



## Export-Import Routes

### A.

#### **Physical migration of entire EDRMS system**

*(EDRMS installation, customisations, content and metadata)*

##### A.1 Physical relocation of existing servers from department to department.

###### Considerations:

- The creation of entire system backups (“disk images”) to be retained by the originating department for an interim period. These are intended to guard against any potential failures in the transfer process and should be deleted as soon as the completed transfer has been fully verified or upon expiry of the agreed interim period, usually only 3-6 months
- The effective (non-retrievable) removal from those servers and backup/relocation of data not intended for transfer (data to be retained by the originating department)
- The uninstallation from those servers of non-EDRMS related software (with particular regard to licensing restrictions)
- The supply of supporting documentation relating to the service history and administration of those servers
- System analysis at the recipient department to ensure the servers can be networked and tolerated without system software (or even hardware) conflicts. Should also include vetting against viruses, spyware and malware.
- Installation on the servers of non-EDRMS related software (usually to do with network-compatibility at the recipient department).

##### A.2 Procurement and installation at the recipient department of the same EDRMS software as the originating department in preparation for a homogeneous export-import (see *also* section b. below)

###### Considerations:

- The creation of entire system backups (“disk images”) to be retained by the originating department for an interim period. These are intended to guard against any potential failures in the transfer process and should be deleted as soon as the completed transfer has been fully verified or upon expiry of the agreed interim period, usually only 3-6 months

- Careful system analysis of the originating department installation (including reference to supporting documentation relating to its service history and administration) to ensure that the recipient installation is fully compatible (or backwards-compatible) with the data transfer. Careful attention to software versions, customisations, patches and service packs applied.

## **B.**

### **Homogenous EDRMS export-import**

*(matching, parallel EDRMS software)*

In theory this should prove a straightforward mechanism if due care has been taken to ensure that both EDRMS system installations are fully compatible (or backwards-compatible) with one another (see above). Other considerations would need to cover:

- The creation of entire system backups (“disk images”) to be retained by the originating department for an interim period. These are intended to guard against any potential failures in the transfer process and should be deleted as soon as the completed transfer has been fully verified or upon expiry of the agreed interim period, usually only 3-6 months
- The format to be used for the export-import. This will often be a proprietary format in this scenario, but it could equally utilise open, standard formats such as XML. Direct database connectivity might be used in the implementation (see C.5. below).
- The type of export, typically whether it is ‘live’ or a ‘disposal’. This will have implications for the ongoing use of the data once imported to the recipient EDRMS. A ‘disposal’ may reduce the imported data to a ‘read-only’ state
- The extent of the export, typically whether it is full or in part. The recipient EDRMS will need to build a corresponding FilePlan framework in order to accept the incoming data structure. There may be issues for future, complementary transfers if the FilePlan framework is different or incomplete. This may also be the case if the recipient EDRMS already contains a working FilePlan framework into which the incoming data structure may need to be incorporated.

## **C.**

### **Heterogeneous EDRMS export-import**

*(different EDRMS software)*

This path is complicated by such factors as mismatching FilePlan frameworks, mismatching field (or even level) customisations, mismatching use of terminology, mismatching interpretation of fields and mismatching use of data types or conventions.

The export-import route will require a detailed and often sophisticated programmatic data conversion methodology, involving the combined input of users, records managers and software engineers (programmers).

A detailed mapping exercise will need to be undertaken to cross-walk the location of data in the donor system to its logical equivalent in the recipient system, or the *nearest* logical equivalent where mismatching data structures exist.

Note that this cross-walk may often be more complicated than a simple one-to-one relationship. It may require the concatenation of several elements in the donor system into one recipient field, or conversely the parsing and separation of a single data element in the donor system into several recipient fields. Differing data types or formatting conventions across the systems may also require that data elements be rearranged or modified as part of the mapping process, for example date/time formatting.

Furthermore, exports may need to be executed in sections rather than in whole, and physically separate files may be produced which represent different logical parts of the system, such as Folders or Parts. These fragments will need to be recombined in the recipient system and this may need to be done as part of the import, if not before.

The programmatic technique used to perform the data conversion will depend on the source and destination of the data formats involved. Five typical scenarios along with suggested programmatic conversion techniques are given below for consideration:

#### C.1. XML *source* – XML *destination*

Consider a native transformation using XSLT. Exceptionally complicated conversion requirements might more easily be accomplished through additional pre- or post-processing using a scripting language with superior string manipulation capabilities, such as Perl.

#### C.2. XML *source* – TSV/CSV *destination*

Consider transformation using XSLT as the core solution, especially given XSLT's ability to process sequentially with contextual hierarchical awareness. Additional conversion requirements can still be accomplished through additional pre- or post-processing using a scripting language with superior string manipulation capabilities, such as Perl.

#### C.3. TSV/CSV *source* – TSV/CSV/XML *destination*

Conversion in a scripting language with superior string manipulation capabilities, such as Perl, is probably the only option. Great care should be taken with CSV files that incidental, textual commas are properly escaped and distinguished from commas used as field-separators. Similarly, the presence of tabs (CHAR 9) within data fields may have an untoward effect on the integrity of TSV exports (and consequent imports).

#### C.4. D/B tables *source* – TSV/CSV/XML *destination*

Consider doing this in separate stages: the data is first exported (or 'dumped') from the source database using native export facilities. Then, depending on the export functions and formats available (usually TSV/CSV/XML), and any further

requirements to do with data conversion, one of the methods outlined above at C.1-C.3 should pertain.

Alternatively, consider connecting directly to the database using an appropriate connector (such as ODBC) in order to read data directly from database fields using SQL and write data directly to files in the appropriate format. Some data conversion may still be required on-the-fly but the entire process could be accomplished using Perl to connect (via DBI), read, convert and write data to file.

#### C.5. D/B tables *source* – D/B tables *destination*

Consider doing this in separate stages: the data is first exported (or 'dumped') from the source database using native export facilities. Then, depending on the export functions and formats available (usually TSV/CSV/XML), and any further requirements arising from data conversion, one of the methods outlined above at C.1-C.3 should pertain.

Once in the appropriate import format (usually TSV/CSV/XML), the converted data might be imported via a standard bulk import process into the recipient system. Alternatively, consider connecting directly to the recipient database using an appropriate connector (such as ODBC) in order to read data from the exported files and write data directly into recipient database fields using SQL updates.

Using Perl it is even possible to combine all of these techniques: connect directly to the donor database using ODBC or DBI in order to read data directly from database fields using SQL, perform data conversion if required on-the-fly, and connect directly to the recipient database using ODBC or DBI in order to write data directly into recipient database fields using SQL updates. Whilst such round-tripping database interaction would be most readily achieved on a local network, it could feasibly also be performed over a remote connection.

*Note that it is very important to ensure that all transferred data complies with the constraints specified by the schema of the recipient database, for example that data types correspond and that maximum field length stipulations aren't exceeded. Attention might also need to be paid to character encoding conventions, especially as regards foreign or extended characters, the use of Unicode and so on.*

*Also pay very close attention to system-specific data and how this is going to be transferred in a MEANINGFUL way. For the point of view of efficiency, it is common practice to store frequently used values as ids or codes which can be looked-up within a particular system in order to reveal their true meaning. For example, '001' might mean 'open record' or 'User6' might refer to an employee called 'John Smith'.*

*Unless this coded data is fully 'translated' in situ as part of the export mechanism, or unless the means to resolve these codes (the look-up tables)*

*are included as part of the complete data transfer, then the recipient system is in danger of receiving nothing more than useless strings of letters and numbers ('001' or 'User6').*

#### **D.**

##### **Unilateral EDRMS export**

*(EDRMS software – FileSystem/Database/Other)*

The appropriate solution in this situation will depend largely on the ongoing functionality required of the data once in the recipient system.

If the recipient system is to act only as a host for the data in an essentially archived state, then the most important considerations will revolve around extracting the data from the donor system in a non-proprietary, possibly text-based, structured format which still retains the full hierarchical context of the original.

The obvious candidate here is XML, but other formats such as CSV or TSV might better apply, especially if the intention is to import the data natively into a recipient database.

In the case of a fundamental, non-resolvable mismatch between the data schemas in the two systems, exported XML could be used to capture the structure of the original and this XML could then simply be stored in the recipient database system as self-contained, large objects.

Most modern EDRMS software supports a non-proprietary export format such as XML, CSV or TSV. If the data is to migrate to the recipient system as a 'live' transfer, the core task will be to convert this donor format to one which complies to the schema and import format of the recipient. A comprehensive data-mapping exercise and one or more of the data conversion methods outlined above under C.1-C.5 should help establish the means to accomplish this.

#### **E.**

##### **Unilateral EDRMS import**

*(FileSystem/Database/Other – EDRMS software)*

The intention here will usually be to migrate data natively to the recipient EDRMS as a transfer which permits its ongoing 'live' use. First establish the import facilities and formats supported by the recipient system. Analysis of the export formats available from the donor system will then dictate the requisite conversion techniques. A comprehensive data-mapping exercise and one or more of the data conversion methods outlined above under C.1-C.5 should help establish the means to accomplish this.

#### **F.**

##### **Non-EDRMS export-imports**

*(different FileSystems/Databases/Others)*

Multiple permutations arise here depending on the export formats available from the source system and the import formats acceptable to the recipient system. No transfer can proceed without a comprehensive data-mapping exercise and detailed specification of the data conversion requirements needed to move from one to the other. Once this is done, an appropriate composite methodology should be found amongst the considerations and suggested solutions already covered under sections C., D. and E. above.