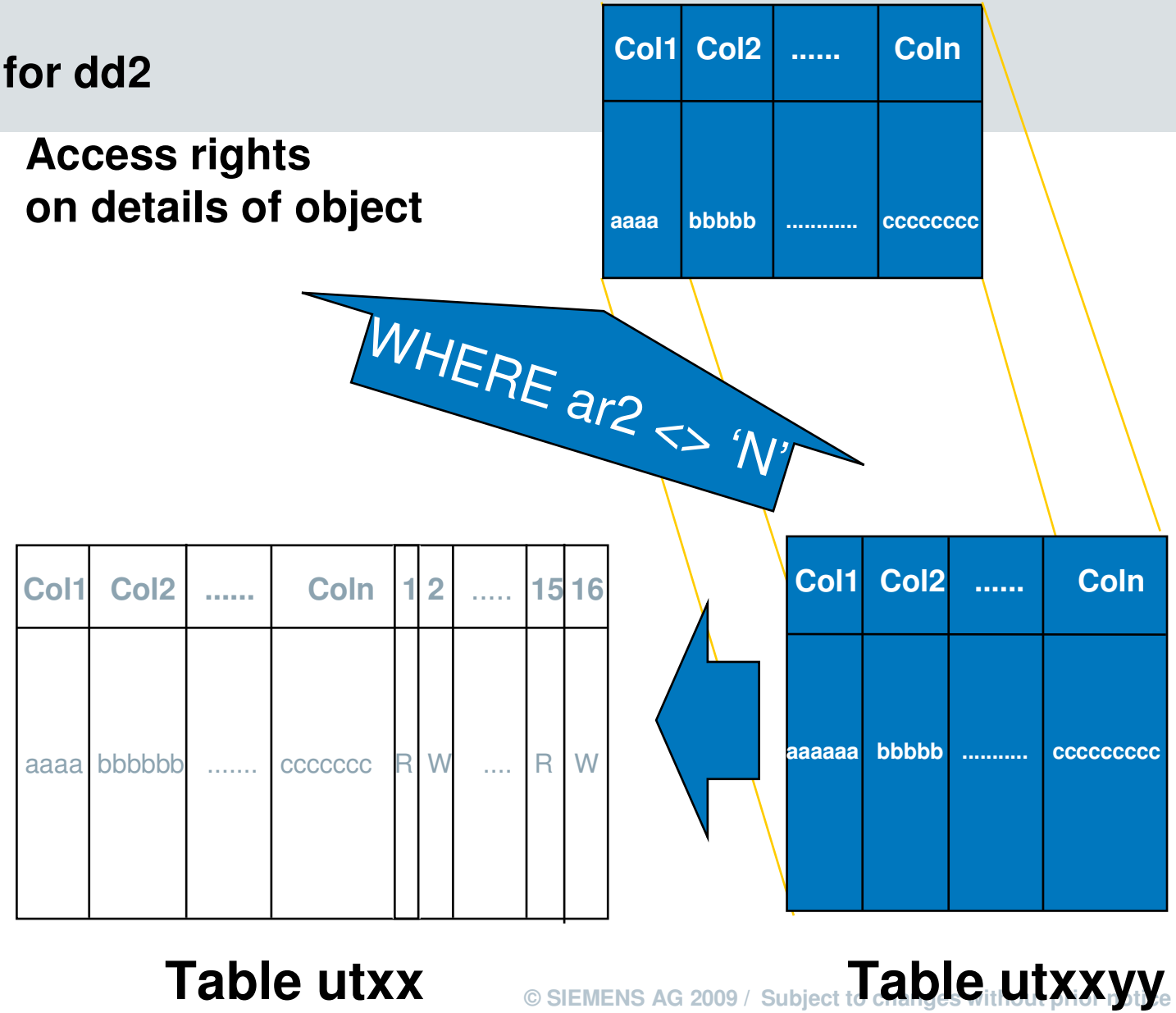


Table utxx

Access rights on main object

View uvxxyy for dd2

Access rights on details of object



Select up

NO up selection

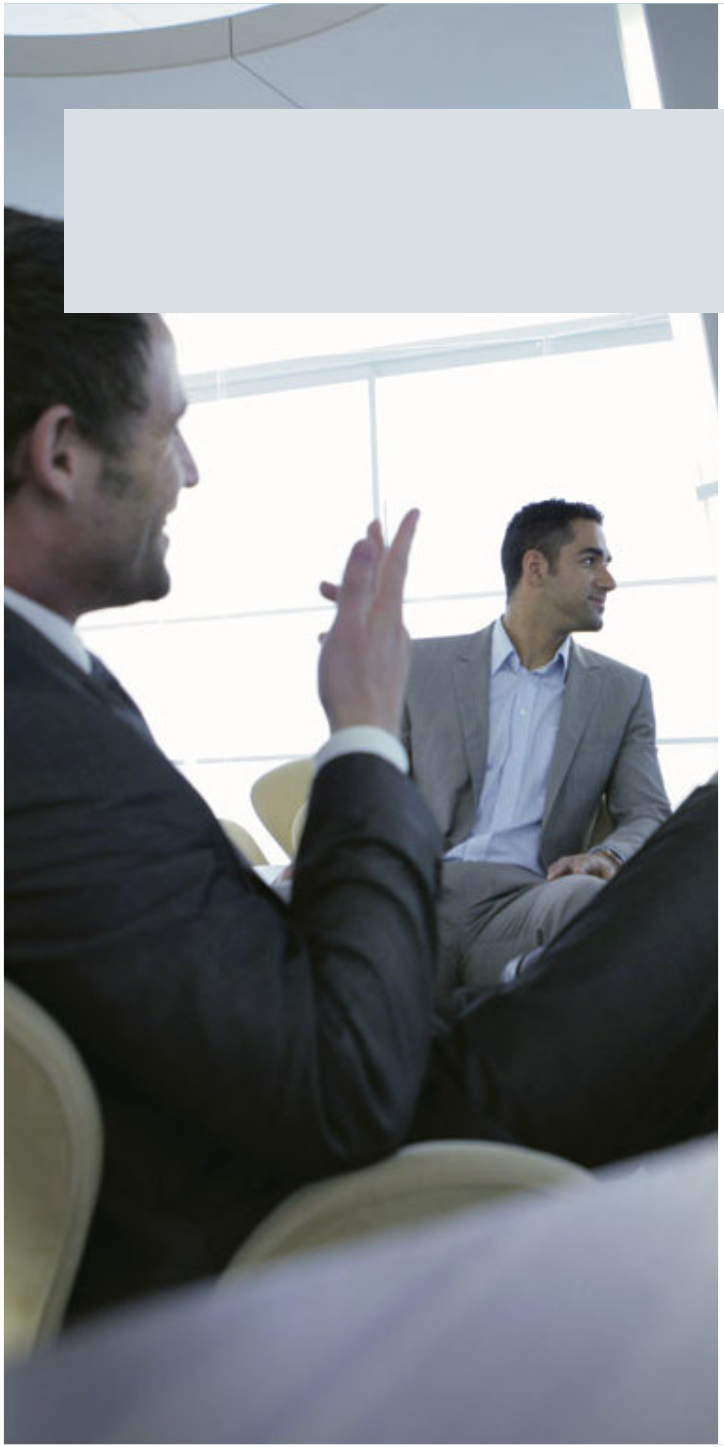
Simatic IT Unilab

Current up: ddx

Default up: ddx

Schema ddx

uvxx

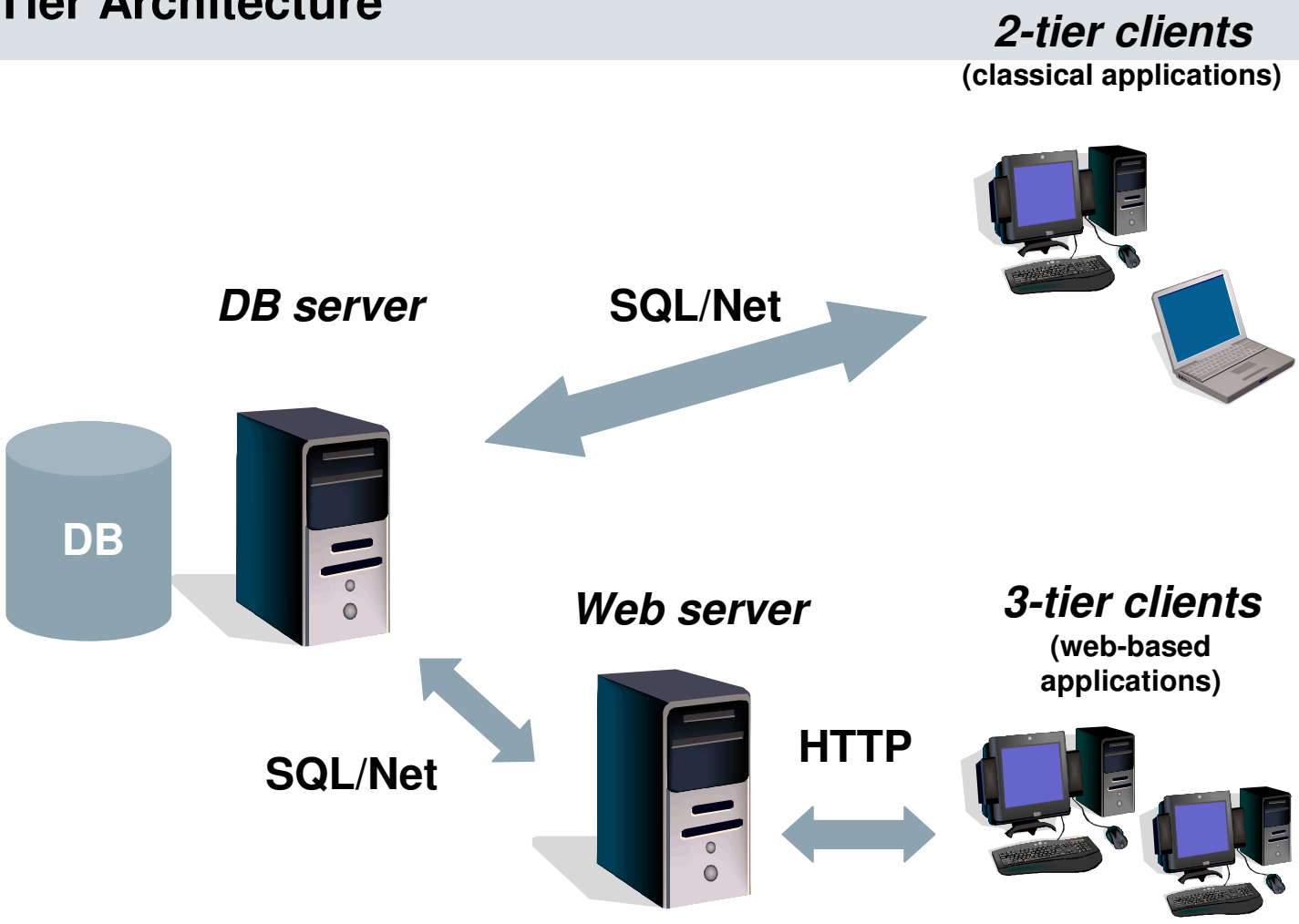


SIEMENS

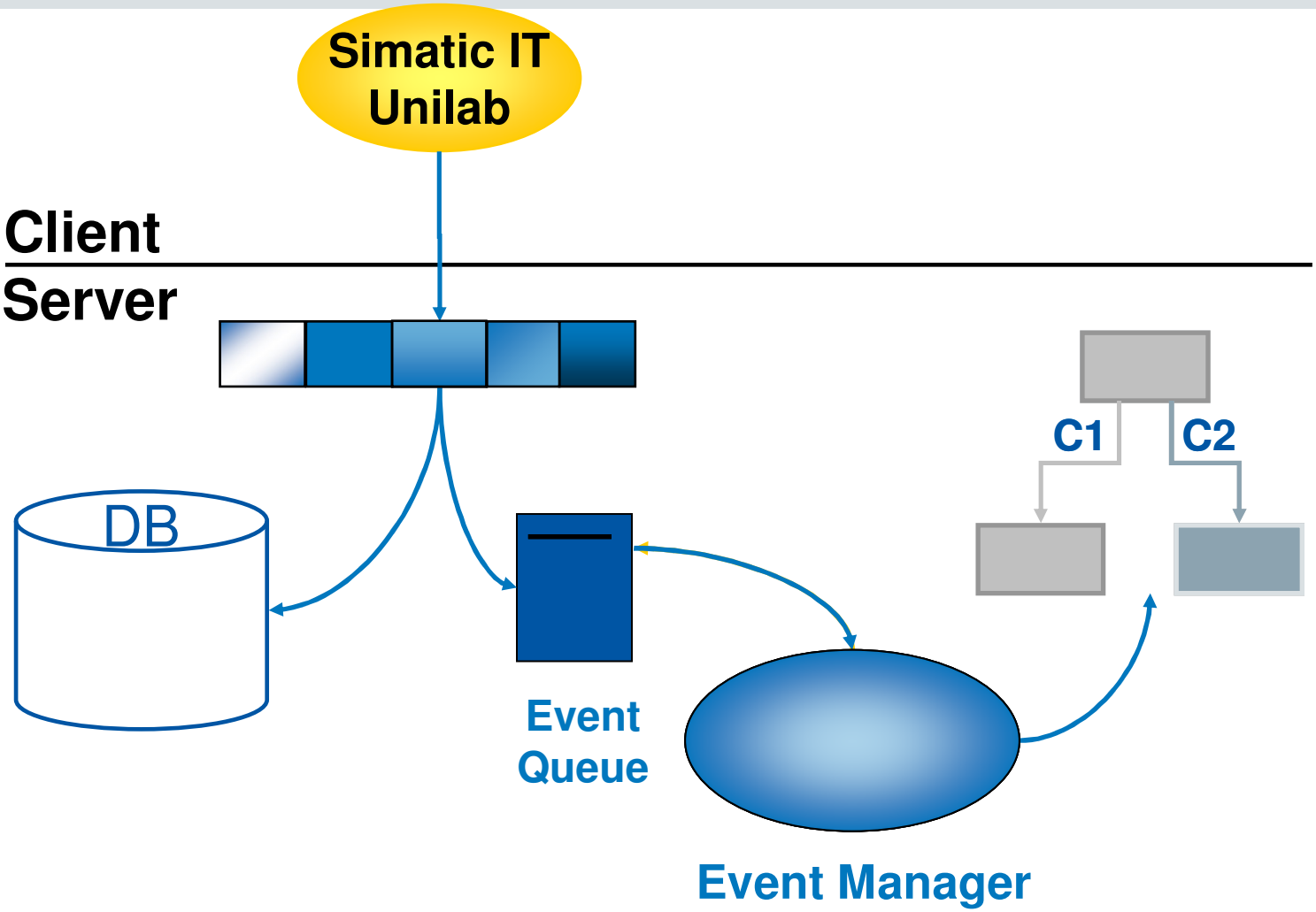
Database

Architecture

2 Tier vs. 3 Tier Architecture



Event Management



Event Manager Tables

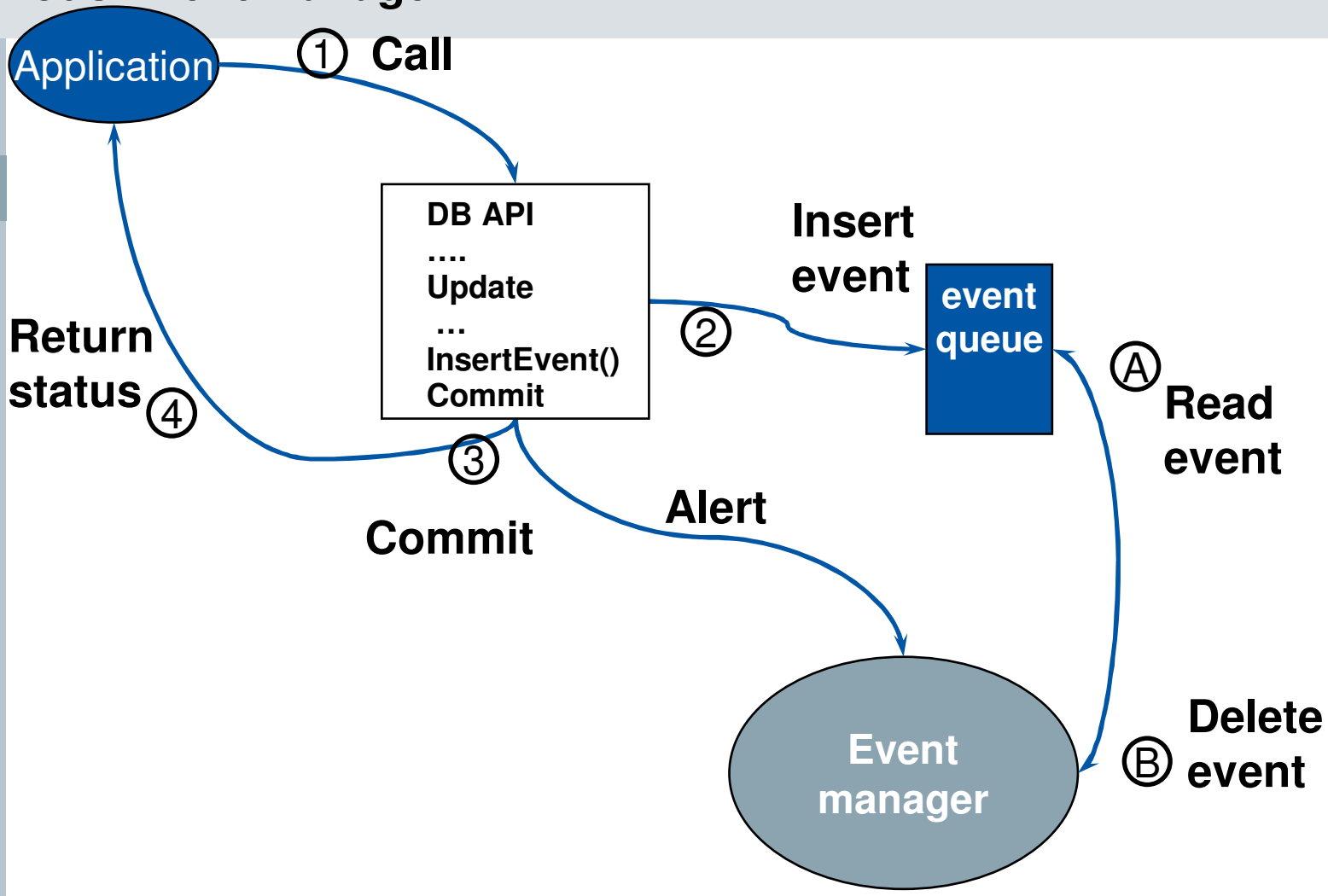
Events:

- utev
- utevtimed: timed events
- utevlog: logging in case of error

Client Event Manager

- utclient
- utclientalert

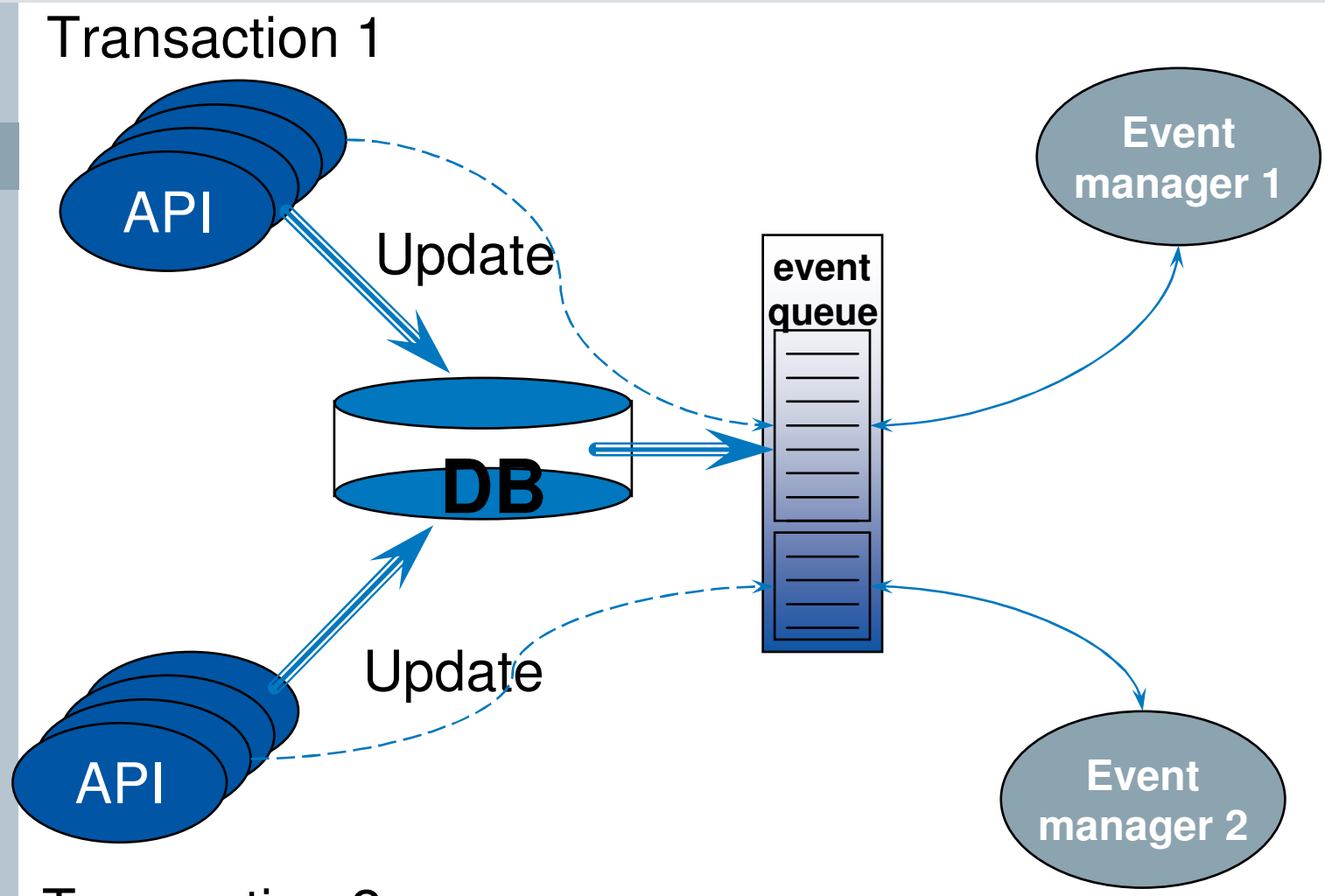
Asynchronous Event Manager



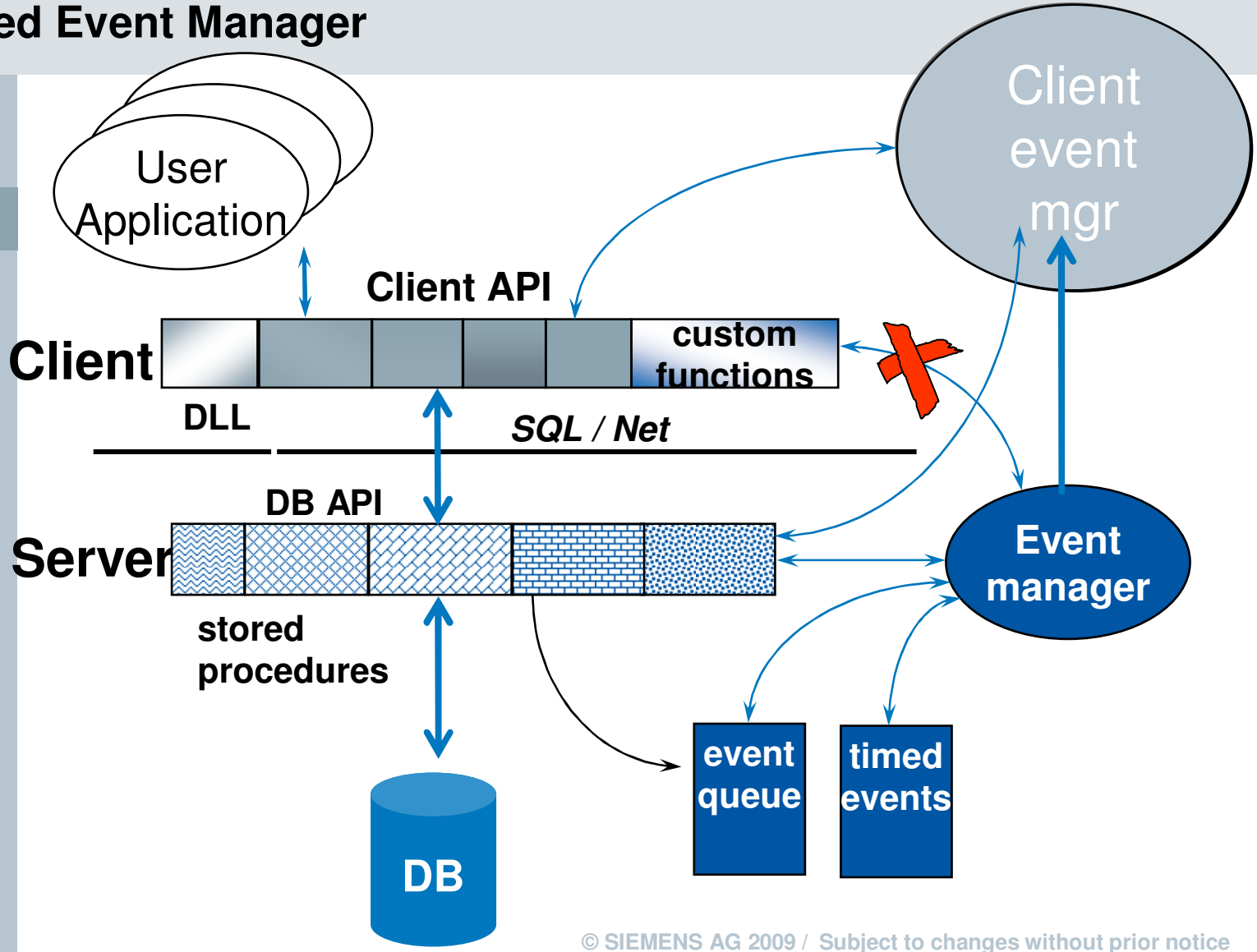
Approach to Process Event

- Step 1: Life cycle evaluation
- Step 2 : Change of status
  - First condition met
  - New status
  - New value for allow_modify and active flag
- Step 3: Evaluation of Worklist Assignment Rules
- Step 4: Execution of Actions

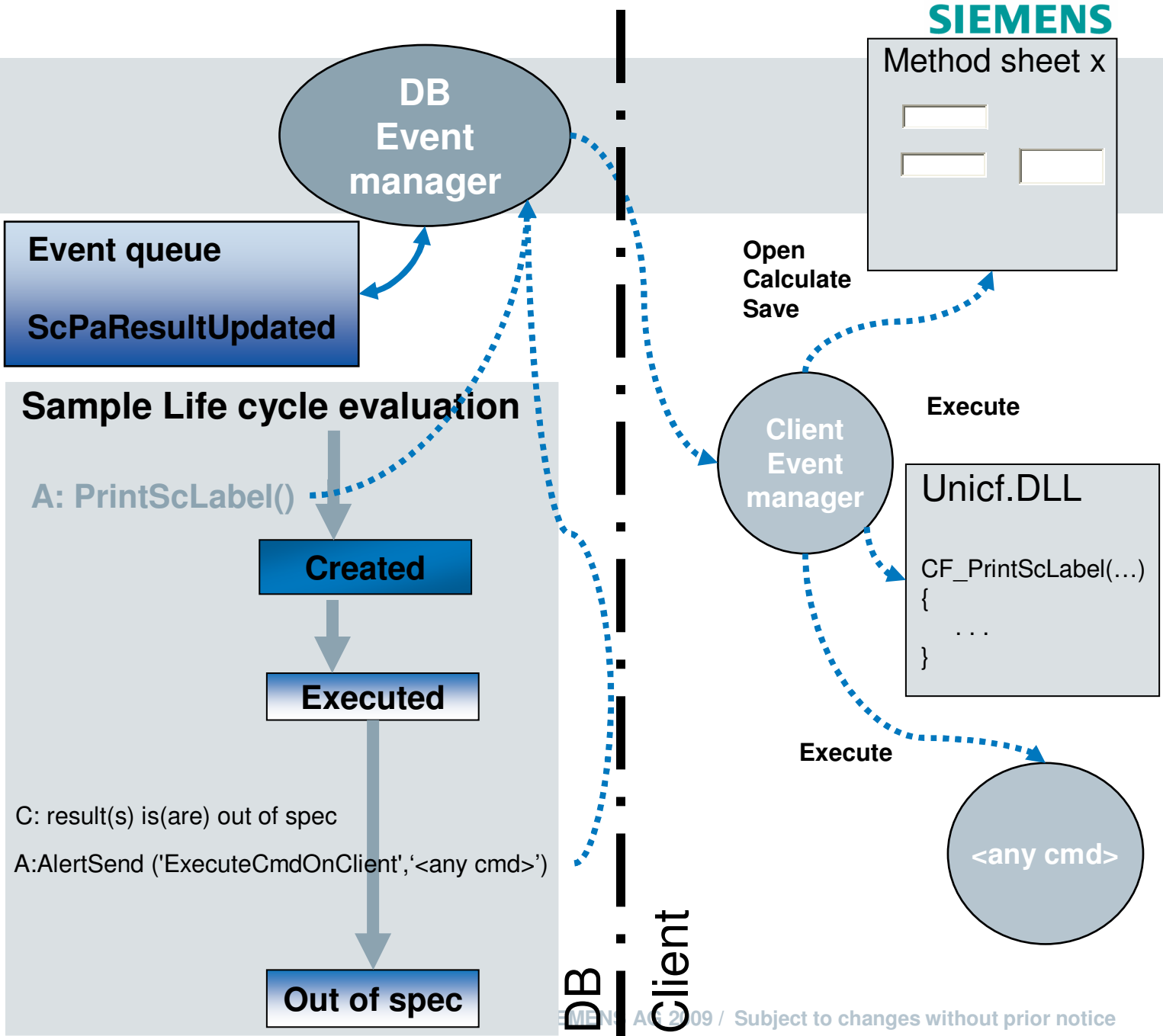
Multiple Event Managers



Distributed Event Manager



Example



Starting / Stopping Evmr

Sys.DBA_Scheduler_Jobs

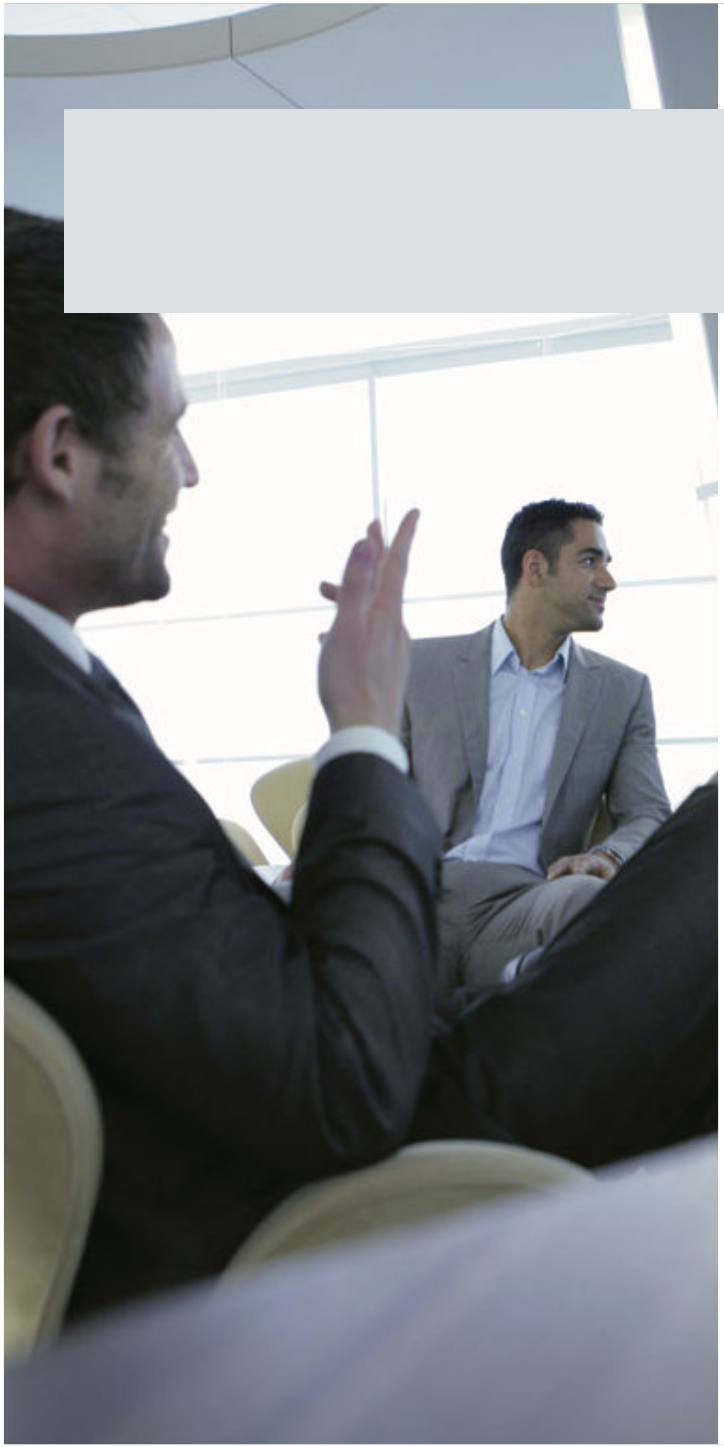
- All running event managers
  - Multiple event managers (multiple instances)
  - TimedEvent manager
  - Equipment manager job

Scripts

- Strtevmr.sql
  - Starts all event managers
- Stopevmr.sql
  - Stops all event managers

CXAPP

- Standard package to be used for start and stop jobs.
- ALWAYS USE THIS IN PROJECTS, also for project JOBS !!!



SIEMENS

Database APIs

Overview

DB API

DB Application Programming Interface

Set of Stored Procedures

- Implements main LIMS functionality
- Provides access to ALL Unilab data

Functions

- Encapsulate data access rights
- Encapsulate implicit joins,...
- Generation events
- Keeps track of full audit trail
- Restrictions on the number of rows returned

DB API General

API Structure
Return Codes
Arguments
Transaction handling

API Structure

APIs = set of Functions

- Return codes
- IN, OUT and IN OUT arguments
- Each function can be called independently

Documentation

- On-line help
- DBAPI.hlp, DBAPIOP.hlp and DBAPICON.hlp

Stored procedures grouped in logical packages

- Grouping of logical functions
- Optimisation of the implementation
- Information hiding
- Package = header + body
  - Header: available in *.h file or DB
  - Body: wrapped

Object Oriented Structure

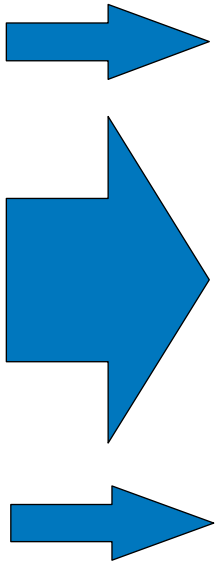
Package naming conventions

- Configuration objects:
  - unapi<xx> where <xx> = object type
  - unapiprp: generic functions
- Operational objects:
  - unapi<xx> and unapi<xx>p where <xx> = object type
  - unapiaut: generic functions
- Generic functions & declarations:
  - unapigen
- Custom packages:
  - un<xxxxx> where <xxxxx> = function type

Template API Function

Generic function definition

```
FUNCTION <MyFunc>
(arg1 IN    VARCHAR2,
 arg2 IN    NUMBER,
 ...
 argn OUT   VARCHAR2)
RETURN NUMBER
```



Naming convention

IN, OUT or
IN OUT arguments
(unbound data types)

Return code

Template API Function (2)

Calling the function

```
DECLARE

ret_code        INTEGER;
argn            VARCHAR2(20);

BEGIN

 l_ret_code := MyPack.MyFunc('first arg',
                             10,...
                             argn);

END;
```

Return code

Arguments
(bound data types)

Function call

DB API General

API Structure

Return Codes

Arguments

Transaction handling

Return Codes

Always NUMERIC value (l_ret_code)

Declared as constants in UNAPIGEN

ALWAYS USE THE CONSTANT DEFINITIONS

Error code classes:

- Generic
- Function specific
- Object specific

... Some common generic codes

- DBERR_SUCCESS (0)
  - Everything OK
- DBERR_GENFAIL (1)
  - Refer to uterror for error logging
- DBERR_NOOBJECT (2)
  - The object does not exist
- DBERR_NOTMODIFIABLE (5)
  - The object cannot be modified (allow_modify = '0')

Return Codes (2)

Most other return codes have common structure:

- DBERR_<*Column name*>
  - <*Column name*> stands for the column that has an incorrect value
  - E.g. DBERR_TEMPLATE, DBERR_PROTECTED, DBERR_OBJID

Return codes are NOT always Error codes

Using Return Codes

```
DECLARE
l_ret_code        NUMBER;
a_sc              VARCHAR2(20);
BEGIN
   l_ret_code := UNAPISC.CreateSample('Chicken', a_sc,
                                      SYSDATE, '', '',
                                     'Training example');
   IF l_ret_code <> UNAPIGEN.DBERR_SUCCESS THEN
      /* Handle the error */
   END IF;
END;
```

General Failure

Return codes often reflect logical or data error

- The calling application should provide corrective actions based upon the return code
- There is no specific error logging

General Failure

- Fatal error occurred in the DB (e.g. missing table, incorrect statement, wrong user privileges)
- Return code = DBERR_GENFAIL (1)
- It is usually followed by proper logging in uterror

General Failure (2)

Entry in uterror:

(client id, application name, user id, logdate, api_name, error message)

Example exception handler

```
EXCEPTION
   WHEN OTHERS THEN
      l_sqlerrm := SQLERRM;
      ROLLBACK;
      INSERT INTO uterror(client_id, applic, who, logdate, api_name, error_msg)
      VALUES(UNAPIGEN.P_CLIENT_ID, UNAPIGEN.P_APPLIC_NAME, USER, SYSDATE,
             'MyFunc', l_sqlerrm);
      COMMIT;
      RETURN(UNAPIGEN.DBERR_GENFAIL);
END MyFunc;
```

General Failure (3)

Use of LogError()

```
PROCEDURE UNAPIGEN.LogError                /* INTERNAL */
(a_api_name      IN        VARCHAR2,       /* VC40_TYPE */
 a_error_msg     IN        VARCHAR2)       /* VC255_TYPE */
```

Example

```
EXCEPTION
   WHEN OTHERS THEN
      IF sqlcode <> 1 THEN
         UNAPIGEN.LogError('MyFunc', sqlerrm);
      END IF;
      RETURN(UNAPIGEN.DBERR_GENFAIL);
END MyFunc;
```

DB API General

Argument Data Types (2)

Simple arguments

- Always pass bound argument!
  - Directly (IN)

```
ret_code := UNAPISC.CreateSample('Chicken meat', '9821_02',...);
```

  - Using variable (IN, OUT and IN OUT)

```
a_st     VARCHAR2(20);
...
a_st := 'Chicken';
ret_code := UNAPISC.CreateSample(a_st, a_sc,...);
```

Argument Data Types (3)

Array arguments

- Definitions in UNAPIGEN
  - Format of array definition:

```
TYPE <Tp><Sz>_TABLE_TYPE IS TABLE OF <Tp>(<Sz>)
  INDEX BY BINARY_INTEGER;
```

  - <Tp> represents the single data type
    - All used single data types have an array equivalent
  - <Sz> represents the data size
  - Example

```
TYPE CHAR1_TABLE_TYPE IS TABLE OF CHAR(1)INDEX BY
BINARY_INTEGER;
```

Argument Data Types (4)

Type	<Tp>	Values for <Sz>
CHAR	CHAR	1,2
VARCHAR2	VC	2,3,4,8,20,40,60,255,1000,2000
NUMBER	NUM	N/A
FLOAT	FLOAT	N/A
DATE	DATE(*)	N/A
RAW	RAW	8
LONG	LONG	N/A

(*) DATE array is equivalent to VC30_TABLE_TYPE

Argument Data Types (5)


```
DECLARE
l_ret_code                    INTEGER;
l_row                         INTEGER;
l_nr_of_rows                  NUMBER;
l_where_clause                VARCHAR2(255);
l_next_rows                   NUMBER;
l_mt_tab                      UNAPIGEN.VC20_TABLE_TYPE;
l_version_tab                 UNAPIGEN.VC20_TABLE_TYPE;
l_version_is_current_tab      UNAPIGEN.CHAR1_TABLE_TYPE;
l_effective_from_tab          UNAPIGEN.DATE_TABLE_TYPE;
l_effective_till_tab          UNAPIGEN.DATE_TABLE_TYPE;
l_description_tab             UNAPIGEN.VC40_TABLE_TYPE;
a_ss_tab                      UNAPIGEN.VC2_TABLE_TYPE;

l_nr_of_rows := NULL;
BEGIN
   l_ret_code := UNAPIMT.GetMethodList
                    (l_mt_tab,
                     l_version_tab, l_version_is_current_tab,
                     l_effective_from_tab, a_effective_till_tab
                     l_description_tab, l_ss_tab,
                     l_nr_of_rows, '', 0);
   FOR i IN 1..a_nr_of_rows LOOP
                    DBMS_OUTPUT.PUT_LINE(a_mt_tab(i));
   END LOOP;
END;
```

Passing Arguments

Always pass ALL arguments

- For single arguments
  - use `NULL` or `''`
- For array arguments
  - initialize with `''`

DB API General

Transaction control

Function types

- Get... functions
  - Read-only access to data
- Save, Create, Delete, etc. functions
  - Read-write access to data
  - Require transaction handling

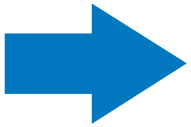
Transaction control is provided by DB APIs

- APIs use Oracle COMMIT or ROLLBACK
- The user does not have to use COMMITs nor ROLLBACKs

Transaction Control (2)

Logical transactions

- Each function = logical discrete transaction, except if it is part of a bigger transaction
  - Function does COMMIT or ROLLBACK at the end
- User defined transactions
  - BeginTransaction()
  - Several functions
  - EndTransaction()



An API that is part of a multi-statement transaction does NOT perform COMMIT or ROLLBACK !

Transaction Control (3)

BeginTransaction()

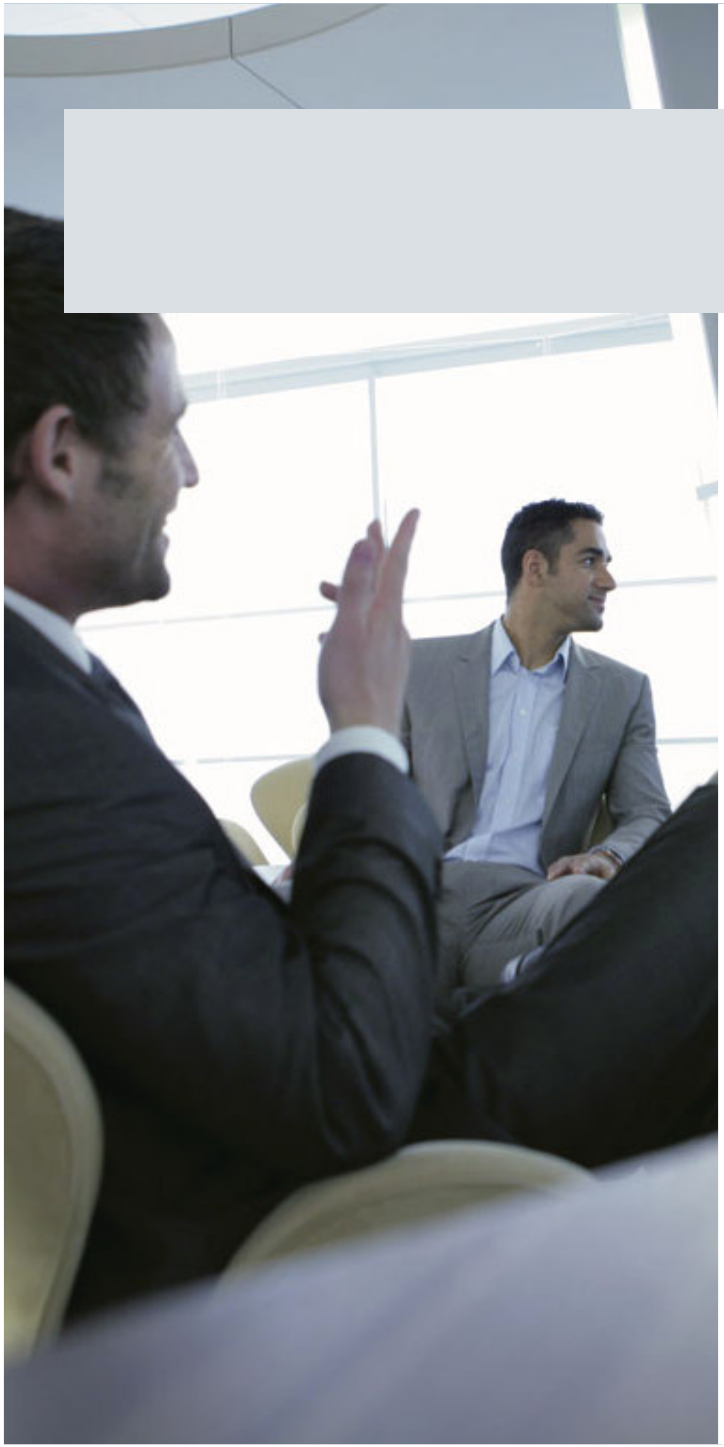
- Initialise transaction variables
- Returns error if previous transaction not ended

EndTransaction()

- If error occurred: ROLLBACK, otherwise: COMMIT
- Resets transaction variables
- Returns (last) error code

Example

```
...
ret_code := UNAPIGEN.BeginTransaction();
ret_code := UNAPIPA.SaveScParameter(...);
ret_code := UNAPIPAP.SaveScPaAttributes(...);
ret_code := UNAPIGEN.EndTransaction(...);

IF ret_code <> UNAPIGEN.DBERR_SUCCESS THEN
   display appropriate error message
END IF;
```



SIEMENS

Database APIs

Generic APIs

Connection Handling

Before using the API

- Connect to Oracle
  - Perform Oracle CONNECT
    - From client (e.g. call DBConnect())
    - On server (e.g. start SQL*Plus)
  - Uses Oracle authorisation
- Setup a connection to Unilab
  - Call SetConnection()
    - Part of DB-API (UNAPIGEN)
  - Performs Unilab authorisation

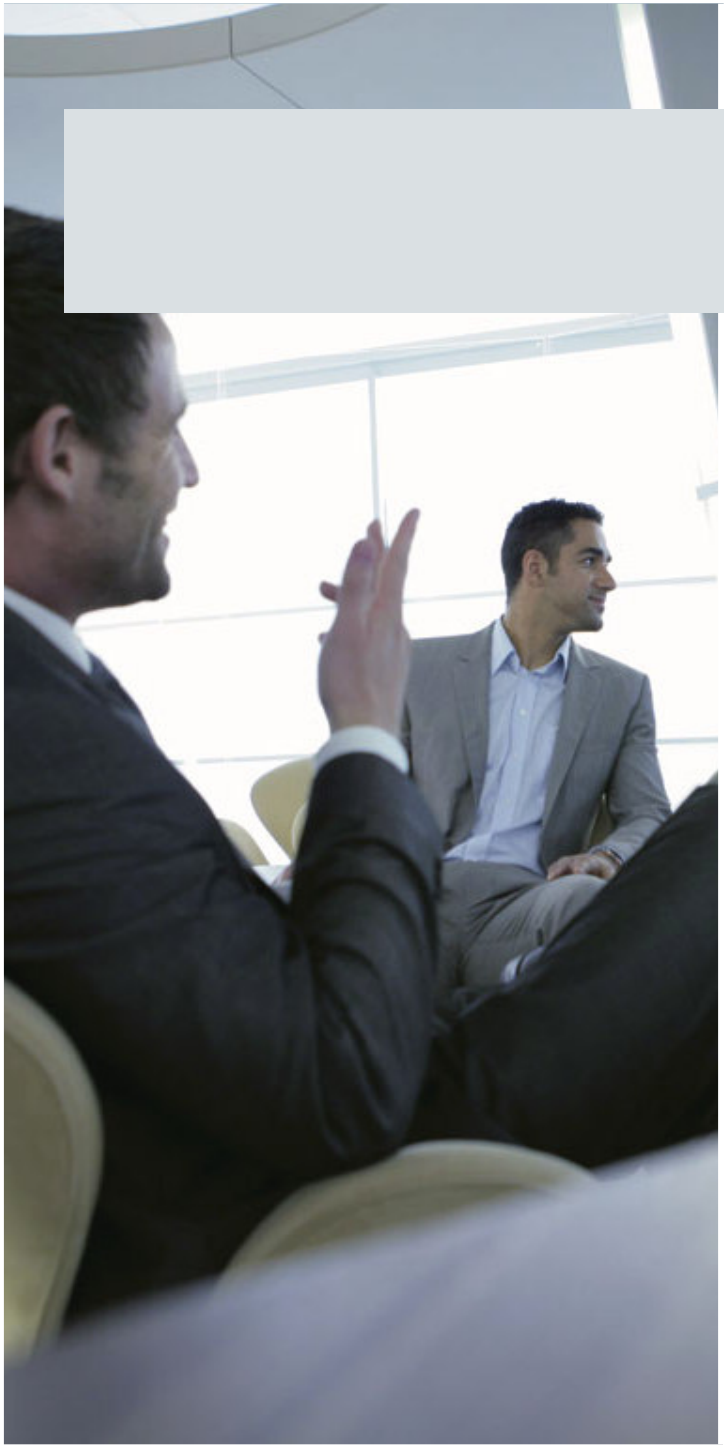
SetConnection()

- Initialises session variables
  - Date format
  - Internal settings (e.g. used for error logging)
- Checks user access rights
- Returns global user settings
  - Default user profile (+language)
  - Default task (if appropriate)

Exercise

Exercise 1

- Set Connection as DBA
- Display the following OUT parameters
  - UP
  - UP_DESCRIPTION

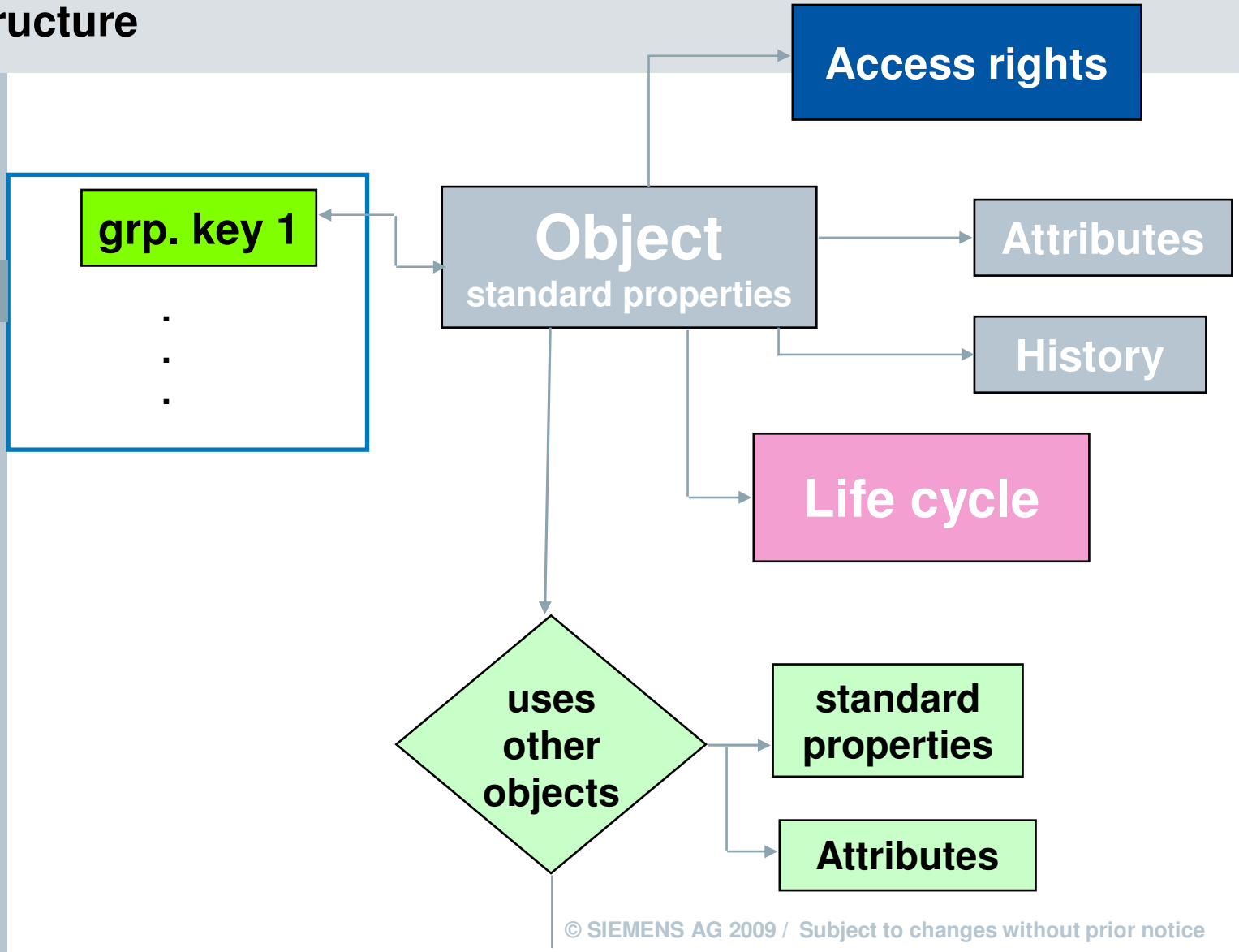


SIEMENS

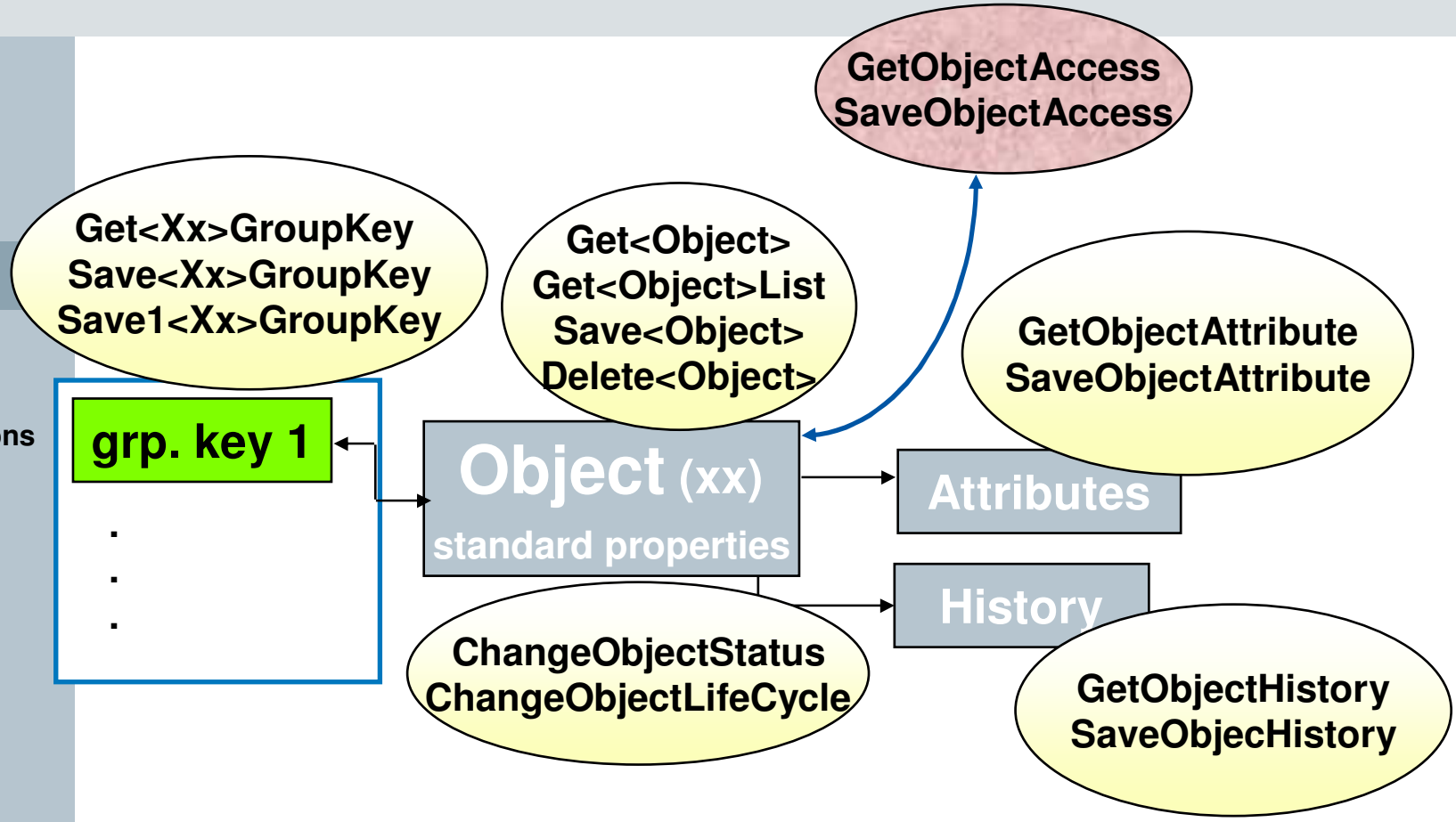
Database APIs

Configuration APIs

Object structure



Configuration APIs



Configuration APIs (2)

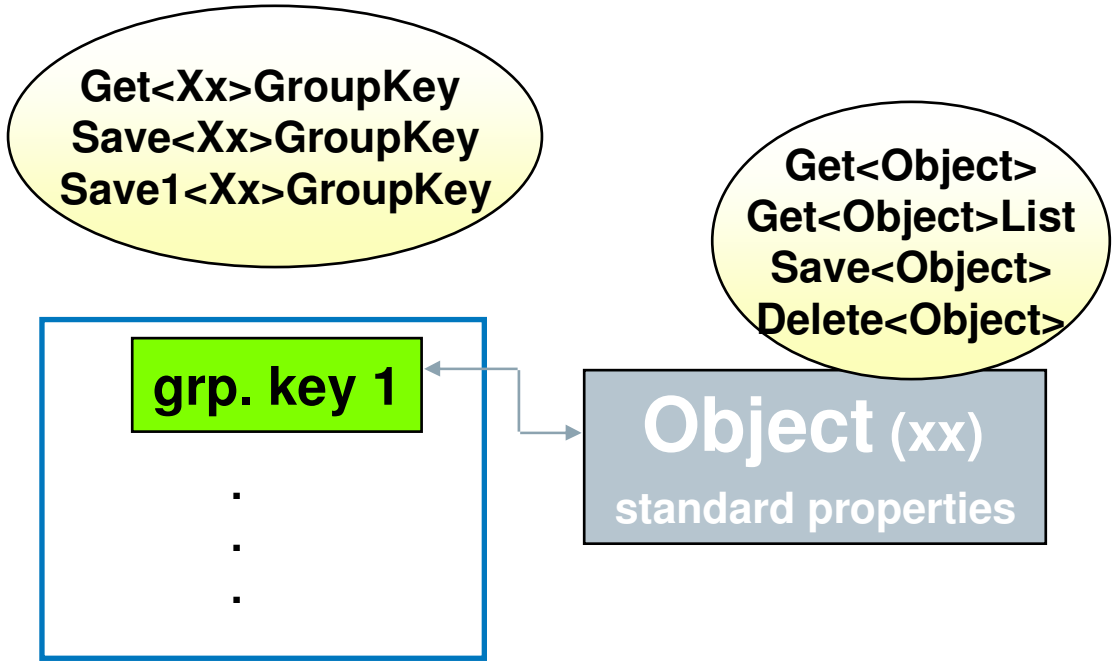
Naming convention valid for

- Main configuration objects
  - st, pp, pr, mt, ip, ie
- Other objects
  - ad, au, lc, uc, up,...

Packages

- Object specific functions in UNAPI<Xx> package
- Generic functions in UNAPIGEN and UNAPIPRP

Object specific API's



Stored in UNAPI<XX>

Get... vs Get...List

Get<Object>

- Gets all standard properties of object
- All objects as specified in a_where_clause

Get<Object>List

- Only object id, description and status
- All objects as specified in a_where_clause
- Allows get...next

Get<Object>

Generic function header

```
FUNCTION UNAPIST.GetSampleType
(a_st                   OUT     UNAPIGEN.VC20_TABLE_TYPE,
 a_version              OUT     UNAPIGEN.VC20_TABLE_TYPE,
 a_version_is_current   OUT     UNAPIGEN.CHAR1_TABLE_TYPE,
 a_description          OUT     UNAPIGEN.VC40_TABLE_TYPE,
 a_description2         OUT     UNAPIGEN.VC40_TABLE_TYPE,
 a_is_template          OUT     UNAPIGEN.CHAR1_TABLE_TYPE,
 ...
 a_allow_modify         OUT     UNAPIGEN.CHAR1_TABLE_TYPE,
 a_active               OUT     UNAPIGEN.CHAR1_TABLE_TYPE,
 a_lc                   OUT     UNAPIGEN.VC2_TABLE_TYPE,
 a_ss                   OUT     UNAPIGEN.VC2_TABLE_TYPE,
 a_nr_of_rows           IN OUT  NUMBER,
 a_where_clause         IN      VARCHAR2)
RETURN NUMBER
```

Get<Object> (2)

a_where_clause

- Object ID
  - E.g. `'pH', 'Moisture'`
- Complete WHERE clause
  - E.g. `'WHERE is_template = ''1'' ORDER BY mt'`
  - Use all columns of the main object table
- `NULL`
  - Default set to return all 'active' objects, ordered alphabetically by object ID

a_nr_of_rows

- IN: Maximum expected number of rows
  - Limit on the rows returned
  - Most often = 1
  - If zero, default provided
- OUT: Real number of rows returned
  - <= IN value
  - UNDEFINED if ret_code <> DBERR_SUCCESS
- NO possibility to fetch the remaining rows!

Get<Object>List

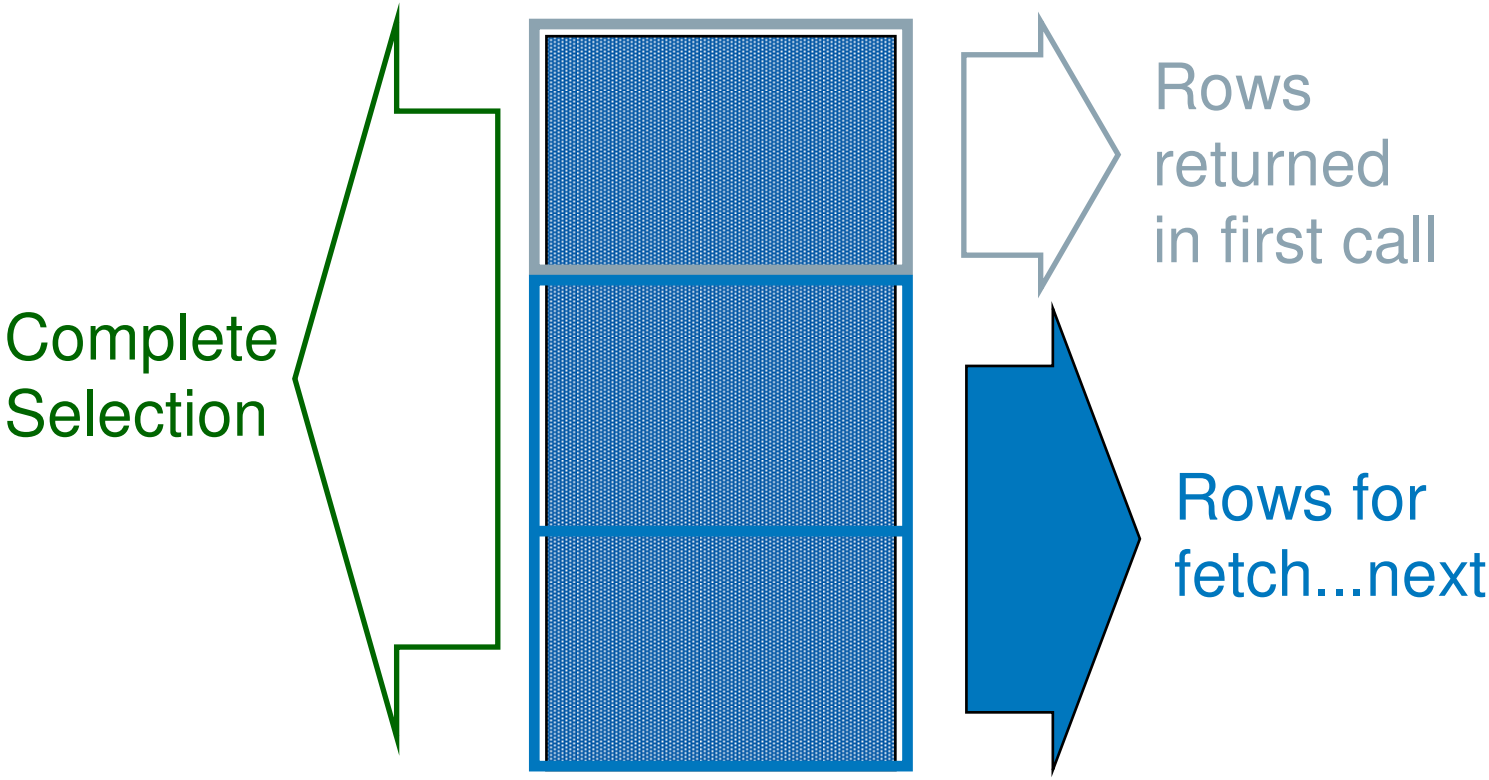
Generic function header:

```
FUNCTION UNAPIST.GetSampleTypeList
(a_st                    OUT      UNAPIGEN.VC20_TABLE_TYPE,
 a_version               OUT      UNAPIGEN.VC20_TABLE_TYPE,
 a_version_is_current    OUT       UNAPIGEN.CHAR1_TABLE_TYPE,
 a_effective_from        OUT       UNAPIGEN.DATE_TABLE_TYPE,
 a_effective_till        OUT       UNAPIGEN.DATE_TABLE_TYPE,
 a_description           OUT      UNAPIGEN.VC40_TABLE_TYPE,
 a_ss                    OUT      UNAPIGEN.VC2_TABLE_TYPE,
 a_nr_of_rows            IN       OUT NUMBER,
 a_where_clause          IN       VARCHAR2,
 a_next_rows             IN       NUMBER)
RETURN NUMBER;
```

a_where_clause and a_nr_of_rows as in Get<Object>

Get<Object>List (3)

Fetch...next allows to fetch the remaining rows



Get<Object>List (4)

First call

- a_next_rows = 0
- New a_where_clause

Next calls

- a_next_rows = 1
- Same a_where_clause (arg not used)
- If all rows returned:
  - a_nr_of_rows OUT <= IN
  - Cursor closed automatically
- If no more rows found
  - ret_code = DBERR_NORECORDS
  - OUT arguments UNDEFINED

Explicitly closing the cursor

- a_next_rows = -1
- No error if cursor was already closed
- OUT arguments are UNDEFINED
- When only one call must be performed, it must be done with a_next_rows=-1.

The cursor has to be always closed when finished

Exercise

Exercise 2

- Fetch all data for the sample type with the *description* 'EA Pizza Cheesy Crust'
  Display the sample type, description, version and status.

Exercise 3

- Get the list of all sample types ordered by sample type
- Display the sample type, description, version and status.

Exercise 3b

- Get this list in chunks of 5

Save<Object>

SAVES the standard properties of one specific object.

- All properties, except
  - ss
  - allow_modify
  - active
  - access right
- Valid values (+required columns) are defined by object type
  - Refer to table definitions

INSERTS if new object

- Initialisation is done by the Event Manager
  - Life cycle if not specified
  - Status
  - Access rights

UPDATES if existing object

- If update is allowed
- Life cycle handling is done by the Event Manager

Delete<Object>

Deletes an object permanently

- If allowed
- Object has to be non-active
- All references are removed as well

Generic function header

```
FUNCTION UNAPI<XX>.Delete<Object>
  (a_<xx>                IN  VARCHAR2,
  a_modify_reason        IN  VARCHAR2)
RETURN NUMBER;
```

Version Management

Configuration objects where version control is implemented

- rt, st, pp, pr, mt, ip, ie, au, wt, cy, pt

Objects where version column is present but not yet implemented

- gk, eq, lc, ad, up, uc

Objects always have a version.

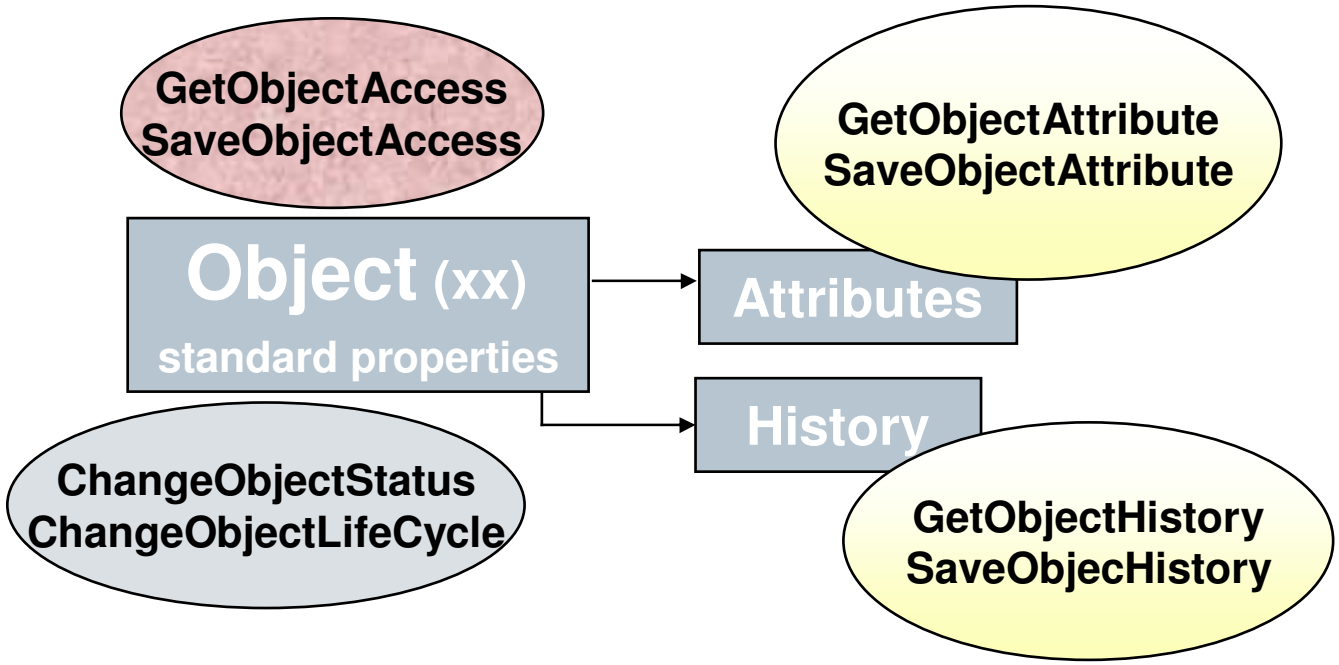
- Objects with version management
  - UNVERSION.P_INITIAL_VERSION
- Objects without version management
  - UNVERSION.P_NO_VERSION

Exercise

Exercise 4

- Create a new sample type
- Do not forget to fill in the description
- Sample Lifecycle : LS
- Do not forget to specify the correct SC_LC_VERSION
- Object Lifecycle : @L
- Do not forget to specify the correct LC_VERSION
- Modify Reason : ‘Exercise 4’

Generic Configuration APIs



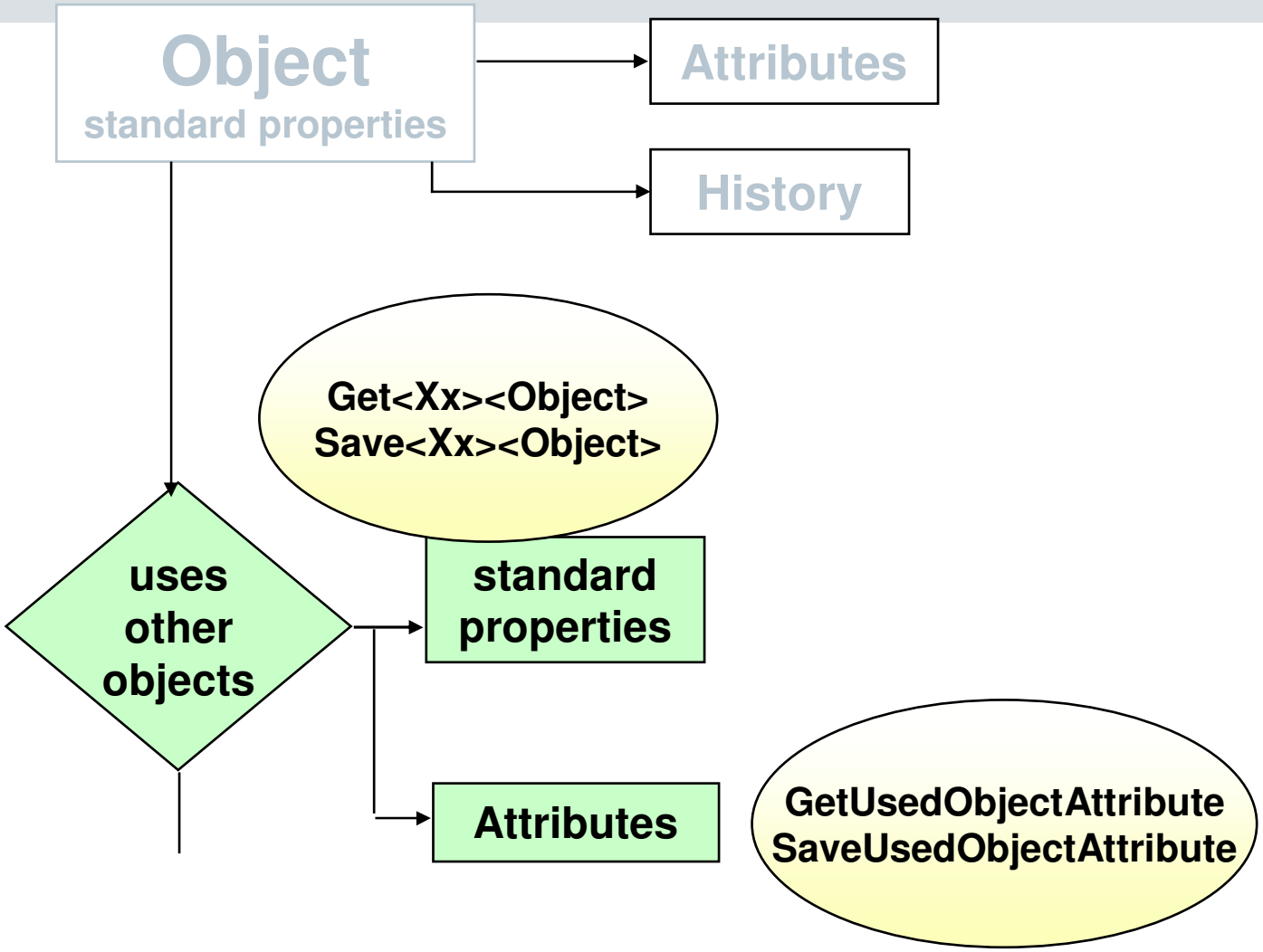
Generic Functions

- Valid for all configuration objects
  - Object type = <xx>
- Functions stored in UNAPIPRP and UNAPIGEN

Exercise 5

- Set the status of the new sample type to ‘Approved’ (@A)
- Modify Reason : ‘Exercise 5’

Configuration APIs – Used objects



Configuration APIs – Used objects (2)

Packages

- Object specific functions in UNAPI<Xx> package with Xx = main object
- Generic functions in UNAPIGEN and UNAPIPRP

Get<Xx><Object>

Get the list of used objects for a main object and their standard properties

- Returns all std. properties of the used objects
- E.g. GetPrMethod

```
FUNCTION UNAPI<Xx>.Get<Xx><Object>
  (a_<xx>         OUT              UNAPIGEN.VC20_TABLE_TYPE,
  a_<yy>          OUT              UNAPIGEN.VC20_TABLE_TYPE,
  ...
  a_nr_of_rows    IN OUT           NUMBER,      /* NUM_TYPE    */
  a_where_clause  IN               VARCHAR2)    /* VC255_TYPE */
RETURN NUMBER;
```

Get<Xx><Object> (2)

a_where_clause

- Object ID
  - E.g. ‘pH’, ‘Moisture’ => ‘ORDER BY seq’
- Complete WHERE clause
  - E.g. ‘WHERE mt like ‘‘M%’’ ORDER BY mt, seq’
  - Use all columns of the used object table
- NULL
  - Default set to return all ‘used’ objects assigned to all objects, ordered by object ID, seq

Save<Xx><Object>

Save the list of used objects for a main object

- Deletes existing entries (!!) and inserts new list
- Is in charge of the sequence numbering
- E.g. SavePrMethod

```
FUNCTION UNAPI<Xx>.Save<Xx><Object>
 (a_<xx>                IN   VARCHAR2,    /* VC20_TYPE */
  a_<yy>                IN   UNAPIGEN.VC20_TABLE_TYPE,
  ...
  a_nr_of_rows          IN   NUMBER,
  a_modify_reason       IN   VARCHAR2)   /* VC255_TYPE */
RETURN NUMBER;
```

Group Key Functions

GetStGroupKey

- Gets the list of group keys assigned to a group of sample types and their values
- Similar to GetUsedObjectAttribute

GetGroupKeySt

- Gets the details of one or more sample type group keys
- Similar to Get<Object>

SaveGroupKeySt

- Saves the definition of a group key for sample types
- Similar to Save<Object>

SaveStGroupKey

- Saves the list of group keys and their values assigned to a single sample type
- Similar to SaveObjectAttribute

Save1StGroupKey

- Saves the values for 1 group key assigned to 1 sample type

Version Management (Part 2)

Questions to ask yourself when creating a new version

- Which version will the next version be based on
  - The current or last version
  - UNVERSION.GetHighestMinorVersion('st', 'waste_water', l_version)
  - UNVERSION.GetHighestMajorVersion ('st', 'waste_water', l_version)
- Is it a minor or a major version upgrade
  - UNVERSION.GetNextMinorVersion(l_version)
  - UNVERSION.GetNextMajorVersion(l_version)

Assigning objects to an other object

- On Each link you have to define the version you want to use
  - ~Current~ : Unilab will always use the Current object.
  - 0001.* : Unilab uses the last minor version for the specified version.
  - 0001.01 : Unilab uses a fixed version.

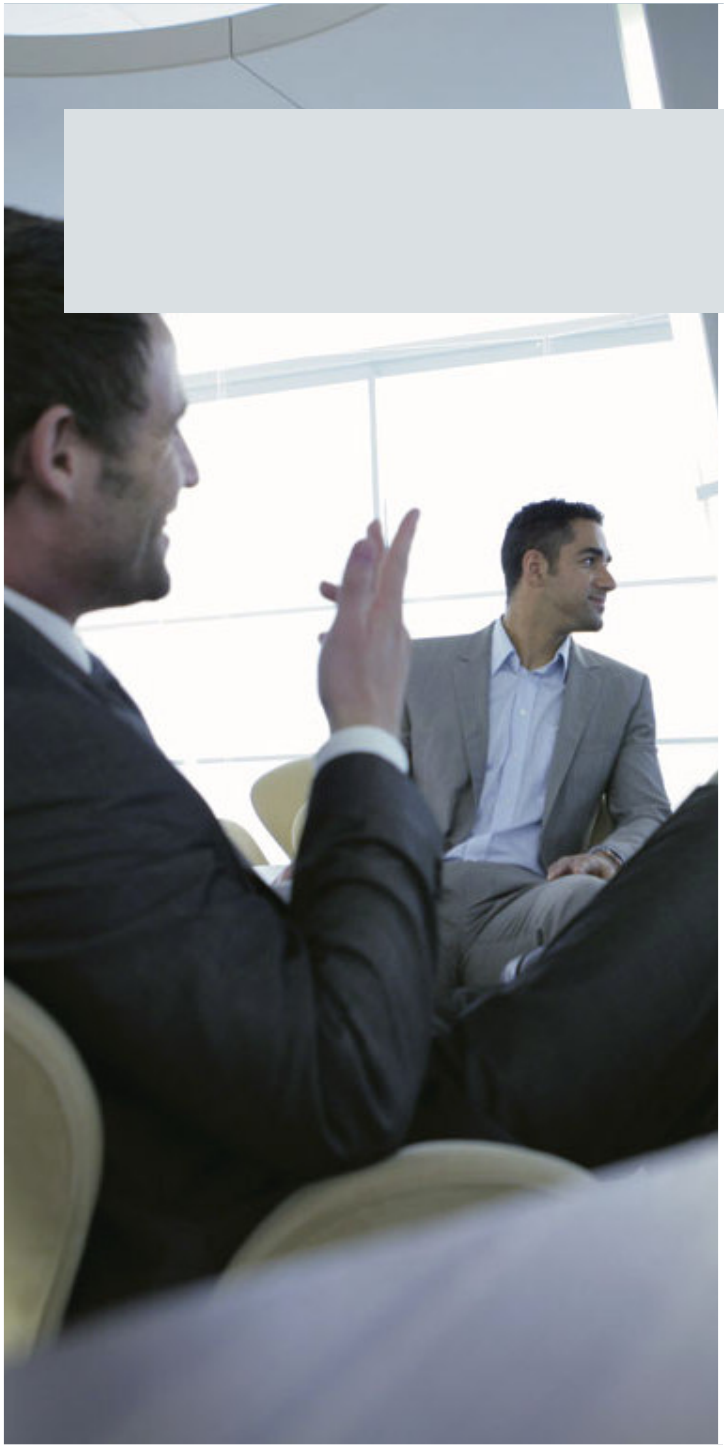
Exercise

Exercise 6

- Create a new sample type version
- Assign the parameter profiles with the *descriptions* ‘Anions’ and ‘Calibration’ to the new sample type.
- Use next_rows = -1 when performing one call.

Exercise 7

- Assign the group key with *description* ‘Product Class’ with the value ‘finished product’ to your new sample type
- GK Version is ignored, but it is best to implement this for future use.



SIEMENS

Database APIs

Operational APIs

Nodes vs Sequences

Common use

- Ordering in list
- Distinction between multiple assignments of same object

Configuration objects: sequences

- List of used objects ordered by seq
- No explicit identification of used object
- Sequence number only internally to DB-API

Operational objects: nodes

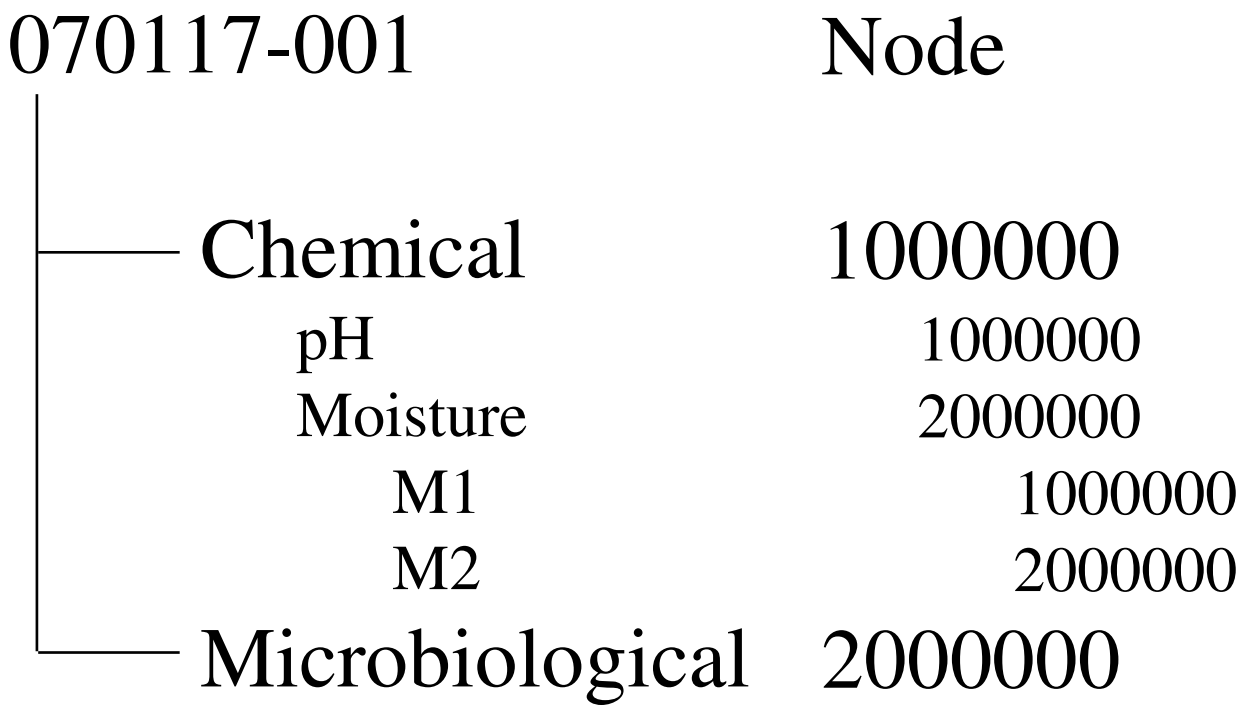
- Each object is a ‘used’ object
- Multi-level hierarchy (sc-pg-pa-me)
  - No re-sequencing !
  - Composite node numbers

Key to each object = id(s) + node(s)

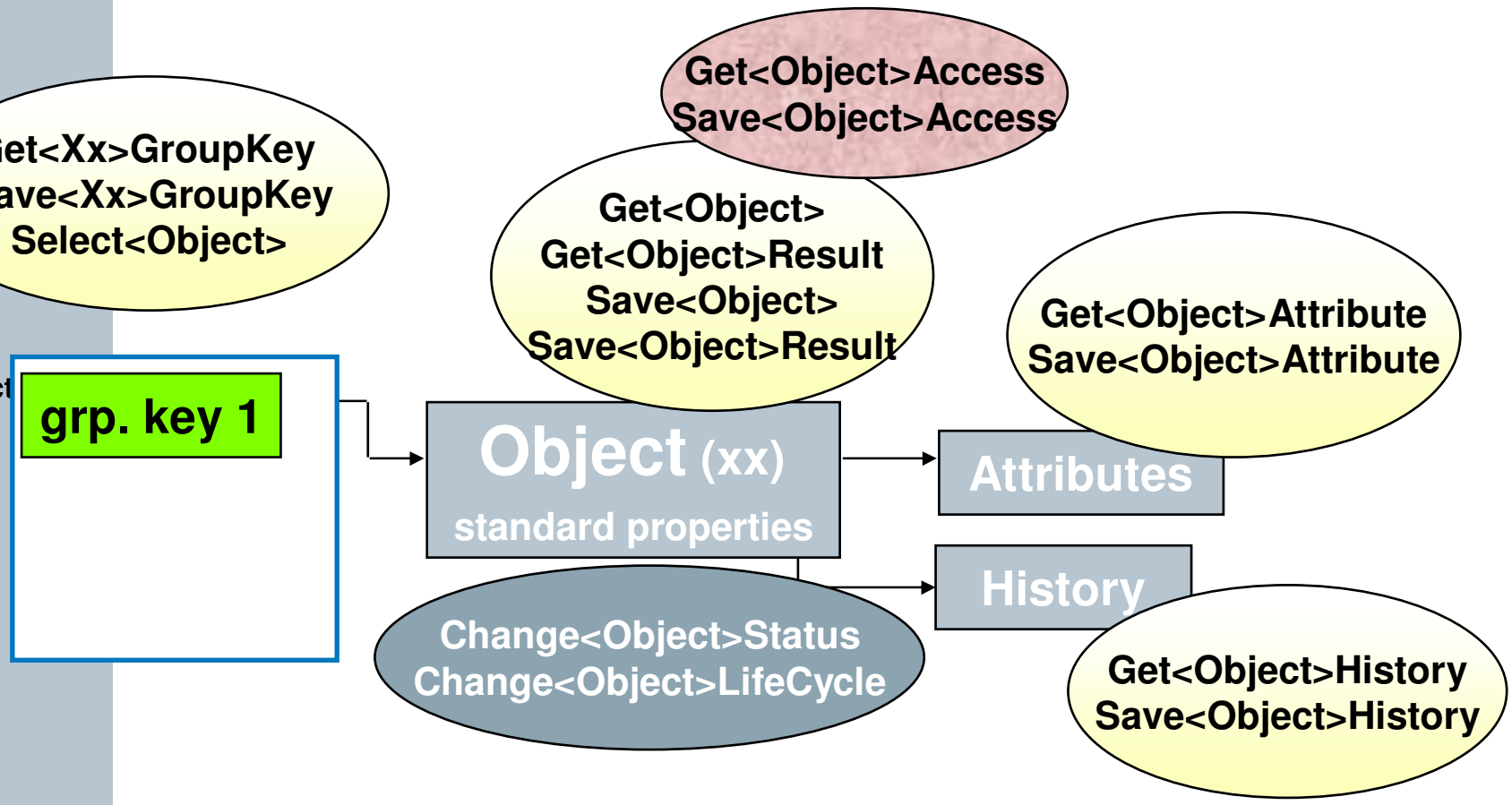
- E.g. operational parameter
  - sc + pg + pgnode + pa + panode
- E.g. operational info field
  - sc + ic + icnode + ii + iinode

Node Numbering

Example



Operational APIs



Operational APIs (2)

Naming conventions

- All operational objects
  - <xx> = sc, scpg, scpa, scme, scic, scii
  - <Object> = Sample, ScMethod,...

Packages

- Object specific functions in:
  - UNAPI<XX> : main functions
  - UNAPI<XX>P : extra functions
- Generic functions in UNAPIAUT

Get<Object>

Get<Object>

- Gets all object details
- All objects as specified in a_where_clause
- E.g. GetScMethod may result in list of methods for one sample
- Access rights will influence the result !!!

There is NO Get<Object>List

Get<Object>Result

- Returns only result (if relevant)

Get<Object> (2)

Generic function header:

```
FUNCTION Get<Object>
(a_sc                OUT       UNAPIGEN.VC20_TABLE_TYPE,
 a_pg                OUT       UNAPIGEN.VC20_TABLE_TYPE,
 a_pgnode            OUT       UNAPIGEN.LONG_TABLE_TYPE,
 ...
 a_allow_modify      OUT       UNAPIGEN.CHAR1_TABLE_TYPE,
 a_active            OUT       UNAPIGEN.CHAR1_TABLE_TYPE,
 a_lc                OUT       UNAPIGEN.VC2_TABLE_TYPE,
 a_ss                OUT       UNAPIGEN.VC2_TABLE_TYPE,
 a_nr_of_rows        IN OUT    NUMBER,              /* NUM_TYPE */
 a_where_clause      IN        VARCHAR2)            /* VC255_TYPE */
RETURN NUMBER;
```

Get<Object> (3)

a_where_clause

- Sample code
  - 'WHERE sc = ... ORDER BY all nodes'
- Complete WHERE clause
  - E.g. 'WHERE sc = '050117-001'
    AND pg = 'Chemical'
    ORDER BY panode'
  - Use all columns of the object table
- NULL !
  - Fetches the entire list

a_nr_of_rows

- IN: Maximum expected number of rows
  - Limit on the rows returned
  - Most often = 1
  - If zero, default provided
- OUT: Real number of rows returned
  - <= IN value
  - UNDEFINED if ret_code <> DBERR_SUCCESS
- NO possibility to fetch the remaining rows !

Exercise

Exercise 8

- Fetch all data for all samples of the sample type *description* 'EA Pizza Cheesy Crust'

Exercise 9

- Get all parameters for the last sample of exercise 8
- Display parameter, description, value_s and status.

Get... vs Select... Functions

Requests, samples and (operational) methods have group keys

- Select... returns same data (columns) as Get...
- Select... restricts selection based on group key selection
  Get... uses where_clause
- Select... only allows order_by_clause

Select<Object>

Generic function header

```
FUNCTION SelectScMethod
(a_col_id                IN       UNAPIGEN.VC20_TABLE_TYPE,
 a_col_tp                IN       UNAPIGEN.VC2_TABLE_TYPE,
 a_col_value             IN       UNAPIGEN.VC40_TABLE_TYPE,
 a_col_nr_of_rows        IN       NUMBER,
 a_sc                    OUT      UNAPIGEN.VC20_TABLE_TYPE,
 a_pg                    OUT      UNAPIGEN.VC20_TABLE_TYPE,
 a_pgnode                OUT      UNAPIGEN.LONG_TABLE_TYPE,
 ...
 a_allow_modify          OUT      UNAPIGEN.CHAR1_TABLE_TYPE,
 a_active                OUT      UNAPIGEN.CHAR1_TABLE_TYPE,
 a_lc                    OUT      UNAPIGEN.VC2_TABLE_TYPE,
 a_ss                    OUT      UNAPIGEN.VC2_TABLE_TYPE,
 a_nr_of_rows            IN OUT   NUMBER,     /* NUM_TYPE */
 a_order_by_clause       IN       VARCHAR2)   /* VC255_TYPE */
RETURN NUMBER;
```

Get<Object>Result

Only for methods and parameters

Get the result of a method/parameter

```
FUNCTION GetScMeResult
(a_sc              IN       VARCHAR2,        /* VC20_TYPE */
 a_pg              IN OUT   VARCHAR2,        /* VC20_TYPE */
 a_pgnode          IN OUT   NUMBER,          /* LONG_TYPE */
 a_pa              IN OUT   VARCHAR2,        /* VC20_TYPE */
 a_panode          IN OUT   NUMBER,          /* LONG_TYPE */
 a_me              IN OUT   VARCHAR2,        /* VC20_TYPE */
 a_menode          IN OUT   NUMBER,          /* LONG_TYPE */
 a_value_f         OUT      FLOAT,           /* FLOAT_TYPE */
 a_value_s         OUT      VARCHAR2)        /* VC40_TYPE */
RETURN NUMBER;
```

Exercise

Exercise 10

- Fetch samples of the product class ‘finished product’ that have status Available (AV)

Save<Object>

Save the std properties of a list of objects assigned to the same main object

- E.g. SaveScParameter saves parameters for 1 parameter group
- Exception: SaveSample
  - Only saves standard properties of 1 sample
- Only saves current object level

Uses NODES to identify objects

SaveSc<Object>

Generic function header

```
FUNCTION SaveSc<Object>
(a_sc                   IN      UNAPIGEN.VC20_TABLE_TYPE,
 a_pg                   IN      UNAPIGEN.VC20_TABLE_TYPE,
 a_pgnode               IN      UNAPIGEN.LONG_TABLE_TYPE,
 ...
 a_lc                   IN       UNAPIGEN.VC2_TABLE_TYPE,
 a_modify_flag          IN OUT   UNAPIGEN.NUM_TABLE_TYPE,
 a_nr_of_rows           IN       NUMBER,
 a_modify_reason        IN       VARCHAR2)/* VC255_TYPE*/
RETURN NUMBER;
```

SaveSc<Object> (2)

a_modify_flag (IN):

- what to do for each row
  - MOD_FLAG_UPDATE
    - Update the data in the database for this record
  - MOD_FLAG_INSERT
    - Insert this row with a node number before the next row in the array
  - MOD_FLAG_CREATE
    - Insert the object and all descending levels
  - MOD_FLAG_DELETE
    - Delete this record from the database
  - DBERR_SUCCESS
    - Leave this record untouched
  - MOD_FLAG_INSERT_WITH_NODES
    - Insert this object in the DB with a given node number

SaveSc<Object> (3)

a_modify_flag (OUT):

- error code in case of PARTIAL save
- DBERR_SUCCESS if OK
- If there is a data error in one of the rows of the arrays
  - The API tries to save as much as possible
  - Return code = DBERR_PARTIALSAVE
  - Check the on-line help of a_modify_flag for details (contains exact return code for the row)

Inserting Op. Objects

Only if allow_modify = '1' on all upper levels

- E.g. adding method measurement only if sample not yet finished

IN node should be 0, OUT value will be generated node number

Insert is done BEFORE the node of the next row in the arrays

- Node = NUMBER(9)
- Binary split to insert before
- If no next specified, append at end

Pay attention to:

- Duplicate entries (same method name, but different node number)
- Inserting parameter without parameter group ('/' parameter group created)
- If result available, implicit call of Save...Result
- Event handling:
  - <Xx>Created
  - <Xx>Result Updated

Updating Op. Objects

Only if allow_modify = ‘1’ on all upper levels

- E.g. info field can only be changed if sample is still modifiable

If result available => implicit call of Save...Result

Event handling

- <Xx>Updated
- <Xx>ResultUpdated

Deleting Op. Objects

Only possible if object is not active

- DBERR_OPACTIVE

Samples, Requests

- DeleteSample(), DeleteRequest()

Other objects:

- SaveSc<Object> with MOD_FLAG_DELETE

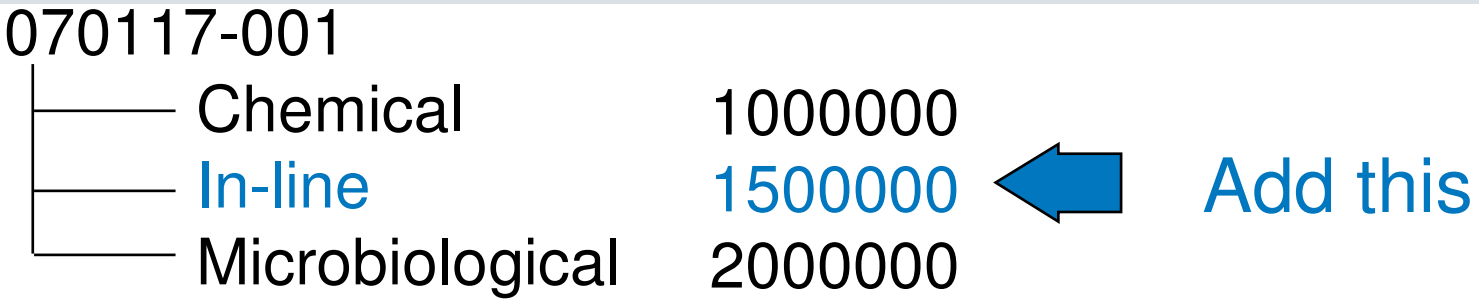
Event handling:

- <Xx>Deleted

SaveSc<Object> - Example

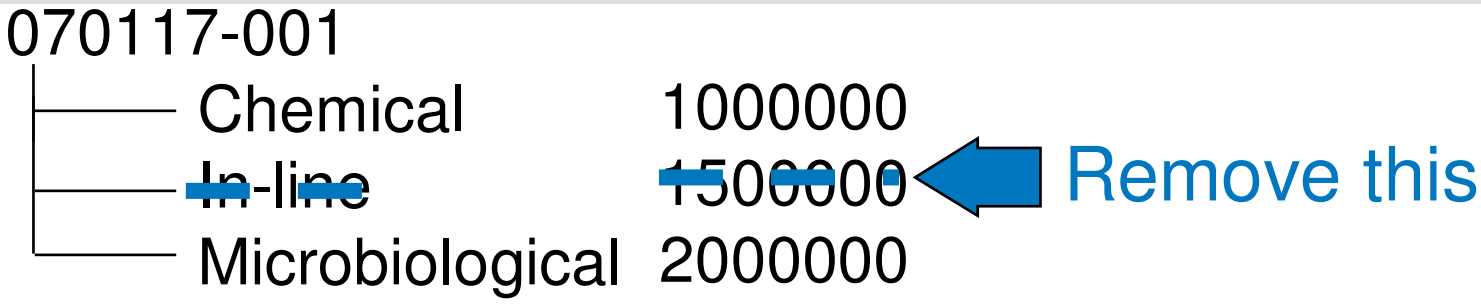

```
a_sc(1) := '070117-001';
a_pg(1) := 'Microbiological';
a_pgnode(1) := 0;
a_modify_flag(1) := UNAPIGEN.MOD_FLAG_INSERT;
a_nr_of_rows := 1;
ret_code := UNAPIPG.SaveScParameterGroup
                (a_sc, a_pg, a_pgnode, ...
                 a_modify_flag, a_nr_of_rows, 'Example1');
```

SaveSc<Object> - Example (2)




```
a_sc(1) := '070117-001';
a_pg(1) := 'In-line';
a_pgnode(1) := 0;
a_modify_flag(1) := UNAPIGEN.MOD_FLAG_INSERT;
a_sc(2) := '070117-001';
a_pg(2) := 'Microbiological';
a_pgnode(2) := 2000000;
a_modify_flag(2) := NULL;
a_nr_of_rows := 2;
ret_code := UNAPIPG.SaveScParameterGroup
                (a_sc, a_pg, a_pgnode, ...
                a_modify_flag, a_nr_of_rows, 'Example2');
```

SaveSc<Object> - Example (3)




```
a_sc(1) := '070117-001';
a_pg(1) := 'In-line';
a_pgnode(1) := 1500000;
a_modify_flag(1) := UNAPIGEN.MOD_FLAG_DELETE;
a_nr_of_rows := 1;
ret_code := UNAPIPG.SaveScParameterGroup
                (a_sc, a_pg, a_pgnode, ...
                a_modify_flag, a_nr_of_rows, 'Example3');
```

Create Sample (Request)

Based on sample type

- Automatic pg, pa, me assignment
  - Evaluate frequency assignments
  - Sampling date is reference
  - Specs handling
- Assignment of au and gk

If no sample type is specified

- Sample created empty (no ic or pg)
- Assignment of gk

Field Types

- Field types are used to set the context
- Possible field types : rq; sd; gk; delay; delay_unit
- When field_type gk, field_name specifies the gk name
- All group keys will be assigned to the object
- The assigned group keys are used for enterprise LIMS

Exercises

Exercise 11

- Create a sample of sample type 01-100 by calling CreateSample()
- Did you create a sample of the last version??

Exercise 12

- Add the parameter with description ‘Zero’ in the first parameter group
- Use LifeCycle ‘LP’ with the correct version.

Creating Object from Configuration

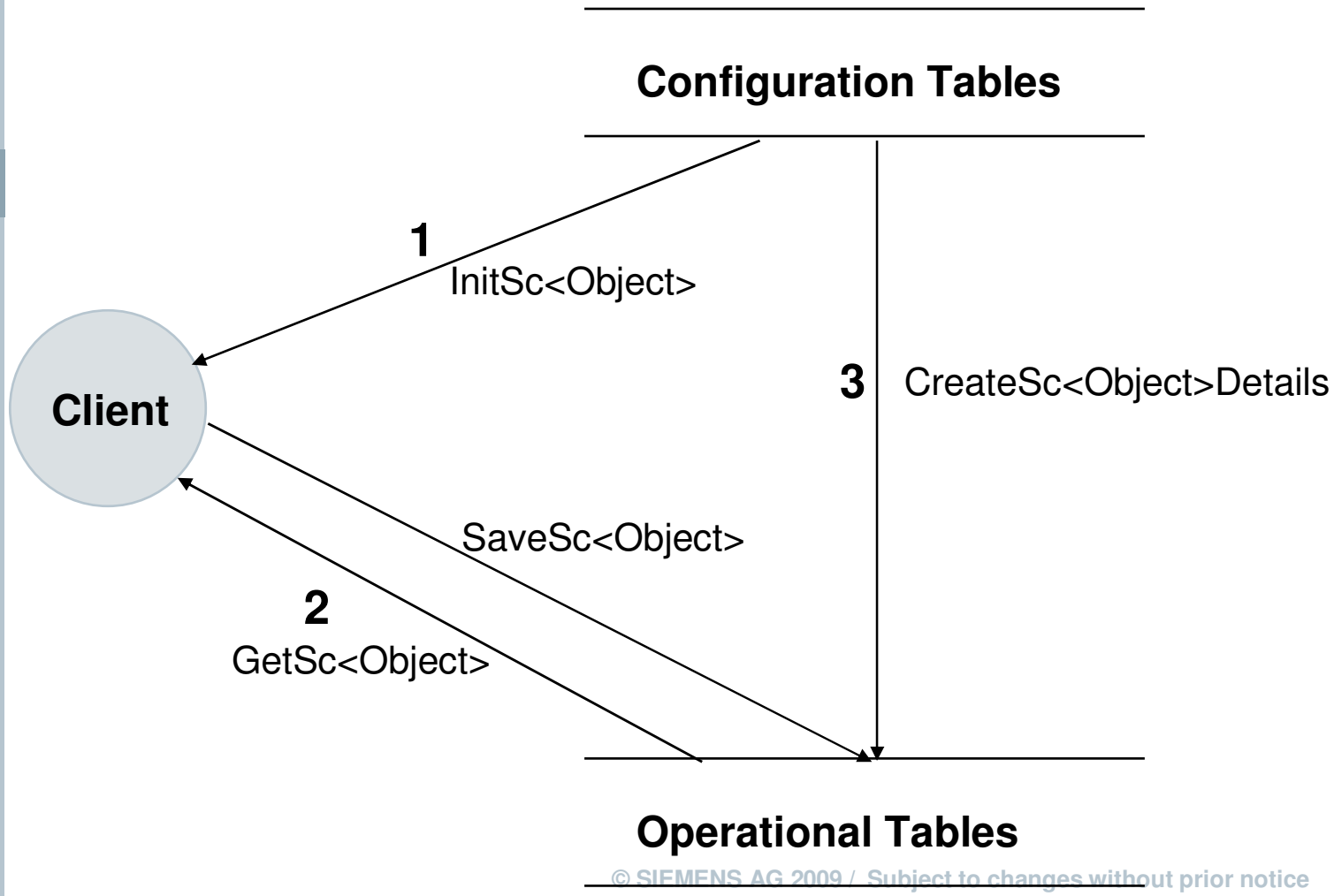
Creating objects from configuration

- Initialisation using InitSc<Object>
  - Only in memory
- Calling SaveSc<Object>
  - In the DB

Sub-tree assignment

- For existing objects!
- Using InitSc<XX>Details
  - Only in memory
- Calling CreateSc<XX>Details
  - In the DB

Assignment of op. objects (2)



Initialising Data

Initialising op. objects: InitSc<Object>

- Using configuration data
- Output arrays can be used directly in SaveSc<Object>
  - All columns provided
  - Output consists of arrays because multiple rows may be returned

Example function header

```
FUNCTION InitScParameter
(a_pr              IN       VARCHAR2,          /* VC20_TYPE */
 a_seq             IN       NUMBER,            /* NUM_TYPE */
 a_sc              IN       VARCHAR2,          /* VC20_TYPE */
 a_pg              IN       VARCHAR2,          /* VC20_TYPE */
 a_pgnode          IN       NUMBER,            /* LONG_TYPE */
 a_description     OUT      UNAPIGEN.VC40_TABLE_TYPE,
 ...
 a_nr_of_rows      IN OUT   NUMBER)            /* NUM_TYPE */
RETURN NUMBER;
```

Creating Sub-objects

In the DB

- With or without frequency filtering
- E.g. method(s) for a parameter

```
FUNCTION CreateScPaDetails
(a_st                    IN VARCHAR2,        /* VC20_TYPE */
 a_pp                    IN VARCHAR2,        /* VC20_TYPE */
 a_pr                    IN VARCHAR2,        /* VC20_TYPE */
 a_seq                   IN NUMBER,         /* NUM_TYPE */
 a_sc                    IN VARCHAR2,       /* VC20_TYPE */
 a_pg                    IN VARCHAR2,       /* VC20_TYPE */
 a_pgnode                IN NUMBER,         /* LONG_TYPE */
 a_panode                IN NUMBER,          /* LONG_TYPE */
 a_filter_freq           IN CHAR,            /* CHAR1_TYPE */
 a_ref_date              IN DATE,            /* DATE_TYPE */
 a_modify_reason         IN VARCHAR2)       /* VC255_TYPE */
RETURN NUMBER;
```

Exercises

Exercise 13

- Assign a new parameter (*description: FAT*) to the sample, using the Init- and Save- functions

Exercise 14

- Assign methods to the new parameter using the Create...Details functions

Result Handling

Saving parameter and method results

- Call SaveSc<XX>Result instead of SaveSc<Object>

SaveSc<Object>Result

- Saves the result
- Evaluates alarm handling
- Distributes results to other parameters/methods

Reanalysis

Reanalyses can be applied to

- Parameter groups: ReanalScPgDetails
  - Reanalysis of all parameters in parameter group
- Parameters: ReanalScParam
- Methods: ReanalScMethod

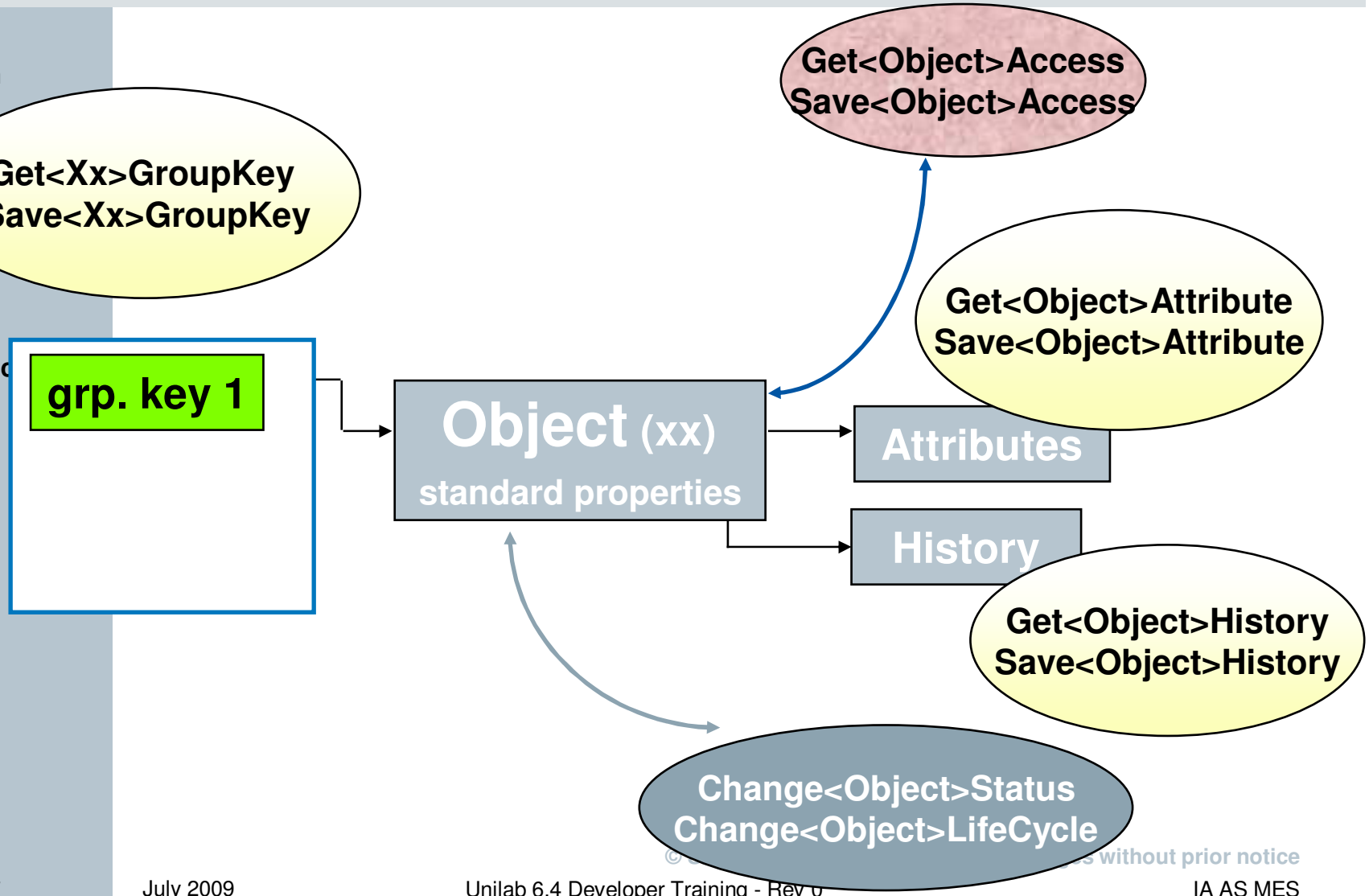
ReanalSc<XX>FromDetails

- Details of analysis are not cleared
  - example: ReanalScPaFromDetails

To get the reanalysis values:

- GetScRe<Object>
  - Identical to GetSc<Object>
  - Selects from utrsc<xx>
  - Gives the complete list of all reanalyses (unless specified otherwise in where_clause)

Common Operational APIs



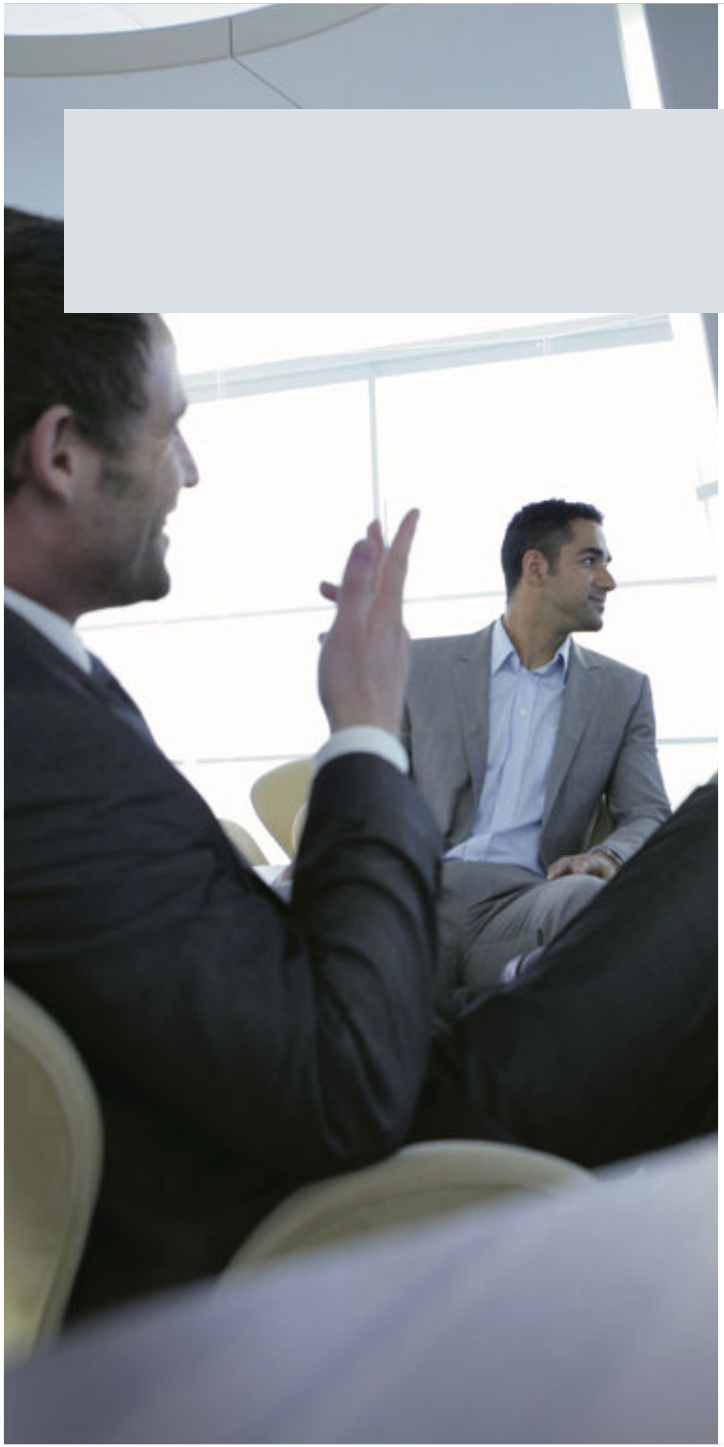
Common Operational APIs (2)

All functions are identical to the Configuration APIs

Except for the key arguments

- No object type, as each type has its own functions
- Object ID = sc + object ID(s) + nodes !!!

No ‘Used’ objects at all



SIEMENS

Life Cycles

Life Cycles

Each individual object has its own life cycle

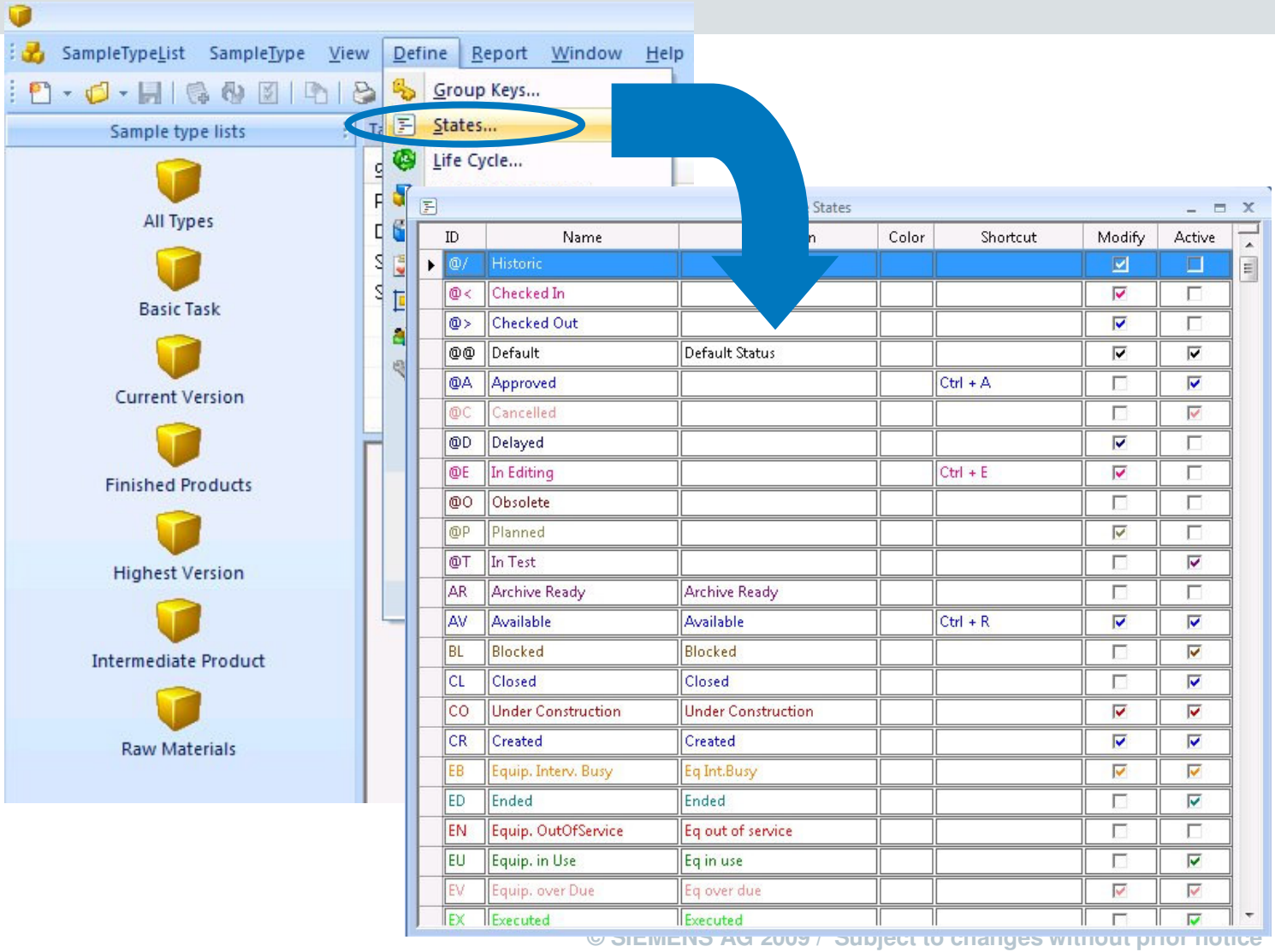
Life Cycles are used to manage the states of a certain object

An object is, at any moment, in a specific state

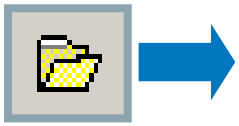
Each state determines

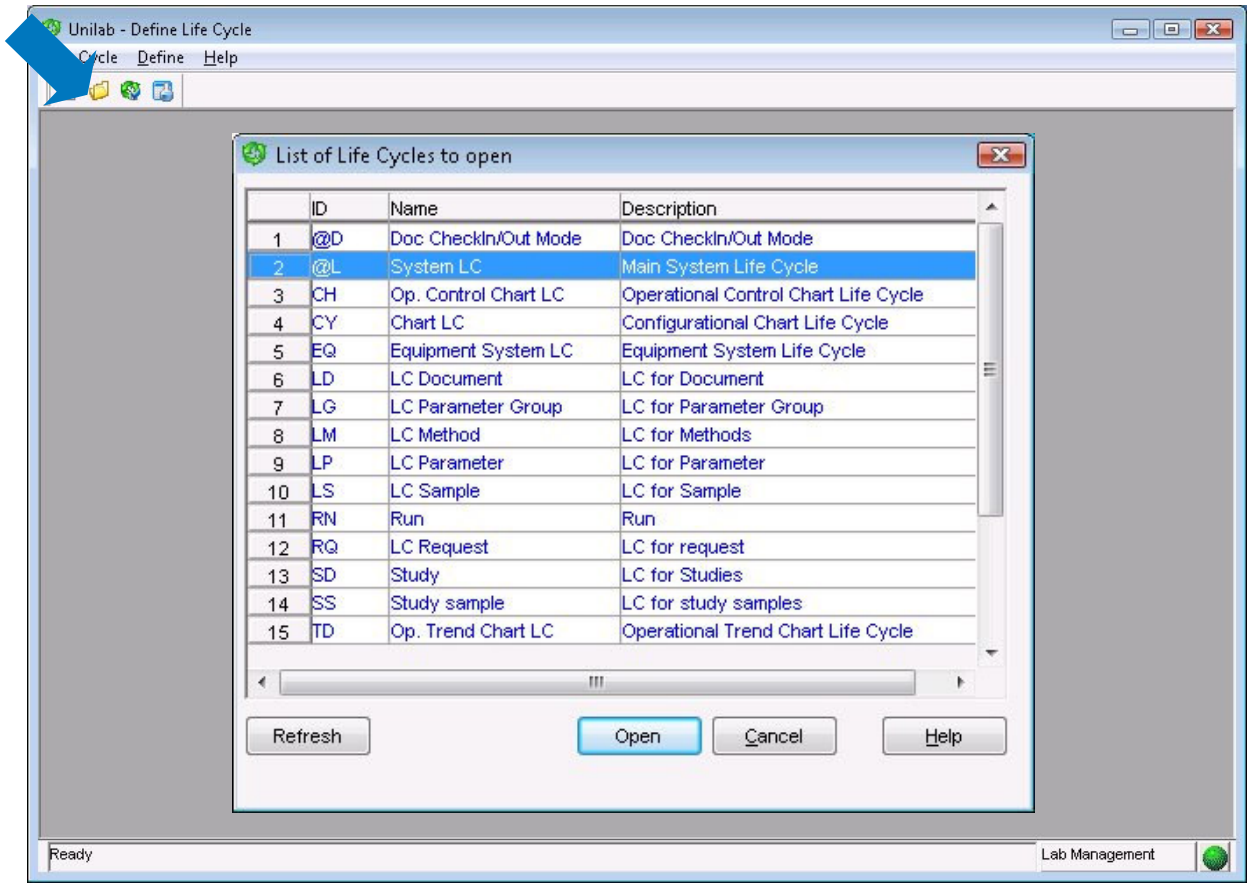
- whether or not modifications are allowed
- whether or not the object is “active”
- what colour the object will be displayed in a list!

Define states

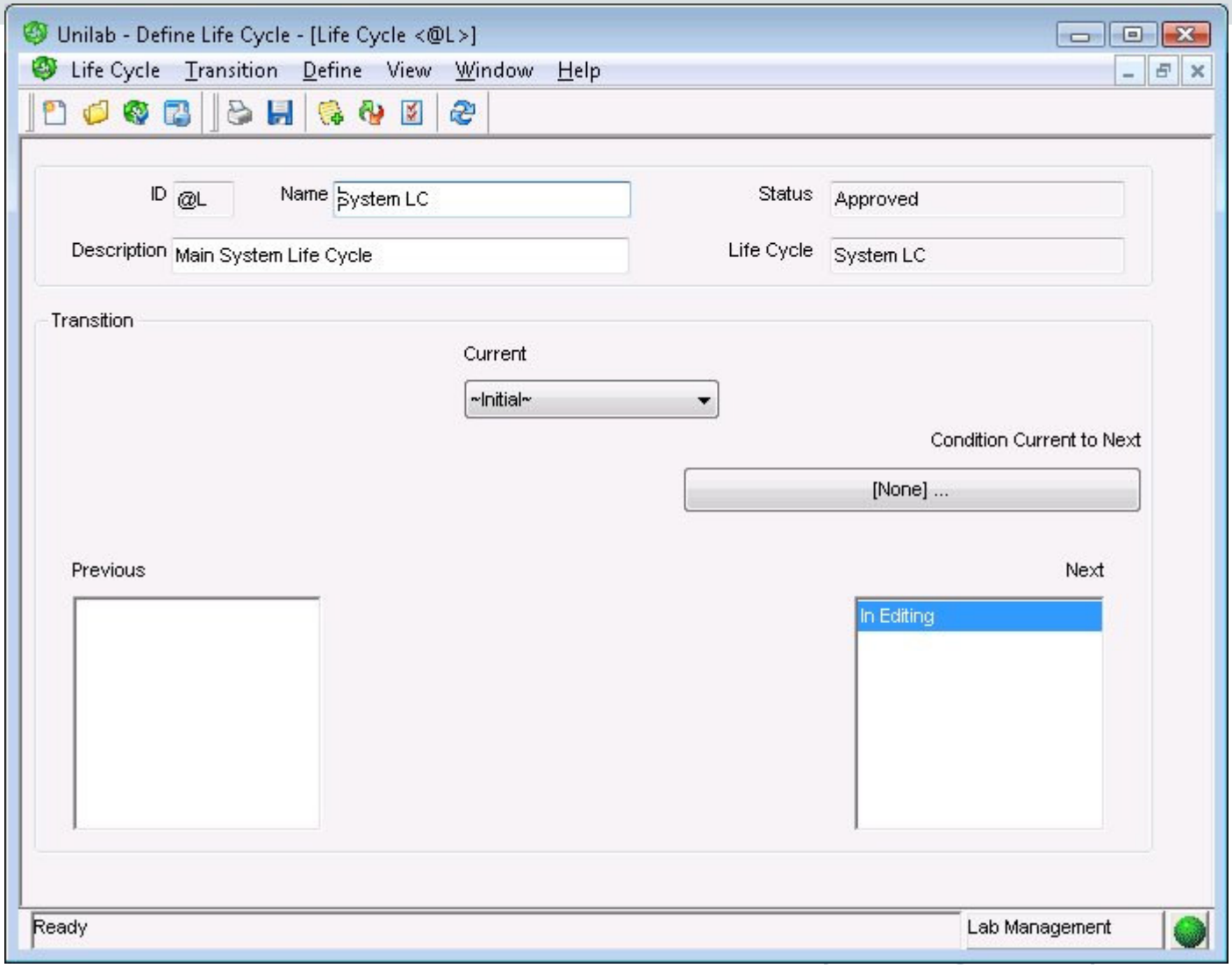


Define Life Cycles

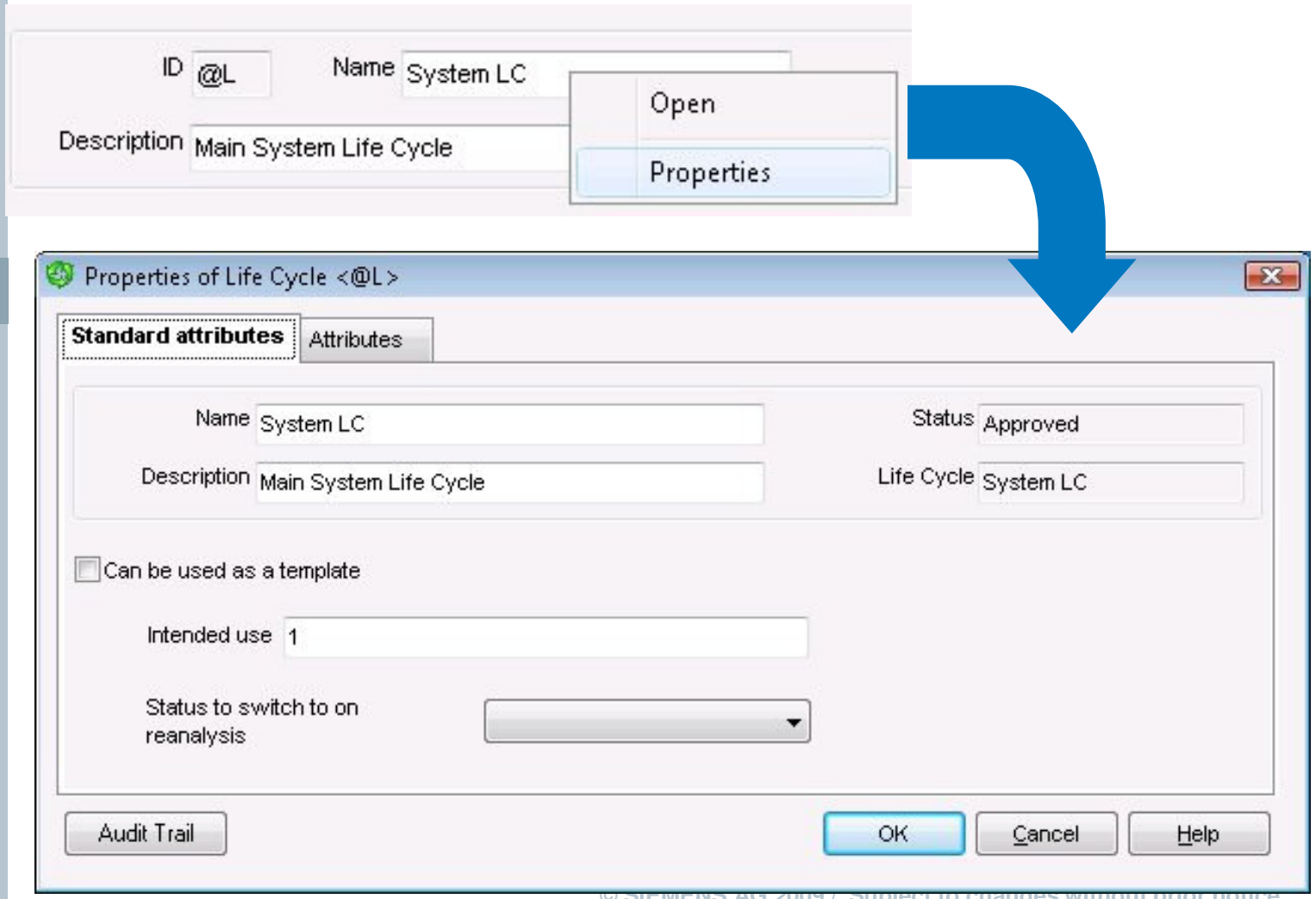




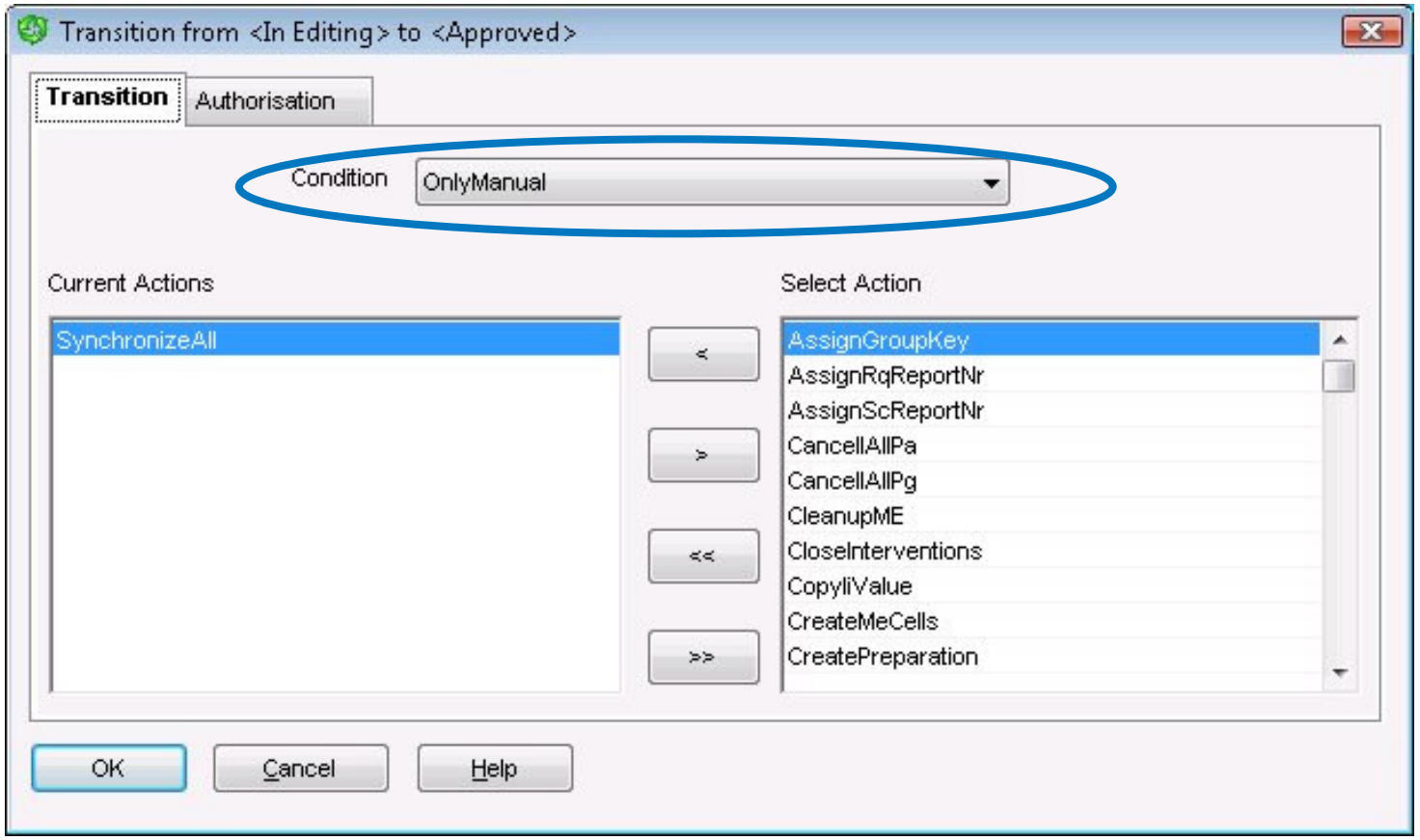
Life Cycle



Life Cycle Properties

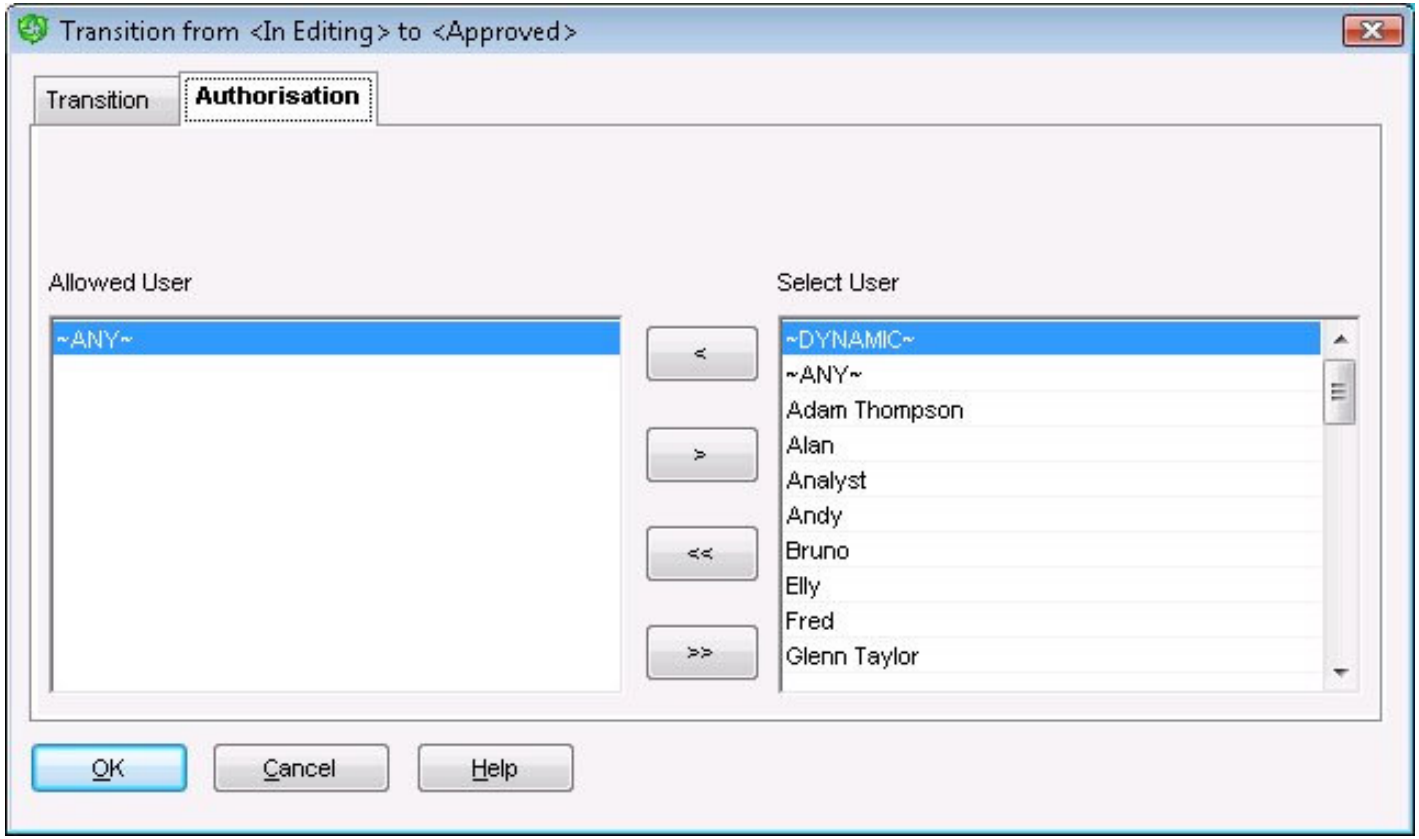


Conditions

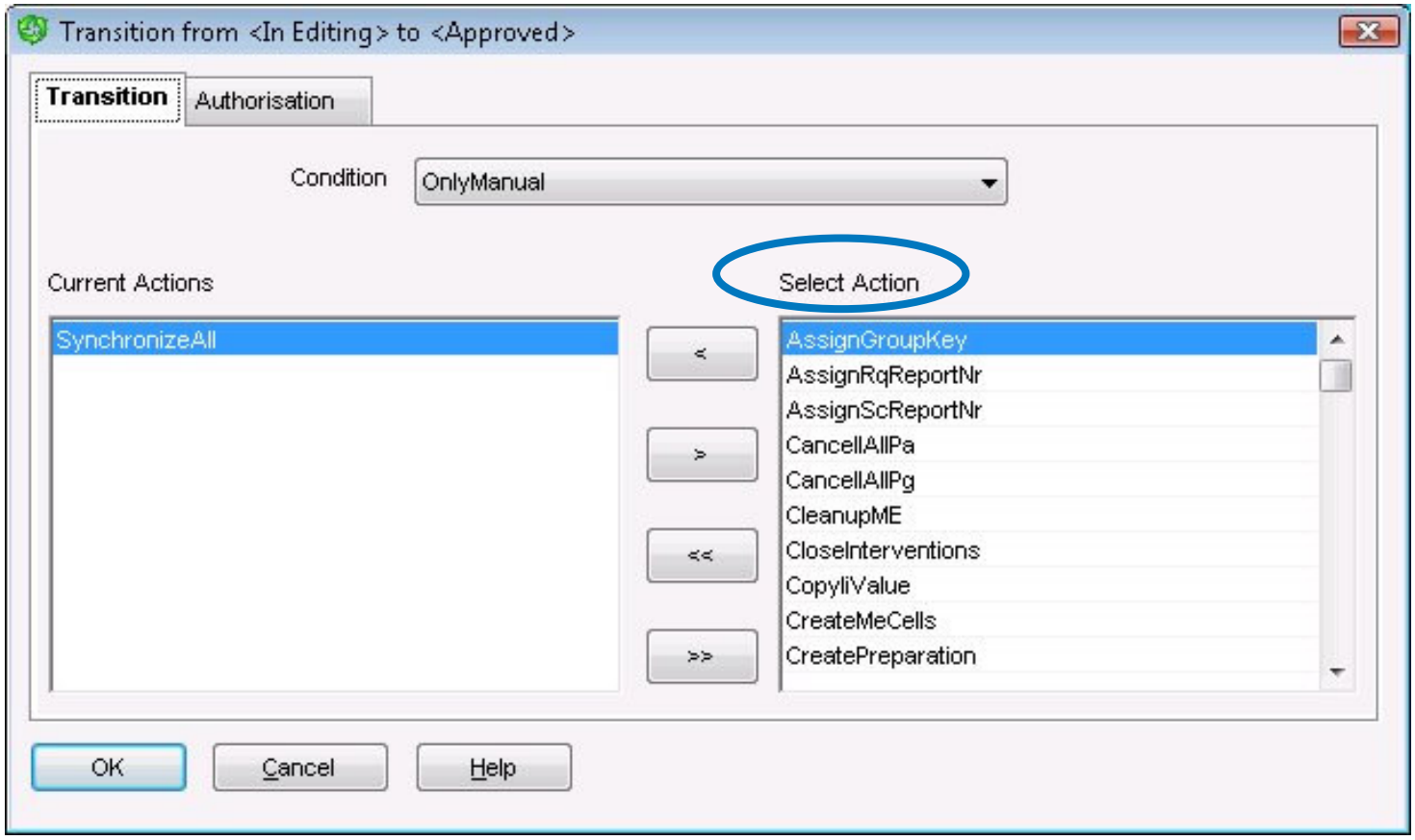


Conditions are PL/SQL Custom Functions

Authorization

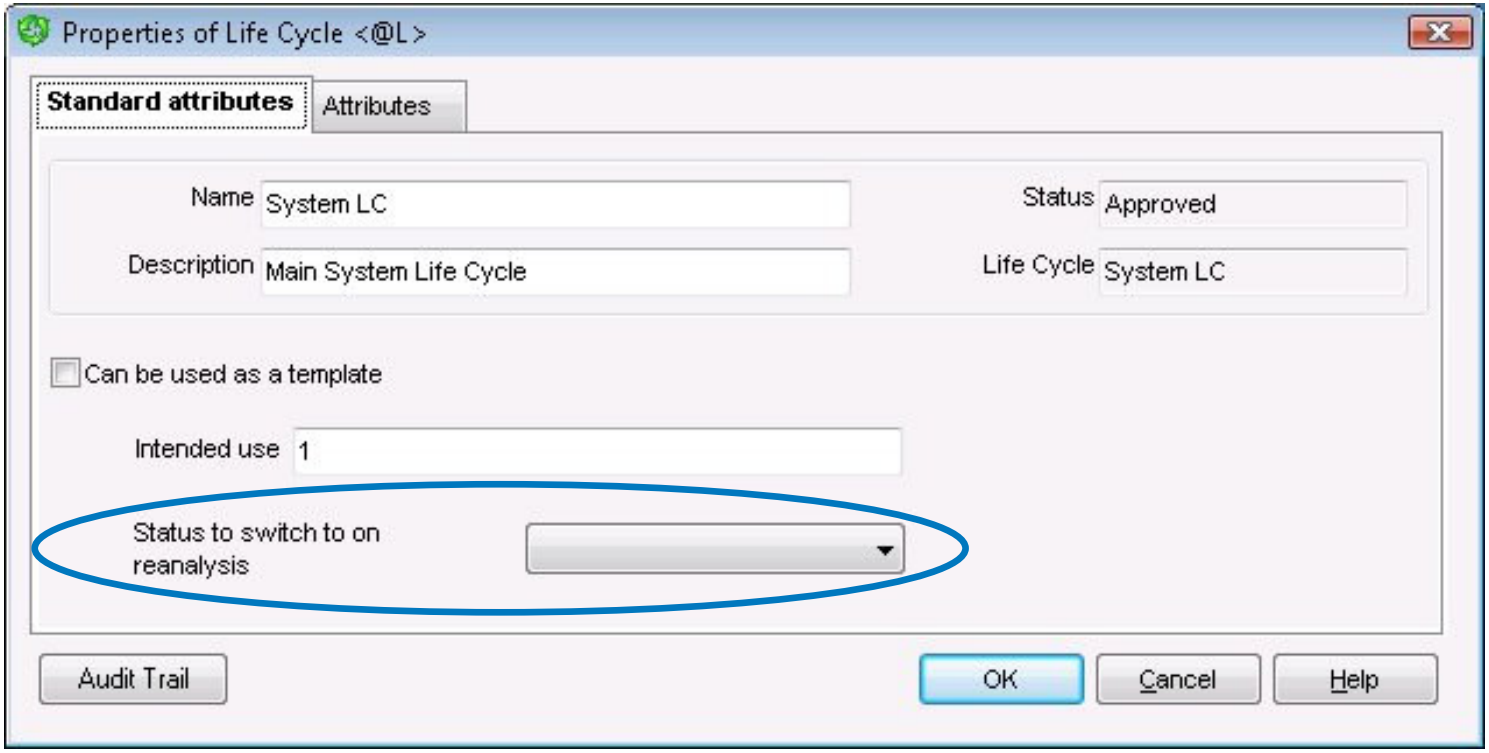


Actions

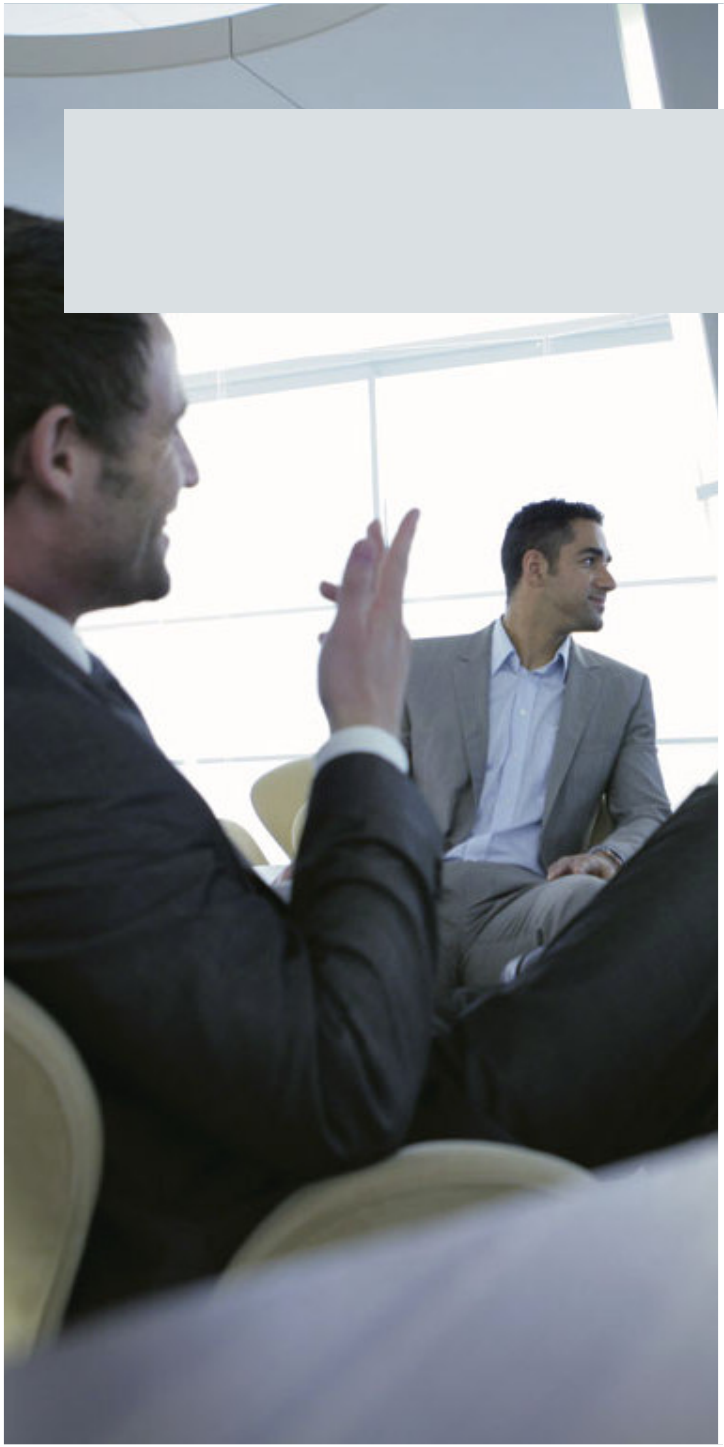


Actions are PL/SQL Custom Functions

Re-analysis



ChangeSs_AllowReanal Preference



SIEMENS

Custom Functions

Introduction

Developing Custom Functions

Custom Functions

- PL/SQL or C++

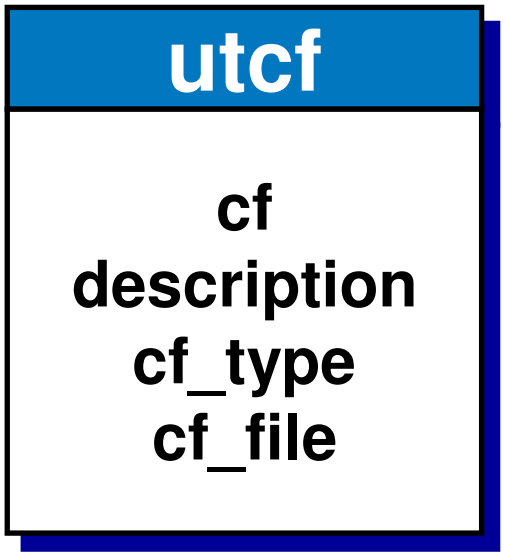
Custom Functions are stored in “packages”

- Custom packages for PL/SQL custom functions
- DLL for C++ custom functions

UTCF

- Reference table for all custom functions

UTCF Table



CF

- name of PL/SQL or C++ function

DESCRIPTION

- info on custom function

CF_TYPE

- Custom function type
  - condition, action,…
- Determines package for PL/SQL functions

CF_FILE

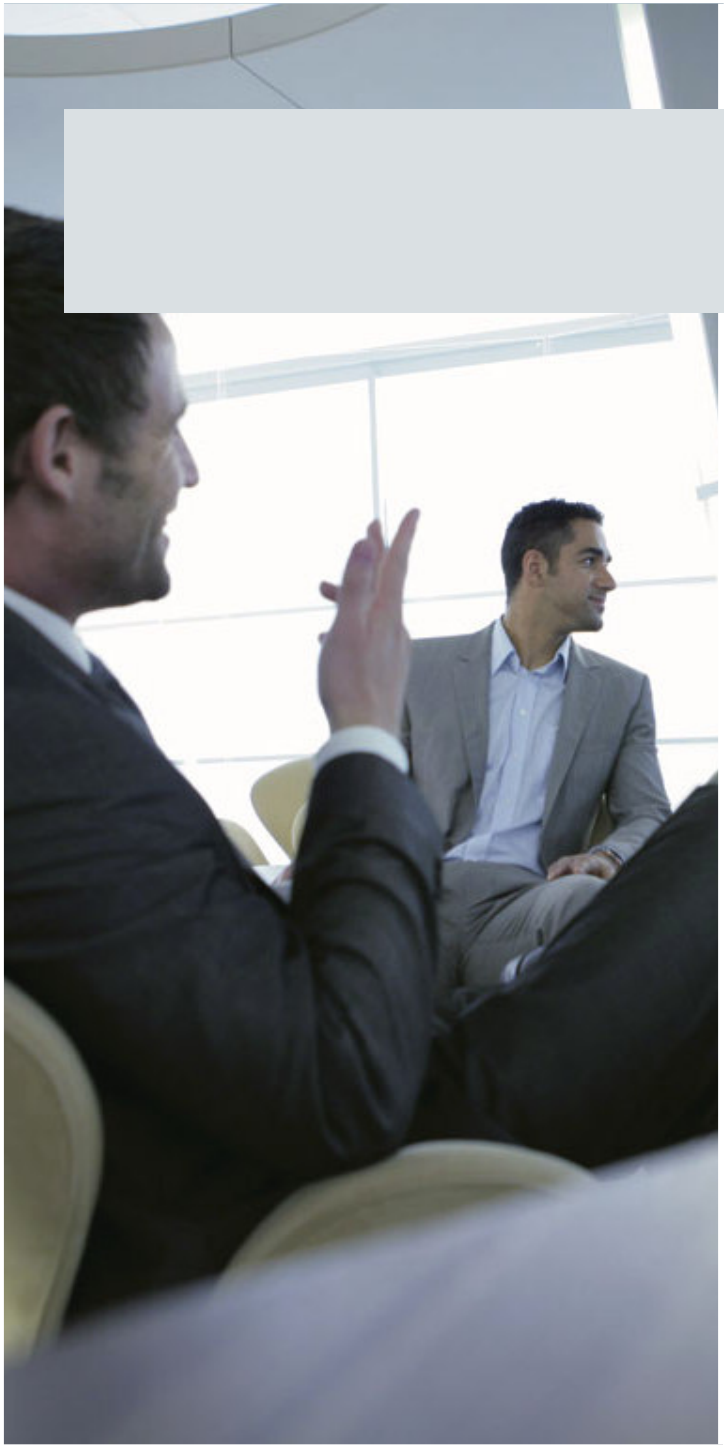
- File for C++ custom functions

UTCF Table - Example

CF	DESCRIPTION	CF_TYPE	CF_FILE
slope	slope	mtcellcalc	slope.cpp
AddExtraParameters	AddExtraParameters	paalarm	AddExtraParameters.cpp
CheckResultPositive	CheckResultPositive	mtcellval	CheckResultPositive.cpp
DistributionLst	DistributionLst	ieval	DistributionLst.cpp
standard label	standard label	printlbl	standard label.cpp
SCSAMPLINGDATE	SCSAMPLINGDATE	scgkcreate	
MEASSIGNRESPONSIBLE	MEASSIGNRESPONSIBLE	megkcreate	
CANCELLALLPA	CANCELLALLPA	lcaction	
ALLPAVALIDATED	ALLPAVALIDATED	lccond	

Client Custom Functions

DB Custom Functions



SIEMENS

Custom Functions

Database Custom Functions

DB Custom functions

PL/SQL Custom Functions

- header + body !

Grouped in Packages:

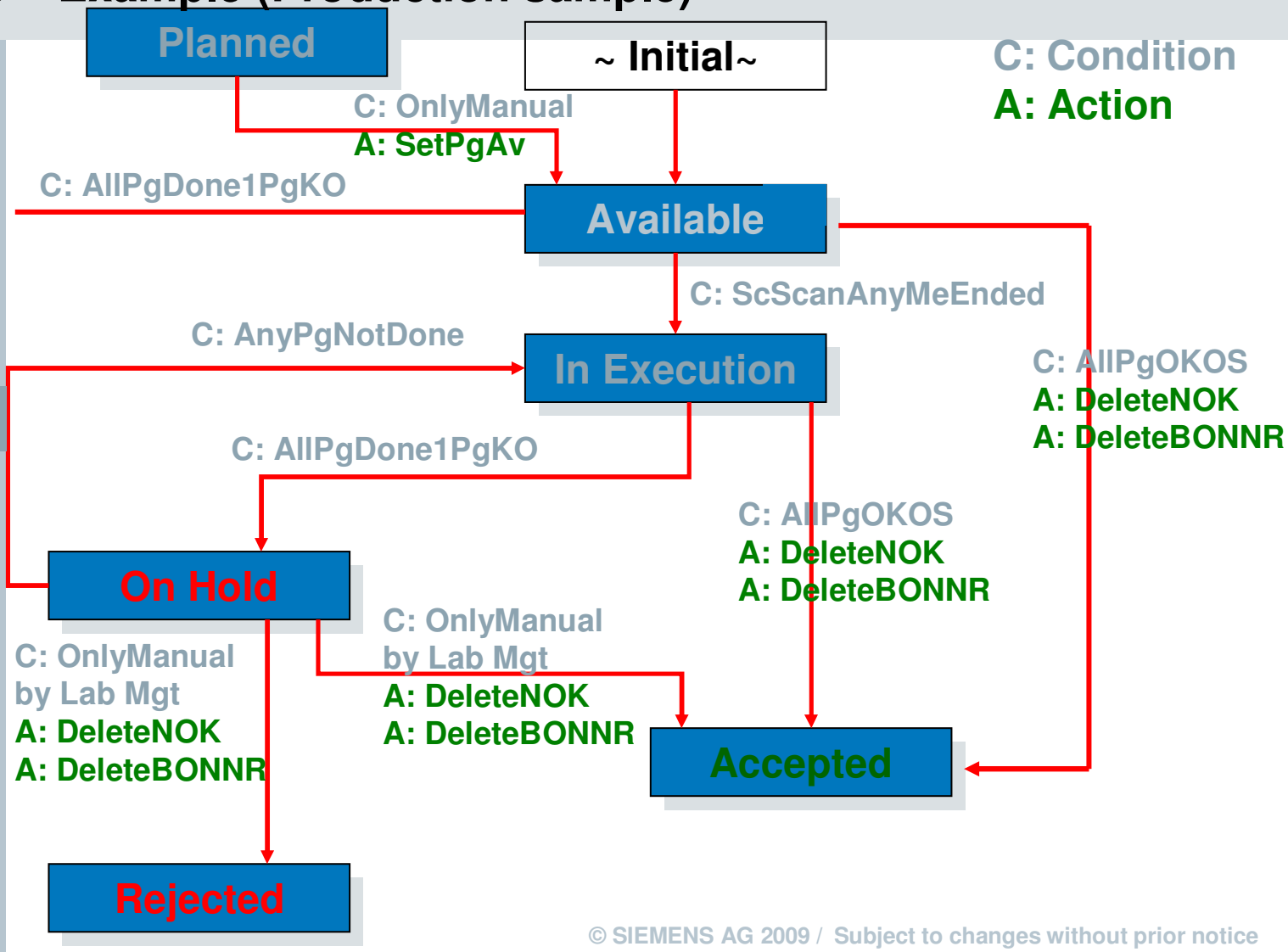
- UN + custom function type
  - Conditions : UNCONDITION
  - Data access rights : UNACCESS
  - ...

Access to DB through DB API's

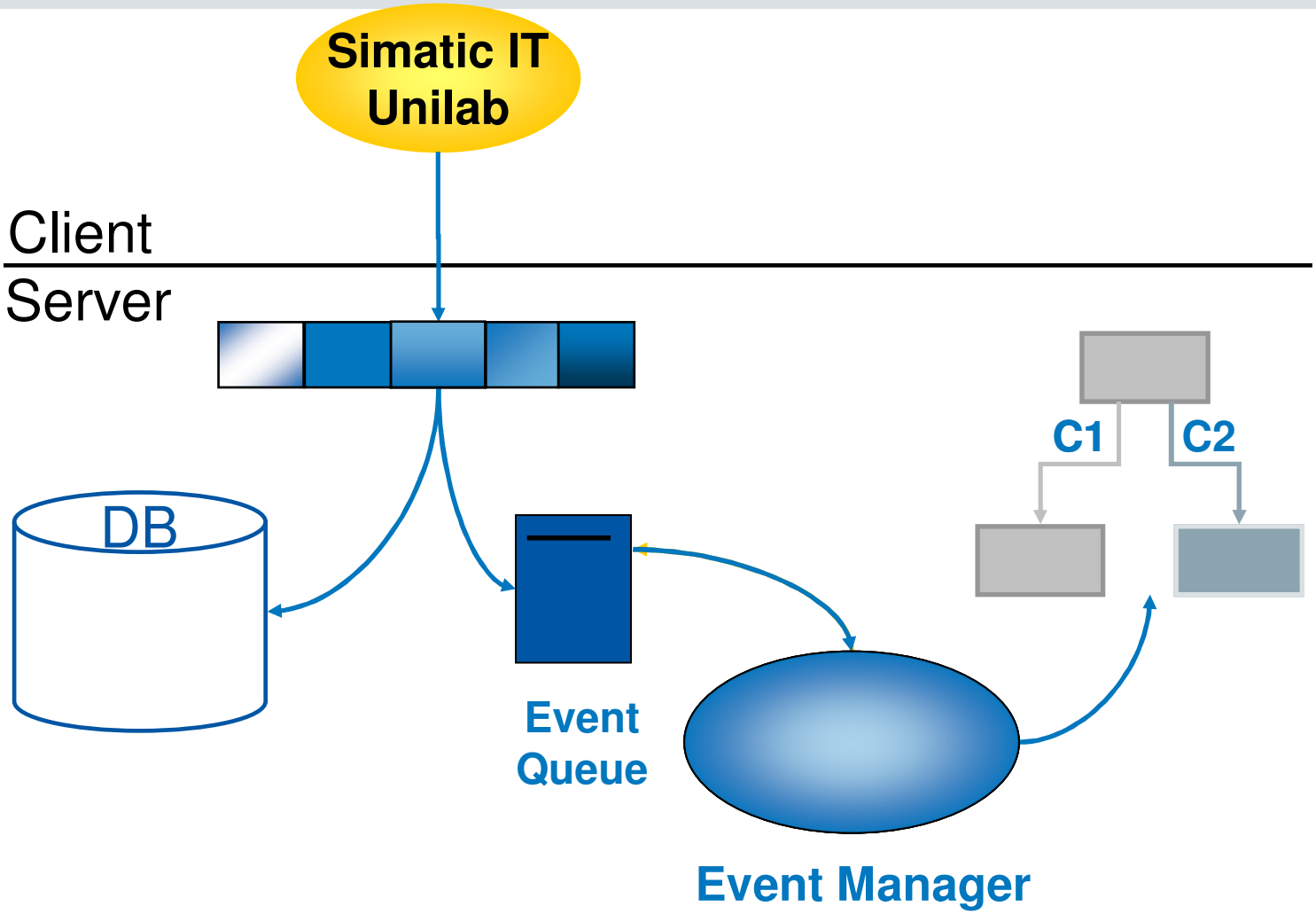
DB Custom Functions

Conditions in life cycle
Actions in life cycle
Access rights assignment
Group key assignment rules
Worklist assignment rules
Custom assignment frequencies
Alarm Handling
Parameter calculations

Life Cycle – Example (Production sample)



Event Management



Conditions

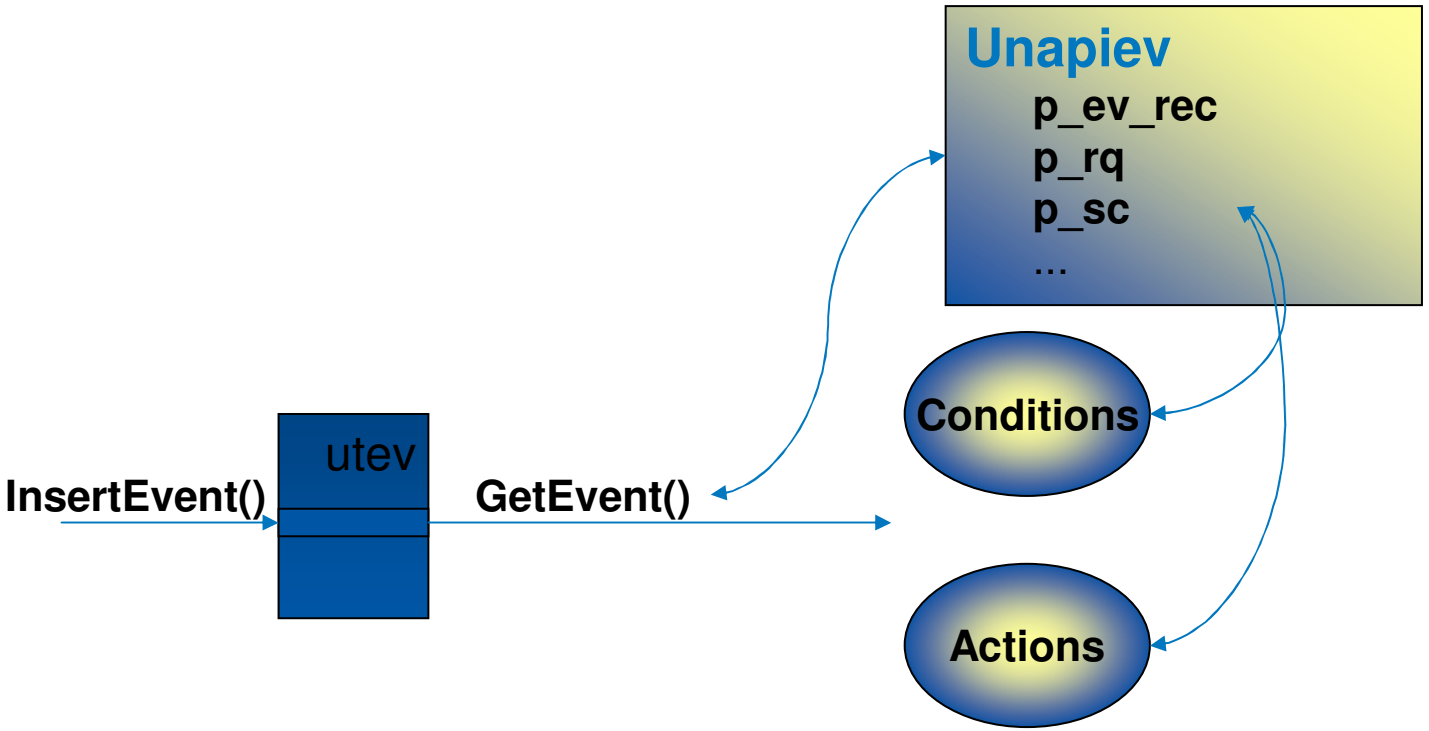
Package: UNCONDITION

CF_type = lccond

Return value (ret_val)

- DBERR_SUCCESS	The condition is met
- <else>	The condition is not OK

Event Manager Package Variables



UNAPIEV package variables

- Event causing the call

```
P_EV_REC        utev%ROWTYPE;
```

- Excerpts from ev_details

```
P_SC            VARCHAR2(20);
P_PG            VARCHAR2(20);
P_PA            VARCHAR2(20);
P_ME            VARCHAR2(20);
P_IC            VARCHAR2(20);
P_II            VARCHAR2(20);
P_PGNODE        NUMBER(9);
P_PANODE        NUMBER(9);
P_MENODE        NUMBER(9);
P_ICNODE        NUMBER(9);
P_IINODE        NUMBER(9);
...
```

Conditions - Examples

Check if parameter started

```
FUNCTION PaStarted
RETURN NUMBER IS
  l_exec_start_date   DATE;
BEGIN
  SELECT exec_start_date
  INTO l_exec_start_date
  FROM utscpa
  WHERE sc = UNAPIEV.P_SC
         AND pg = UNAPIEV.P_PG
         AND pgnode = UNAPIEV.P_PGNODE
         AND pa = UNAPIEV.P_PA
         AND panode = UNAPIEV.P_PANODE;
  IF l_exec_start_date IS NOT NULL THEN
         RETURN (UNAPIGEN.DBERR_SUCCESS);
  ELSE
         RETURN (UNAPIGEN.DBERR_NOOBJECT);
  END IF;
END PaStarted;
```

Reference to UNAPIEV package variables

Conditions - Examples

OnlyManual

- Always returns an error to avoid automatic transitions

```
FUNCTION OnlyManual
RETURN NUMBER IS

BEGIN
  RETURN(UNAPIGEN.DBERR_GENFAIL);
END OnlyManual;
```

DB Custom Functions

Conditions in life cycle

Actions in life cycle

Access rights assignment

Group key assignment rules

Worklist assignment rules

Custom assignment frequencies

Alarm Handling

Parameter calculations

Actions

Package: UNACTION

Cf_type = lcaction

- ret_val
  - Any value allowed
  - If not DBERR_SUCCESS, following actions are no longer executed
- Executed after a status transition has happened
  - Object is already in new status
- Secondary events
  - Executed within the same transaction
  - Executed with the same access rights as the primary events

Actions - Examples

Set access rights to ‘R’ for up5 for parameter

```
FUNCTION SetAccessToUp5
  RETURN NUMBER IS
BEGIN
  UPDATE utscpa SET ar5='R'
  WHERE sc = UNAPIEV.P_SC
          AND pg = UNAPIEV.P_PG
          AND pgnode = UNAPIEV.P_PGNODE
          AND pa = UNAPIEV.P_PA
          AND panode = UNAPIEV.P_PANODE;
  IF SQL%ROWCOUNT = 0 THEN
          RETURN(UNAPIGEN.DBERR_GENFAIL);
  ELSE
          RETURN(UNAPIGEN.DBERR_SUCCESS);
  END IF;
END SetAccessToUp5;
```

Transaction Handling

Avoid Get… functions

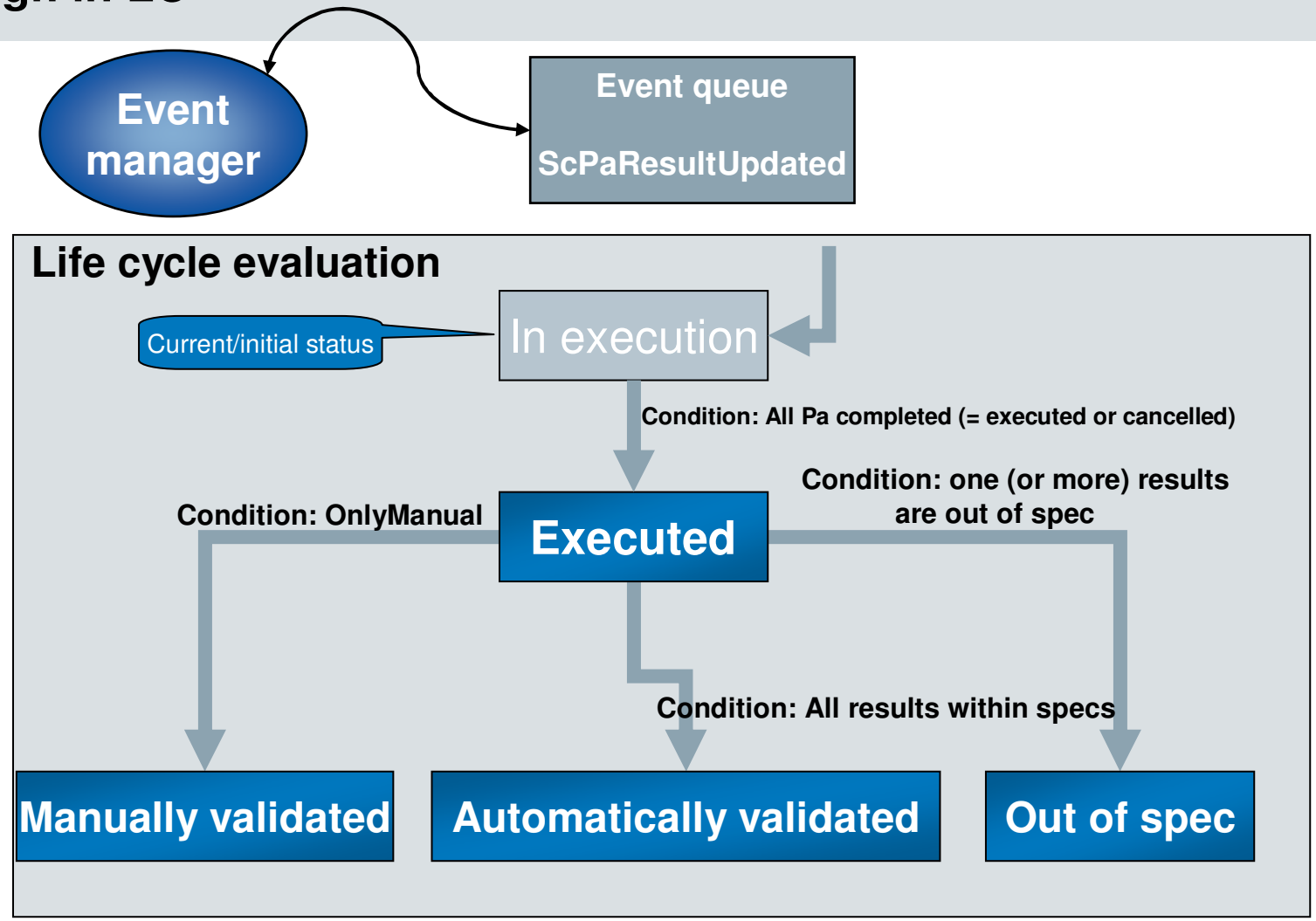
Event manager handles transaction control

- No BeginTransaction() - EndTransaction()

No Rollback on failure of action

Never ignore the return code of an API

Fall Through in LC



Insert Event


```
BOOL PaCompleted = TRUE;  -- we start optimistically
BOOL AllResultsWithinSpecs = TRUE ;
BOOL ResultsOutOfspecs = FALSE ;
FOR EACH pa IN pa_list_of_this_pg
        IF NOT ( pa.ss IN ("Executed", "Cancelled") ) THEN
                PaCompleted = FALSE ;
        END IF
        IF ( pa.value_f IS NULL ) OR ( pa.ss <> "Executed" )
THEN
                AllResultsWithinSpecs = FALSE ;
        ELSE IF pa.value_f is out of spec THEN
                ResultsOutOfspecs = TRUE ;
                AllResultsWithinSpecs = FALSE ;
        END IF
NEXT pa
IF AllResultsWithinSpecs = TRUE THEN
        ev_details = "All results within specs"
        InsertEvent ('<ThisAction>', UNAPIGEN.P_EVMGR_NAME,
                object_tp, object_id, object_lc,
                object_ss, ev_tp, ev_details, seq_nr ) ;
ELSE IF ResultsOutOfspecs = TRUE THEN
        ev_details = "One (or more) results are out of spec"
        InsertEvent('<ThisAction>', UNAPIGEN.P_EVMGR_NAME,
                object_tp, object_id, object_lc,
                object_ss, ev_tp, ev_details, seq_nr ) ;
END IF
```

DB Custom Functions

Conditions in life cycle

Actions in life cycle

Access rights assignment

Group key assignment rules

Worklist assignment rules

Custom assignment frequencies

Alarm Handling

Parameter calculations

Access Rights

- Initialized on object creation
- Additional Event on object creation
  - Life cycle assignment
  - Group key assignment
  - Initialization object access rights
- Logic in UNACCESS
- InitObjectAccessRights
  - For a new object
  - Only once (on object creation)
- UpdateAccessRights
  - For each status change of any existing object
  - Executed after life cycle evaluation
    - Before execution of actions!
- Define additional functions
  - InheritObjectAccessRights

Authorization for LC Transition

- The UNACCESS.TransitionAuthorised function is evaluated each time a Change…Status function is called for a life cycle transition where the authorised user has been set to ~DYNAMIC~
- Package variables (similar to event manager variables)
  - **P_LCTRUS_REC** **utlcus record for this transition**
    P_OBJECT_ID **object name**
    P_OBJECT_TP **object type**
    P_LC **life cycle**
    P_SS_FROM **from status**
    P_LC_SS_ **FROM life cycle ss_from (can be different of P_SS_FROM when Default status is used)**
    P_SS_TO **target status**
    P_TR_NO **transition number**
    P_RQ
    P_SC
    P_PG
    P_PGNODE
    …

DB Custom Functions

Conditions in life cycle
Actions in life cycle
Access rights assignment
Group key assignment rules
Worklist assignment rules
Custom assignment frequencies
Alarm Handling
Parameter calculations

Group Key Assignments

- Group key assignment
  - Group key assignment rules
  - Inheritance at group key level
  - Inheritance at object level
- Assignment of a group key value
  - On object creation
- Package: UNGKASSIGN
- Return value:
  - Any value allowed

Group Key Assignments (2)

Logic:

- Check if event is relevant for the assignment to be done
- Load the list of group key values for the group key at hand (Get<Xx>GroupKey)
- Adjust this list to match the new assignments
- Save the list again using Save1<Xx>GroupKey

Access to UNAPIEV

- UNAPIEV.P_EV_REC
- Current group key assignment rule (UNAPIEV.P_GKRULE_REC)

DB Custom Functions

Worklist Assignments

Assignment of a group key value

- Depends on status in life cycle

Package: UNWLASSIGN

Return value

- Any value allowed

Access to UNAPIEV

- UNAPIEV.P_EV_REC
  - Full object key

Logic

- Define the new group key assignments to enter the method in the required worklists.
- Save the value list using Save1<Xx>GroupKey. To delete the entry, use this function with nr_of_rows = 0.

Access to UNAPIEV.P_EV_REC

DB Custom Functions

Assignment Frequencies

- Specific custom frequency
- Package: UNFREQ
- ret_val:
  - DBERR_SUCCESS Assignment to be done
  - <else> Assignment not to be done
- Evaluation on object creation

Assignment Frequencies (2)

Example

- Assign if attribute present

```
FUNCTION AuBased RETURN NUMBER IS
...
BEGIN
  SELECT value
  FROM utstau
  WHERE st = l_st
  AND au = 'custom_freq'
  AND ROWNUM = 1;
EXCEPTION
WHEN NO_DATA_FOUND THEN
  RETURN(UNAPIGEN.DBERR_GENFAIL);
END;
RETURN(UNAPIGEN.DBERR_SUCCESS);
```

DB Custom Functions

Alarm Handling

- Specific mechanism to handle the fact that a parameter result is out of a specification set
- Package: UNALARM
- Cf_type: paalarm
- Return value:
  - Any value allowed

Alarm Handling (2)

Syntax:

```
FUNCTION EvalAlarm
(a_sc               IN      VARCHAR2,          /* VC20_TYPE */
 a_pg               IN      VARCHAR2,          /* VC20_TYPE */
 a_pgnode           IN      NUMBER,            /* LONG_TYPE */
 a_pa               IN      VARCHAR2,          /* VC20_TYPE */
 a_panode           IN      NUMBER,            /* LONG_TYPE */
 a_valid_specsa     IN      CHAR,              /* CHAR1_TYPE */
 a_valid_specsb     IN      CHAR,              /* CHAR1_TYPE */
 a_valid_specsc     IN      CHAR,              /* CHAR1_TYPE */
 a_valid_limitsa    IN      CHAR,              /* CHAR1_TYPE */
 a_valid_limitsb    IN      CHAR,              /* CHAR1_TYPE */
 a_valid_limitsc    IN      CHAR,              /* CHAR1_TYPE */
 a_valid_targeta    IN      CHAR,              /* CHAR1_TYPE */
 a_valid_targetb    IN      CHAR,              /* CHAR1_TYPE */
 a_valid_targetc    IN      CHAR)              /* CHAR1_TYPE */
RETURN NUMBER;
```

DB Custom Functions

Conditions in life cycle
Actions in life cycle
Access rights assignment
Group key assignment rules
Worklist assignment rules
Custom assignment frequencies
Alarm Handling
Parameter calculations

Parameter Calculations

- Define a specific mechanism to calculate the parameter result out of the available method results.
- Package: UNCALC
- Syntax:

```
FUNCTION CalculationName
  (a_sc          IN      VARCHAR2,        /* VC20_TYPE */
   a_pg          IN      VARCHAR2,        /* VC20_TYPE */
   a_pgnode      IN      NUMBER,          /* LONG_TYPE */
   a_pa          IN      VARCHAR2,        /* VC20_TYPE */
   a_panode      IN      NUMBER,          /* LONG_TYPE */
   a_value_f     OUT     FLOAT,           /* FLOAT_TYPE */
   a_value_s     OUT     VARCHAR2)        /* VC40_TYPE */
RETURN NUMBER;
```

Parameter Calculations (2)

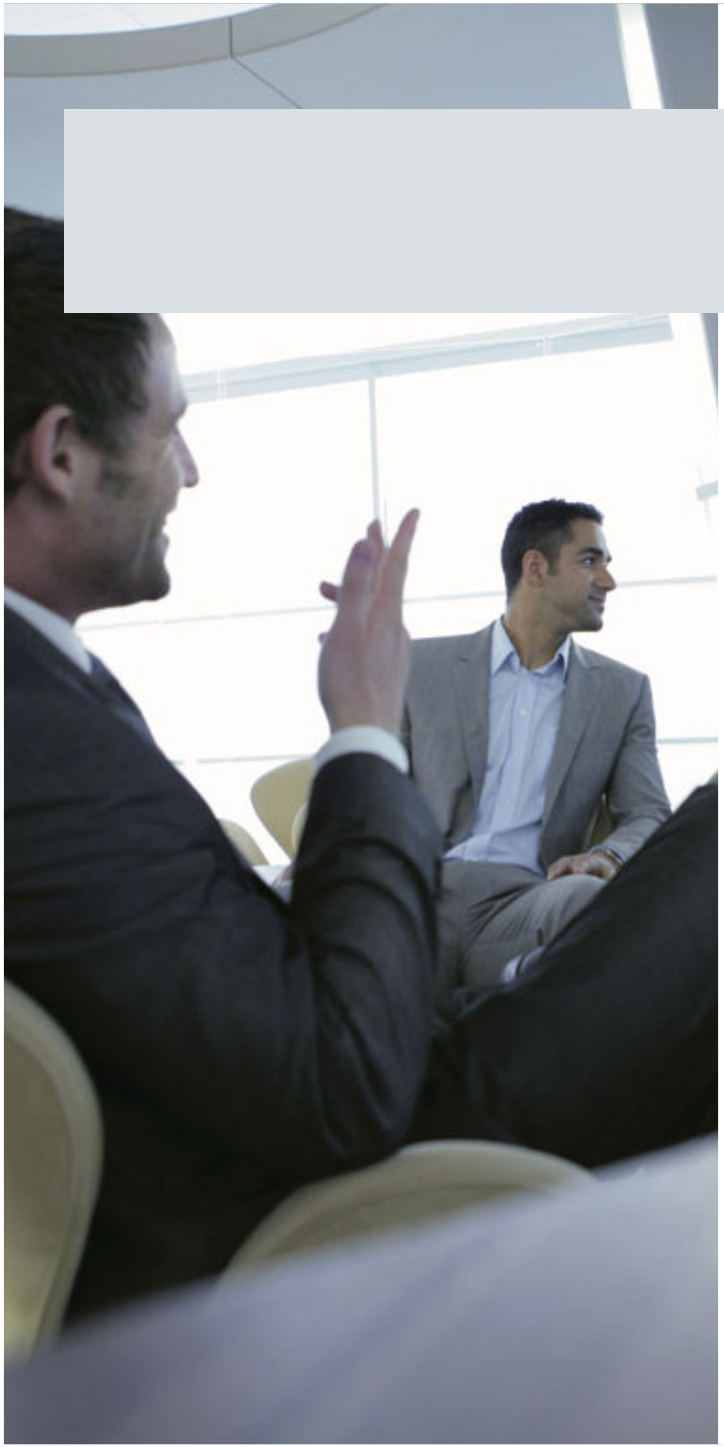
Return code

- DBERR_SUCCESS: result is calculated
  - OUT arguments will be saved as parameter result
- Otherwise: result not (yet) available
  - Nothing changed

Parameter Calculations (3)

Example

```
FUNCTION CalcMethod
(...) RETURN NUMBER IS
BEGIN
  SELECT MAX(value_f)
  INTO a_value_f
  FROM utscme
  WHERE sc = a_sc
          AND pg = a_pg
          AND pgnode = a_pgnode
          AND pa = a_pg
          AND panode = a_panode;
  a_value_s := a_value_f;

   RETURN(UNAPIGEN.DBERR_SUCCESS);
END CalcMethod;
```

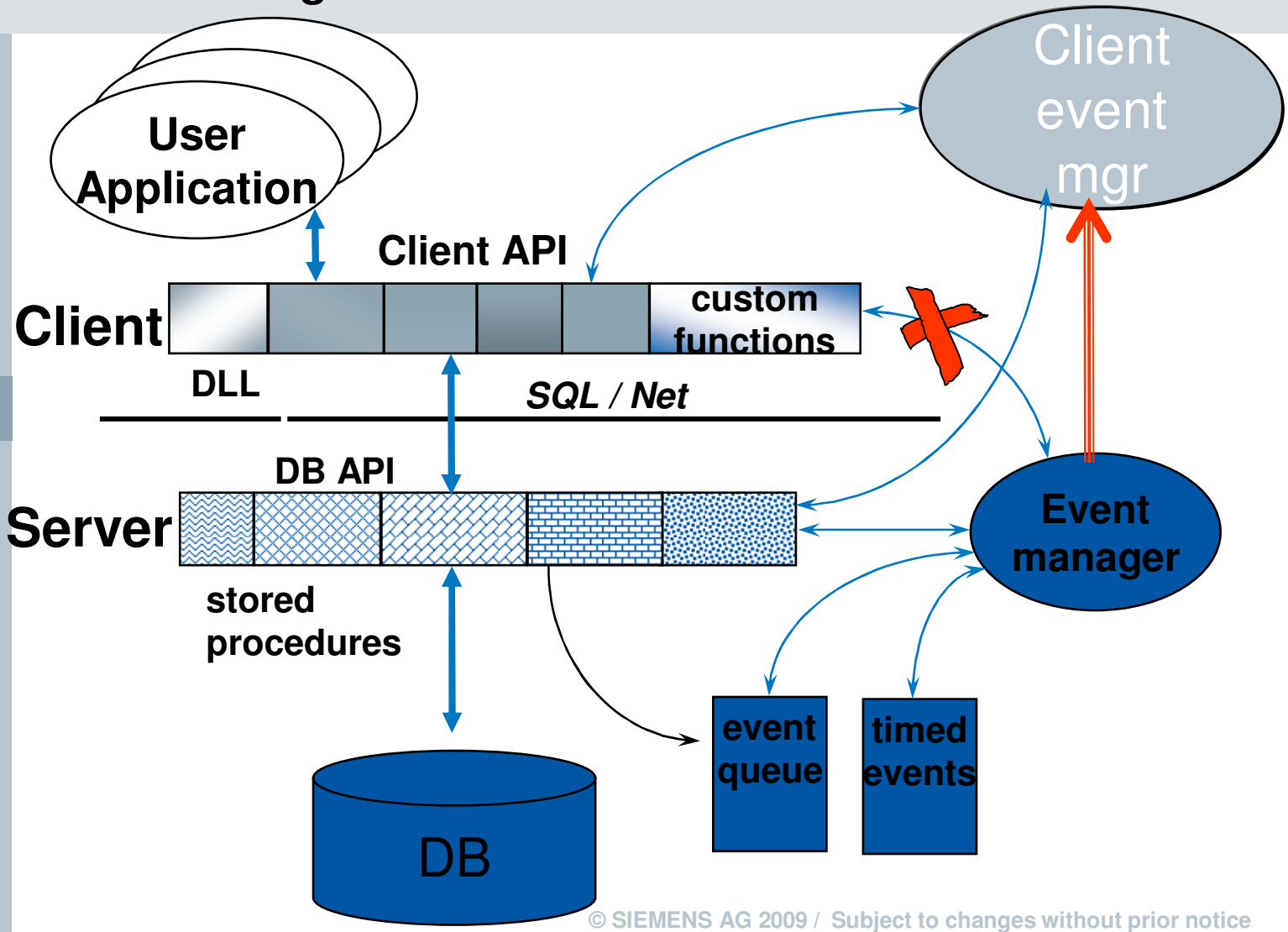


SIEMENS

Custom Functions

Client Event Manager

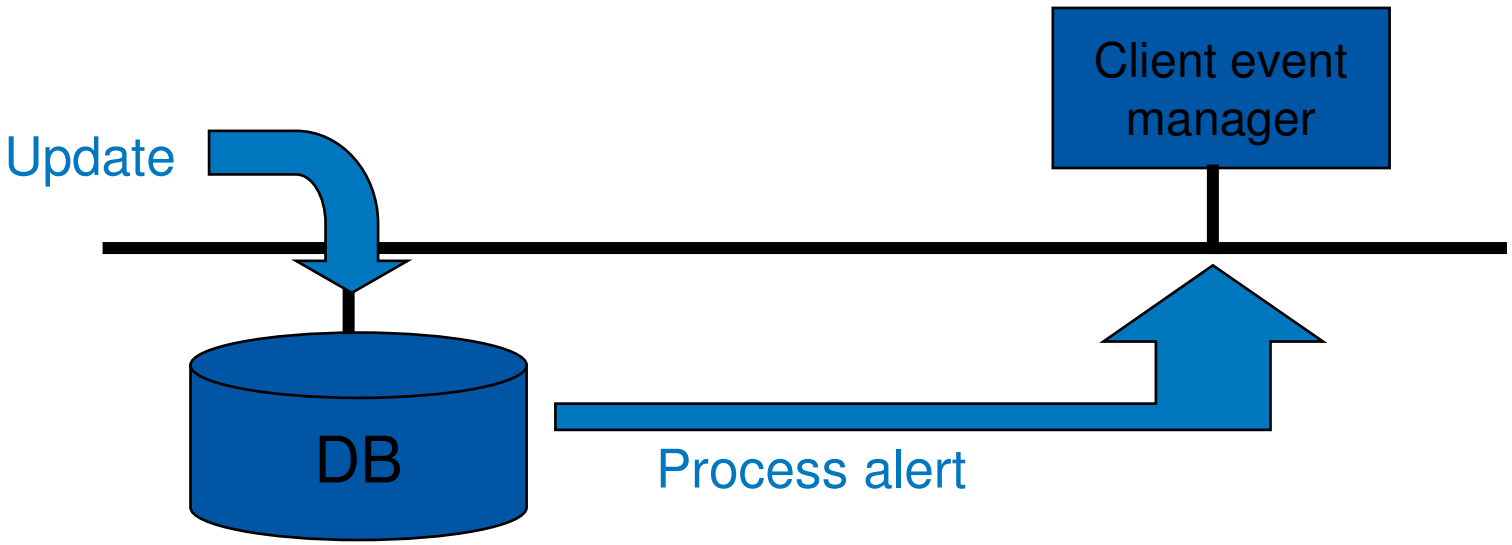
Distributed Event Manager



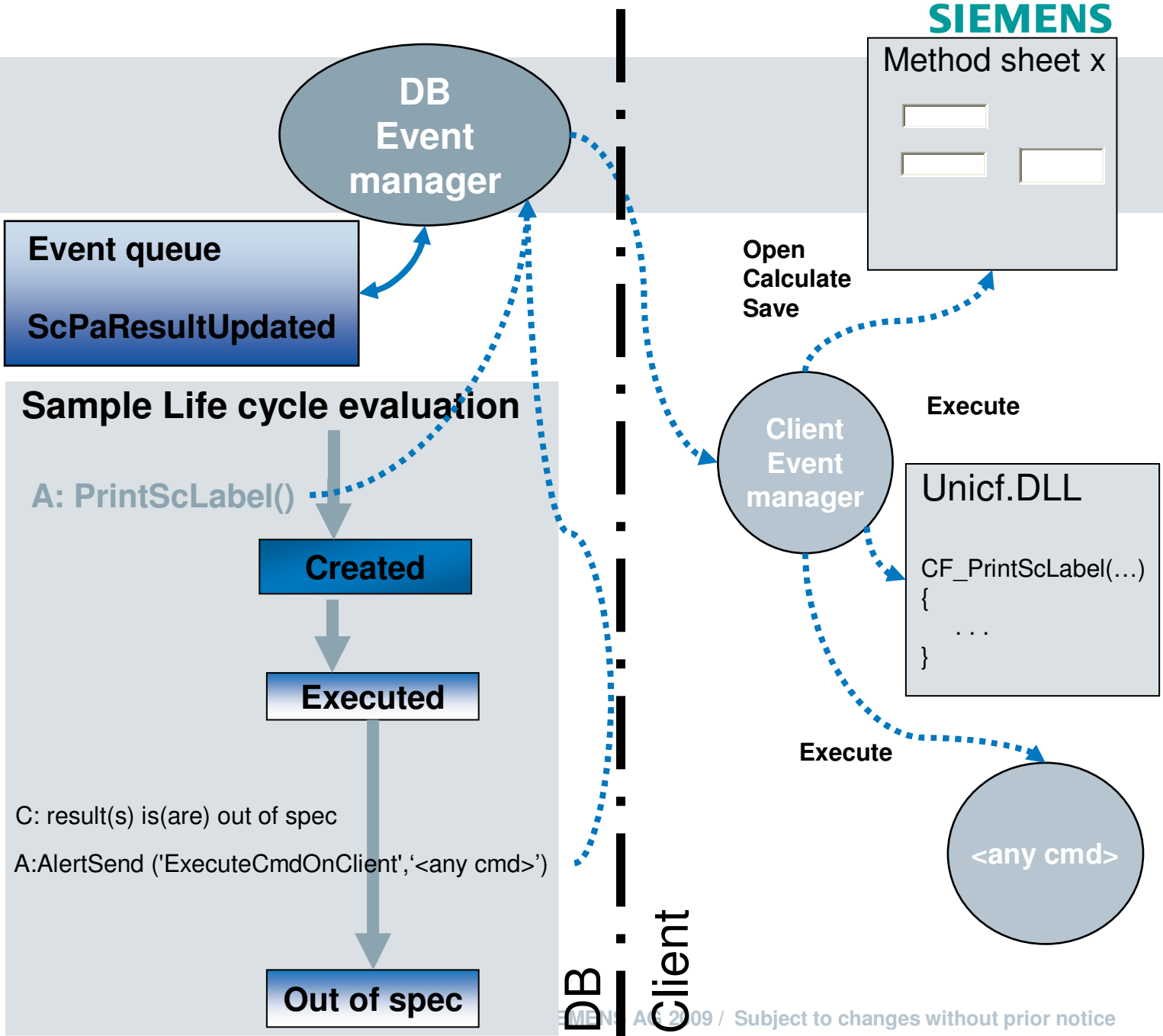
Concept of the Client Event Manager

Client Event Manager

- DB event manager sends alert to client event manager
- Remains “idle” while waiting

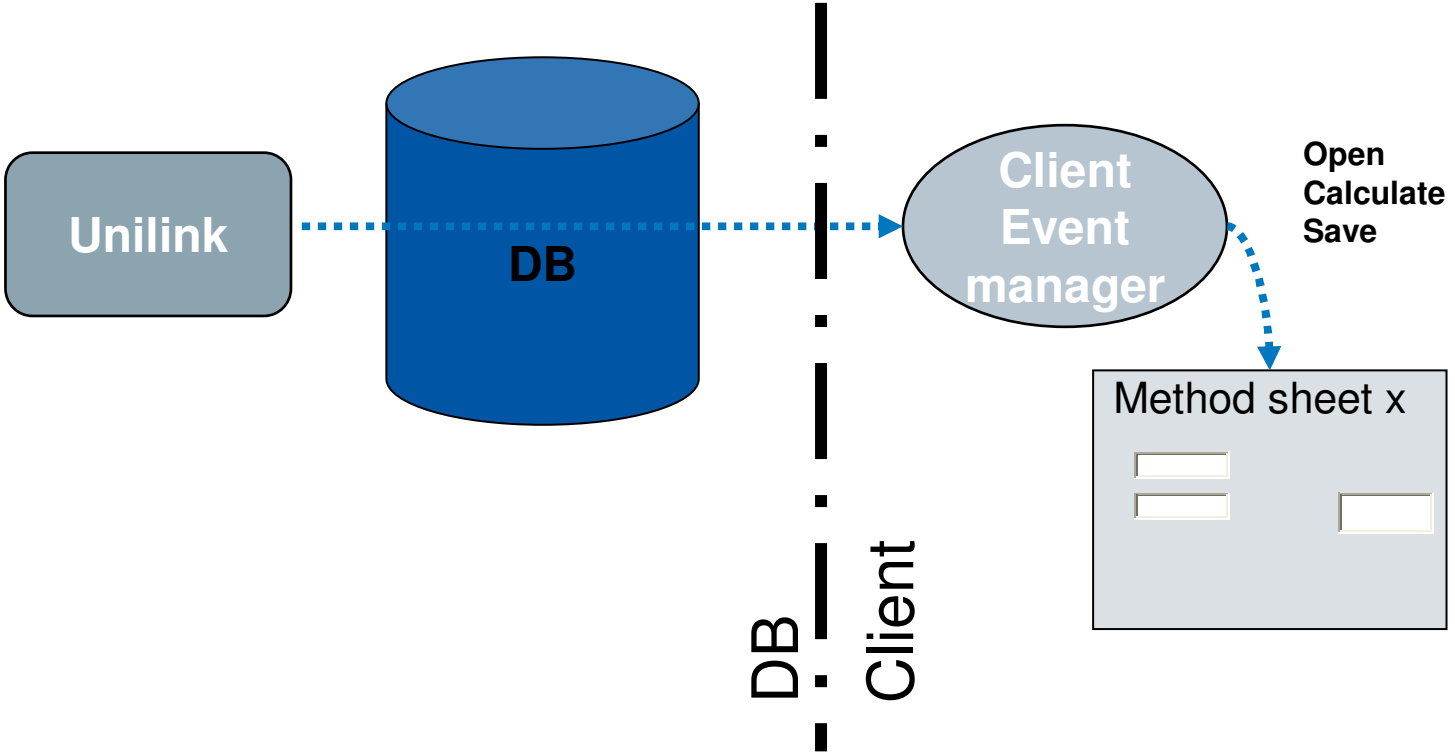


Example

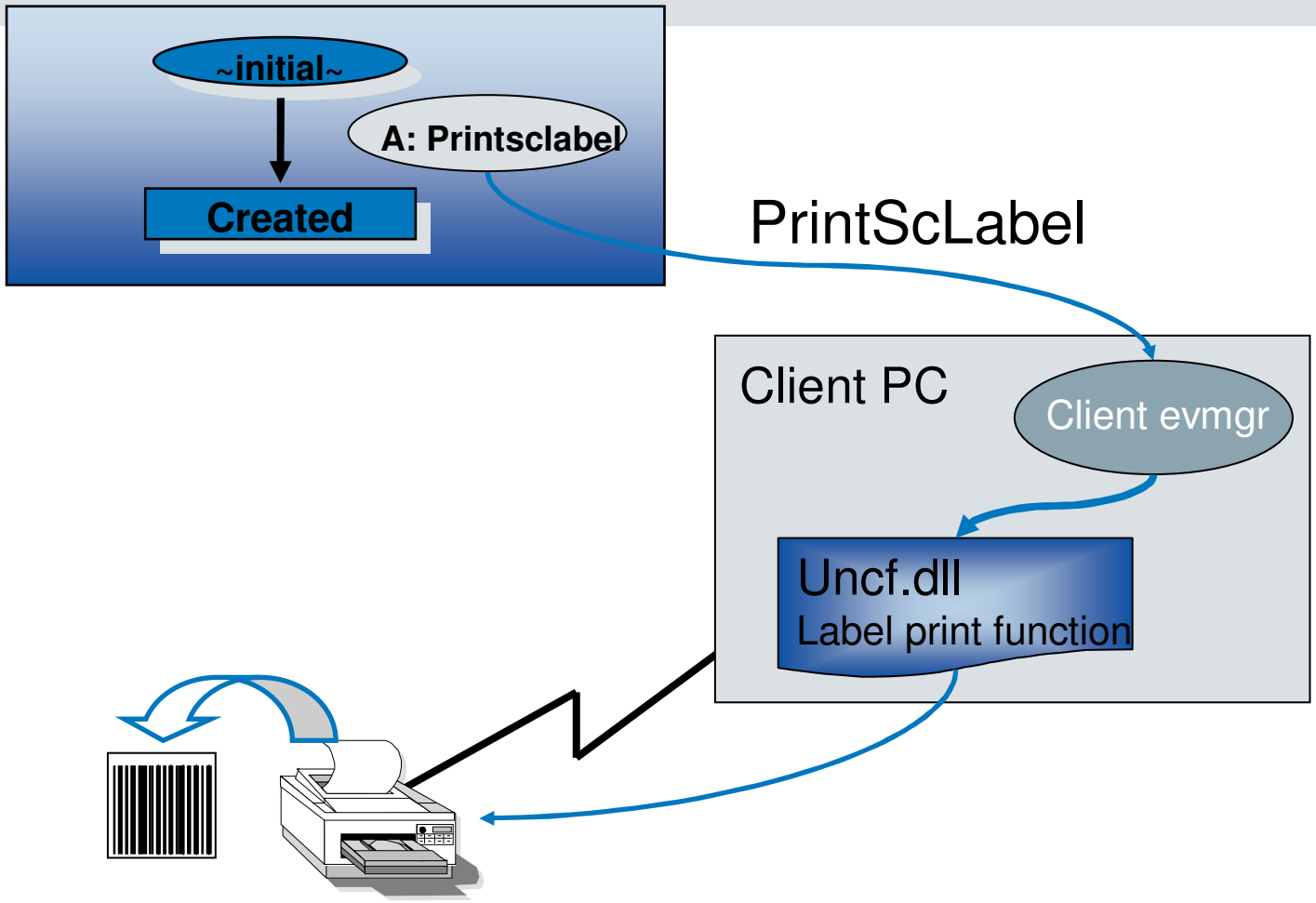


Update Method Sheet

- **Alert EvaluateMeDetails Example:**



Print Label



Execute Command

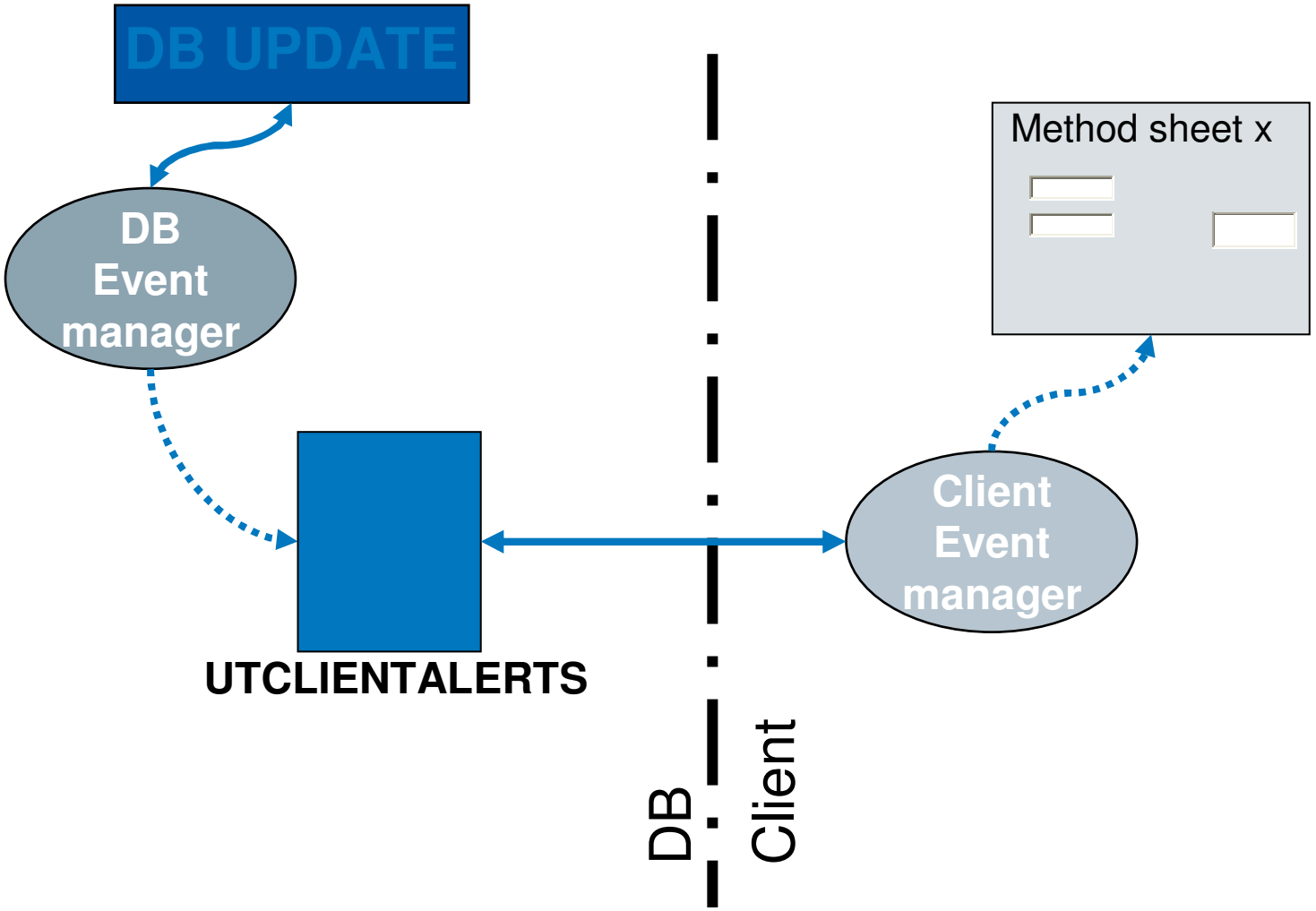
Syntax

```
L_retcode := UNAPIEV.Alertsend ('ExecuteCmdOnClient', 'l_alert_data')
```

- L_alert_data := 'cmd as in DOS box'

- Example
  - **l_alert_data := 'busobj -user ~BOUSER~ -pass ~BOPASS~ -script 'Distribute_sc' -nologo**

Client Event Manager - Mechanism



Running the Client Event Manager

Clevtmgr.exe

- Runs on 1 machine in the net
- Automatically minimized on startup

Settings

- In Registry
  - SqlError = False (not TRUE)
  - TimerInterval = 1000
- In system settings
  - Client_evmgr_used = Yes

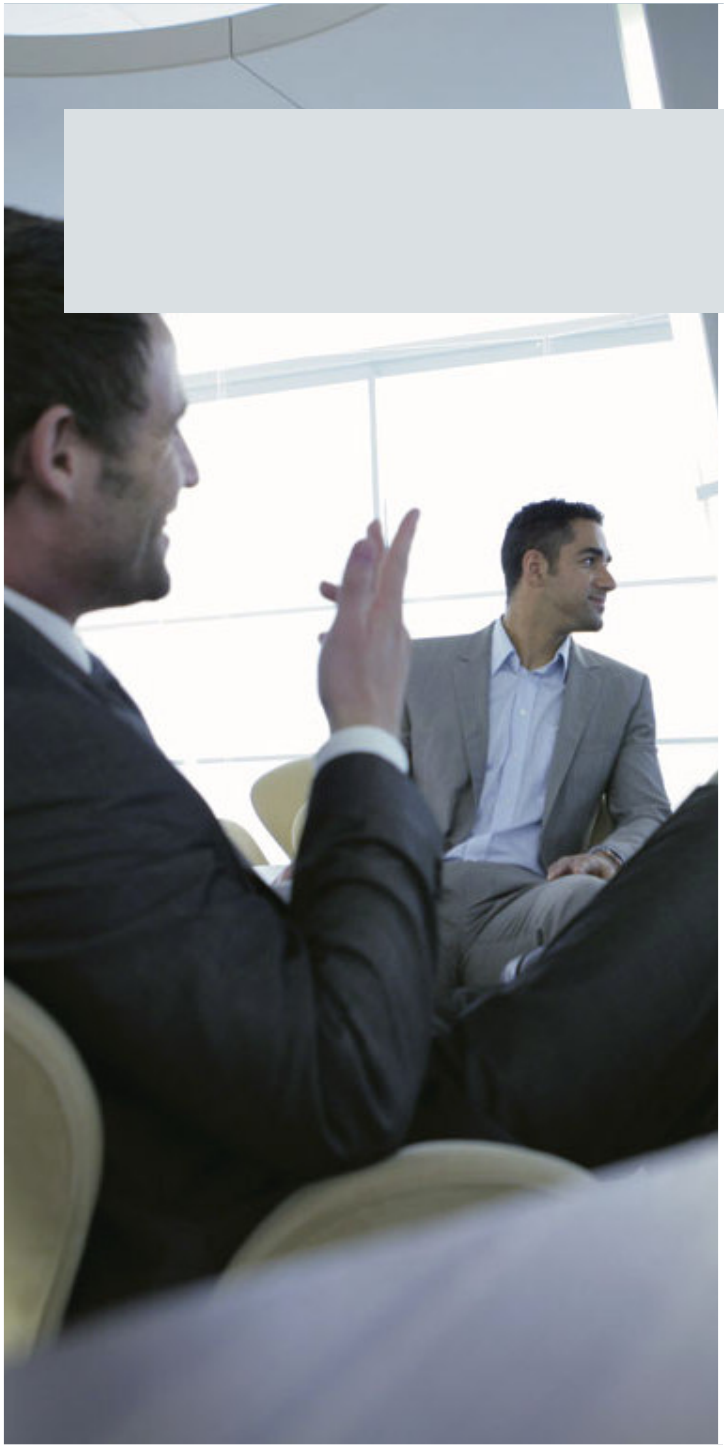
Running the Client Event Manager

Error Logging

- unerror.log
- In directory of Clevtmgr.exe

KeepDBConnectionOpen

- Client event manager keeps on running in case of missing DB

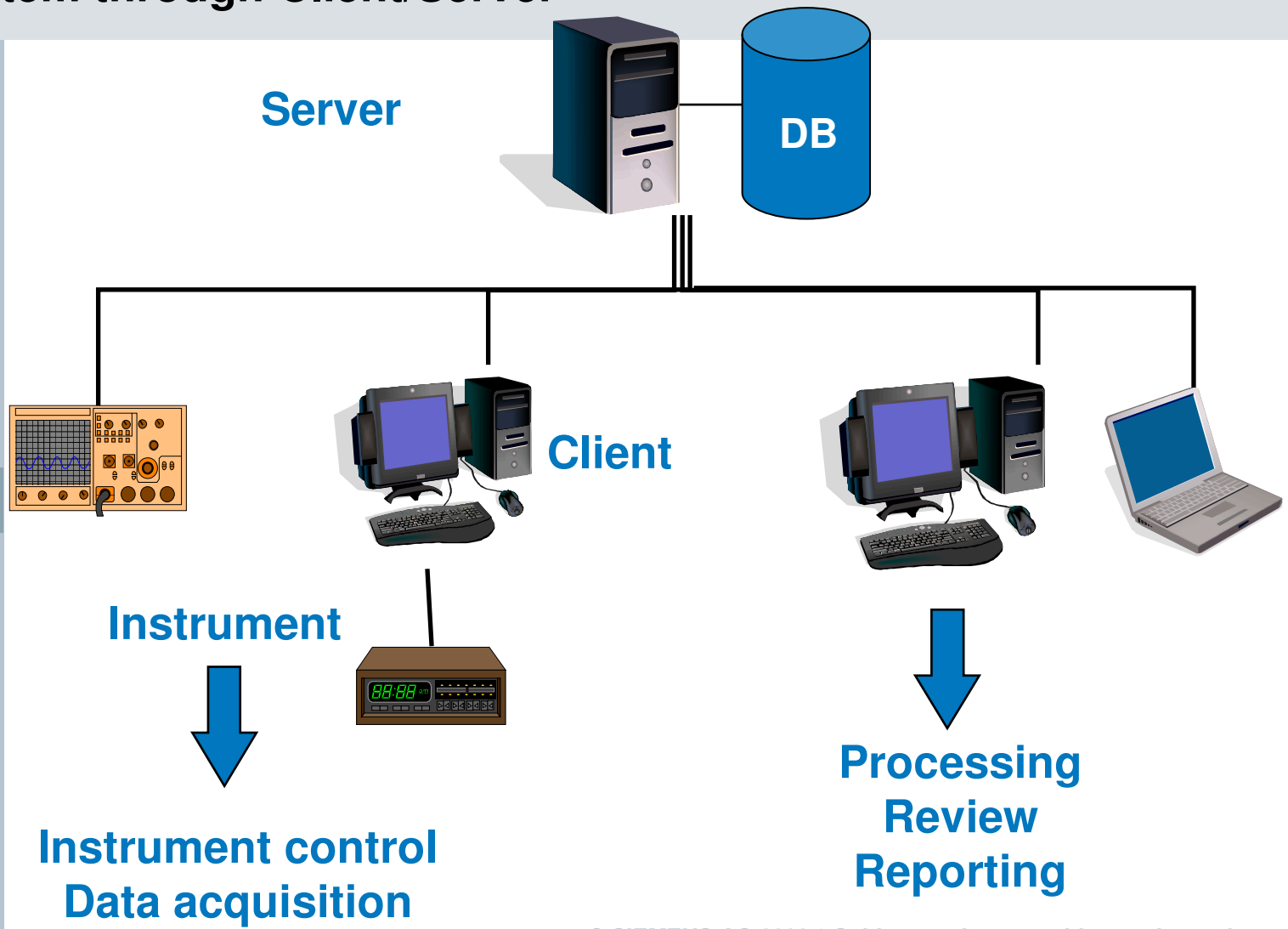


SIEMENS

Connecting Instruments

Overview

Open System through Client/Server



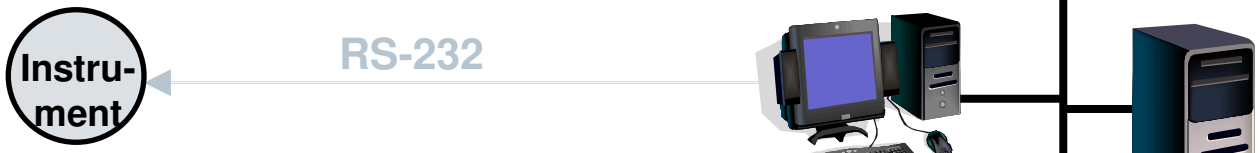
Advantages of Instrument Connections

Instrument connections lead to

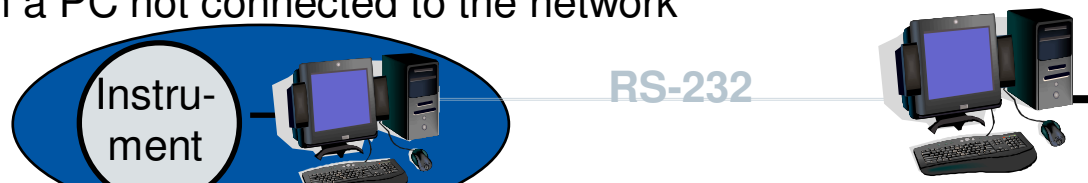
- More reliable results → Quality Improvement
- Less manual labour → Cost Reduction

3 Different Types of Instrument Connections

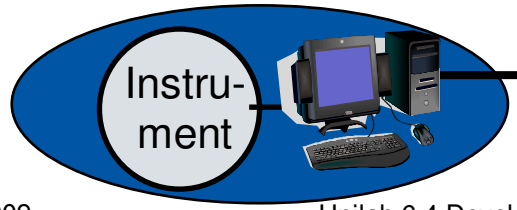
- Instruments with output on RS-232 or printer port



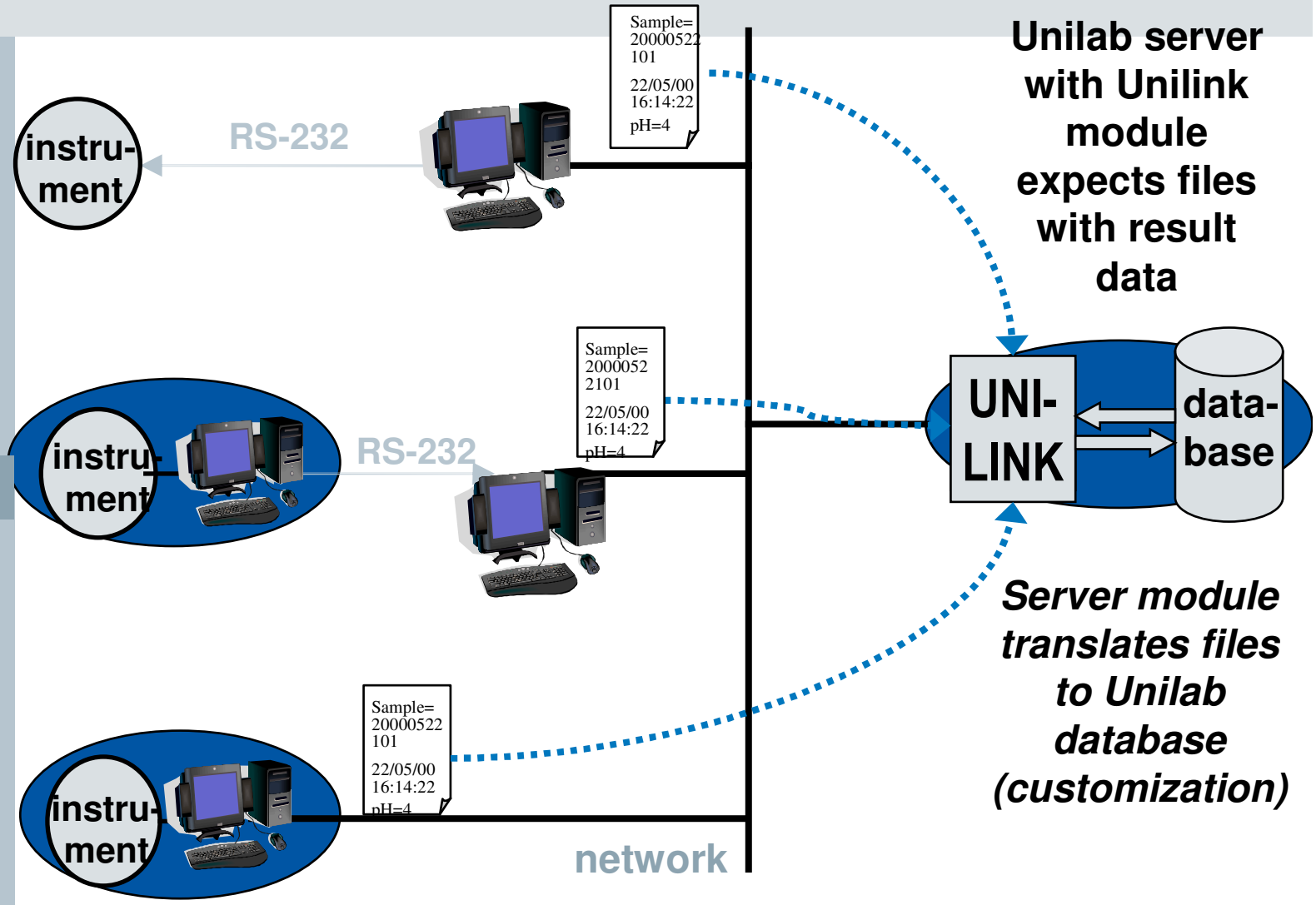
- Instruments that deliver files
  - on a PC not connected to the network



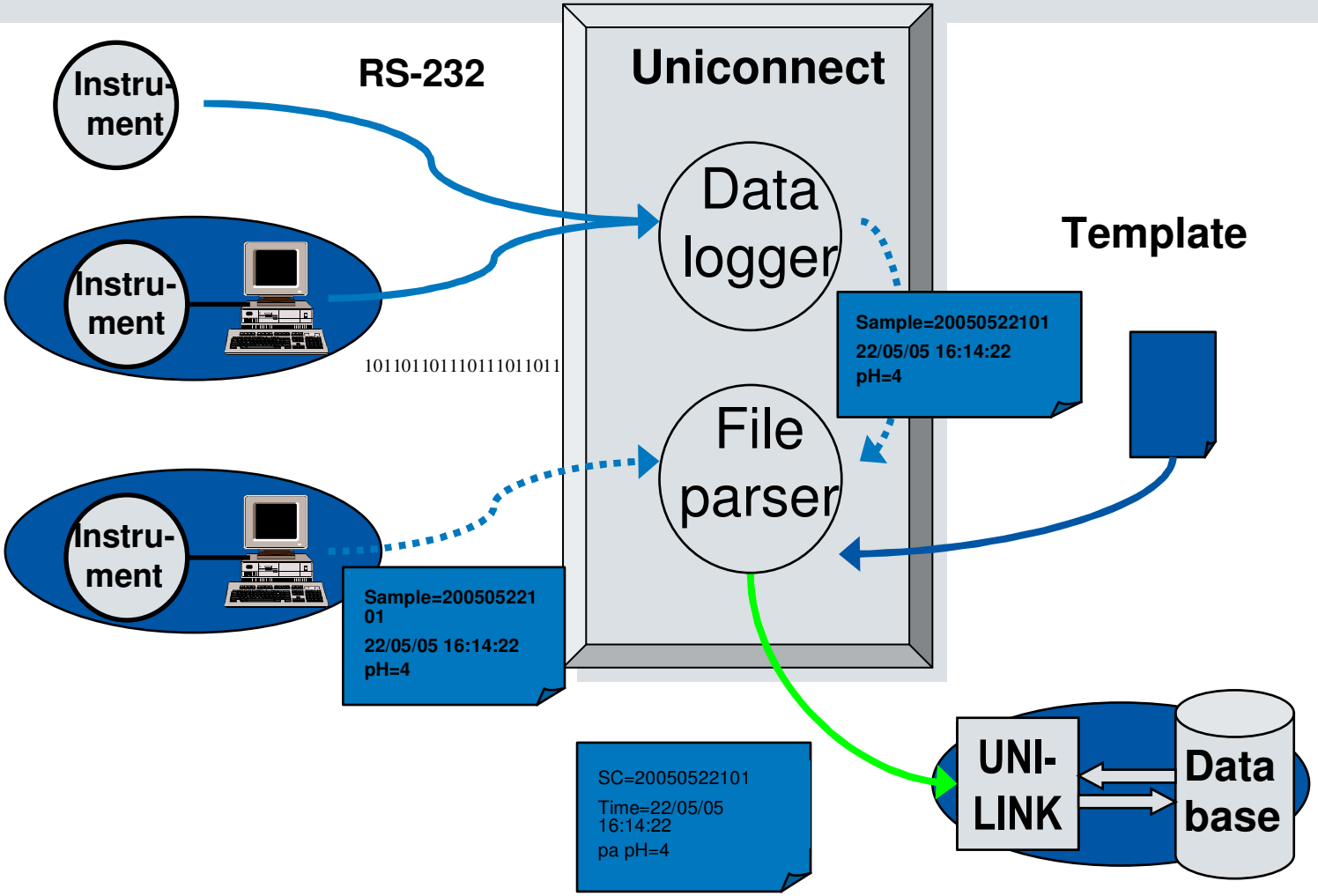
  - on a PC connected to the network



On Server Side - Unilink



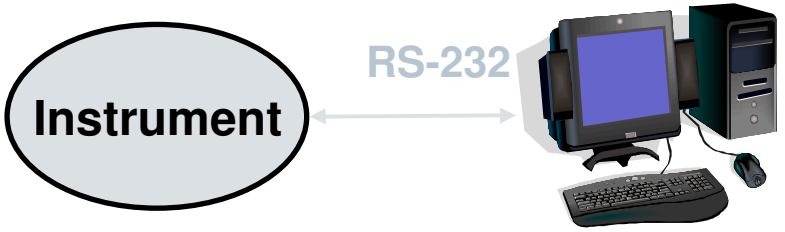
Uniconnect - Unilink



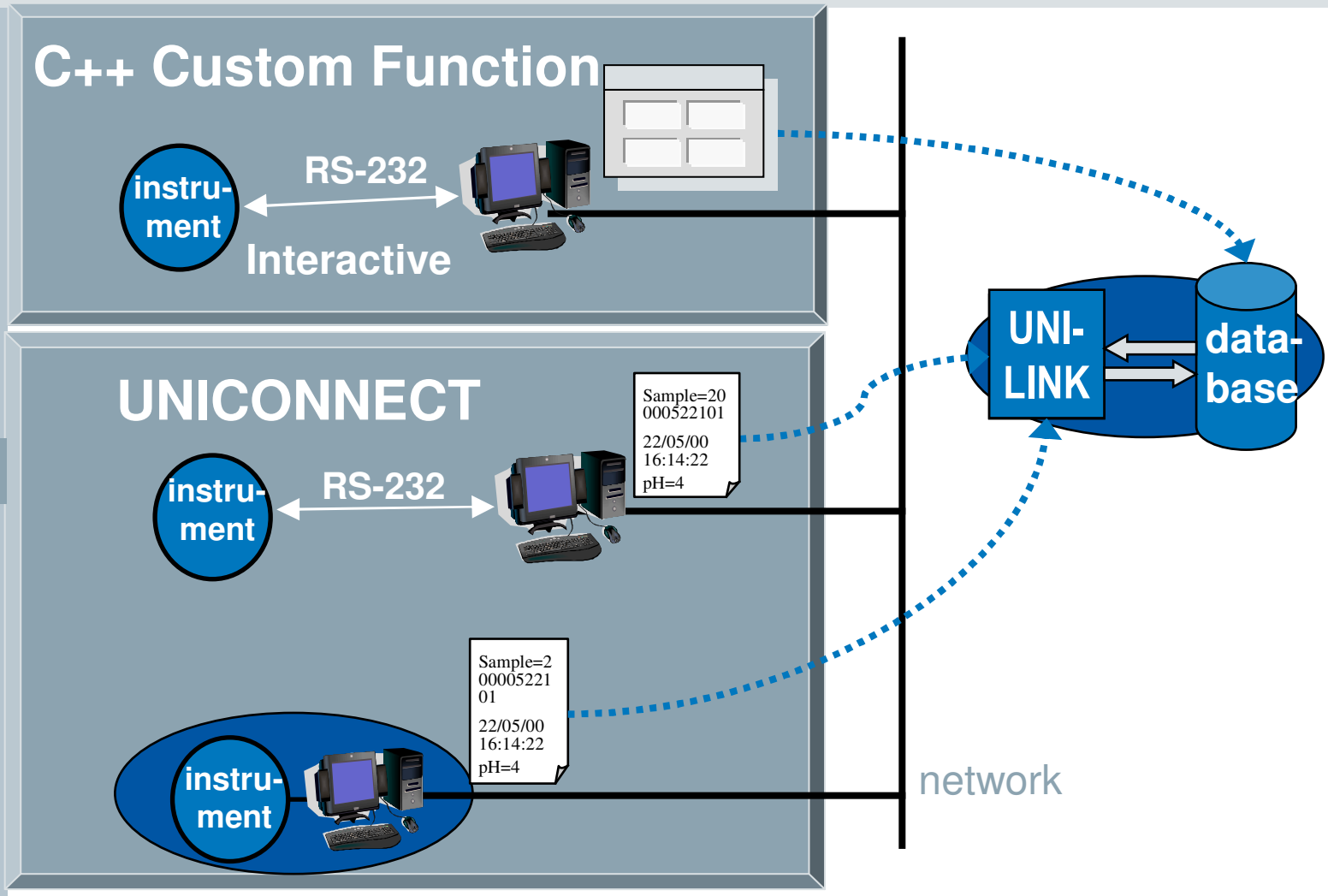
Point-to-Point Connections

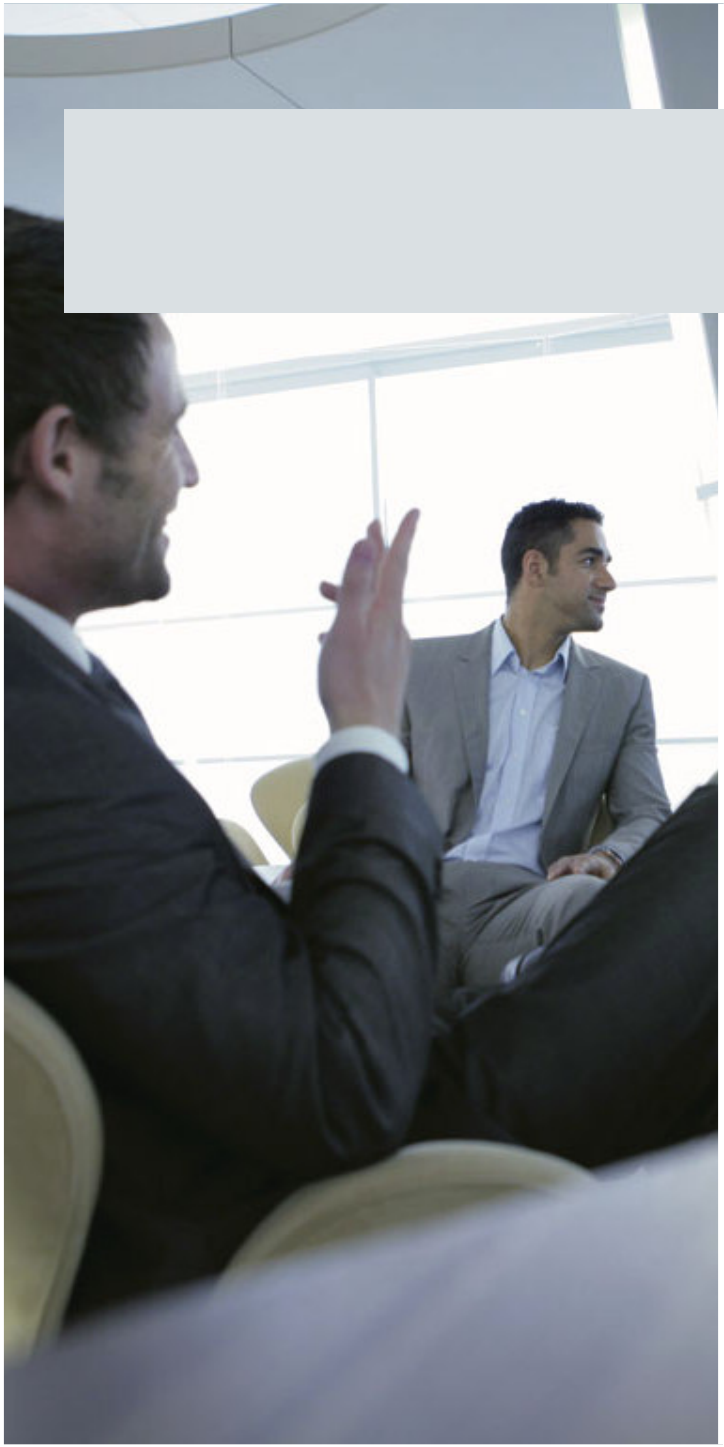
- To method sheet
  - Interactive
    - pH probe, balance,…
  - Connections using C++ custom function
  - Save to DB using Simatic IT Unilab functionality

- To PC
  - No interaction from user required
  - Processing in background
  - Uniconnect - Unilink



Connecting Instruments - Overview





SIEMENS

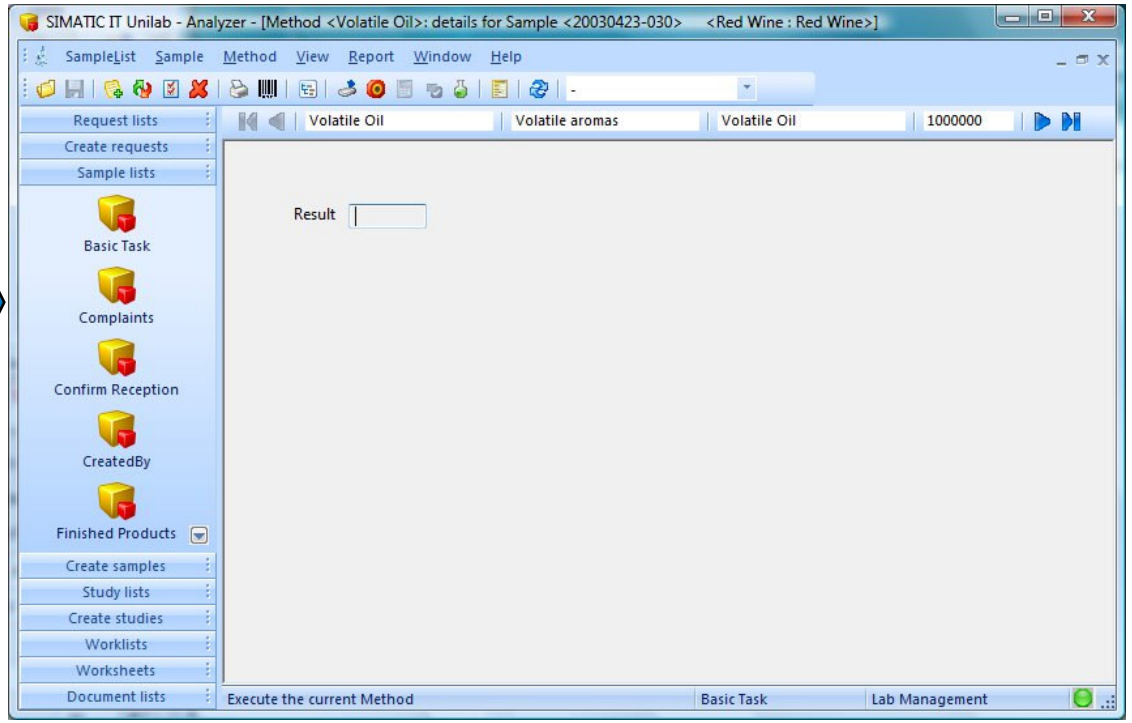
Connecting Instruments

Point-to-point connections

Point-to-Point Connection from Method Sheet

Interactive connections from method sheet

Connection through C++ custom functions

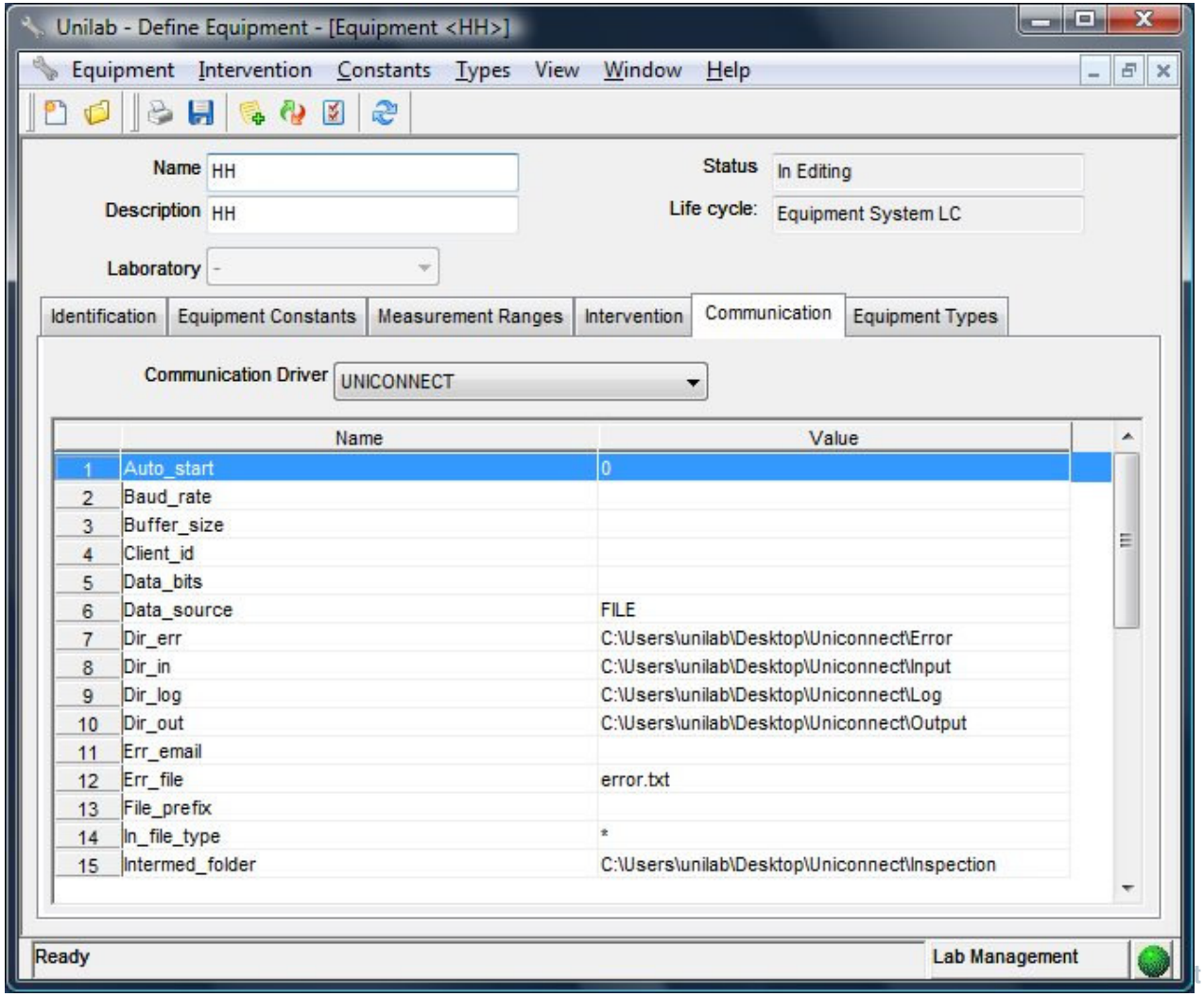


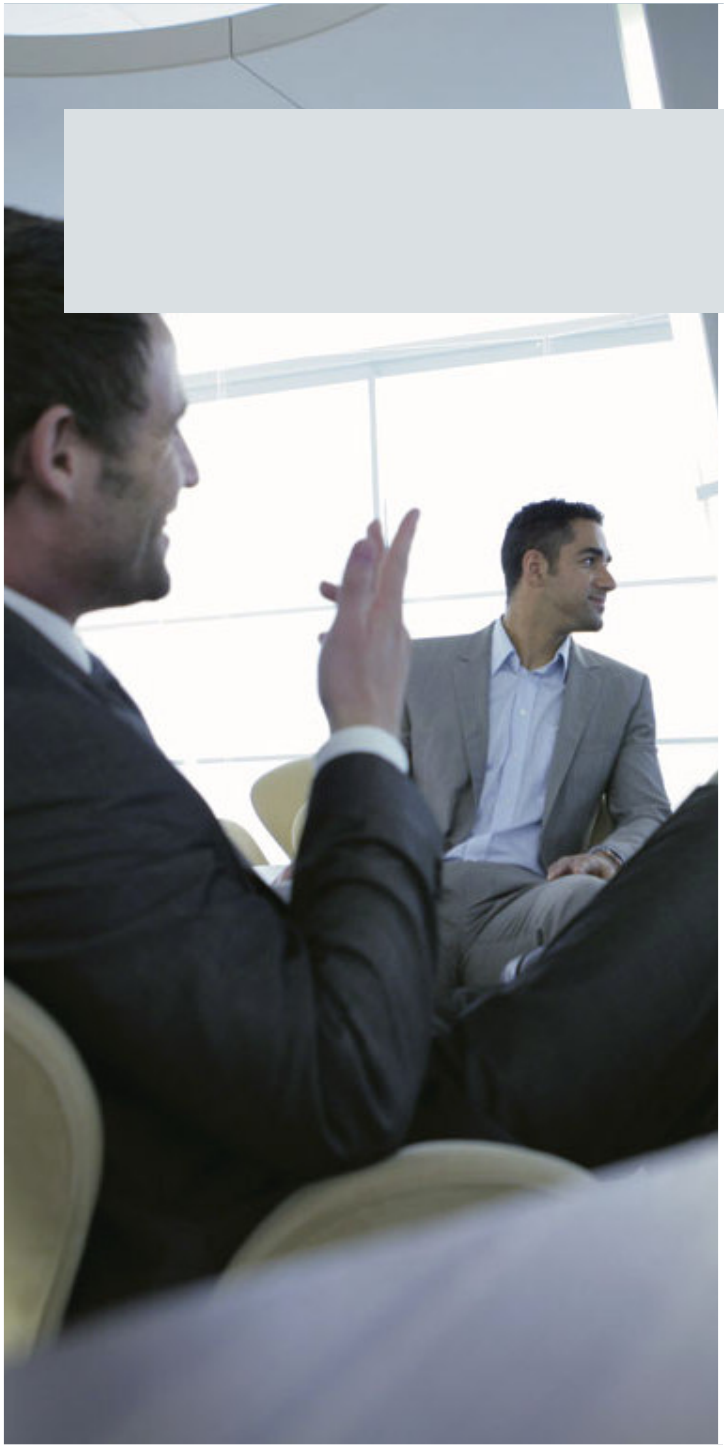
Setup

- C++ custom functions for instrument connection
  - In CF_Measure( ) function
- utcf
  - No entry needed in utcf
  - Possibility to specify additional interface parameters

```
CF_Measure (CMece* mece, const char* EqName,
            const char* customFunctionName)
```

Setup (2)





SIEMENS

Connecting Instruments

Uniconnect

Uniconnect

Introduction

Execution Client

Configuration

Template files

Examples

Uniconnect

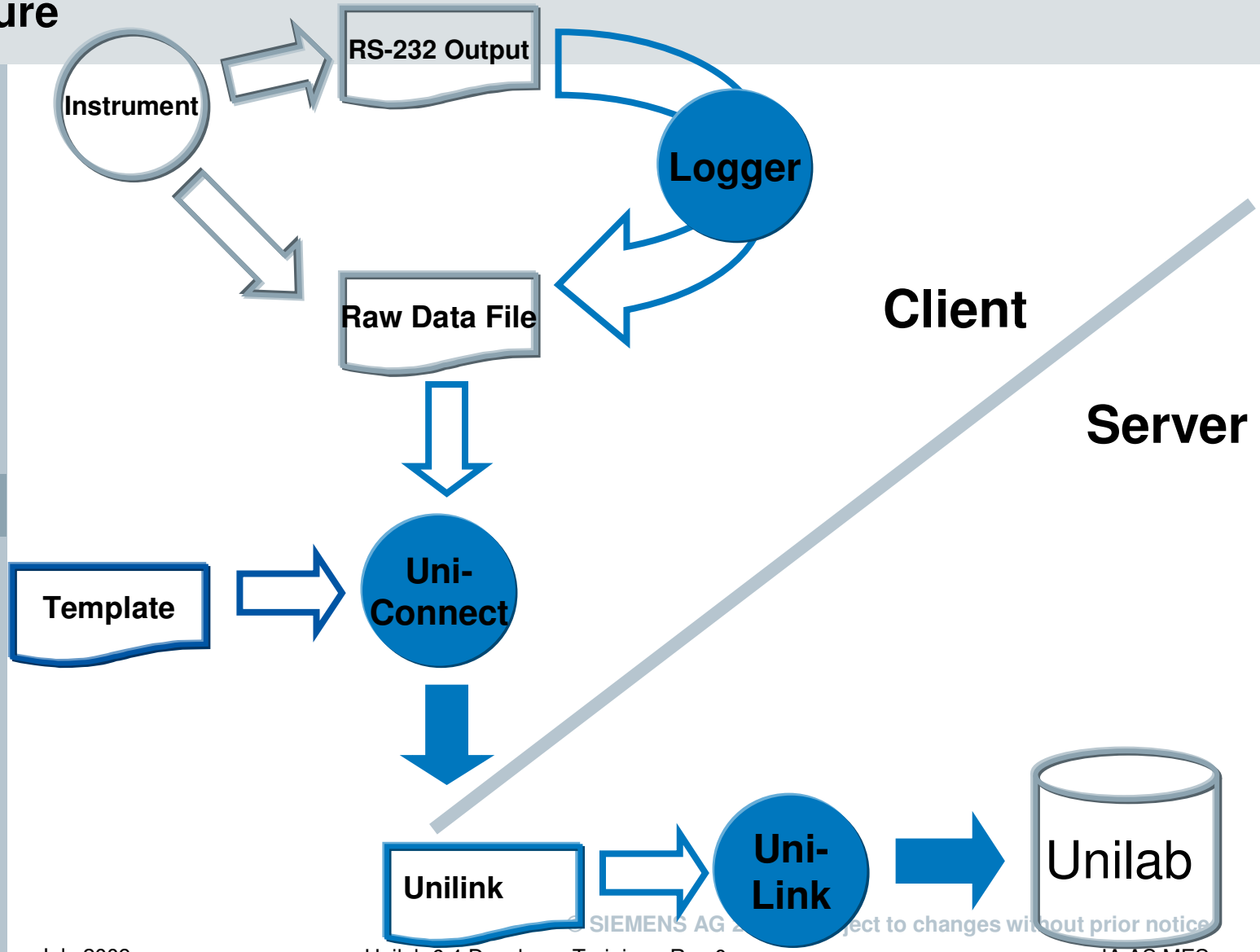
Purpose

- Front end module for Unilink
  - Client software + Unilink configuration
- Instruments producing ASCII file or sending ASCII characters via RS232

Environment

- One dedicated PC
- Simultaneously on multiple clients
  - Spread of workload

Architecture



Application Architecture

Configuration window

- Setup of equipment
- Stored in the database

Execution client

- Performs operational connections
- Runs as background job

Uniconnect

Introduction

Execution Client

Configuration

Template files

Examples

Uniconnect configuration in the database

Connection to the database is required:

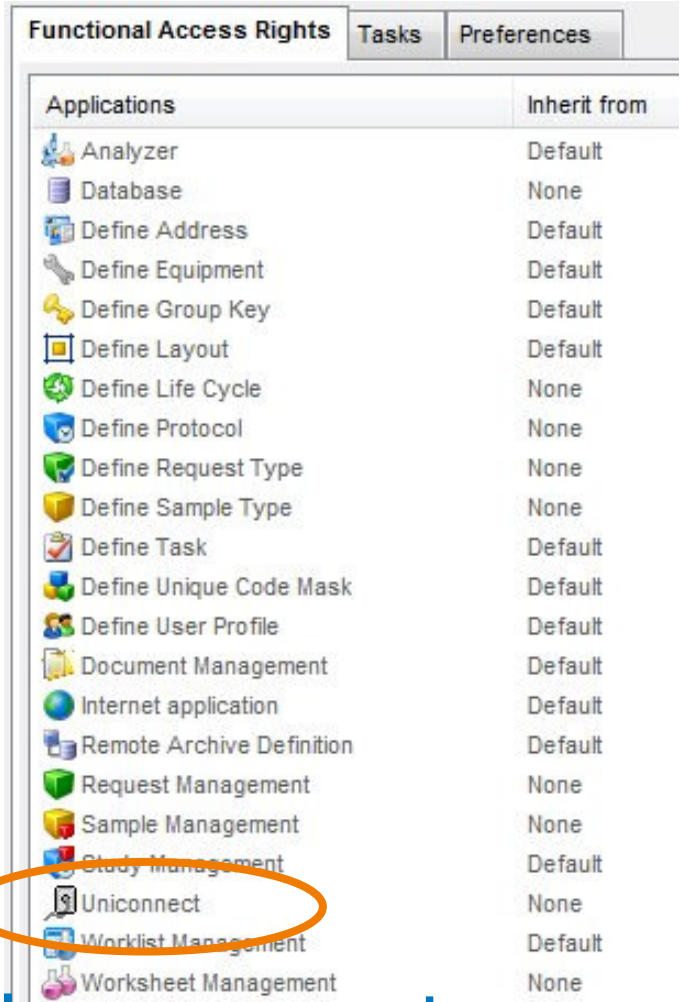
- At start-up
- At log-off
- When making changes to the configuration options

Retrieves required information from database and then continues working without any database connection

Following Uniconnect configuration information is retrieved from the database:

- Instrument list
- Communication settings of each instrument
- Lookup tables

Logon procedure – Functional Access Rights



User management

User authorization

Information stored locally after authorization check

The following information is downloaded from the DB and stored locally:

- **List of authorized users but connection is needed to logoff**
- **Functional Access Rights to change the instrument's properties (configuration)**
- **Instrument list**
- **Communication settings of each instrument**
- **Lookup tables**

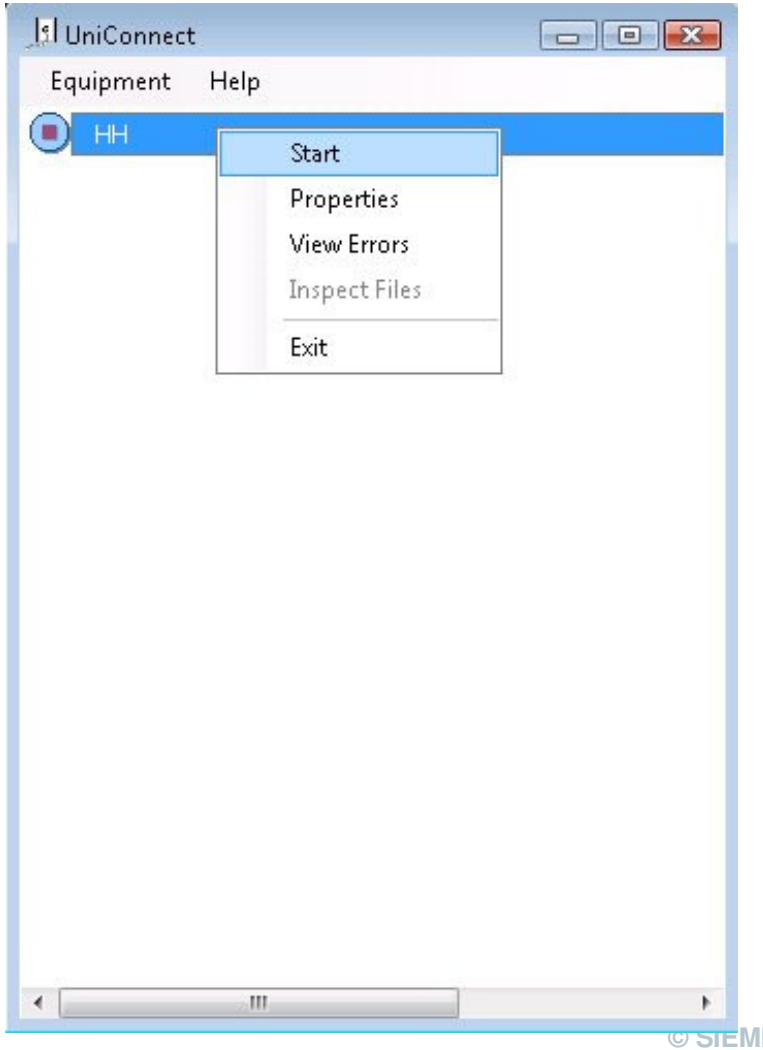
Then the database connection closes and

Uniconnect operates independently

Logoff procedure

- Same user that started Uniconnect?
- User has Functional Access Rights to start or stop Uniconnect?

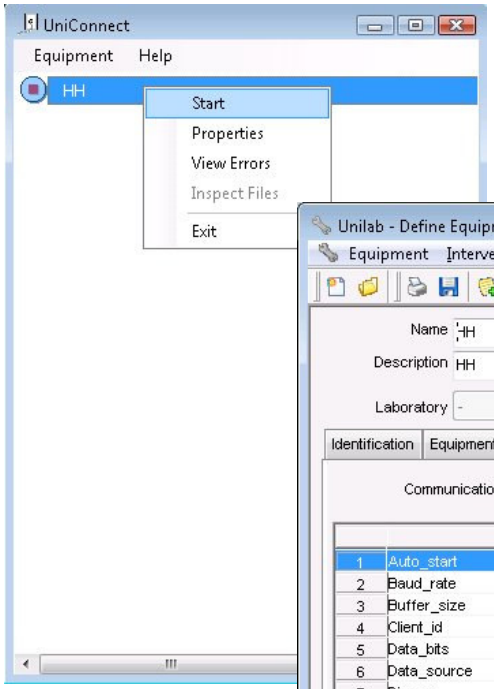
Execution Client



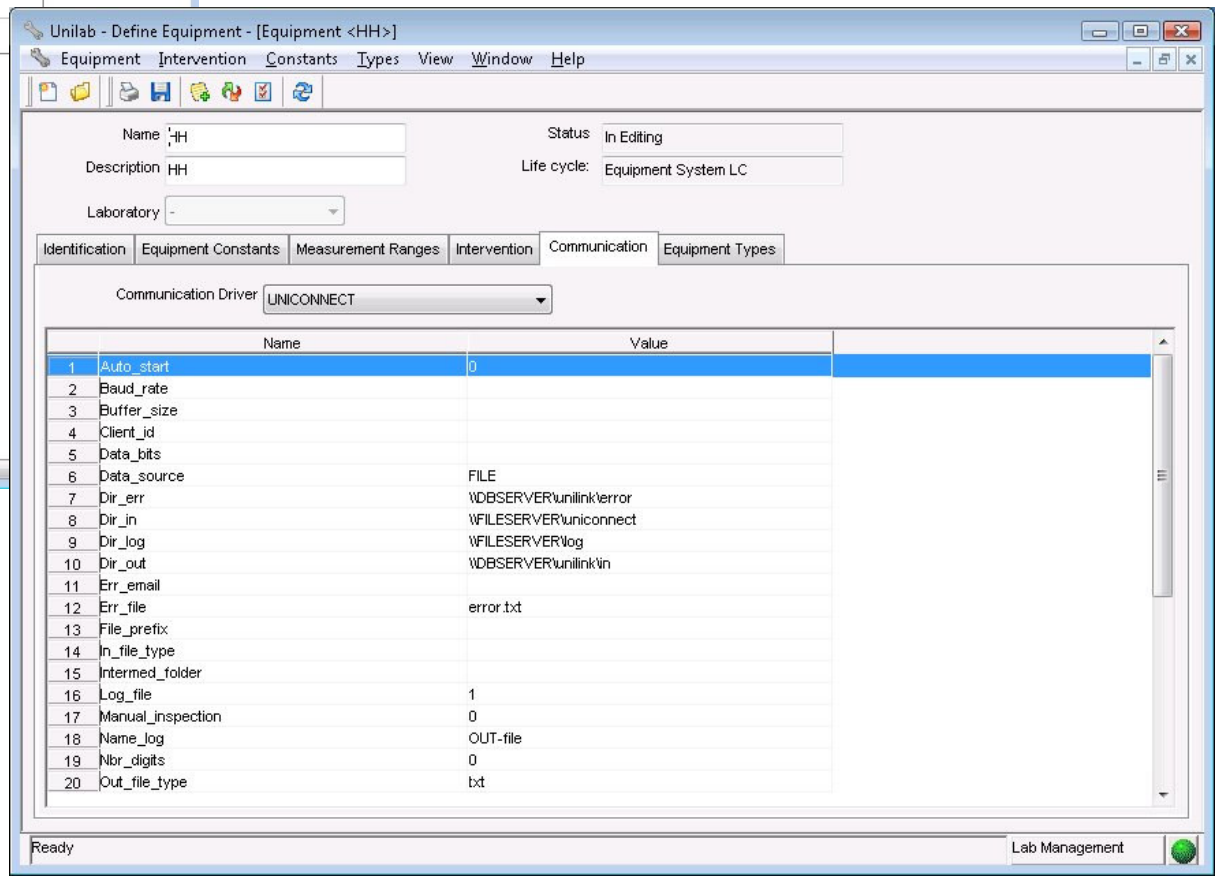
- Start/stop connection
- Change configuration (if allowed)
- View error status
- Inspect files

Local to 1 PC !

List of instruments



The list of available instruments depends on the protocol chosen in the “Define Equipment” application



Status Equipment

Instrument is not running/not started

Instrument is running

Processing error occurred

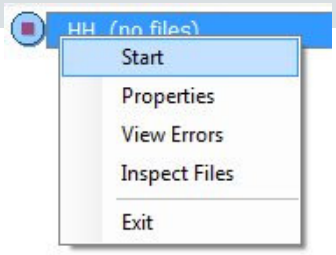
- Instrument is still running
- Check error file

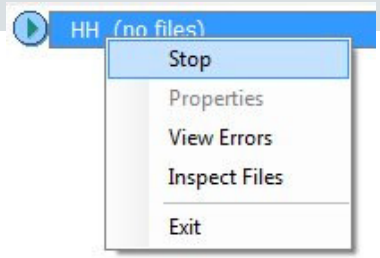
Configuration error occurred

- Instrument is stopped
- Check error file

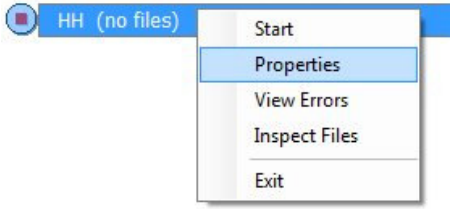
Processing

- Start/Stop equipment

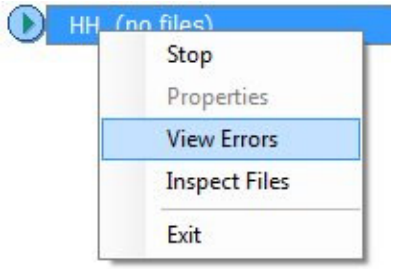




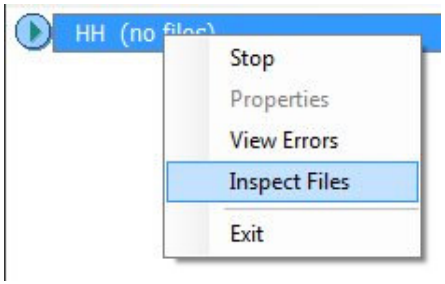
- Properties – Configuration (if allowed)



- View error file

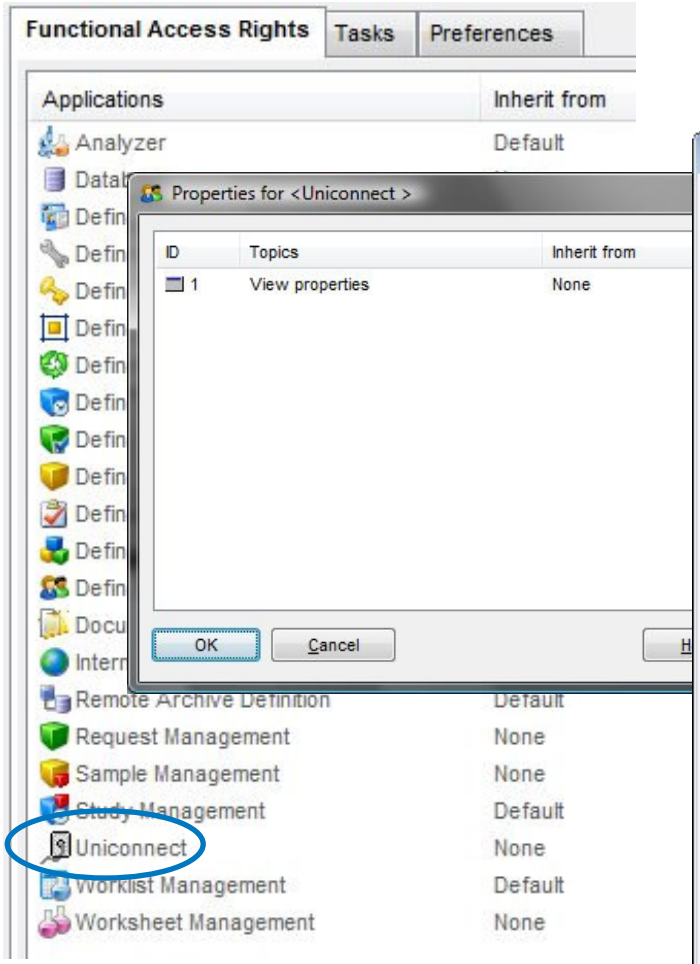


- Inspect files

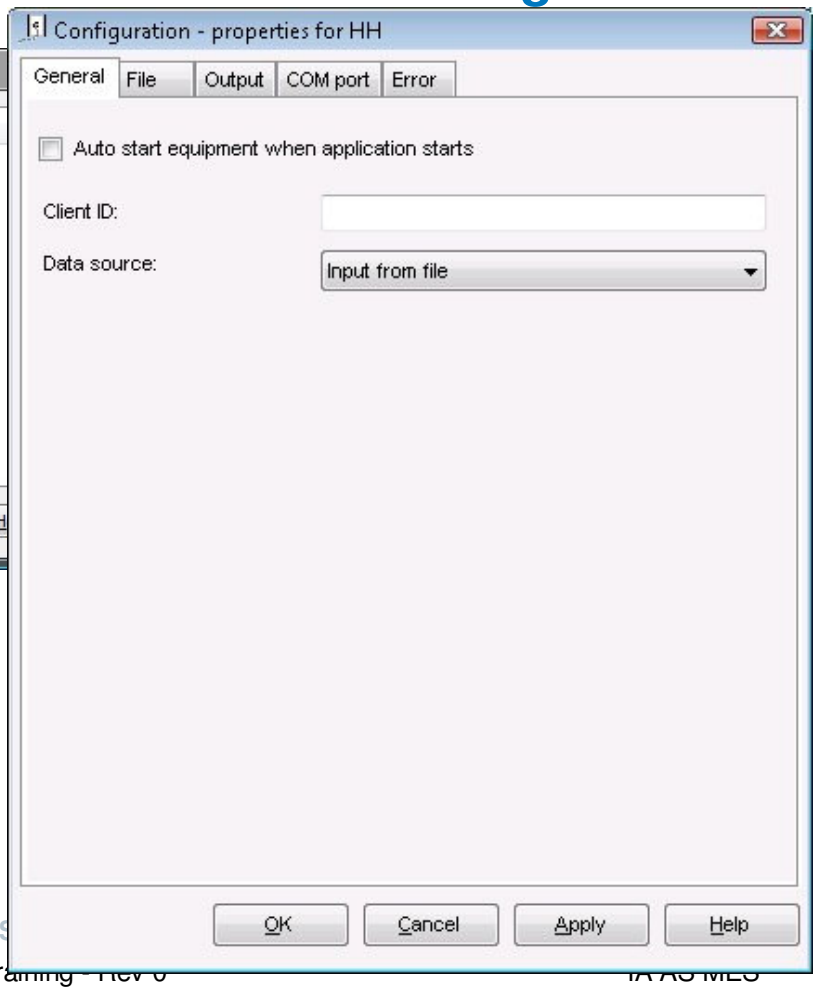


Uniconnect

Configuration – Properties tab



Configuration



User management

Configuration

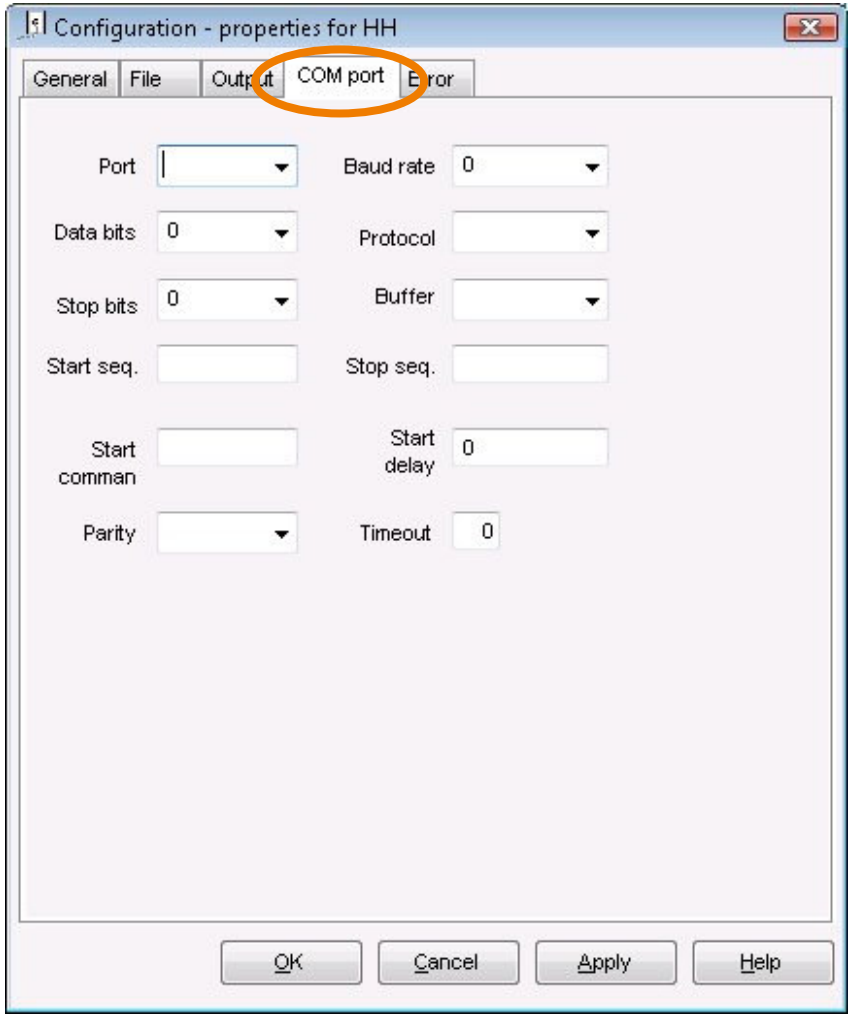
Directory structure

- IN: Raw Data Files
- OUT: Unilink file
- ERROR: Raw Data Files in case of error
- LOG: Copy of Raw Data Files
- TEMP: Temporary folder for post-processing
- INTERMEDIATE: Intermediate folder for manual inspection

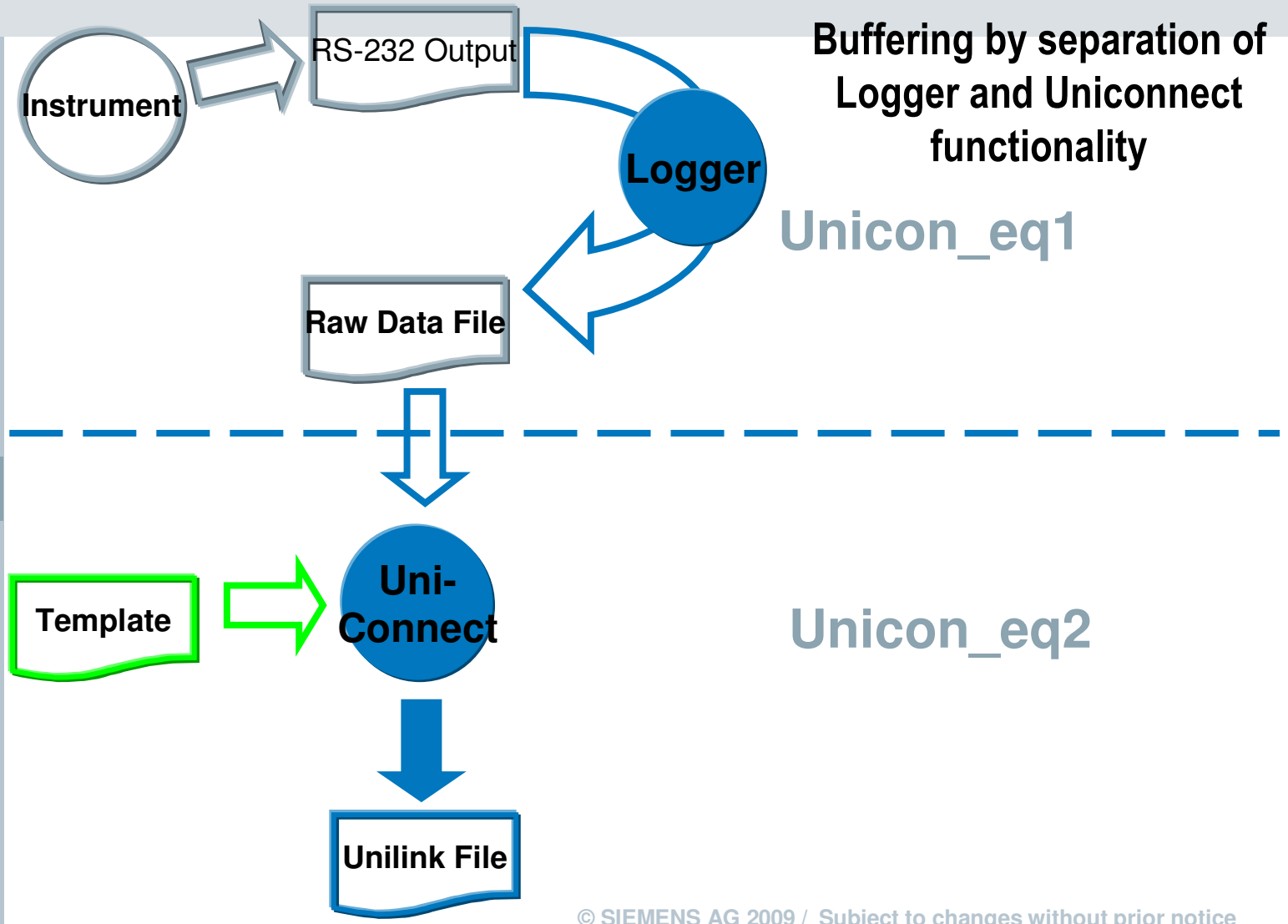
- Configurable per instrument
- Can be on different machines

Setup is stored in the database

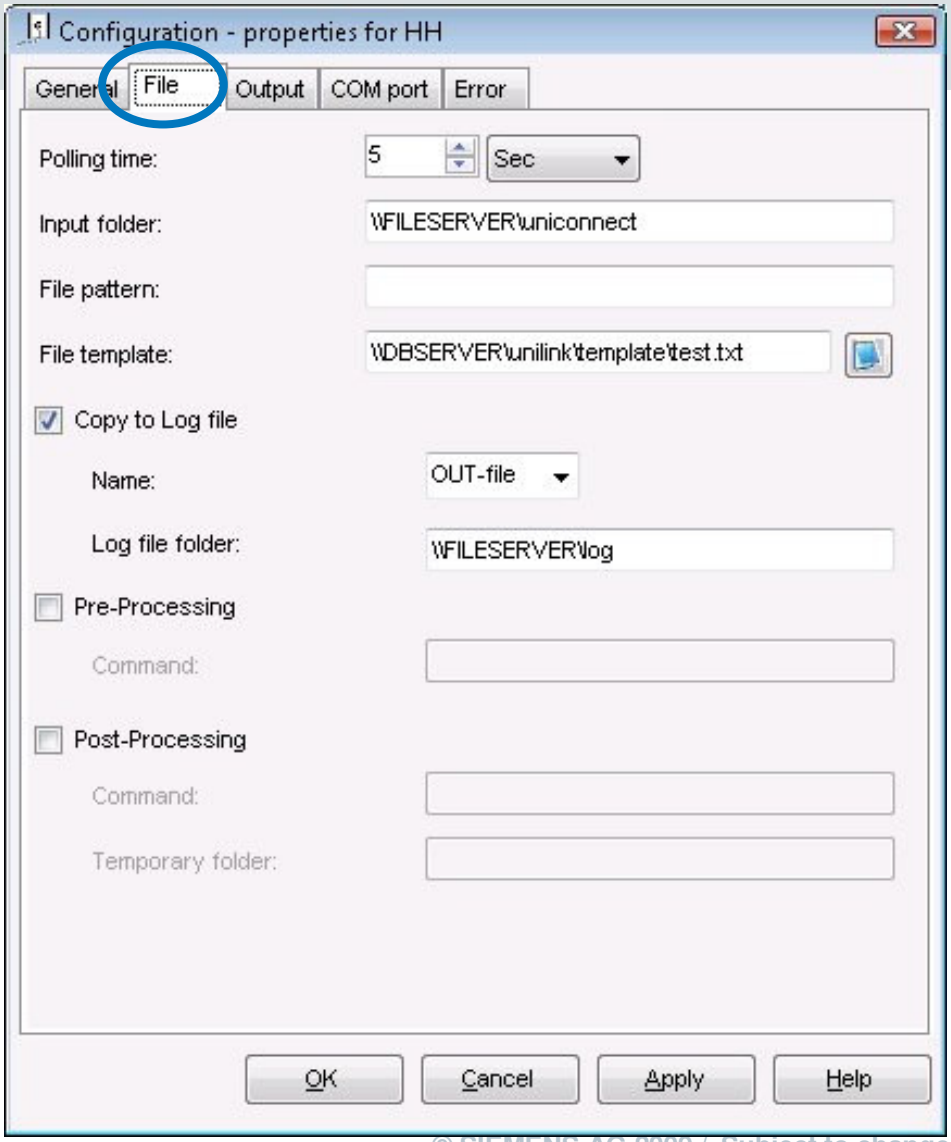
Input from COM Port



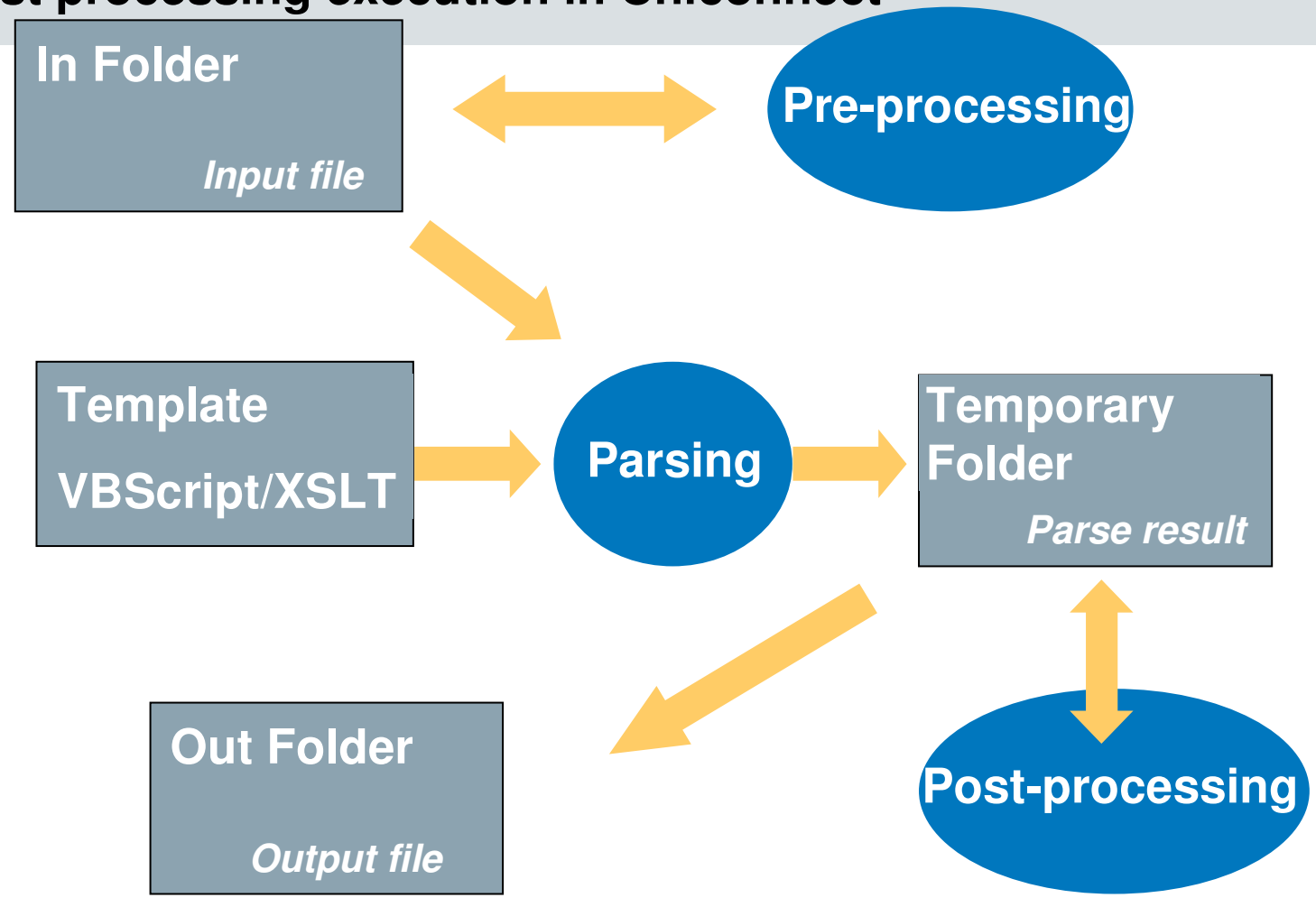
Buffering in Case of Error



Input from File

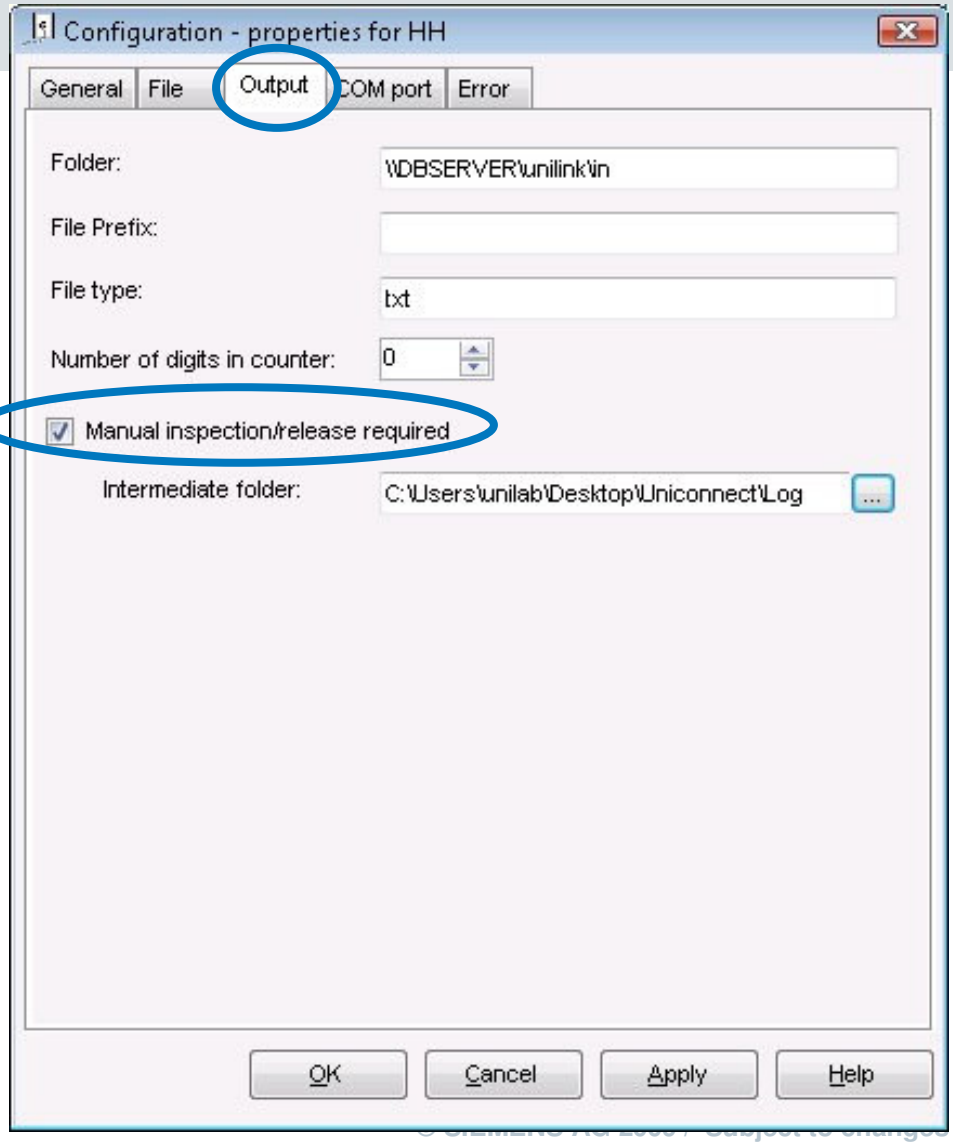


Pre and post processing execution in Uniconnect



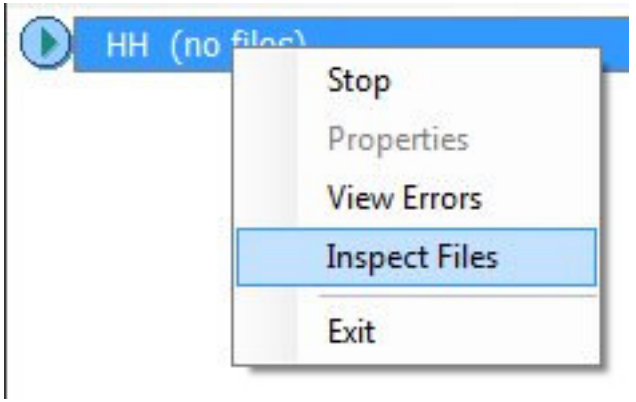
As post–processing we recommend to remove the scls.xml file

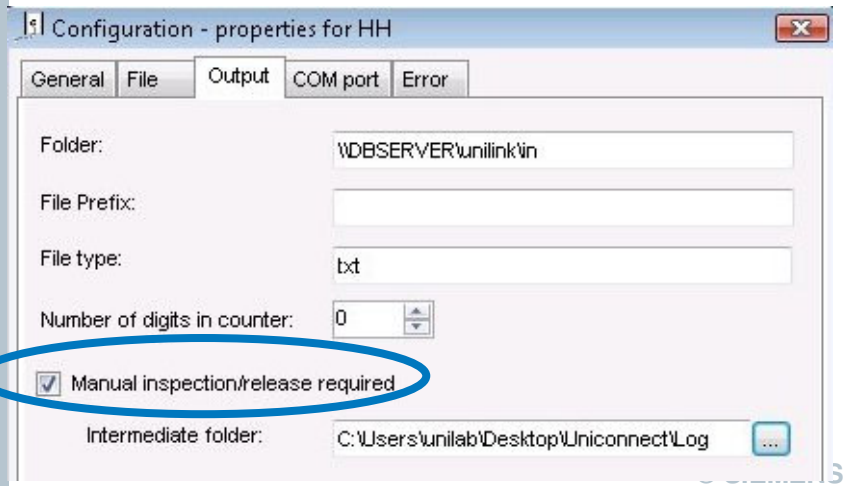
Output Files



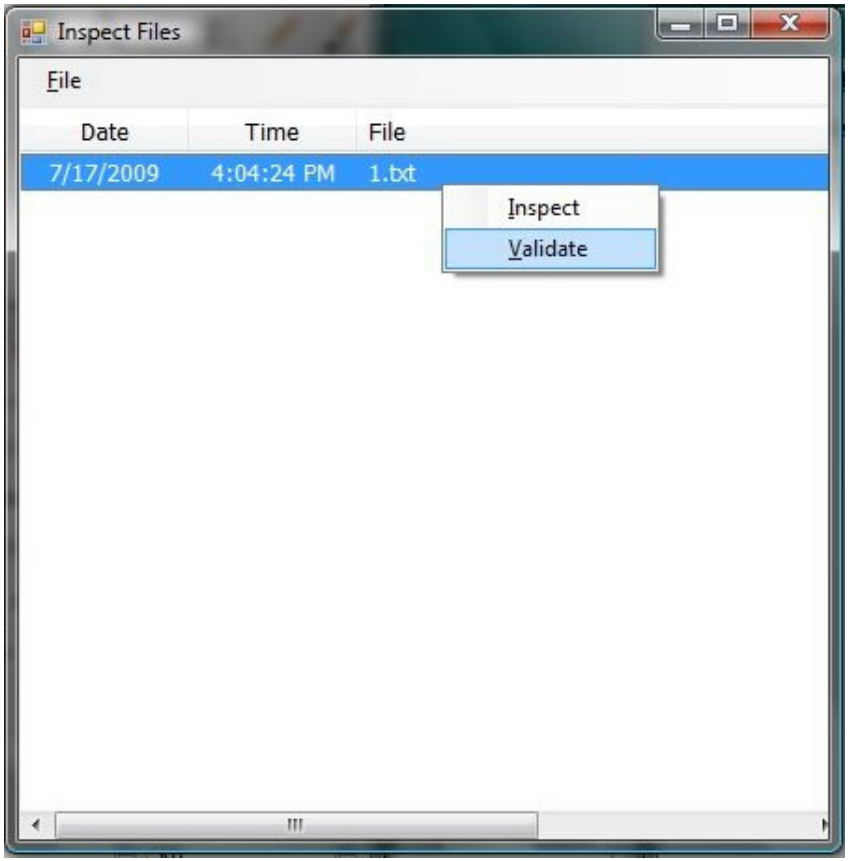
Manual inspection / release required

Inspect manually the output files





Inspect Files dialog



Inspect: opens the file

Validate: moves the file to the output directory

Error logging

General error file

- Application errors/warnings
- Location defined in the registry

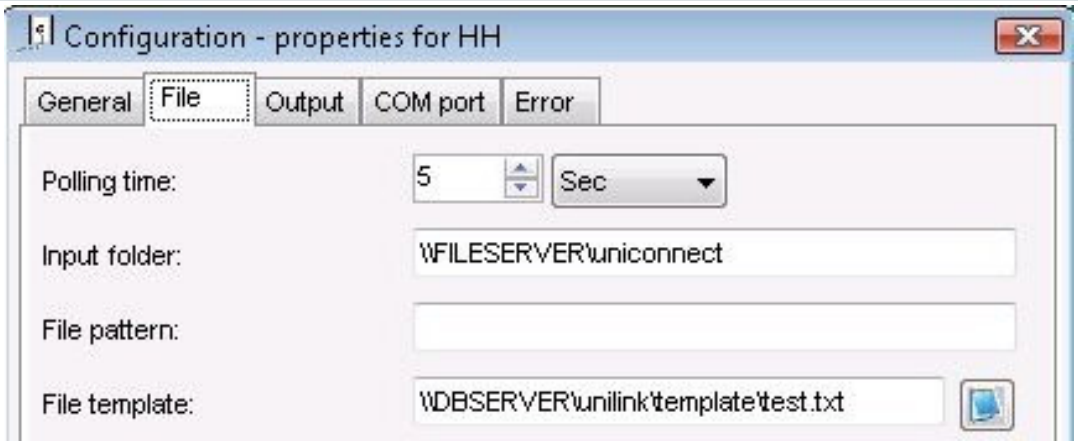
Equipment specific error file

- Processing and configuration errors for the instrument
  - Review from Exec main screen
    - 29/03/2005 14:31:02 - Folder <> not found
    - 05/04/2005 08:10:40 - Invalid COM-port specified
    - 22/04/2005 14:32:59 - Invalid line/position specified

E-mail address (per equipment)

- Error messages sent by e-mail
- MAPI based (Microsoft Exchange / Outlook)

Template File



Specify the template file

How must the raw data file be converted into an output file?

Uniconnect

Templates

ASCII file

- Reflects Output File Format
- Contains specific notations about where to insert data from Raw Data File

Contains all instrument specific info, e.g.

Output File

```
[sc]
sc=20050520-001
[pa]
pa=pH
value_f=5.7
```

Template

```
[sc]
sc=~1:1-20~
[pa]
pa=pH
value_f=~2:1-10~
```

Output File Format

Paragraphs in INI notation

- Each paragraph = 1 logical transaction
- Allowed sections:
  - [sc], [pg], [pa], [me], [cell], [cell table], [ic], [ii]
  - [BeginTransaction], [EndTransaction]
  - [CloseParameterGroup]

Any setting remains valid until overruled

Sample Section

Specifying the sample code

- Exactly
  - sc=<sample_code>
- Searching for the sample
  - sc.<std_prop>=<std_prop_value>
  - scgk.<gk>=<gkvalue>
    - E.g.: sc.st=Milk cream
    - scgk.factory=Ninove
    - scgk.packaging=bottle
  - sc=<SELECT-statement>

Sample Section (2)

Modifying sample properties

- SaveSample is called automatically
  - <std_prop>=<std_prop_value>

Creating samples

- create_sc=Y|N|W
- <std_prop>=<std_prop_value>

Other Object Sections

- Same rules as for [sc] section apply
- Save…Result optimisation is implemented
- Settings remain valid across sections!
  - E.g. sample code specified in [sc] section remains valid across [pa] section
  - No difference between

```
[sc]
sc=20070520-001
[pa]
pa=pH
value_f=5.7
```

```
[pa]
sc=20070520-001
pa=pH
value_f=5.7
```

Specials

- allow_reanalysis=Y|N
- pg[x] notation (Object location Syntax)
  - pg=Chemical
    - First parameter group Chemical
  - pg[2]=Chemical
    - Second parameter group Chemical
  - pg[]=Chemical
    - First not executed group Chemical

Special Sections

- Transaction handling
  - 1 section = 1 transaction
  - Overrule with [BeginTransaction] and [EndTransaction]
  - No nested transactions are possible
  - Synchronous vs. Asynch. Handling
- [CloseParameterGroup]
  - Insert Event ‘ClosePg’ for the current parameter group

Templates

Simple data fields

- E.g. take sample code from line 1 (columns 1 to 20), parameter result is on line 5 on position 5-15
- Text types
  - Static (defined in the Template)
    - sc=20070902-001
  - Partially copied from the Raw Data file
    - sc=~1:1-20~
  - Partially processed, based on data from Raw Data File (custom format function)
    - sampling_date=~FormatDate('2:1-30')~

Dynamic Text Assignment

~LineNumber:StartPosition-EndPosition~

- Allows exact positioning of text to use
- If input is surrounded by literal text, use:
  - PositionBefore('literal', 'occurrence')
  - PositionAfter('literal', 'occurrence')

Raw Data File

```
20070520-001
BA123
          | Results    |
----------+------------+
Measure1  |     5.7    |
```

Template

```
[sc]
sc=~1:1-20~
[pa]
pa=pH
value_f=~5:PositionAfter('|', '1')
-PositionBefore('|', '2')~
```

Using Custom Functions

~CustomFunctionName('Argument1', …, 'ArgumentN')~

To allow flexibility

- Concatenating strings, re-formatting dates, translation of identifiers,…
- Use separate configuration tables

Development in custom VB-DLL's

- Generic
- Lab specific

```
[pa]
sc=~ConvertSampleCode(`1:1-20')~
pa=pH
value_f=~5:1-20~
exec_start_date=~ConvertDate(`07/5/20')~
```

Variable Lists of Data

Raw data file

```
20070520-001
BA123
           | Results     |
-----------+-------------+
Na         |          57 |
K          |         100 |
Cl         |         230 |
```

Dynamic list of values

- Defined as Block
- Use the individual lines for output file or only perform custom function on the whole block

[BlockStart] and [BlockEnd]

Variable Lists of Data (2)

Start of Block

`[blockstart name=BlockName firstline=FirstLine lastline=LastLine]`

For FirstLine or LastLine, use:

- A fixed line number
  - E.g. firstline=5
- A string identifier
  - E.g. firstline=Equals(~:1-5~, '-----')
- A custom function (returns number)
  - E.g. firstline=FindMyResults()

PreviousLine and NextLine

Examples


```
20070520-001
BA123
           | Results    |
-----------+------------+
Na         |       57   |
K          |      100   |
Cl         |      230   |
```

```
[blockstart name=Myblock
   previousline=Equals(~:1-5~, '-----')]
```

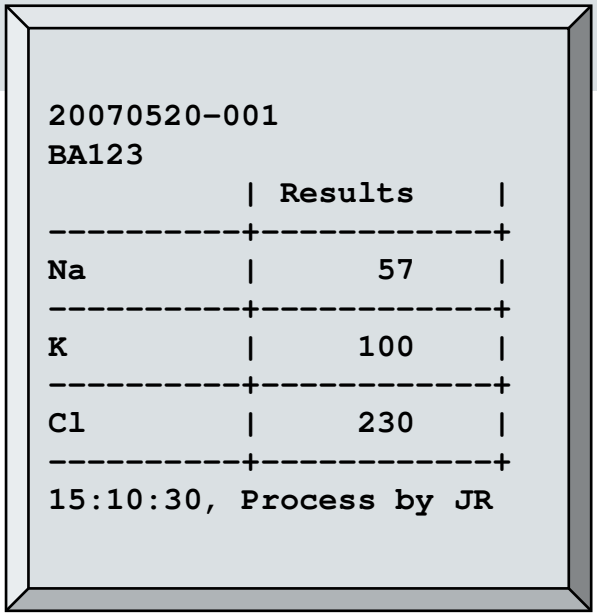
```
Na         |       57   |
K          |      100   |
Cl         |      230   |
```

```
[blockstart name=Myblock
   firstline= ~1~]
```

```
20070520-001
BA123
           | Results    |
-----------+------------+
Na         |       57   |
K          |      100   |
Cl         |      230   |
-----------+------------+
15:10:30, Process by JR
```

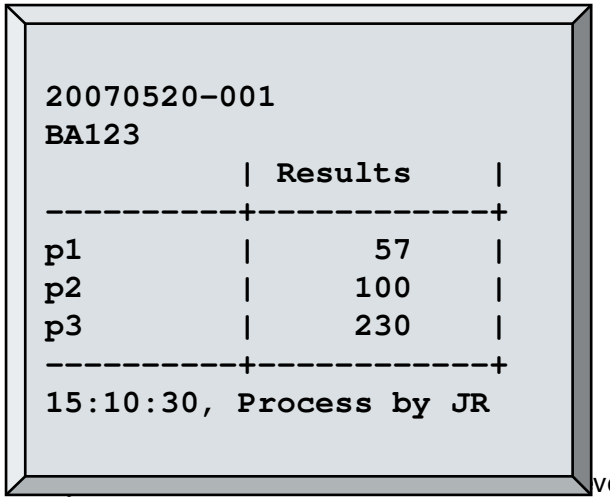
```
[blockstart name=Myblock
   previousline=Equals(~:1-5~, '-----')
   nextline=Equals(~:1-5~, '-----')]
```

Examples



```
20070520-001
BA123
          | Results    |
----------+------------+
Na        |      57    |
----------+------------+
K         |     100    |
----------+------------+
Cl        |     230    |
----------+------------+
15:10:30, Process by JR
```

```
[blockstart name=Myblock
   previousline=Equals(~:1-5~, '-----')
   nextline=~MyEndCheck()~]
```



```
20070520-001
BA123
          | Results    |
----------+------------+
p1        |      57    |
p2        |     100    |
p3        |     230    |
----------+------------+
15:10:30, Process by JR
```

```
[blockstart name=Myblock
   previousline=Equals(~:1-5~, '-----')
   lastline=Count(3)]
```

Data from a Block

All statements between [blockstart] and [blockend] are repeated for each line of the block

Line indication:

- Use ‘current line’
  - E.g. pa=~:1-10~
- Specify relative line number
  - E.g. sc=~MyBlock+3:5-20~

Examples


```
20070520-001
BA123
           | Results    |
-----------+------------+
Na         |       57   |
K          |      100   |
Cl         |      230   |
```

Raw Data File

Template

```
[sc]
sc=~1:1-20~
[blockstart name=MyBlock previousline=Equals(~:1-5~, `-----`)]
   [pa]
   pa=~:1-10~
   value_s=~:12-24~
[blockend]
```

Examples (2)


```
20070520-001
BA123
           | Results     |
-----------+-------------+
Na         |          57 |
K          |         100 |
Cl         |         230 |
```

Raw Data File

Template (only sum of components is required)

```
[sc]
sc=~1:1-20~
[blockstart name=MyBlock previousline=Equals(~:1-5~, `-----`)]
[blockend]
pa=Ions
value_s=~MyCalculateSum(MyBlock)~
```

Comments

Multiple blocks are allowed in 1 file

No nesting of blocks

SkipLines:

- Skip a number of lines, e.g.
  - ~SkipLines(3):10-PositionBefore('-', '4')~

SetLine:

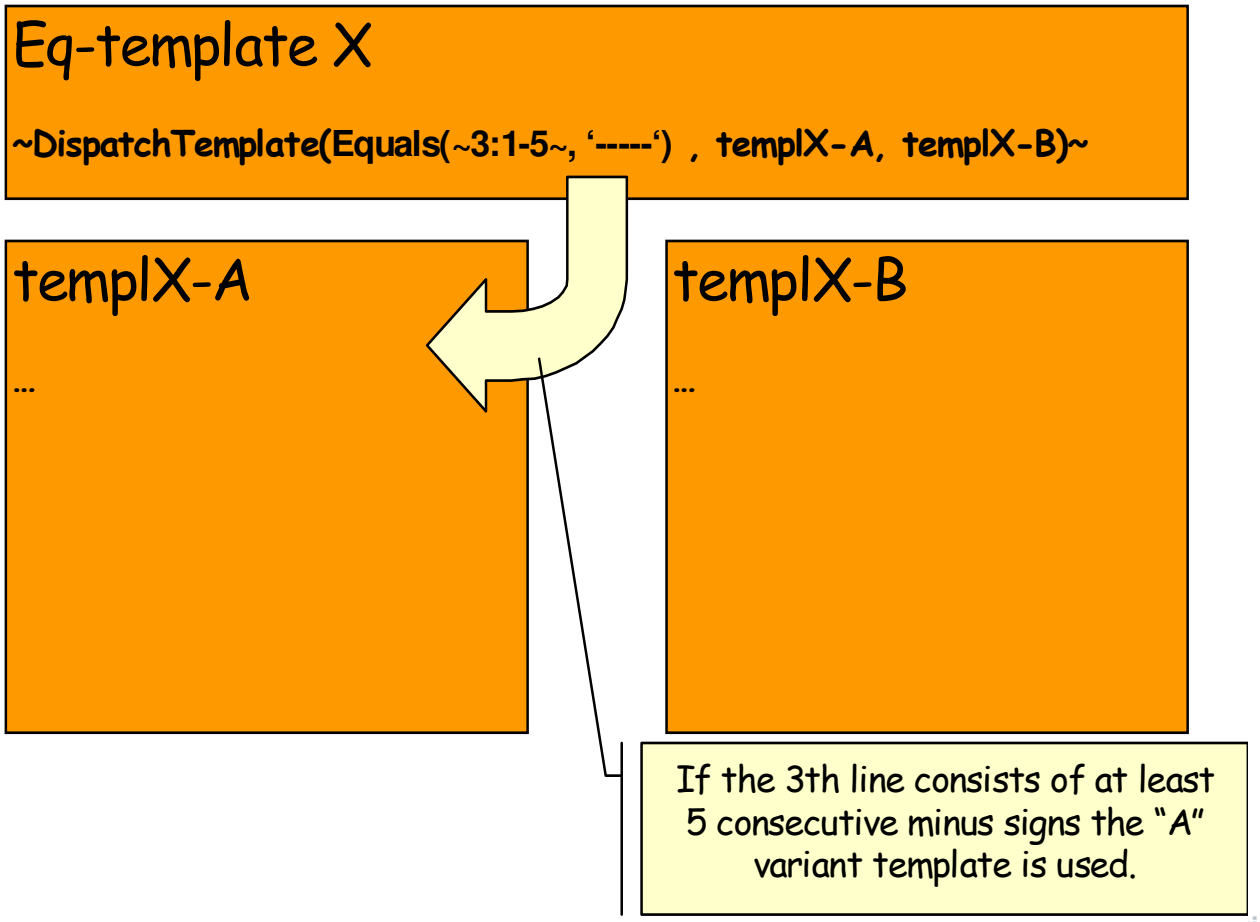
- Move the line pointer to the specified line, e.g.
  - ~SetLine(10)~

Templates - Conditional Evalution

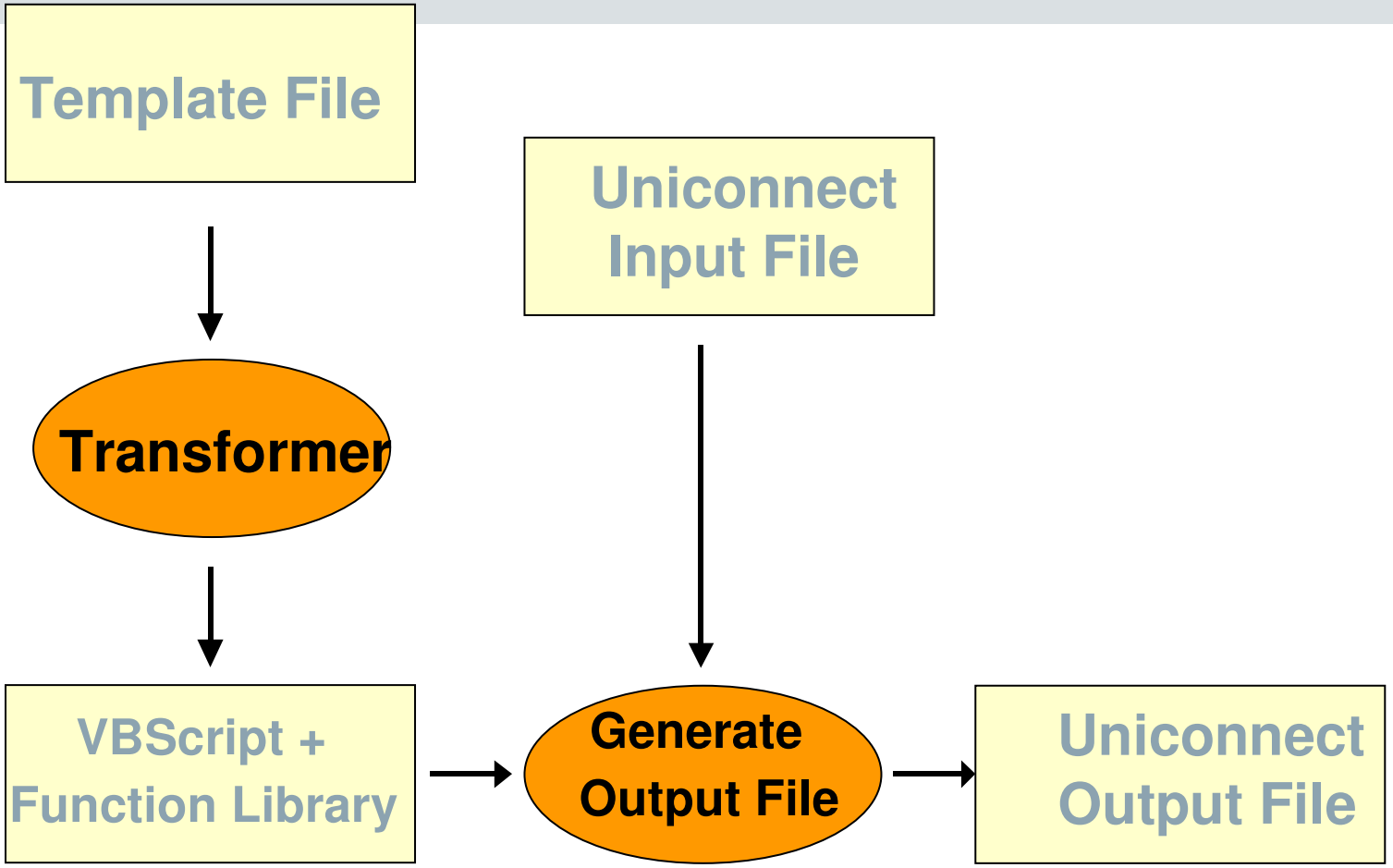
- Enable selection and switching between different templates
  - At run-time
  - Depending on the data that needs to be processed
  - Instruments generate different output formats
  - Define separate “variant” templates
  - Conditional evaluation statements

- Template needs to support programming logic - VBScript

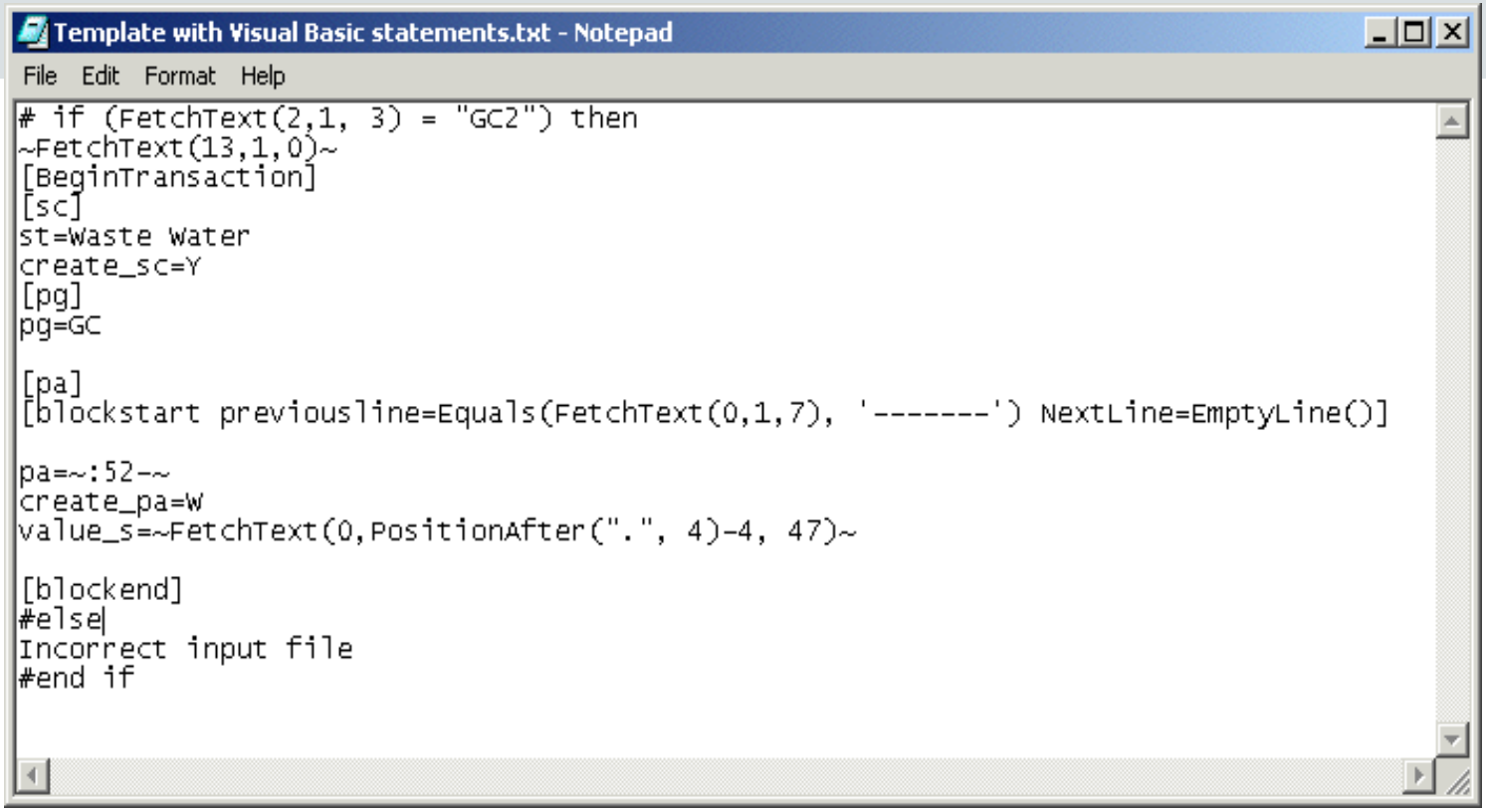
Conditional Evaluation - Example



Concept – Transformer VBScript



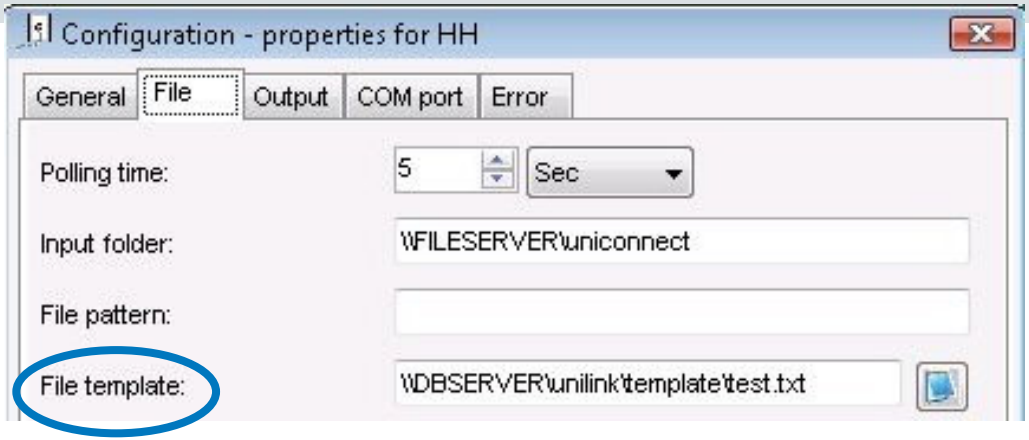
Template with Visual Basic statements - Example



Advised to use the function:

~FetchText((int) linenumber, (int) start_index, (int) end_index)~

Other parsing options

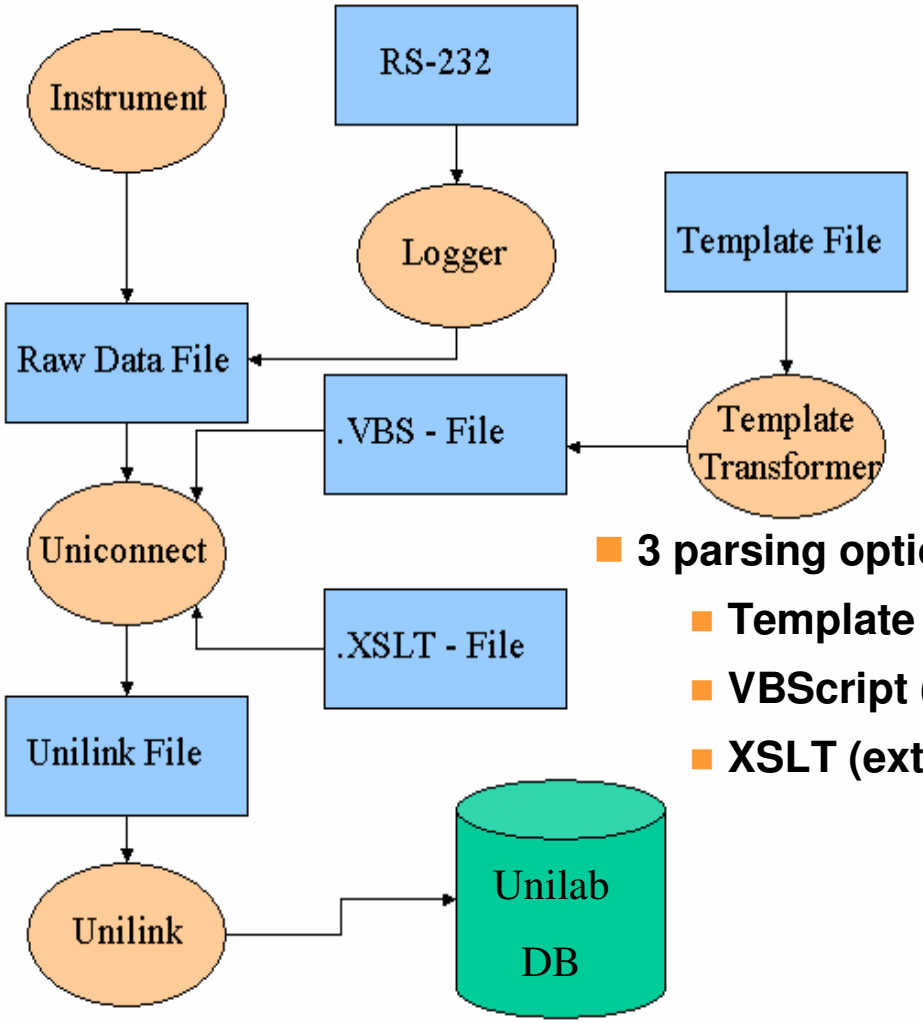


Parsing options besides the use of a template:

- Specify a VBScript
- Specify a XSLT file

Uniconnect detects the appropriate parsing option by looking at the file extension

Parsing options - Summary

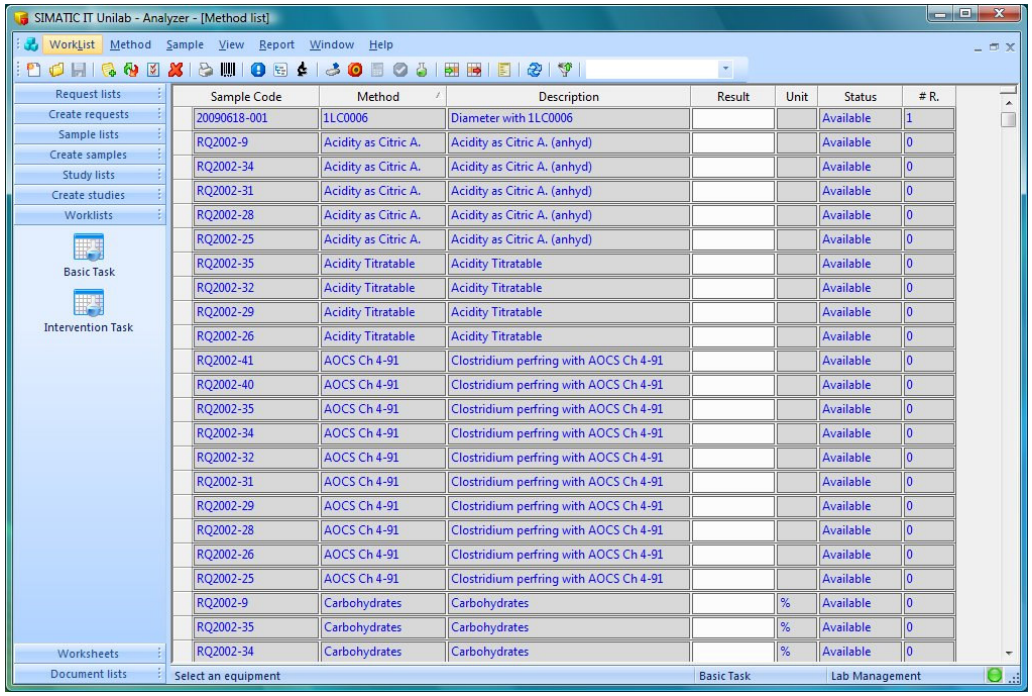


- 3 parsing options:
  - Template
  - VBScript (extension .vbs)
  - XSLT (extension .xsl)

Uniconnect

Example 1 - GC

- Results are transferred in background
- No user interaction
- Sample is created automatically
- Simatic IT Unilab automatically adds all (but only those) components that were sent through



GC

TEMPLATE_GC.txt - Notepad

File Edit Search Help

```
[BeginTransaction]
[sc]
st=Water
create_sc=Y

[pg]
pg=GC

[pa]
[blockstart previousline=Equals(~:1-7~, '-------')
nextline=EmptyLine()]
pa=~:52-~
create_pa=W
value_s=~:38-47~

[blockend]
[EndTransaction]
```

GC template file

GC003.txt - Notepad

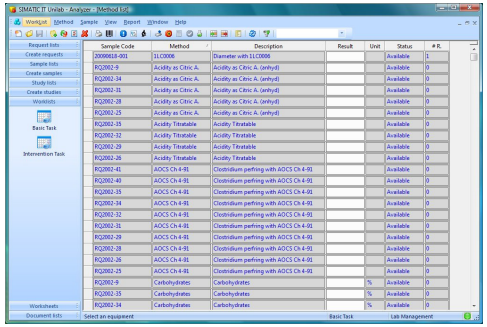
File Edit Search Help

```
Data File D:\HPCHEM\GC1\DATA\052F0601.D Sample Name: 2111A01
GC2 01-10-1999 11:39:31

================================================================================
Injection Date  : 01-10-1999 10:59:23          Seq. Line :   6
Sample Name     : 2111A01                          Vial :  52
Acq. Operator   :                                   Inj :   1
                                                Inj Volume : 1 µl
Sequence File   : D:\HPCHEM\GC1\SEQUENCE\DIRK.S
Method          : D:\HPCHEM\GC1\METHODS\TD(C).M
Last changed    : 22-09-1999 14:14:47
================================================================================
                        Normalized Percent Report
================================================================================

Sorted By             :      Signal
Calib. Data Modified  :      14-09-1999 08:48:32
Multiplier            :      1.0000
Dilution              :      1.0000

Signal 1: FID1 A,
Uncalibrated peaks RF :      1.00000


RetTime  Type     Area     Amt/Area     Norm    Grp   Name
 [min]          [pA*s]                   %
-------|------|----------|----------|----------|--|------------------
  6.874           -          -          -          123TRICHLOORBENZEEN
  8.000           -          -          -          123TRICHLOORPROPAAN
 11.698 BV       12.75306    1.00000   0.007514    124TRICHLOORBENZEEN
 11.761 VP        3.02078    1.00000   0.001780    2CHLOORTOLUEEN
 15.507 BP       12.05278    1.00000   0.007102    4CHLOORTOLUEEN
 16.846 BB      652.82306    1.00000   0.384661    CHLOROFORM
 17.987 PV        3.30110    1.00000   0.001945    FOSGEEN
 18.124 VB       10.29629    1.00000   0.006067    METHYLEENCHLORIDE
```

GC input file

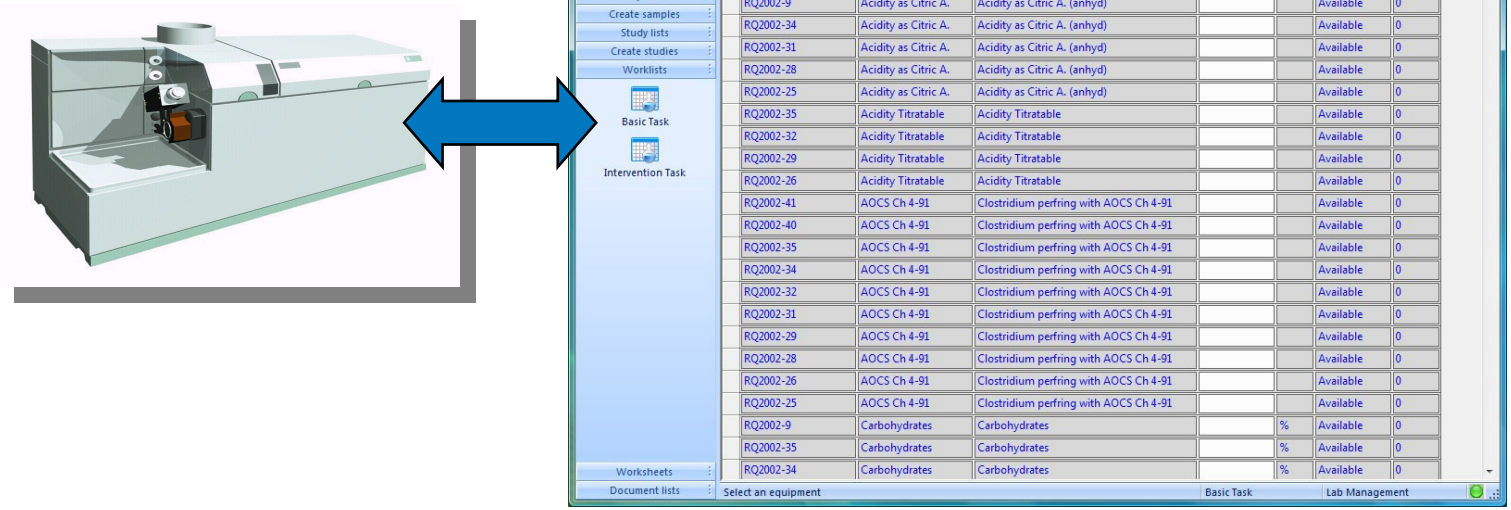
Uniconnect

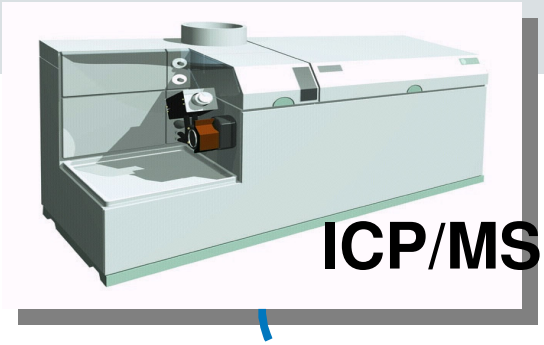
Unilab



Example 2 - ICP/MS

- Results are transferred in background,
- User can change results and is requested to validate manually
- Sample should already exist
- Simatic IT Unilab automatically adds all elements as soon as validation is complete





ICP/MS

TEMPLATE_ICPMSUC.txt - Notepad

File Edit Search Help

```
'Template ICP-MS-UC
[BeginTransaction]
[sc]
sc=~2:10~
st=Water
create_sc=W

[pg]
pg=ICP MS
create_pg=W

[pa]
pa=Som Metalen

[cell table]
me=ICP-MS
cell=TABEL
value_s[3,0]=~7:PositionAfter('/', '1')-PositionBefore('/', '2')~
value_s[2,0]=~7:PositionAfter('/', '4')-PositionBefore('/', '5')~
value_s[3,1]=~8:PositionAfter('/', '1')-PositionBefore('/', '2')~
value_s[2,1]=~8:PositionAfter('/', '4')-PositionBefore('/', '5')~
value_s[3,2]=~9:PositionAfter('/', '1')-PositionBefore('/', '2')~
value_s[2,2]=~9:PositionAfter('/', '4')-PositionBefore('/', '5')~
value_s[3,3]=~10:PositionAfter('/', '1')-PositionBefore('/', '2')~
value_s[2,3]=~10:PositionAfter('/', '4')-PositionBefore('/', '5')~
value_s[3,4]=~11:PositionAfter('/', '1')-PositionBefore('/', '2')~
value_s[2,4]=~11:PositionAfter('/', '4')-PositionBefore('/', '5')~
[EndTransaction]
```

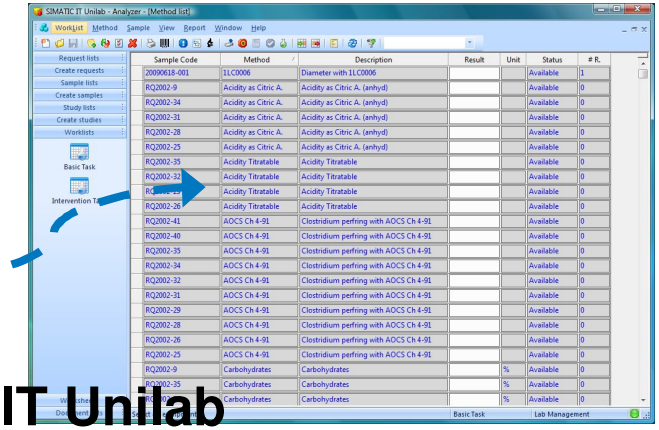
ICP/MS template file

ICPMS05.txt - Notepad

File Edit Search Help

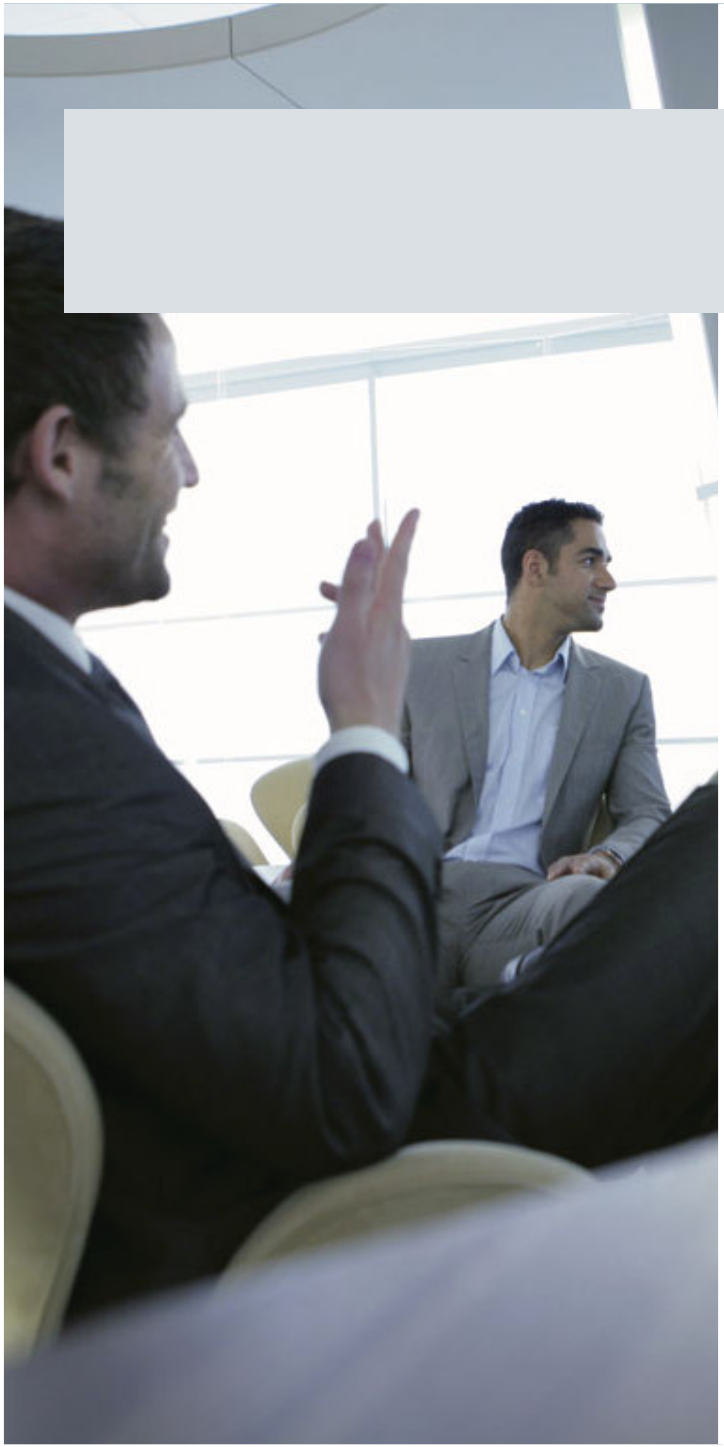
```
Methode : ICP-MS
Sample : SW_20000525_105
Run : run02
Analist : CXFJ
Experiment datum : 2000-04-17
Toestel : ICP-MS-1
Mg/2.28/1/1/100/V////
K/23.34/1/1/100/B/commentaar op K meting///
As/21.28/1/1/100/V////
Mn/43.54/1/1/100/B/commentaar op Mn meting///
Na/1.58/1/1/100////
```

ICP/MS data file



Uniconnect

Simatic IT Unilab



SIEMENS

Connecting Instruments

Unilink

Unilink

Introduction

File Processing by Unilink

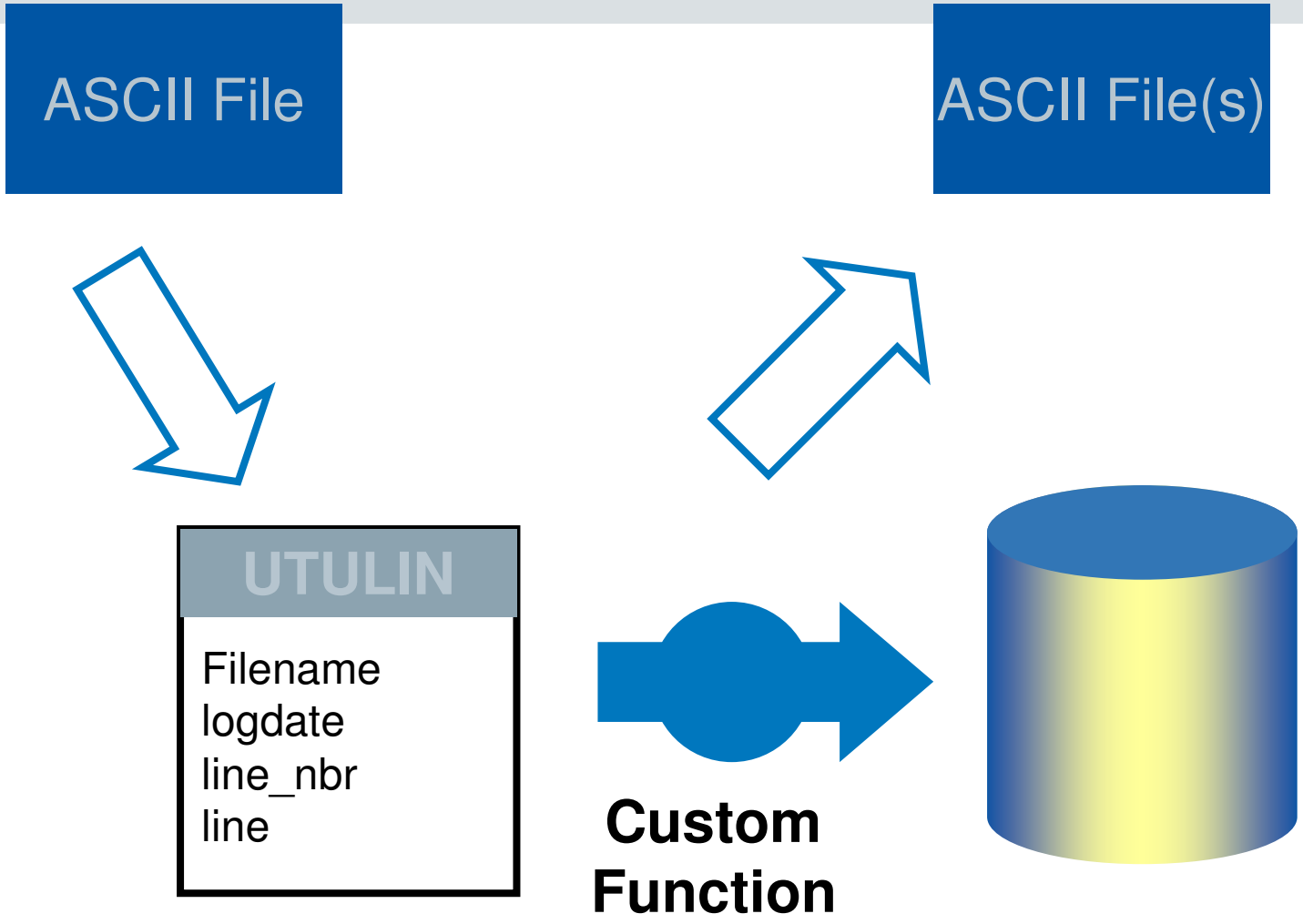
Configuration

Unilink

Unilink allows

- Parsing of ASCII files on server
- Transfer of the data to the Simatic IT Unilab database
- Processing of data
  - Parsing and/or interpreting data

Functional Overview



Introduction

File Processing by Unilink

Configuration

Unilink Directory Structure

In directory (LOC_IN_DIR)

- Storage of files to be processed
- Example: files coming from Uniconnect

Log directory (LOC_LOG_DIR)

- Storage of trace files

Out directory (LOC_OUT_DIR)

- Storage of output files

Err directory (LOC_ERR_DIR)

- Storage of error files

Running Unilink

Options

- Run as a database job
- Called from within application

Unilink as database job

- StartUnilink / StopUnilink
- Polling Interval (system setting UL_POLLING_INTERVAL)
  - ChangeULInterval

Unilink called directly

- Unilink function call

End of File Detection

Options

- By OS or calling program (OS)
- Based on string (EOF_STRING)
- Based on file size (EOF_FILESIZE)
- Based on *.rdy file (EOF_FLAGFILE)

Argument for functions

- StartUnilink
- Unilink

Custom Function

PL/SQL CF

Specified on equipment communication settings

Package

- Default: Unilink

Uses

- DB API for Unilink
- DB API for sample creation, save parameter results,…
- Custom functions

Uniconnect Parser

- Template
- Extendable

Developing Custom Code

Package variables

- UNAPIUL
  - Refer to On Line Help
- UNAPIUL.P_EQUL_REC
  - loc_dir, file_name, cf, after_process, unilink_id, eof_string

Transaction logic

- BEGINTRANSACTION
- ENDTRANSACTION

Custom Function - Return Code

- UNIAPIGEN.DBERR_SUCCESS
- Something else
  - Return code logged in Unilink log file
  - Copy of log file in err directory
  - No logging in uterror
- Does not interrupt Unilink
  - After_process setting is executed

Customizing Uniconnect Parser

Uniconnect parser

- Template
  - Uniconnect package
- Extendable for specific purposes

Customizing the parser

- Implement in another package
- Refer to On Line Help for customization

After Processing

ASCII file can be

- Deleted
- Marked as processed
  - utulfilestatus

Specified in equipment communication settings

- AFTER_PROCESS setting

Tracing and Logging

Arguments

- in StartUnilink and Unilink
- a_ul_trace
- a_ul_trace_loc_dir
- a_trace_level

Trace Level

a_trace_level

- In Unilink and StartUnilink

UNAPIUL.UL_TRACE_NONE	0	Very low level debugging
UNAPIUL.UL_TRACE_LOW	1	Debug information
UNAPIUL.UL_TRACE_NORMAL	2	Informative
UNAPIUL.UL_TRACE_HIGH	3	High priority message but no error
UNAPIUL.UL_TRACE_ALERT	4	Alert fatal error

Using Unilink without ASCII file

Using Unilink without ASCII file

- Parsing without reading file first
- Filling utulin
  - No API

Using Unilink DB APIs

- RemoveFile
- GetDirectory
  - Returns all files in an array

Unilink

Introduction

File Processing by Unilink

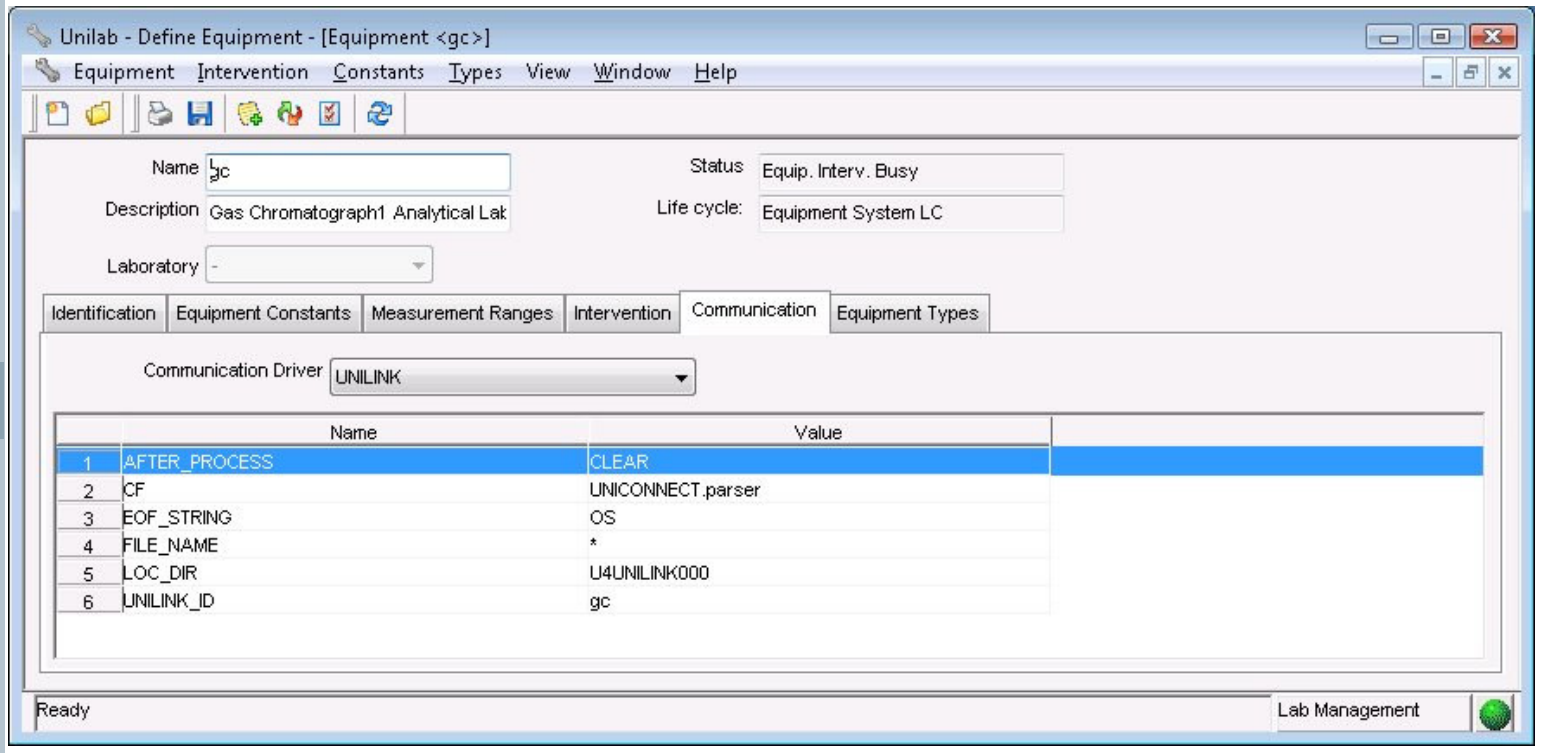
Configuration

Configuration

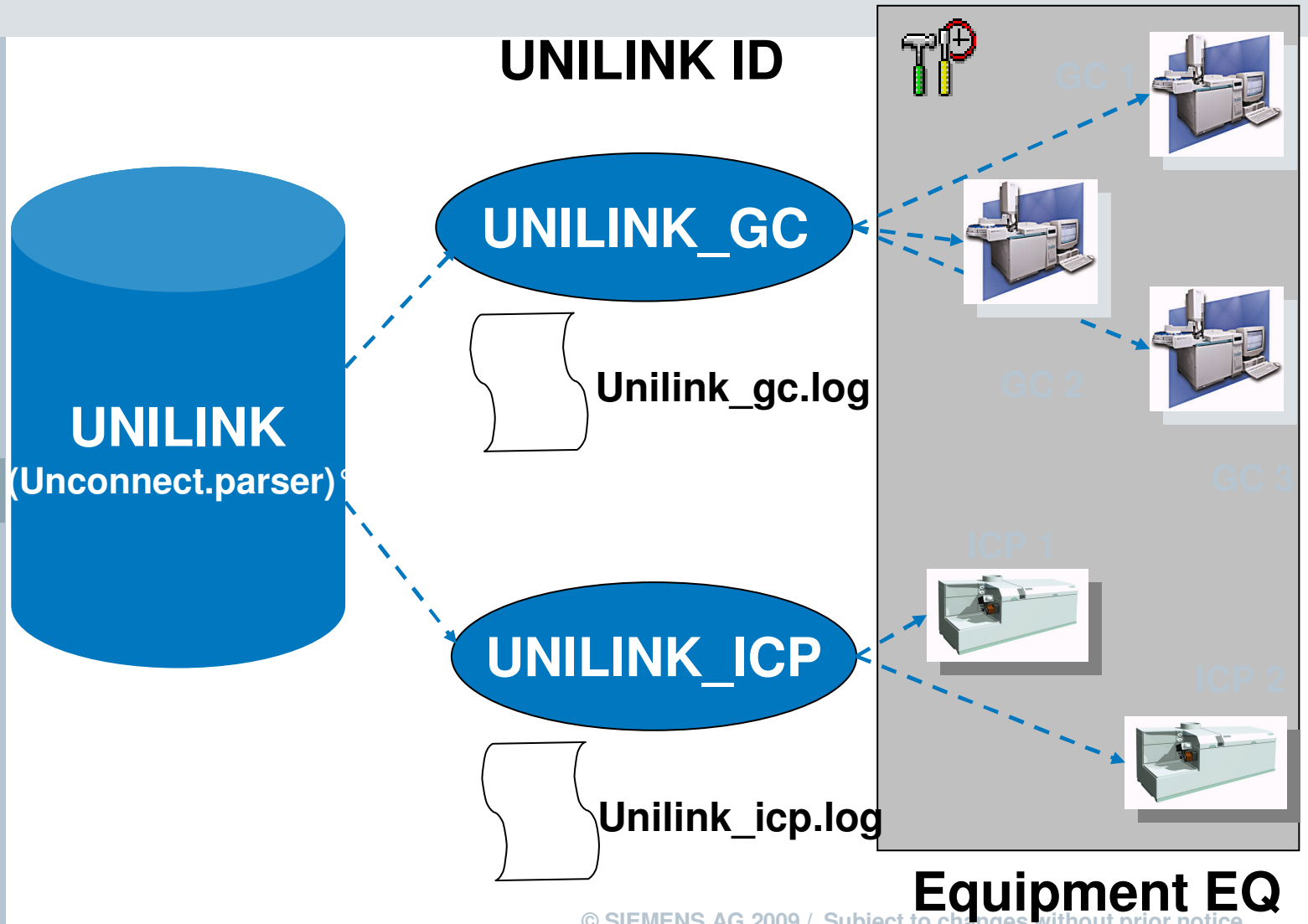
System settings

- UL_POLLING_INTERVAL
  - Interval between 2 Unilink jobs in seconds
  - Default = 300 s
- UL_TRACING_ON
  - Boolean indicating if the tracing should be activated or not
- UL_LOG_TO_FILE
  - Boolean indicating whether the logging is switched on/off for UNILINK

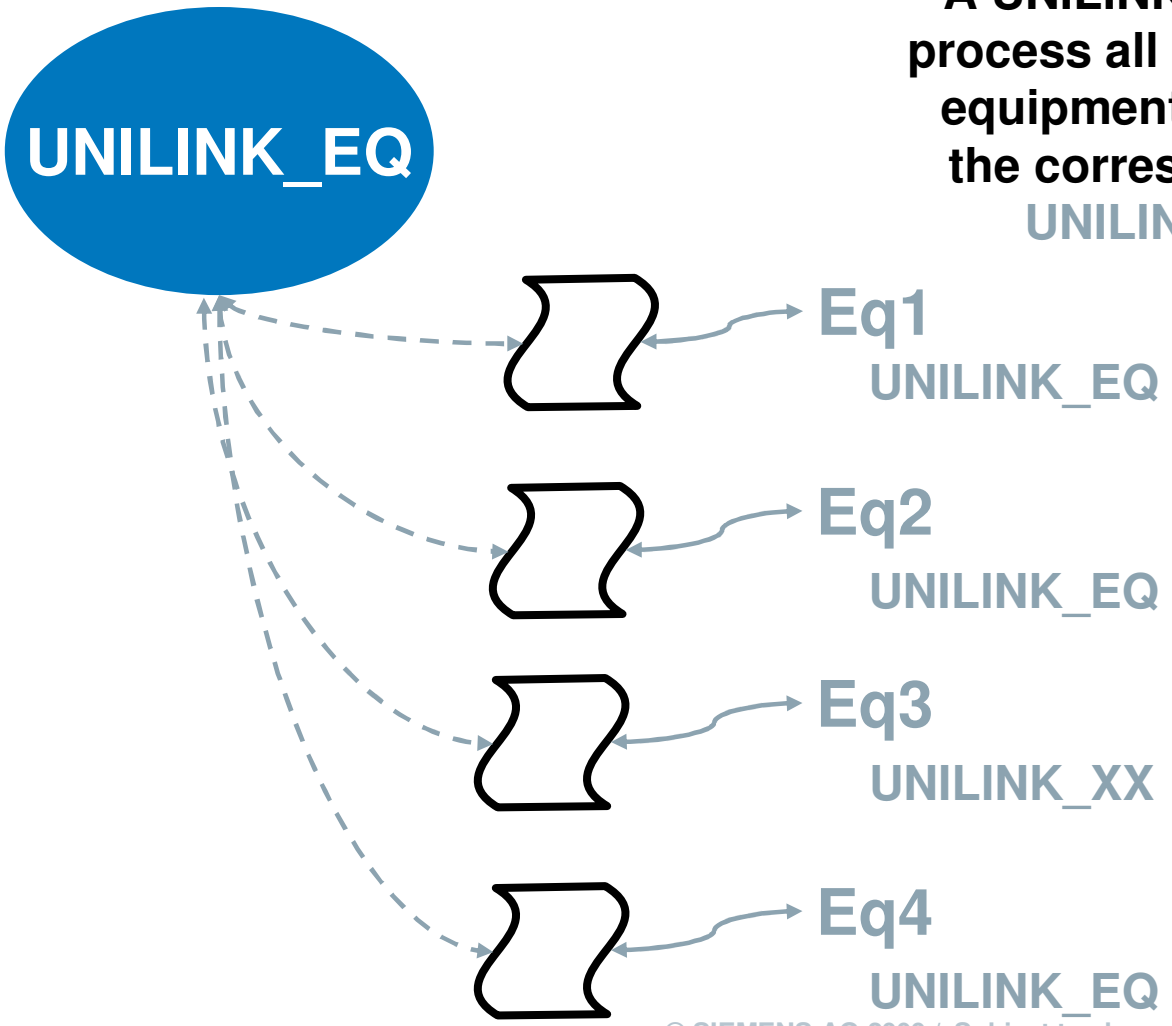
Equipment Configuration



Unilink ID



Unilink as DB Job



Thank you!