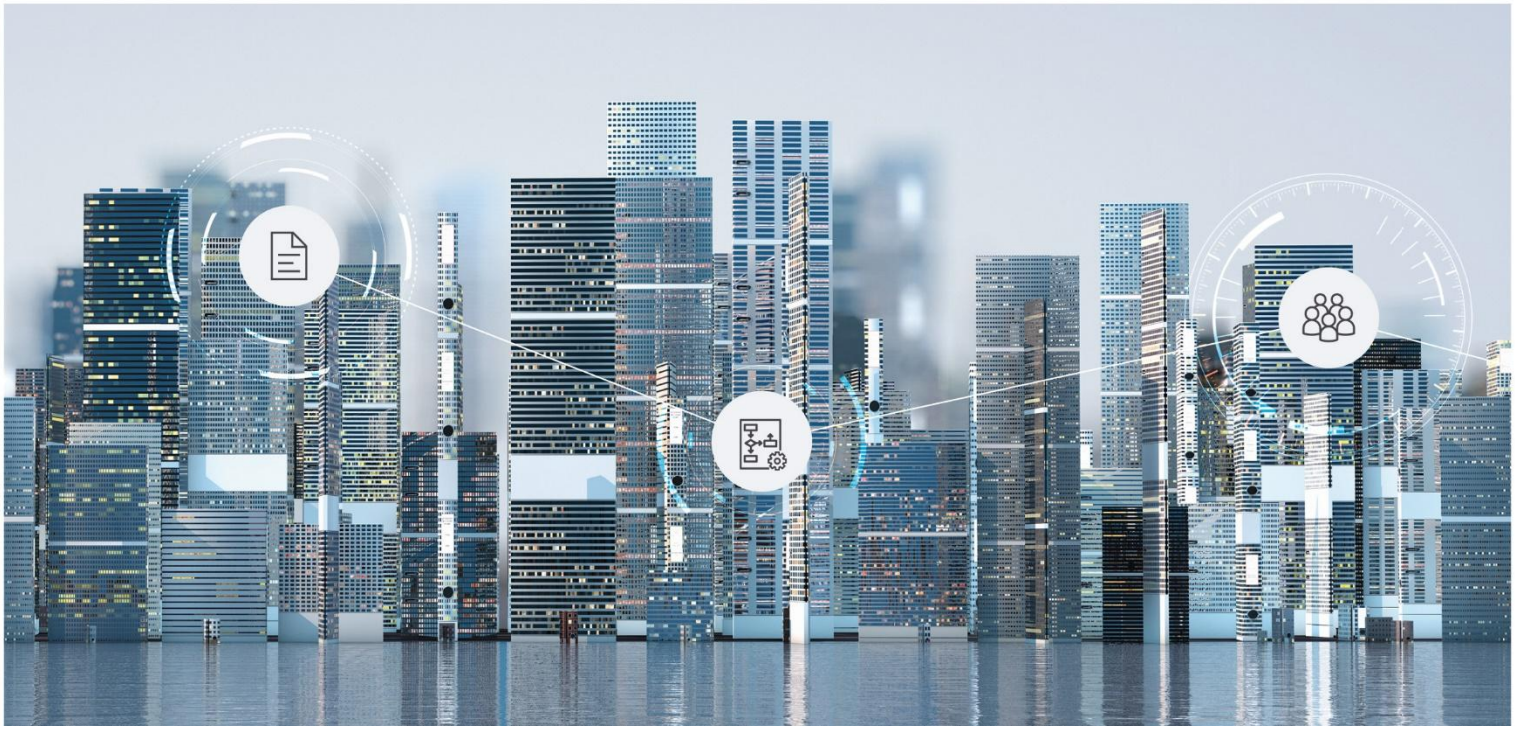
Fabasoft



White Paper

Fabasoft Folio/COLD

Fabasoft Folio 2025 Update Rollup 1

Contents

1 Introduction

This document describes how to create or update Fabasoft Folio objects from external data sources via a standard interface with the software product *Fabasoft Folio/COLD*. This includes the definition of data sources, configuring the data import, updating of data and logging the data transfer.

2 Software and Hardware Requirements

System environment: All information contained in this document implicitly assumes a Microsoft Windows environment.

Supported platforms: For detailed information on supported operating systems and software see the software product information on the Fabasoft distribution media.

Descriptions in this document are based on the following software:

- Microsoft Windows Server 2019 Standard

3 Definition of Data Sources

Fabasoft Folio/COLD offers several ways to specify the access to data sources.

3.1 ODBC

ODBC (Open Database Connectivity) is a standardized method developed by Microsoft that allows access to data sources, regardless of the application that manages the data. Appropriate drivers are available for a variety of applications.

Examples of data sources:

- Database Systems
- CSV files (Character Separated Values)
- Excel spreadsheets

The data source must be defined as a file data source or a system data source using operating system tools.

1. Click “Start” > “Administrative Tools” > “Data Sources (ODBC)”.
2. Use the “System DSN” tab and the “File DSN” tab to define data sources.

Note:

- File data sources are specified by DSN files and system data sources can be entered into the registry (under `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI`).
- If the project-specific situation permits, OLE DB should be used instead of ODBC.

3.2 OLE DB

OLE DB (Object Linking and Embedding Database) is a collection of interfaces developed by Microsoft that provides access to a variety of data sources. Compared to ODBC, it is a newer technology with better support.

Examples of data sources:

- Microsoft SQL Server
- Oracle Database
- Microsoft Office Access

Fabasoft Folio/COLD is an OLE DB consumer and therefore can import data that is provided by an OLE DB provider.

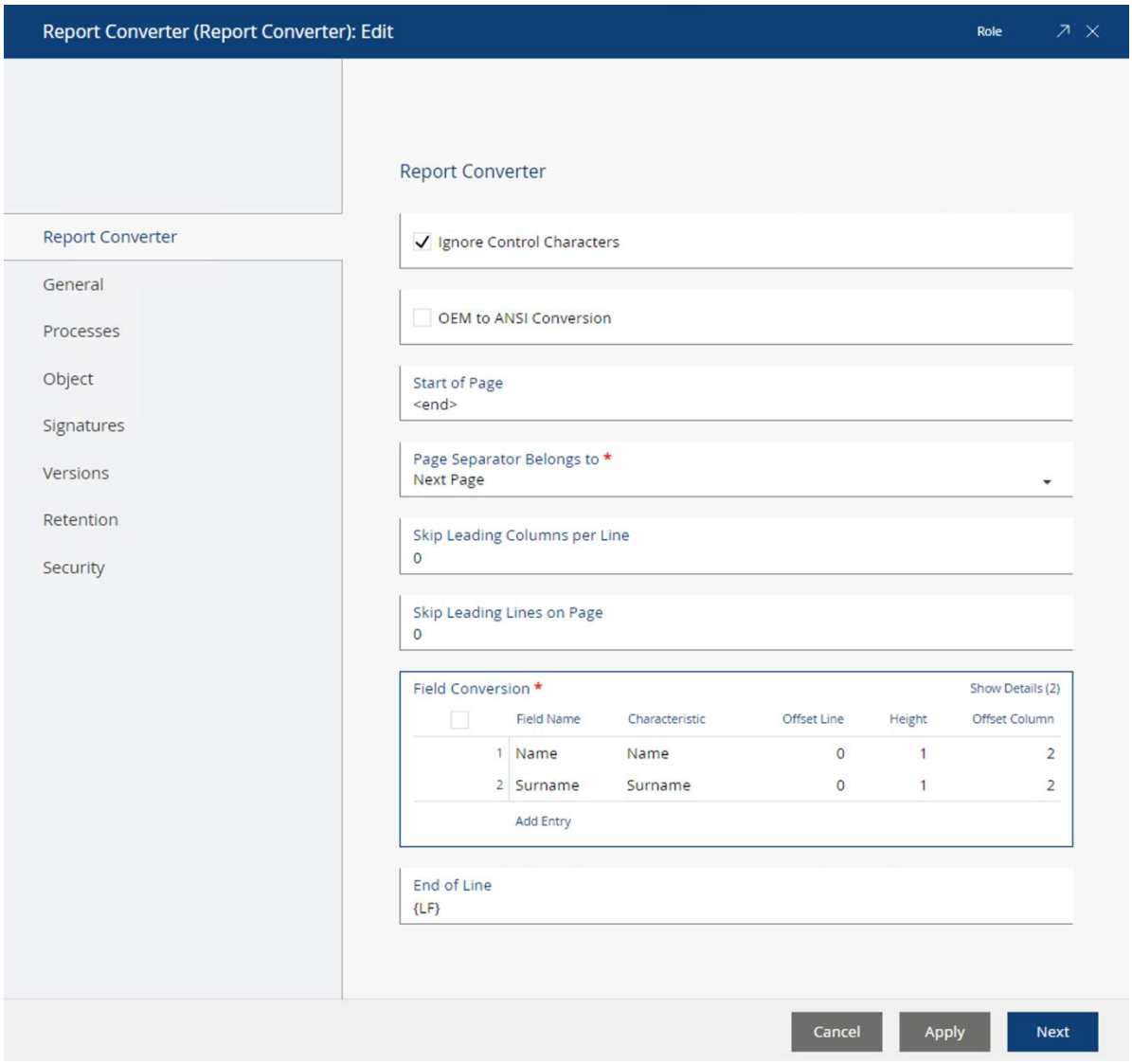
As part of *Fabasoft Folio/COLD* also an OLE DB provider for the Microsoft Windows file system is installed ("FileProvider OLE DB Provider"). This can be used to read directory structures and files from the file system and to generate Fabasoft Folio objects (see chapter 6 "The Fabasoft Folio/COLD File System Import").

3.3 Report Converter

Configuration objects of the object class *Report Converter* (also available as a component object) can be used to read and split print files (such as from mainframe systems) and host export files, and to pass them to the data import mechanism. The files to be read must be available as text files. The maximum field size should not exceed 1 MB.

Each record is handled as one page. The *Start of Page* has to be defined, as well as the position and length of the fields (*Field Conversion*).

Example for a *Report Converter:*



This report converter can read the following TXT file:

```
<end>
Name: Otto
Surname: Maler
<end>
Name: Karin
Surname: Winkler
```

3.4 Script

The data to be imported are handed to *Fabasoft Folio/COLD* using scripts. This possibility can be used for generating test data and for integration of data sources that can be addressed by scripts. An example for such a data source is XML.

The following features are defined:

- `Init(Table)`
  Initialize the data source. The content of the *Table* property of the *Data Import* object (see chapter 4 “Configuration of the Data Import”) is passed as parameter.

- `Skip(Rows)`
  Optionally. Skips the number of records defined in `Rows`.
- `GetHeader(ColumnNames)`
  The names of the columns have to be returned as string array.
- `GetRecord(Values, IsValid)`
  For each call of `GetRecord`, in the `Values` parameter a string array with the values of a record has to be returned. If there are no records left, `IsValid` has to be set to `False` otherwise it has to be `True`.
- `Terminate()`
  Optionally. This function is called after reading the last record. In this case e.g. locked objects can be shared again.

3.5 LDAP

LDAP is a network protocol used in directory services. It mediates communication between the LDAP client and the directory (Directory Server). From such a directory, object-related data (e.g. personal data) is read. The communication takes place query-based.

Examples for data sources:

- OpenLDAP Directory
- Active Directory

3.6 ADE-DB

ADE-DB is the name for the database connectivity developed by Fabasoft. It can be used to import data of database systems that can be used for Fabasoft Folio services as well.

Examples for data sources:

- Microsoft SQL Server
- Oracle
- PostgreSQL

3.7 Spreadsheet Document

Spreadsheet Documents working with Fabasoft Folio COLD Loader are CSV, Microsoft Excel or LibreOffice Calc documents.

A CSV file is a text file for storage or exchange of simply structured data. CSV stands for ”Character Separated Values“.

The first line of the file contains the column names. The separator for the Fabasoft Folio COLD Loader is the semicolon by default. If a semicolon should be contained in the values, the whole value needs a double quote. The used separator can be configured by means of the data source.

The CSV file can be ANSI or UTF-8 format.

3.8 XML

XML files are text files for storage or exchange of hierarchically structured data.

Column names can be defined in a syntax similar to XPath (e.g. “address/nr“).
Fabasoft Folio COLD Loader supports the following variants of column names:

- relatively, e.g. ”person/name“
- globally, e.g. “//artnr“
- attributes, e.g. “contactdata/person@id“ (in this case ”id“ is an attribute of the element “person“)

Additionally in the *Table* property of the *Data Import* object (see chapter 4 “Configuration of the Data Import”) the path to the desired element can be defined (e.g. “/staff/contactdata/person“).

3.9 File System

Using the datasource “File System” files from the file system or from a ZIP file can be imported.

4 Configuration of the Data Import

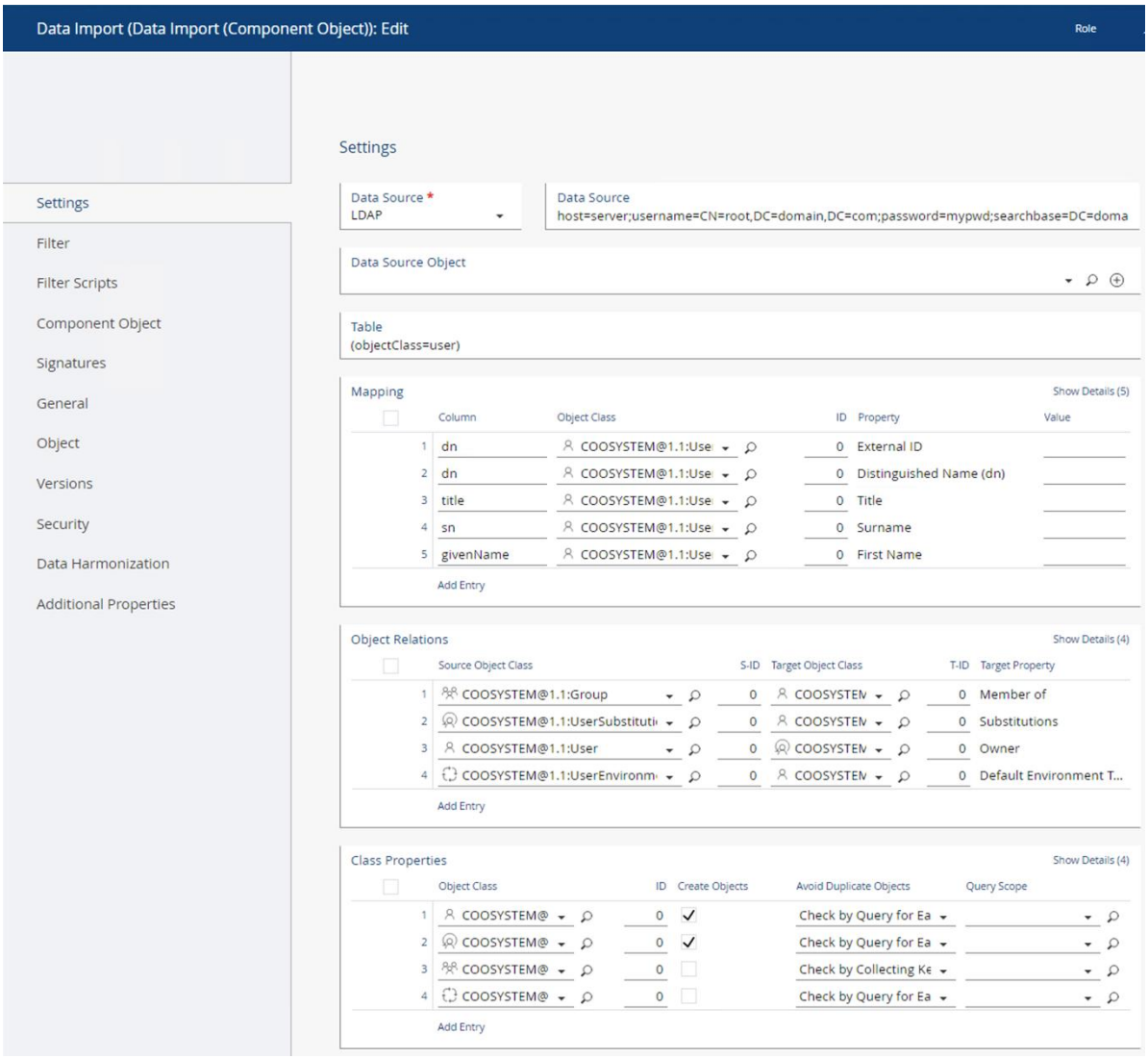
In *Data Import* objects or *Data Import Component Objects* (can be delivered with a software component) is configured how single records or fields of a data source are converted to Fabasoft Folio objects and their properties.

Apply the following principles:

- From a single record of the data source, one or more objects of the same class or objects of different classes can be created.
- In a separate allocation table, object relations can be mapped.
- In addition, functions that affect the data import can be implemented.

Note: In a Linux system environment the Fabasoft web service user (`:<hostname>:\fscsrv`) needs writing properties (“Change Properties“) for the *Data Import* object and the *Data Import Log* object. So for the *Data Import* object an appropriate ACL has to be selected. Additionally in the *Log Object* field or the *Data Import*, a *Data Import Log* has to be created manually before the first import and an appropriate ACL has to be selected.

Example of a *Data Import:*



4.1 Data Source

In the *Data Source* property, select the type of data source (ODBC, OLE DB, Report Converter, Script, LDAP, ADE-DB, Spreadsheet Document or XML) and enter a reference to the data source. The available types of data sources are described in chapter 3 “Definition of Data Sources”.

Examples of references:

- Using “OLE-DB”, for example the path to a UDL file can be defined
  e.g. “E:\migration.udl”
- Using “ODBC”, for example a Microsoft Office Access database can be defined
  e.g. “DSN=Migration;DBQ=E:\migration.mdb;DriverId=25;FIL=MS Access;MaxBufferSize=2048;PageTimeout=5;”
- Using “Report Converter”, define a report converter object in a Fabasoft product environment
  e.g. “LOCAL@1.115:ReportConverterRC1”
- Using “Script”, define a script component object in a Fabasoft product environment
  e.g. “LOCAL@1.115:SkriptS3”

- Using “LDAP”, for example an LDAP directory can be defined as data source (default timeout: 10 seconds). The parameter transport can be “ldap” or “ldaps”.
  e.g. “transport=ldap;host=192.168.8.138;port=389;
  username=CN=root,DC=adminusers,DC=faba,DC=local;password=mypassword;
  searchbase=DC=faba,DC=local;timeout=100”
- Using “ADE-DB”, for example a Microsoft SQL Server connection can be defined as data source
  e.g. “Provider=SQLOLEDB;Datasource=(local);Catalog=FSCReporting”
- Using “ADE-DB”, for example an Oracle connection can be defined data source
  e.g. “Transport=OCI;Location=oradb;Username=system;Password=manager”
- Using “ADE-DB” for example a PostgreSQL connection cab be defined as data source

  e.g.
  “Transport=PGSI;Location=192.168.100.83;Catalog=postgres;ConnectString=PORT=5432;
  Username=postgres;Password=postgres”
- Using “Spreadsheet Document”, the file path is defined in the *Table* field. Optionally the separator in case of a CSV file can be defined in the *Data Source* line.
  e.g. “separator=';'”
- Using “XML“, define the path to an XML file
  e.g. “E:\migration.xml“
- Using “File System“, define the path to the directory
  e.g. “E:\importdata“

ADE-DB Parameter:

Transport:	e.g. “OCI” for Oracle, “PGSI” for PostgreSQL
Provider:	e.g. “SQLOLEDB” for Microsoft SQL Server
Location:	For defining the location
Datasource:	For defining the data source
Catalog:	For defining the name of the database catalogue
Username:	Name of the database user
Password:	Password of the database user
Connectstring:	Additional settings (e.g. “PORT=5432” for PostgreSQL)

4.2 Data Source Object

This property can be used to reference a Fabasoft object as data source.

Using “Spreadsheet Document”, “XML” or “File System”, data of a *Content (Unknown Type)* can be imported, which needs the value “xlsx”, “xls”, “sxc”, “ods”, “csv“, “xml” or “zip” in the property *File Extension* (`COOSYSTEM@1.1:contextension`).

4.3 Table

In this property the name of the table (or the view) is defined, that contains the data to be imported.

Note:

- Using “OLE DB”, a SELECT statement can be defined.
  e.g. “SELECT * FROM Organization WHERE id>10;“

- Using “LDAP”, define an LDAP filter.
  e.g. “(objectClass=inetOrgPerson)“
- Using “Spreadsheet Document”, the file path can be defined.
  e.g. “C:\personen.xlsx“
  Alternatively a Fabasoft Folio object can be selected in the *Data Source Object* field (see chapter 4.2 “Data Source Object“).
- Using “XML”, the path to the desired elements can be defined.
  e.g. “/staff/contactdata/person“
- Using “File System”, define the extensions considered in the import.
  e.g. “{jpg;png;gif}“

4.4 Mapping

This property is used to map external data to Fabasoft Folio properties.

- *Column*
  In this field enter the name of the database column, the attribute name of the LDAP directory, or using “XML”, a path similar to XPath (see chapter 3.8 ”XML“).
- *Object Class*
  In this field select the object class of the object to be filled with the data of the specified data source column.
- *Object ID*
  From a record, several objects of the same object classes can be created. Use this field to distinguish objects of the same object class. For each different object a unique object id is required (integer).
- *Property*
  In this field select the property that is to be filled with the data from the specified data source column. For compound properties, a property path can be defined.
  Note: Which properties are available depends on the *Object Class*. Defining a property path, only the available parts of the compound property can be selected.
- *Value*
  This field is used if a property gets a constant value, regardless of the data source values. In this case the *Column* field is left blank.
- *Key Mode/Update Mode*
  In this field, the import behavior is determined.
  - *Key*
    The value of the specified column is set when Fabasoft objects are created. The key is used for finding objects.
    Following property types can be used as a key: string, integer, enumeration, date and (conditioned) float.
    Following property types cannot be used as a key: field, content, dictionary and object pointer.
    Note: All properties with selection “Key“ are used together as key for unique identification of objects of one object class having the same id.
  - *Set if Created*
    The value is only set if a new object is created.

- *Update*
  The value is set or added to a list. Existing values are not overwritten by blanks.
- *Overwrite*
  The value is set or added to a list. Existing values are overwritten by blanks.
- *Delete and Recreate*
  All values in a list are deleted and added again. Useable only for object lists.
- *Ignore*
  The line is ignored.

- *Options*
  For the values to be imported further conditions can be defined.
  - String List
    This option can be selected to split a string into a list of strings.
  - Aggregate Key
    For aggregates, instead of using the keys defined in the data model, this option can be used to define own key properties.
  - Group by
    This option can be selected to keep certain changes of an object within one transaction. Selecting this option, a variable transaction size is achieved. Usually “Group by” is used for key properties.
  - Content as Value / Filename
    When assigning values to content properties, the value is interpreted as a file name. If there is no matching file, then the value is written into a temporary file and the file is assigned to the property.
    Using “OLE DB” and “ODBC”, values longer than 4000 characters are written to a temporary file and the file name is passed as value.
    This behavior can be changed through the options “Content as Value“ and “Filename”.
    - Content as Value
      The value is interpreted as a string, never as a file name. Using “OLE DB”, values are not written to temporary files but kept in the memory as strings. Values exceeding a size of 500 KB are cut.
    - Filename
      The value is interpreted as file name.
  - Unique Entries in List
    Before adding a value to a list, the list is searched. If the list already contains a certain value, it is not added. Existing double entries are not removed.
    Note: This Option can be used to repeat data imports without producing double entries.
  - Must Be Defined in Aggregate
    An aggregate entry is only produced if the property at which this option is set has a value. (e.g. used for a list of phone numbers).
  - Skip Create Value Method
    The action, defined in the *Constructor Action* field of the property will not be executed.
  - Skip Set Method
    The action, defined in the *Action Called Before Property is Saved* field of the property will not be executed.

4.5 Object Relations

In this field, imported objects can be assigned to object pointer properties.

- *Source Object Class*
  In this filed select the object class of the source object. The source object is the object you want to assign to a certain object pointer property.
- *S-ID*
  In this field enter the object ID defined for this objects in the *Mapping* field.
- *Target Object Class*
  In this field select the object class of the target object. The target object is the object to whose object pointer property the source object shall be assigned.
- *T-ID*
  In this field enter the object ID defined for this objects in the *Mapping* field.
- *Target Property*
  In this field, select the object pointer property you want to fill.
- *Update Mode*
  In this field, the import behavior is determined.
  - *Set if Created*
    The value is only set if a new object is created.
  - *Update*
    The value is set or added to a list. Existing values are not overwritten by blanks.
  - *Overwrite*
    The value is set or added to a list. Existing values are overwritten by blanks.
  - *Ignore*
    The line is ignored.
- *Options*
  For the values to be imported further conditions can be defined.
  - Aggregate Key
    For aggregates, instead of using the keys defined in the data model, this option can be used to define own key properties.
  - Unique Entries in List
    Before adding a value to a list, the list is searched. If the list already contains a certain value, it is not added. Existing double entries are not removed.
    Note: This Option can be used to repeat data imports without producing double entries.
  - Must Be Defined in Aggregate
    An aggregate entry is only produced if the property at which this option is set has a value. (e.g. used for a list of phone numbers).
  - Skip Create Value Method
    The action defined in the *Constructor Action* field of the property will not be executed.
  - Skip Set Method
    The action defined in the *Action Called Before Property is Saved* field of the property will not be executed.

4.6 Class Properties

This field is used to control the behavior of creating and updating objects of certain object classes. To search for objects different methods are available.

- *Object Class*
  Create an entry for each object class to be imported. If for an object class different IDs are used, create an entry for each ID of the object class.
- *ID*
  Enter the ID assigned to the object classes in the *Mapping* field.
- *Create Objects*
  Select this option to create objects of that object class that do not yet exist in Fabasoft Folio.
- *Avoid Duplicate Objects*
  To avoid creating duplicate objects, different methods are available. They are described in chapter 4.6.1 “Avoide Duplicate Objects“.
  - “No Check of Existing Objects”
  - “Check by Collecting Keys of All Objects“
  - ”Check by Query for Each Object“
  - “Check by Query for Each Object (No Cache)“
  - “Check by Hash Table Query“
  - “Check by Local Query“
  - “Use Same Object as Lower Object ID“
- *Query Scope*
  In this field, a *Query Scope* object can be used for restricting the search. This allows improving the performance of the search.
- *Options*
  These options can be used to prevent running certain actions for objects of the selected object class.
  - “Skip Prepare Commit Method“
    The action `COOSYSTEM@1.1:ObjectPrepareCommit` will not be executed.
  - “Skip Finalize Commit Method“
    The action `COOSYSTEM@1.1:ObjectFinalizeCommit` will not be executed.
  - “Skip Committed Method“
    The action `COOSYSTEM@1.1:ObjectCommitted` will not be executed.
  - “Skip Object Constructor Method“
    The action `COOSYSTEM@1.1:ObjectConstructor` will not be executed.

4.6.1 Avoide Duplicate Objects

This section describes the various configuration options in property *Avoid Duplicate Objects*.

No Check of Existing Objects

This setting is recommended if the import process does not create objects that already exist in the Fabasoft product environment. A key property is not required. Objects are always created (even if *Create Objects* is set to "No"), except when key properties (one or more) are defined, but these are all empty.

Advantages:

- Quick
- Simple

Disadvantages:

- No check for duplicate objects
- Difficult error handling

Check by Collecting Keys of All Objects

In an initialization, a local search tree (Cache) is set up in which the key values and the object address of all objects of the object class are stored.

During the import, these keys are searched for a matching object. If no matching object is found, a new object is created and the key also is added to the list of existing objects. This method is useful if many changes are carried out, references are placed on objects or new objects are created and the number of existing objects is not too extensive.

Advantages:

- Quick search in the local search tree

Disadvantages:

- Initialization needs much time
- High memory usage
- Problems, if objects of many object classes shall be collected

Check by Query for Each Object

For each imported object, a search is performed, which checks whether the object already exists. The keys of found objects are stored locally and thus quickly found again. This method is much slower in finding than the method described above. It is particularly well suited when only few objects of a class are accessed. Moreover, less memory is needed.

Advantages:

- No initialization

Disadvantages:

- Slow search per record

Check by Query for Each Object (No Cache)

This option corresponds to the option "Check by Query for Each Object", but keys of found objects are not stored. It makes sense when objects are not used more than once or the memory consumption would be too great.

Advantages:

- No initialization
- Low memory usage

Disadvantages:

- Slow search per record
- No cache for already found keys

Check by Hash Table Query

The object address is calculated from the key value using a hash algorithm.

Advantages:

- In special cases more efficient

Disadvantages:

- Creating objects only possible via COLD or scripts
- Only makes sense for a limited number of objects

Check by Local Query

The keys of the objects that are created by the current import are stored. Not the server is searched for objects but the stored keys. This method is suitable if the keys of the imported data are not already in use for existing objects.

Advantages:

- No initialization
- No search per record
- Quick search in the local search tree

Disadvantages:

- Existing objects are not found

Use Same Object as Lower Object ID

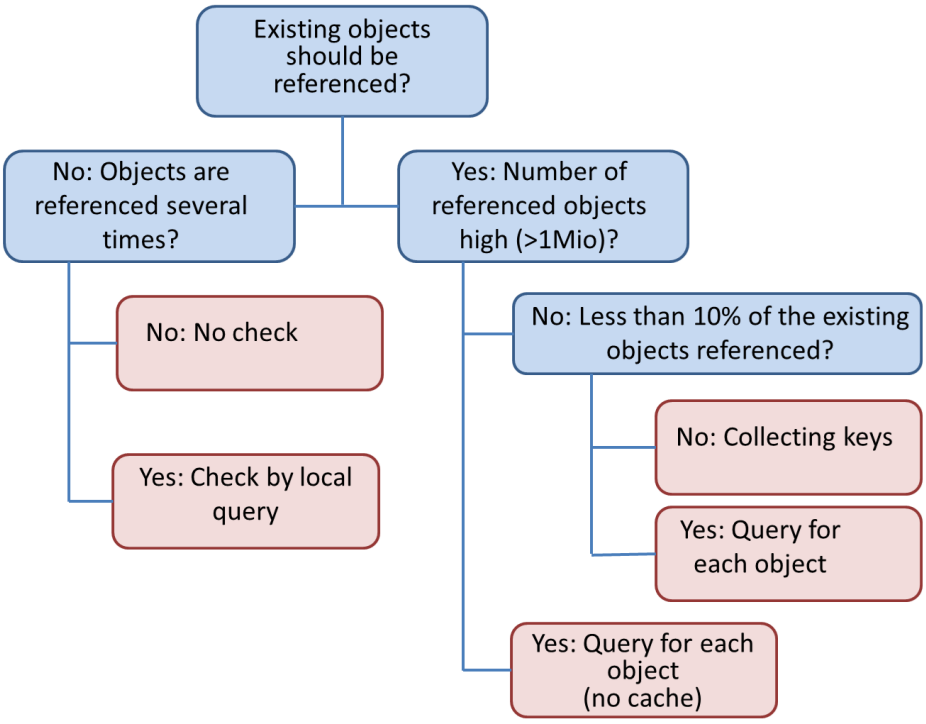
This method can only be used in combination with another search method.

Using this option it is possible to make several list entries (based on one record) within one object, e.g. several phone numbers of a person.

This requires defining the keys and the search method for the object class. Further list entries can be defined using a higher object ID and selecting the search method “Use Same Object as Lower Object ID“.

4.6.2 Digression: Selecting the Search Method

The following graphic may serve as a decision aid for selecting the search method, but for each loader the right method must be verified.



Search via object address

If the object address of an object is known, it can be used as a key. Through that the performance is very high.

To do so, before the import static object classes should be exported and it should be possible to get the object address in a `JOIN` on the database using the key properties.

Search for the reference

The reference name of component objects can be used for searching. Thereby two possibilities are available: It is possible to use the reference including the software component or, if the reference is unique, it can also be used without software component. To use this option, for the *Mapping* the property *Reference* (`COOSYSTEM@1.1:reference`) has to be selected.

Get methods for key properties

Get methods for key properties can have the consequence that the value in the database does not match the value in the object property. So the query for each object will have another result than the check by collecting keys of all objects.

Set methods for key properties

If the value of a key property is changed by a set method, this change is not considered in the local search tree, whereby the search can get inconsistent.

Optimization

It is advantageous to use only one key property per object class. Using search methods that search per object, in the database this property should be provided with an index on the value.

Particularly suitable as an index are *External Key* (`COOSYSTEM@1.1:objexternalkey`) and *Subject* (`COOSYSTEM@1.1:objsubject`). The property *External Key* has been defined for this purpose, but has the disadvantage that it is not efficiently indexed. The property *Subject* is displayed in many forms and is only suitable if the key values contain “readable” information. This property is after definition of an index very efficient in finding.

4.7 Number of Records for Commit

Use this property to determine the maximum number (and using the group change also the minimum number) of records to be processed in one transaction.

Some actions when loading objects can be performed on multiple objects (for example, the "commit", which writes the data to the database), whereby the number of communication steps is greatly reduced. However, very large transactions involve the risk that they collide with other transactions, and thus take longer than several small transactions. In many cases the default value of 150 is a value that leads to a good result.

4.8 Number of Threads

Use this field to define the number of threads that are used by the client to generate objects. By multiple threads, it is possible to parallelize the processing. Depending on the type of import (e.g. creating or modifying objects) and, depending on the performance of individual components, a large increase in performance can be achieved. On the other hand, a certain synchronization effort is connected with the use of parallel transactions, so that an optimum value mostly is found in a small number (less than 5). The optimum value should be determined experimentally for the particular application.

The use of numerator properties prevents the use of multiple threads, because numerators remain locked until the end of the "Commit" and therefore allow no more parallelism. The restriction applies only when new objects of that class may be created (hence if *Create Objects* is set to "Yes" or "Undefined").

4.9 First Record

In this property, specify the record number that is applied in the database to the record from which starts the loading process.

4.10 Number of Records to Be Read

Use this property to determine the number of records to be loaded. Otherwise all records are loaded.

4.11 Logging Mode

Use this property to determine whether a log is created or not and which information of each import process is logged.

Available options:

- “No Log“
  No log is created.

- “Log Errors“
  Only errors (and their raw data) are logged.
- “Full Log“
  All raw data, the addresses of the created objects and error messages are logged.

4.12 Log Object

The created logs are written to a *Data Import Log* object which is assigned to this field.

4.13 Maximum Number of Entries in Log

Use this property to determine the maximum number of logs in the log object (*Data Import Log*).

Note: If the maximum number of logs is exceeded the eldest log is deleted.

4.14 Skip Methods Without Check?

Use this property to determine whether a review action is called, which determines whether object classes and attributes match the specified object classes and attributes.

4.15 Skip Setting Modification Properties

Use this property to determine whether the properties *Last Change on/at* and *Last Change by* are updated for the objects that are changed during the data import.

4.16 Abort Data Import Immediately on Error

Use this property to determine whether the data import is aborted if an error occurs.

4.17 Skip Object Refresh

Use this property to determine whether objects are updated during import or not. If it is ensured that problems such as inconsistent caches cannot occur, the data import can be accelerated using this option (e.g. for the initial data import).

4.18 Filter

On the “Filter” tab of the data import object, filters can be configured. The available properties are described in the following chapters.

4.18.1 Filter Expression for RAW Data

This property defines an expression, which is called for each record of the database. The expression can be used to modify raw data. More information about the parameters contained in the global scope can be found in the reference documentation.

For modifying raw data, a Fabasoft app.ducx expression can be used.

The modification of raw data can include

- the calculation of values

- the formatting and
- filtering.

4.18.2 Filter Expression for Objects

This property defines an expression, which is called for each handled record after all properties from the current record have been set and before the transaction is committed. More information about the parameters contained in the global scope can be found in the reference documentation.

The modification of objects can include

- versioning,
- setting additional properties and
- key numerators.

4.18.3 Filter Expression for Objects After Commit

This property defines an expression, which is called for each data record after the transaction has been committed. More information about the parameters contained in the global scope can be found in the reference documentation.

4.18.4 Digression: Numerators

The value of numerators is calculated by default from the property constructor. This is done as follows: The numerator is locked in a separate transaction, subsequently the value is increased and finally the process is finished by a "commit".

Using *Fabasoft Folio/COLD* the following possibilities are available for handling numerators:

Automatically

- The numerator is locked in the COLD transaction.
- All objects within the transaction use the numerator.
- COLD transaction commit.

Note: The import can be done using only one thread.

Manually

- The property constructor is ignored.
- The value is set explicitly.
- In an own step the numerator object is updated using the action `NUMERATOR@1.1001:UpdateValue`.

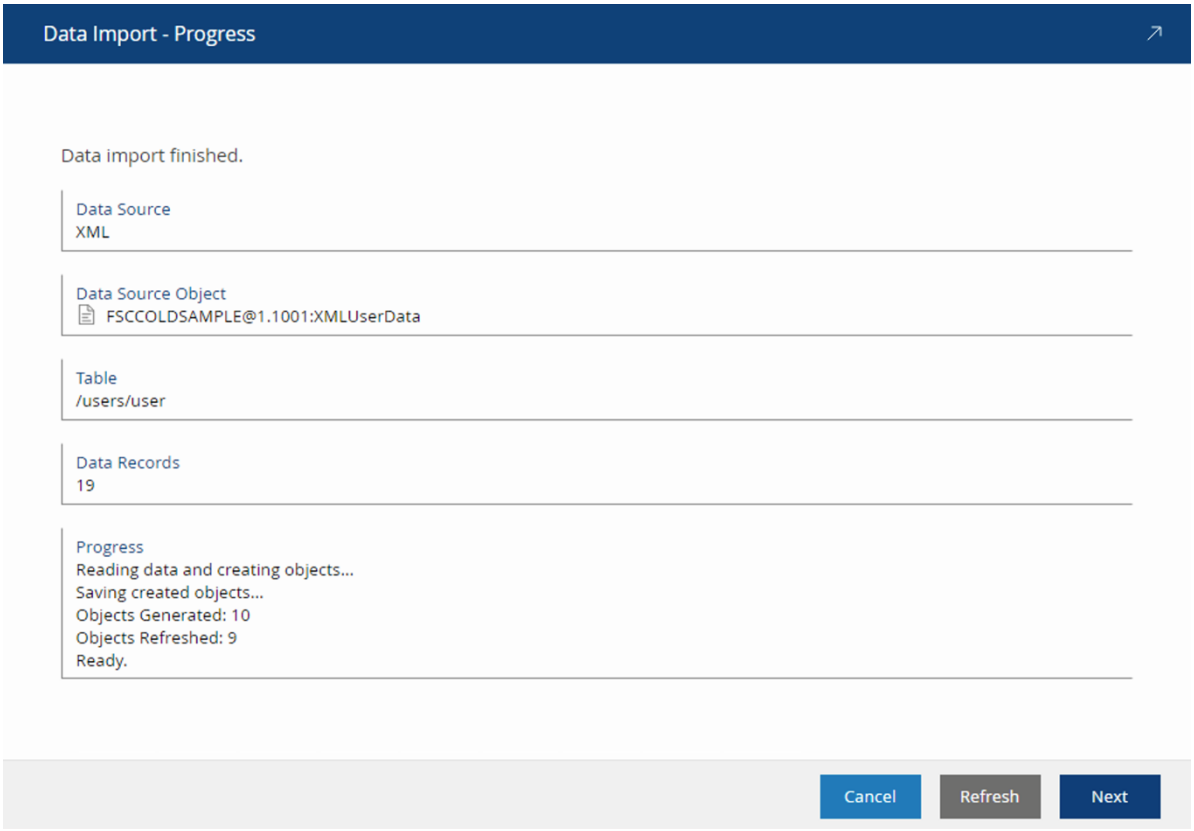
5 Import and Update of Data

For importing or updating of data a *Data Import* object has to be created and configured (for more information, see chapter 4 “Configuration of the Data Import”.)

5.1 Manual Import

To start the import, use the context menu command “Import Data“ of a data import object.

The import progress is displayed in an own window.



If during an import process no objects can be created anymore because e.g. the server goes down, read records are not lost. The data are written to a log file and the import process can be continued at a later date (context menu command “Roll Forward”).

Note: When importing data from Microsoft Office Excel to Fabasoft Folio, the files have to be closed that the import process can be processed successfully. Attention: If files are open, no error message is displayed.

5.2 Programmatical Import

For programmatical import (e.g. via script) the action *Remote-controlled Import* (`FSCCOLD@1.1001:ImportRemote`) is available with the following parameters:

- `Showdialog`
  Use this parameter to determine whether the progress dialog is displayed.
- `closedialog`
  Use this parameter to determine whether the progress dialog is closed at the end of the import without confirmation.
- `reportcreatedobjects`
  If this parameter is set to `TRUE` and the parameter `asynchron` is set to `FALSE`, the created objects are returned in the `createdobjects` parameter.
- `createdobjects`
  In this parameter the created objects can be returned.

- `rollforward`
  Use this parameter to determine whether an abortes import process is continued.
- `asynchron`
  If this parameter is set to `TRUE`, the import takes places in the background.
  Note: `FALSE` is only suitable for a small number of objects, because the client cache can not be cleaned up in this case.
- `paramobject`
  This parameter can contain any COM object.
- `finishedscript`
  This parameter can contain a content that is called as a script at the end of the import process.
- `synchobj`
  If the `asynchrony` parameter is `TRUE`, in `synchobj` a COM object is returned.
  The COM object is not registered and therefore cannot be created explicitely. It implements `IDispatch` and makes available the following functions:
  - `Wait()`
    Waits for the end of the import.
  - `IsFinished()`
    Returns `TRUE` if the import is finished, otherwise `FALSE`.
  - `Cancel()`
    Cancels the import.

5.3 Example

In this section basis of a very simple, concrete example, all steps of a data import are demonstrated.

5.3.1 Initial Situation

Organizations and their employees shall be imported. Exemplary the following data are available:

Organization:

- Name
- Branch of industry

Contact Person:

- Last name
- First name
- E-mail address

Employee:

- Relation between organization and person (m:n Relation)

The following Microsoft SQL server tables can be produced in the data base “migration“:

Organization

	Column Name	Data Type	Allow Nulls
	ID	int	☐
	Name	varchar(MAX)	☑
	Branch	varchar(MAX)	☑

Content:

	ID	Name	Branch
	1	AA GmbH & Co. KG	Research
	2	BB AG	Pharma

Contact Person

	Column Name	Data Type	Allow Nulls
	ID	int	☐
	LastName	varchar(MAX)	☑
	FirstName	varchar(MAX)	☑
	EMail	varchar(MAX)	☑

Content:

	ID	LastName	FirstName	EMail
	1	Maler	Otto	otto.maler@aa.at
	2	Winkler	Karin	karin.winkler@bb.at

Employee

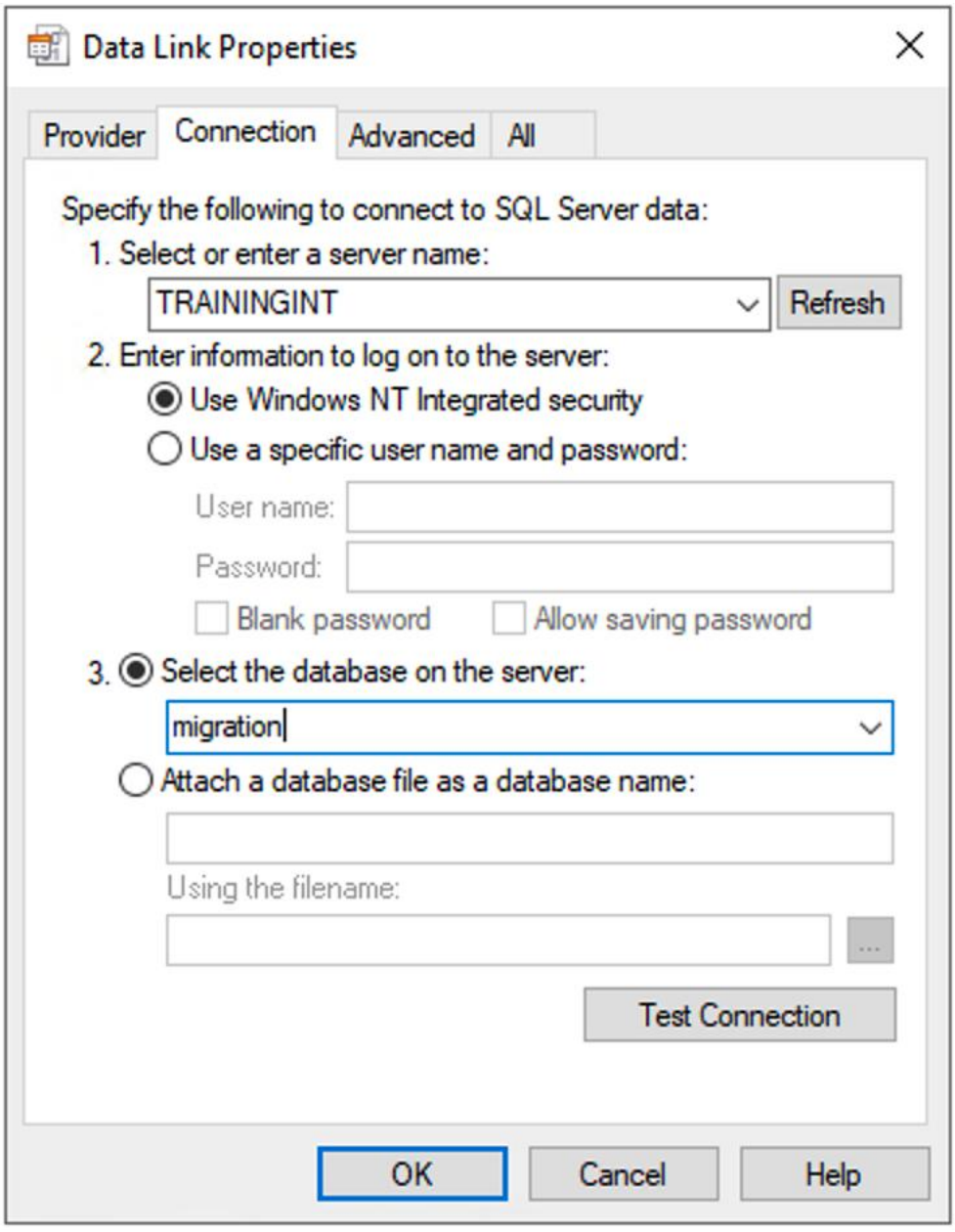
	Column Name	Data Type	Allow Nulls
	OrgID	int	☐
	PersID	int	☐

Content:

	OrgID	PersID
	1	1
	2	2
	1	2

5.3.2 Procedure

To be able to access the data, the OLE-DB interface is used and for that an UDL file is created (see chapter 3.2 ”OLE DB“). As provider, in the properties of the UDL file (“Provider” tab) select the “Microsoft OLE DB Provider for SQL Server”. On the “Connection” tab select the relevant Microsoft SQL server, enter the connection information and specify the data base.



In the Fabasoft Folio domain, create a *Data Import* object for each table of the data base (Organization, Person and Employee).

Organizations

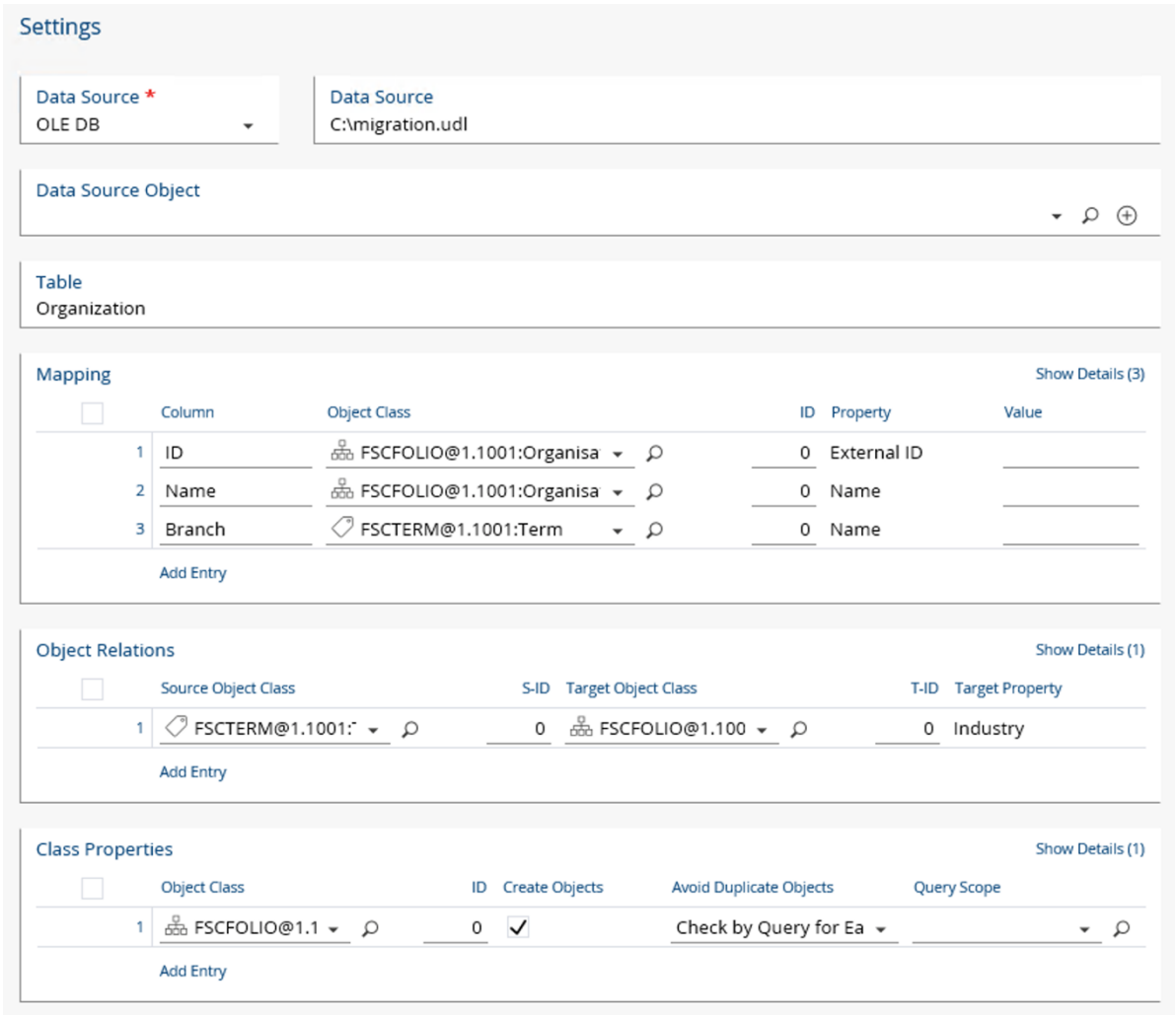
The organizations are imported first.

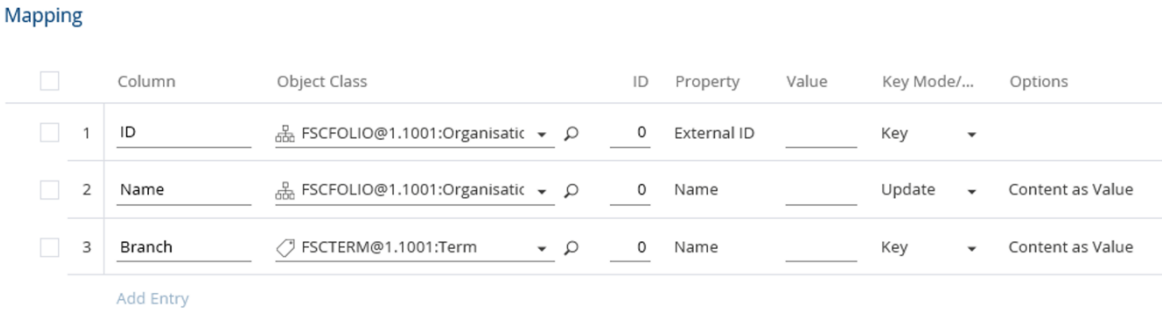
For each object property to be set an entry in the *Mapping* field is necessary. For unique identification, the ID is assigned to the property *External Key* (`COOSYSTEM@1.1:objexternalkey`) that is part of each object. The name is used as object name.

The property for the branch of industry needs a term object. Therefore to steps are necessary. First, in the *Mapping* field, for the branch select the object class `FSCTERM@1.1001:Term`, select the Property *Name* and determine to use it as a "Key" (for each branch the same object is used). Second, in the *Object Relations* field, assign the term to the property *Industry* of the organization.

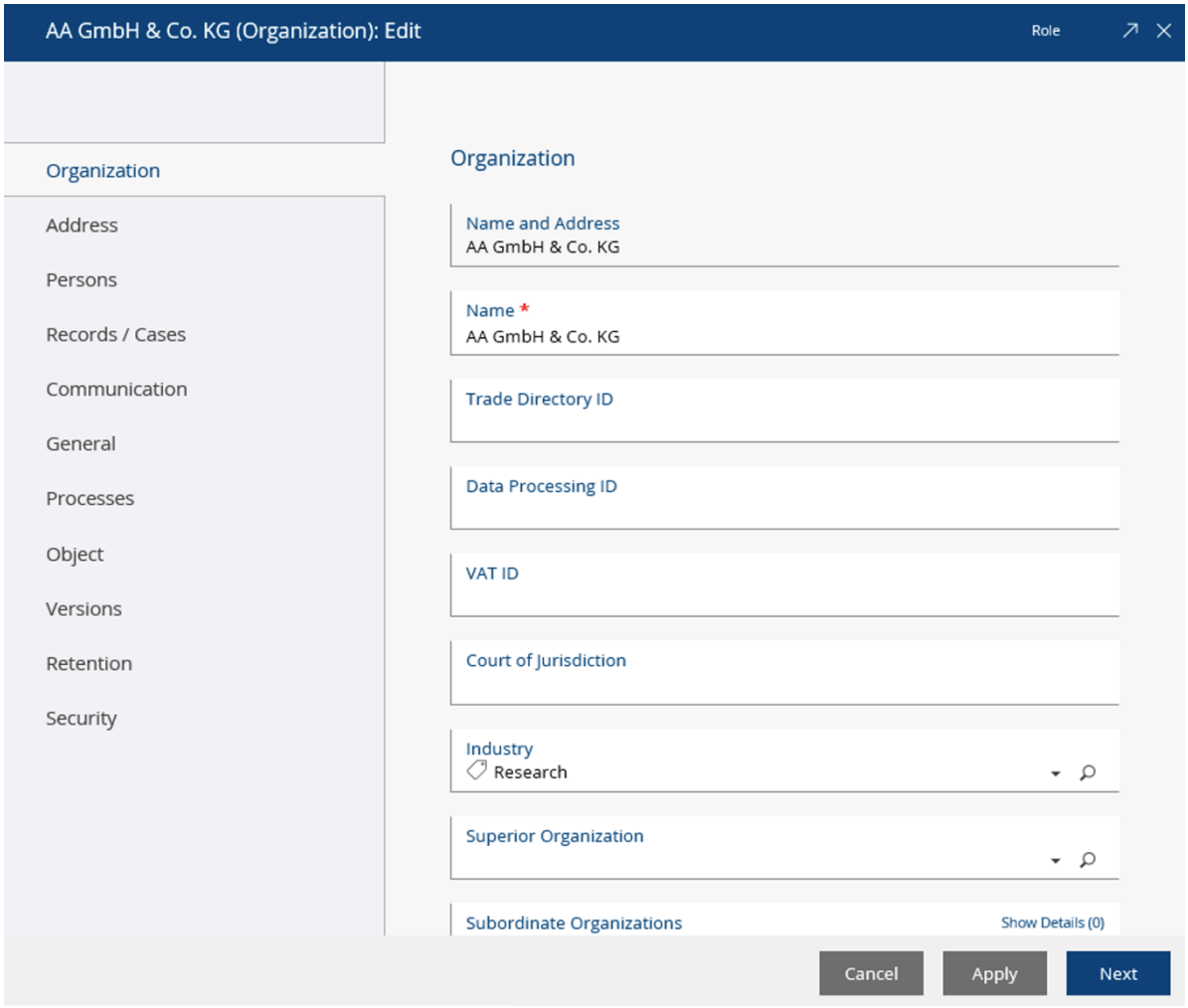
In this example no further values are specified in the table.

To avoid duplicate organization objects, in *Class Properties* select "Check by Query for Each Object" for object class (`FSCFOLIO@1.1001:Organisation`).





Result (Based on the data of chapter 5.3.1 “Initial Situation”)



Contact Persons

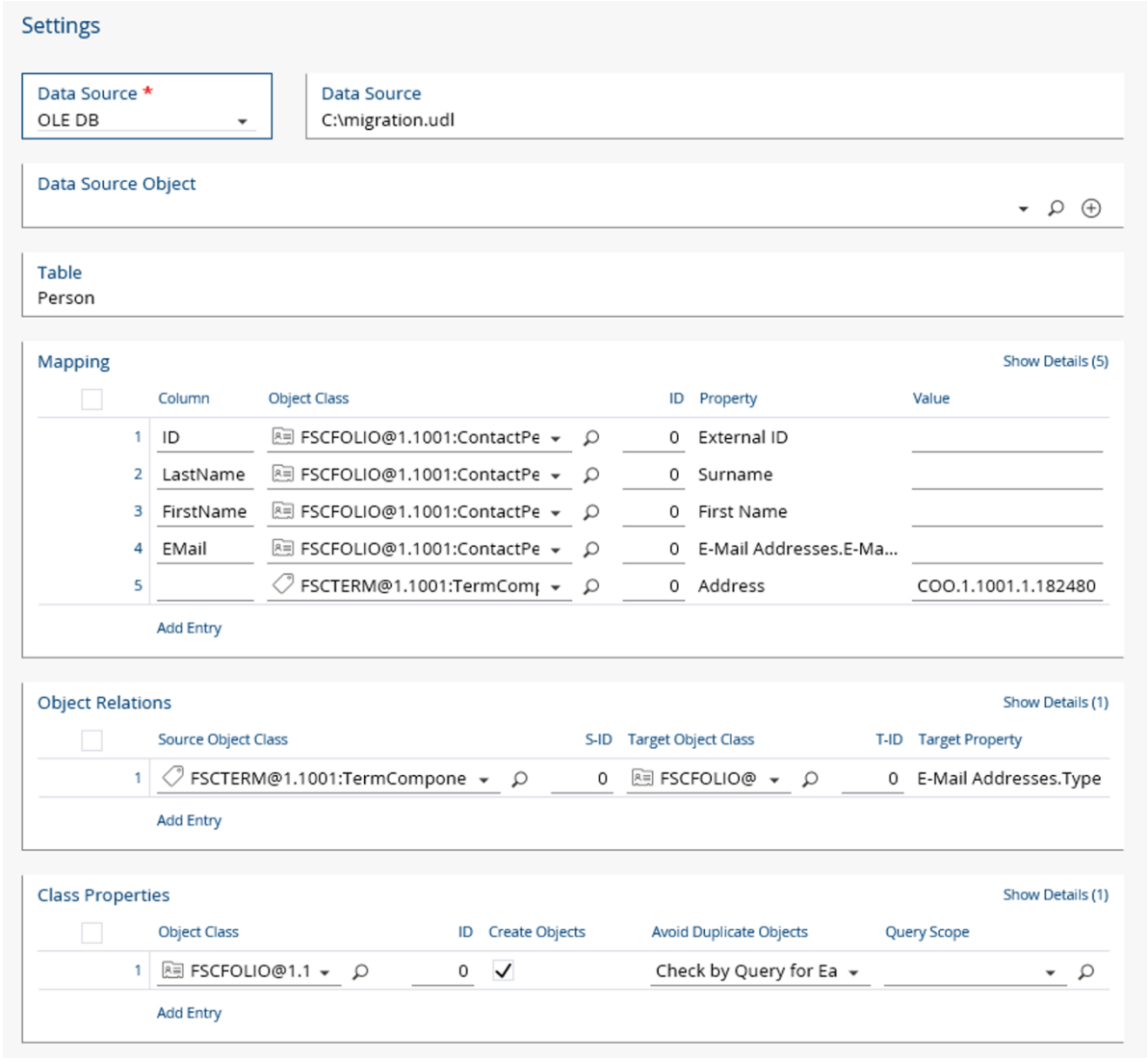
For each object property to be set an entry in the *Mapping* field is necessary. For unique identification, the ID is assigned to the property *External Key* (`COOSYSTEM@1.1:objexternalkey`) that is part of each object. For surname and first name two further entries have to be created.

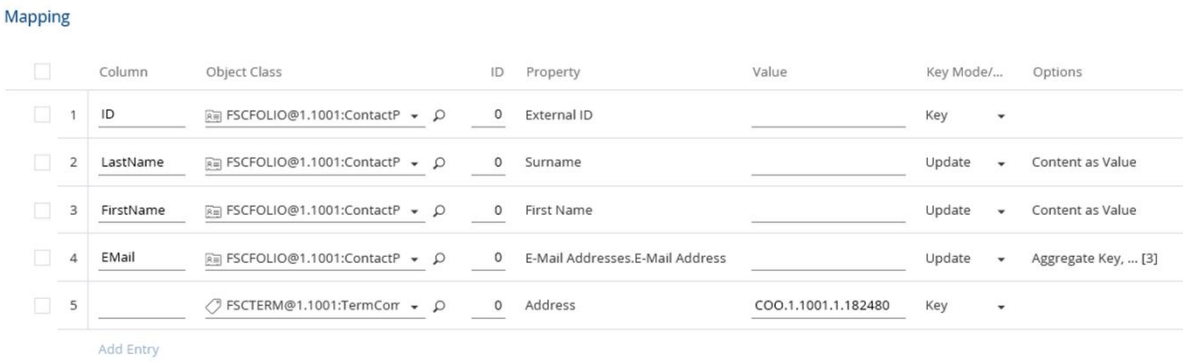
For the e-mail address, two things have to be considered:

The property *E-Mail Addresses* (`COOMAPI@1.1:emailinformation`) is an aggregate. The single parts of the compound property can be addressed by a property path. To avoid duplicate entries (during multiple imports) the e-mail address is used as aggregate key (option “Aggregate Key”). Additionally select the option “Must Be Defined in Aggragate“ to avoid entries when there is no e-mail address and the option “Content as Value” to interpret the contents of the database as a value.

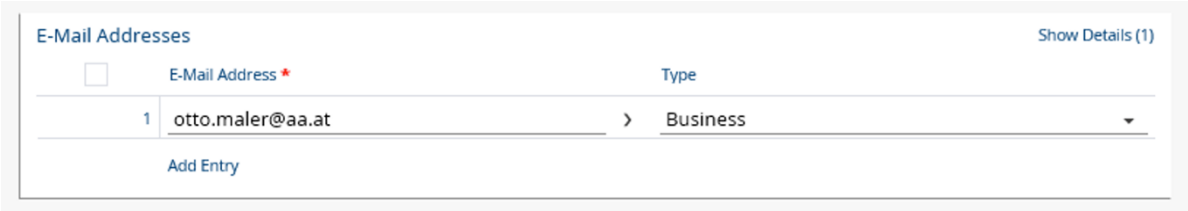
As address topic three terms (`FSCTERM@1.1001:TermComponentObject`, “Business”, “Private”, “Other”) are available. Our e-mail addresses get the topic “Business”. In the *Mapping* field, for object `class FSCTERM@1.1001:TermComponentObject`, select the Property *Address* (`COOSYSTEM@1.1:objaddress`) and as value type the object address of the desired term. In the

Object Relations field, assign the term object to the property path “E-Mail Adresses.Topic” of *Persons*.





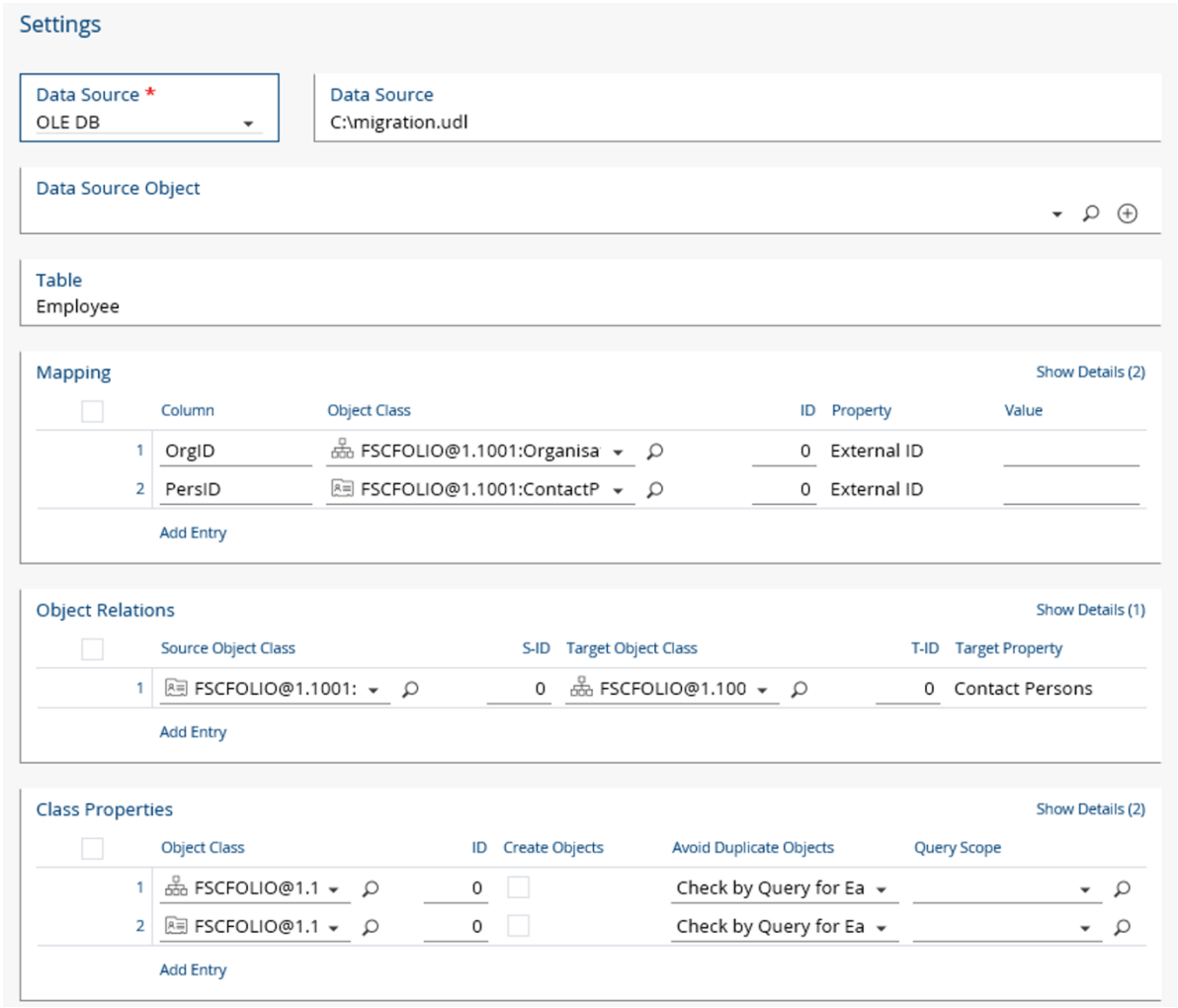
Result (Based on the data of chapter 5.3.1 “Initial Situation”)



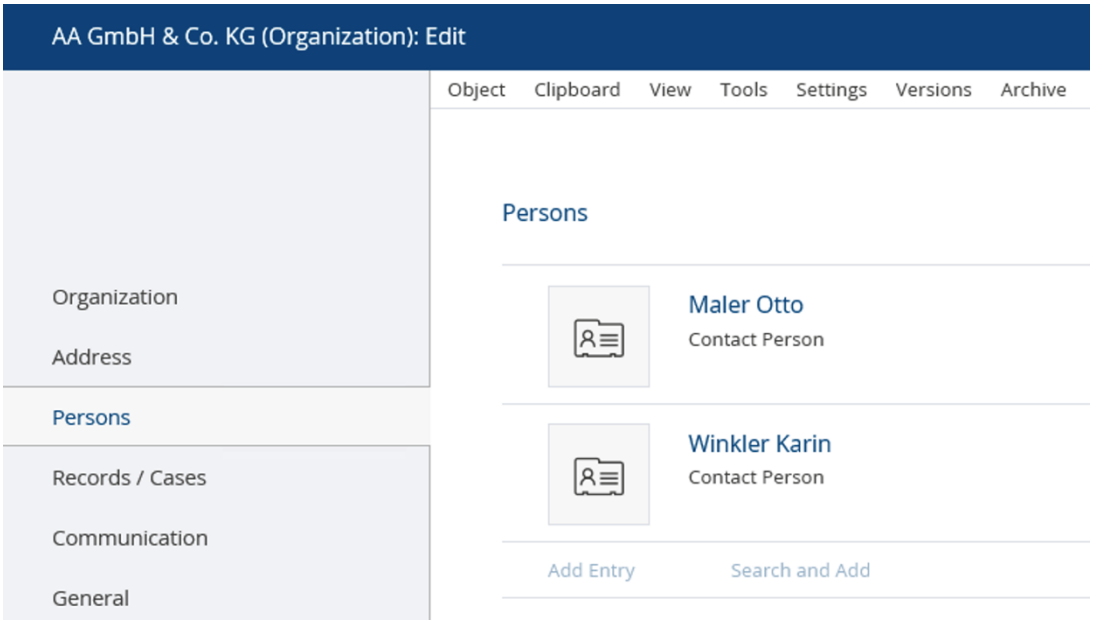
Relations

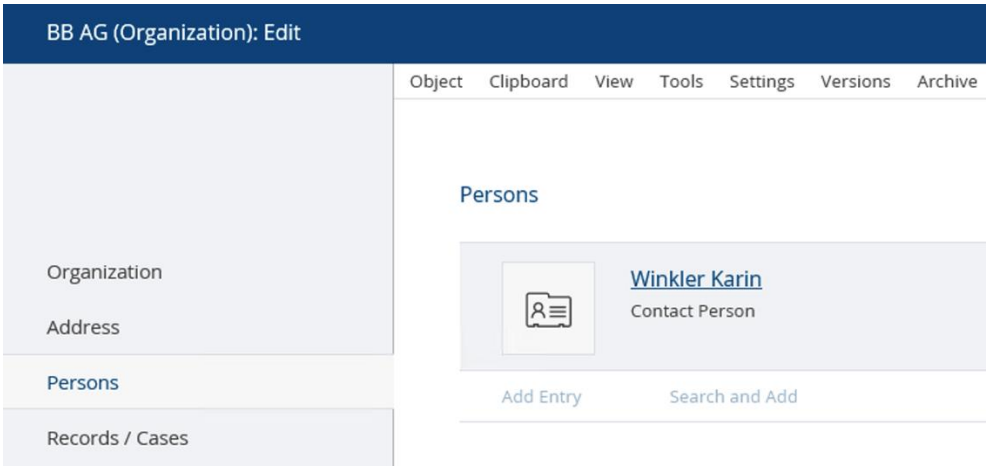
The relation between persons and organizations is created through a further import. Therefore the property *External Key* of *Organizations* and *Persons* is used. To avoid double entries of

persons in organizations, in the *Object Relations* field use the option “Unique Entries in List”. In the *Class Properties* field is defined, that objects that do not exist yet are not created.



Result (Based on the data of chapter 5.3.1 “Initial Situation”)





The next chapter gives an example for a file import.

6 The Fabasoft Folio/COLD File System Import

Using the datasource type “File System” directory structures and files from the file system can be read and used to generate Fabasoft Folio objects.

The Fabasoft Folio/COLD File System Import makes available file names associated with directories and their sub-folders in a table.

The table is structured as follows:

- ParentName
  Contains the name of the parent folder (e.g. “import“).
- ParentPath
  Contains the path of the parent folder (e.g. “e:\import“).
- ParentKey
  Contains a combination of the import address and the relative path to the parent folder.
- <ext>Name
  You have to specify all file extensions that occure among the files that should be imported. For each extension, a column of the form "<ext>Name" is produced, e.g. for the extension “docx”, a “docxName” column is produced and all DOCX files are entered in this column, e.g. “Document1.docx”.
- <ext>Path
  For each specified extension, a column of the form “<ext>Path“ is produced, e.g. for the extension “docx”, a “docxPath” column is produced and the path of each DOCX file is entered in this column, e.g. “e:\import\Document1.docx“.
- <ext>Key
  For each specified extension, a column of the form “<ext>Key“ is produced, e.g. for the extension “docx”, a “docxKey” column is produced and a combination of the import address and the relative path of each DOCX file is entered in this column. This column can be used as key in the Fabasoft Folio/COLD import.
- Extension
  Contains the file extension of the file.

Note:

- “Folder” is a known extension which can be used to import folders.
- “*” can be used as extension which means that all other file formats of the source directory are contained in columns with the * extension.
- Using Fabasoft Folio/COLD File System Import it is possible to import from both directories and ZIP files. Using the import via a ZIP file the content can assigned via the property “Data Source Object” (`FSCCOLD@1.1001:datimpobjdatasource`).

7 Logging the Data Transfer

If the data transfer is logged (see chapter 4.11 “Logging Mode”), in the *Log Object* a new entry is created. If there is no *Data Import Log* for the *Data Import* object, in the *Log Object* property a new object is created.

A complete protocol can contain the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Data Import Record for 'DI', Montag, 10. Mai 2004, 16:52:03-->
<Log><Entry type="data">import;e:\import;e:;;Remarks;
e:\import\Remarks.doc</Entry>
<Entry type="data">import;e:\import;e:;;;</Entry>
<Entry type="update">COO.1.115.2.1002246 (Ordner)</Entry>
<Entry type="create">COO.1.115.2.1002247 (Word-Objekt)</Entry>
</Log>
```

8 Optimizations

For the acquisition of many records, it is important to perform appropriate optimizations to ensure adequate performance.

8.1 Fabasoft Folio/COLD

The following configuration options (see chapter 4 “Configuration of the Data Import”) influence the performance of *Fabasoft Folio/COLD*:

- Sorting/Group by (see chapter 4.4 “Mapping“ Group by)
- Skip running actions (see chapter 4.6 “Class Properties“ - Options)
- Search methods (see chapter 4.6.1 “Avoide Duplicate Objects“)
- Transaction size (see chapter 4.7 “Number of Records for Commit“)
- Parallelization (see chapter 4.8 ”Number of Threads“)
- Summarizing loaders: Avoid refresh of objects

In addition, a kernel trace can bring information about performance problems.

8.2 Client

The client can be optimized by the following parameters:

- Sufficient client cache
- Appropriate bandwidth for network connectivity
- A powerful CPU instead of multiple weaker CPUs
- Distribute parallel loaders on multiple clients (Fabasoft Folio kernel)

8.3 COO Service

The performance monitor can bring information about performance bottlenecks. Creating data base tables in the mentioned cases leads to a performance enhancement.

- Performance monitor (eventual adaption of client cache, optimizing the number of threads)
  - COO-COOService\% Cache Hit Rate
  - COO-COOService\% Cache Used
  - COO-RPCs\Allocated Threads
- Table definition for services
  - Define tables before loading data
  - For properties that are used for finding, an index should be created.

8.4 MMC Service

- Performance monitor (identify bottlenecks)
  - % Disk Time
  - Disk Queue Length
  - Disk Read Bytes/s
  - Disk Write Bytes/s
- RAID
  - Use RAID-1/10
  - Usually RAID-5 is slower to write
- Enter the following values to the registry under `HKey_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Filesystem` (boot after changing):
  - NtfsDisable8dot3NameCreation=1
  - NtfsDisableLastAccessUpdate=1
  - NtfsMftZoneReservation=2...4

8.5 Data Base System

- Optimize indexes
  - As few additional indexes
  - Create indexes for searching
- Update statistics
- Recovery mode: simple/full (use “Simple” for initial load)
- Avoid automatic grow/shrink
- Device placement