# XML Structure Import/Export in 4D v11 SQL

By Atanas Atanassov, Technical Services Team Member, 4D Inc.

Technical Note 08-26

# Table of Contents

## Abstract

This Technical Note introduces the new Structure Export/Import features in 4D v11 SQL. 4D v11 SQL Release 1 introduces support for XML-based structure manipulation. This exciting new feature allows you to export a database structure, create new databases from exported structures, and even copy and paste XML structure data between databases.

For those unfamiliar with XML, this Technical Note will also cover the basics of XML in 4D.

## Introduction

A database structure (Structure definition) can be exported to an XML file. Then, you can use this XML file to create a brand new database in 4D. This feature is available in 4D v11 SQL release 1 and above.

**Note:** *The Structure definition should not be used as a backup or for data recovery routines. The Structure definition consists of tables, fields, indexes, relations, structure editor settings and their attributes which are necessary for a complete definition of the structure. A Structure definition does not affect the database's data file. It is very important to know that there is no data inside the XML structure file.*

All exported objects create their own tags in the XML structure. All together, they form the Structure definition.

Example:

```
<root>
   <table>
          <field>
           . . . . .
          </field>
   </table>
   <index>
   . . . . .
   </index>
   <relation>
   . . . . .
   </relation>
</root>
```

Besides creating a new database, the structure definition file can be used with other technologies that support XML like HTML documents. At this point you need to use XSLT (XSL Transformation) language, which is a part of XML style sheets.

This Technical Note will cover the basics of XML language along with some syntax rules that are important for fully understanding the 4D Structure definition. We will take a quick look at 4D's XML commands and how 4D creates and manipulates XML files. Later, we will define what a Structure Definition in 4D is, and how 4D creates this structure definition and uses it.

## Introduction to XML

***Extensible Markup Language (XML)*** is a rapidly evolved technology for management, display and organization of data. It emphasizes data exchange, based on the use of tags for precise description of the data and the data structure. XML files can be opened and manipulated with any text editor. Also, it has strict syntax rules.

- Has a root element.
- Any custom defined opening tag should have a closing tag.
- XML is case sensitive. The opening and closing tag should be in the same case.
- Tags must be properly nested within each other.
- All tags attributes must be quoted.
- You can not use "<" or ">" inside XML element. It will generate an error because the parser interprets it as the start of a new element. You can use an entity reference "&lt" (Less than) and "&gt"(Gteat than).
- Comments: <!- - This is a comment - ->

Example:

```
<?xml version =”1.0” encoding=”ISO-8859-1”?>
<name>                          // root element
<first>John</first>             // opening and closing tag
<last>Smith</last>              // case sensitive
</name>                         // close tag for the root element
```

In this example <name> is a root element and <first> and <last> are child elements. Also, <first> and <last> are siblings to each other.

XML is not an extension or upgrade of HTML. They both have different purposes. HTML displays data on the web site, and the main focus is how the data looks. XML on the other hand does not do anything besides transport and store data, with the focus on what data is. It is a text file and it is parsed by the application importing the data. In order to display an XML file, one needs to use eXtended Style Sheet Language (XSL).

XML standards are recommended by the W3 consortium. According to these standards an XML document needs to be *Well Formatted* and *Valid.* A well formatted document is a document with correct syntax. A valid document is one that is well formatted and also conforms to the rules of a *Document Type Definition (DTD).*

A Document Type Definition (DTD) defines the legal elements structure and their attributes and values located inside the element's tag. The DTD can be defining inline or as reference to an external file (.dtd).

## XML in 4D

For XML support 4D uses Xerces.dll and Xalan_C_1_6_0.dll, developed by the Apache Foundation.

*Xerces* is a library for parsing and manipulating XML. The library implements a number of standards including DOM, SAX and SAX2.
More information: http://xerces.apache.org

*Xalan* is an open source library that implements the XSLT and XPath languages.
More information: http://xalan.apache.org

There is a 4D command which allows you to transform an XML document using the existing style sheets:

**APPLY XSLT TRANSFORMATION** (xmlSource; xslSheet; result{; compileSheet})

This command applies an XML transformation to a document or BLOB containing an XML structure and generates a document or BLOB. The arguments are:

| | |
|---|---|
| xmlSource | Name or access path to a document or BLOB containing the XML source. |
| xslSheet | Name or access path to a document or BLOB containing the XSL style sheets. |
| result | Contains the document or BLOB that receives the contents of the XSLT transformation. |
| compileSheet | Is an optional parameter. When this Boolean type argument is set to “True” the XML source file is parsed on the first call of the command, compiled and stored in the memory. |

4D allows you to modify the XLS style sheet parameters by using:

**SET XSLT PARAMETER** (paramName; paramValue)

paramName - Name of the parameter to look for in XSL sheet.
paramValue - Value of the parameter to use in the transformed document.

Both commands should be used together and run in the same process.

### Difference between DOM and SAX

These are two different parsing modes for XML in 4D.

DOM or (Document Object Model) builds the XML structure in the memory. The access to each element is very fast. On the other side this can be overwhelming for the memory with big XML structures.

The quantity of memory needed for SAX depends on the maximum depth of the XML file (Children, grandchildren and so forth) and the maximum data stored in XML attributes in a single element. In other words in SAX mode only one element, its sub elements and its data are stored in the memory. When this element structure is parsed, the next element structure is loaded. The elements structure is smaller than the whole tree structure. In other words the SAX model is recommended for large size XML documents, regardless of the amount of memory available.

## XPath notation

XPath notation comes from the XPath language. The primary purpose is to access parts of an XML document. Also, it provides basic facilities for manipulation of strings, numbers and Booleans.

Example of XPath notation to access element 3:

```
<root>
   <element1>
          <element2>
                 <element3>
                 </element3>
          </element2>
   </element1>
</root>
```

The XPath notation to access element 3 is:

```
/root/element1/element2/element3
```

4D uses XPath reference style in the **DOM Create XML element**, **DOM Find XML element** and **DOM SET XML ELEMENT VALUE** commands. In this way an element easily can be created, accessed and manipulated programmatically from 4D**.**

SAX commands in 4D create, manipulate and save XML on disk. They use “document type” references and commands from the “Document” theme commands like: **OPEN DOCUMENT**, **APPEND DOCUMENT**, **SEND PACKET** and so forth.

## Examples

In these examples we will create the same XML file by using the DOM and SAX theme commands.

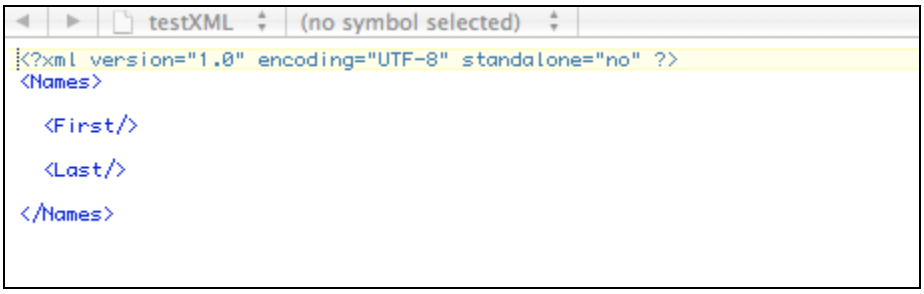
### Using DOM commands theme

This code will create an XML structure file named “testXML” inside the active 4D folder.

```
C_String(16;Ref)

Ref:=DOM Create XML Ref(“Names”)              //Creates root reference
DOM Create XML Element(Ref;”/Names/First”)    //Creates a child named First
DOM Create XML Element(Ref;”/Names/Last”)     //Creates a sibling of First
DOM EXPORT TO FILE(Ref;”testXML”)            //Saves the XML structure into
field
DOM CLOSE XML(Ref)                            //Frees the memory
```

Here is the file:



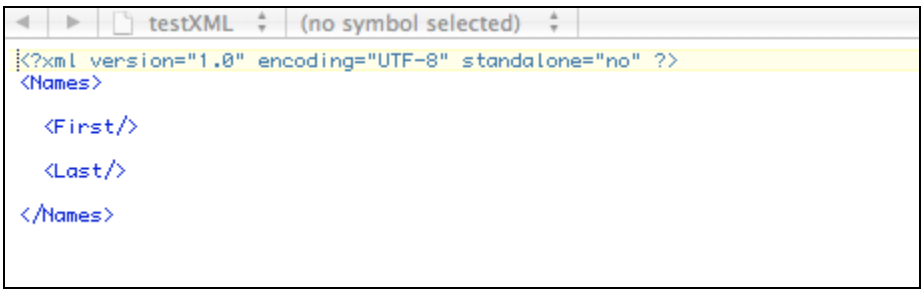

### Using SAX commands theme

This code will create an XML structure file named “testXML” inside the active 4D folder.

```
C_TIME(docRef)

docRef:=Create document(“TestSAX”)      //Create a document
SAX OPEN XML ELEMENT(docRef;”Names”)    //Create the root element
SAX OPEN XML ELEMENT(docRef;”First”)    //Create a child node
SAX CLOSE XML ELEMENT(docRef)           //Close the child node
SAX OPEN XML ELEMENT(docRef;”Last”)     //Create a sibling of “First” note
SAX CLOSE XML ELEMENT(docRef)           //Close the child node “Last”
CLOSE DOCUMENT(docRef)                  //Close the document
```

Here is the file:




Notice the file is the same.

***Note:*** *For XML error handling: The error codes are between 9911 – 9935 and they are listed in the Language reference.*

*http://www.4d.com/docs/CMU/CMU02024.HTM*

## 4D DTD folder

The 4D DTD folder is located in the “Resources” folder of 4D software. For Mac OS X the “Resources” folder is in the 4D package, and for Windows it is inside the 4D folder. The DTD folder consists of all syntax rules which 4D needs in order to validate an XML Structure.

The XML structure does not need to be validated in order to copy a table, or open a new 4D database. In other words you can leave the following line out of the structure and 4D will validate this document internally.

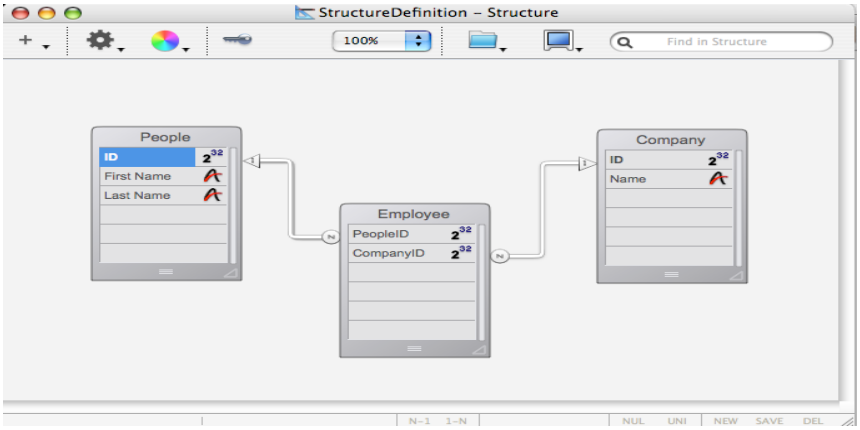
```
<!DOCTYPE base SYSTEM “http://www.4d.com/dtd/2007/base.dtd>
```

An error will occur if the XML structure is not well formed and does not conform to the syntax rules defined by the files “base_core.dtd” and “common.dtd” located inside the DTD folder. 4D will open the newly created database without any tables in it.

## 4D v11 SQL Structure Definition

The 4D Structure Definition includes all objects which can be created with the 4D Structure Editor window. These objects are tables, fields, indexes, relations and the Structure Editor settings.

Example:




Our example has:

- 3 tables (People, Company and Employee).
- 2 relations (Link_1 and Link_2).
- Field types (Long Integer and Alpha types).
- Indexes (**[People] ID** and **[Company] ID** have *Automatic* index set).

All these objects and their properties have been created in the Structure Editor window.

## How to use a Structure Definition

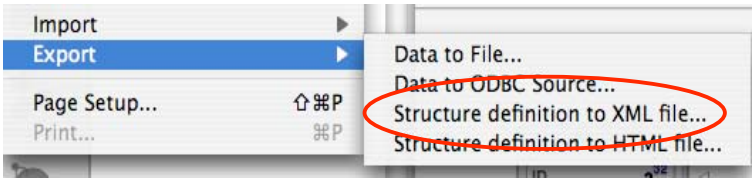
Earlier, I mentioned that a 4D Structure Definition can be used for creating a new database or move structure objects between 4D databases.

***Note:*** *In order to move structure objects between 4D and other database applications which support XML, the XML structure should be well formatted by conforming the syntax rules and using XSLT stylesheets defined for the application which will use the structure definition. Later, this application will try to validate it following its langusge rules defined by DTD*

This section will give a detailed explanation how to perform these tasks and how 4D manages the Structure Editor settings inside the elements tags

## Exporting a Structure Definition

4D v11 SQL allows you to export a structure with Export features to an XML file. This can be done in Design mode from the “File” menu.




The “Save” dialog box will prompt you to choose the name and the destination for the XML file.

This is our database structure exported as an XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE base SYSTEM "http://www.4d.com/dtd/2007/base.dtd" >
<base name="StructureDefinition" uuid="9B4C8D6FD59A4918B7F6DEC2799F85DF"
collation_locale="en">
    <table name="People" uuid="31B158A370724E0187AA0D46981CC90A">
            <field name="ID" uuid="5A646F2A697B4703A0E7F956AE6FFC2A" type="4"
unique="true" never_null="true">
                    <index_ref uuid="92F6B07050B946B09F635770000AD527"/>
                    <field_extra mandatory="true"/>
            </field>
            <field name="First Name" uuid="C8E8B9F5E96F4BA5AFFFFD3C1A1E71F8" type="10"
limiting_length="80" never_null="true"/>
            <field name="Last Name" uuid="AC19ABE1445D4174994206C0671B760E" type="10"
limiting_length="80" never_null="true"/>
            <table_extra>
                    <editor_table_info>
                            <color red="255" green="255" blue="255" alpha="0"/>
                            <coordinates left="39" top="40" width="120" height="168"/>
                    </editor_table_info>
            </table_extra>
    </table>
    <table name="Company" uuid="357C9E3C506D430FA47659EA7FBE402F">
            <field name="ID" uuid="854616B40C0C4D17A9716CAC0201E153" type="4"
unique="true" never_null="true">
                    <index_ref uuid="095B2E5A58814501BF8CF72FC2779236"/>
                    <field_extra mandatory="true"/>
            </field>
            <field name="Name" uuid="86DC6F07FF744F85AF338003E907B6C6" type="10"
limiting_length="100" never_null="true"/>
            <table_extra>
                    <editor_table_info>
                            <color red="255" green="255" blue="255" alpha="0"/>
                            <coordinates left="444" top="43" width="120" height="168"/>
                    </editor_table_info>
            </table_extra>
    </table>
    <table name="Employee" uuid="C0EBFEA32D40456AB9A28C37E0422FFE">
            <field name="PeopleID" uuid="02634015D43A4257821327C866DB07D6" type="4"
never_null="true"/>
            <field name="CompanyID" uuid="934220AB9B644F28B93A75A219D3F749" type="4"
never_null="true"/>
            <table_extra>
                    <editor_table_info>
                            <color red="255" green="255" blue="255" alpha="0"/>
                            <coordinates left="238" top="124" width="120" height="168"/>
                    </editor_table_info>
            </table_extra>
```

```
    </table>

    <relation uuid="ABFA4A957C8B4AA6B52CD25069A86CAE" name_Nto1="Link_1"
name_1toN="Link_1_return" auto_load_Nto1="true" auto_load_1toN="true" foreign_key="false"
state="1">
            <related_field kind="source">
                    <field_ref uuid="934220AB9B644F28B93A75A219D3F749" name="CompanyID">
                            <table_ref uuid="C0EBFEA32D40456AB9A28C37E0422FFE"
name="Emploee"/>
                    </field_ref>
            </related_field>
            <related_field kind="destination">
                    <field_ref uuid="854616B40C0C4D17A9716CAC0201E153" name="ID">
                            <table_ref uuid="357C9E3C506D430FA47659EA7FBE402F"
name="Company"/>
                    </field_ref>
            </related_field>
            <relation_extra entry_wildchar="false" entry_create="false" choice_field="0"
entry_autofill="false">
                    <editor_relation_info via_point_x="0" via_point_y="0"
prefers_left="false" smartlink="true">
                            <color red="255" green="255" blue="255" alpha="0"/>
                    </editor_relation_info>
            </relation_extra>
    </relation>
    <relation uuid="341AA380539344E2BF5EEBC2FEB39742" name_Nto1="Link_2"
name_1toN="Link_2_return" auto_load_Nto1="true" auto_load_1toN="true" foreign_key="false"
state="1">
            <related_field kind="source">
                    <field_ref uuid="02634015D43A4257821327C866DB07D6" name="PeopleID">
                            <table_ref uuid="C0EBFEA32D40456AB9A28C37E0422FFE"
name="Emploee"/>
                    </field_ref>
            </related_field>
            <related_field kind="destination">
                    <field_ref uuid="5A646F2A697B4703A0E7F956AE6FFC2A" name="ID">
                            <table_ref uuid="31B158A370724E0187AA0D46981CC90A"
name="People"/>
                    </field_ref>
            </related_field>
            <relation_extra entry_wildchar="false" entry_create="false" choice_field="0"
entry_autofill="false">
                    <editor_relation_info via_point_x="0" via_point_y="0"
prefers_left="true" smartlink="true">
                            <color red="255" green="255" blue="255" alpha="0"/>
                    </editor_relation_info>
            </relation_extra>
    </relation>
    <index kind="regular" unique_keys="true" uuid="095B2E5A58814501BF8CF72FC2779236"
type="7">
            <field_ref uuid="854616B40C0C4D17A9716CAC0201E153" name="ID">
                    <table_ref uuid="357C9E3C506D430FA47659EA7FBE402F" name="Company"/>
            </field_ref>
    </index>
    <index kind="regular" unique_keys="true" uuid="92F6B07050B946B09F635770000AD527"
type="7">
            <field_ref uuid="5A646F2A697B4703A0E7F956AE6FFC2A" name="ID">
                    <table_ref uuid="31B158A370724E0187AA0D46981CC90A" name="People"/>
            </field_ref>
    </index>
</base>
```

The root element has seven children (three tables, two relations and two indexes).

## Elements in the XML Structure

This section describes each element that can appear in the Structure Definition file.

### base

This is the root element for this structure. This does not create any conflict with the XML syntax rules, because XML works with customized tags.

### table

This is a child element of the root element. All tables are siblings inside the root element. The attributes are:

- Name – this is the name of the table which is set in the structure editor.
- uuid – the unique ID which automatically is set from 4D. Developers can not control or alter this number. Inside the XML document it is used as an element reference number.

### field

This is a child element of the table element. The number of <field> elements is equal to the number of table fields. All <fields> elements are siblings. In other words, you can not nest field elements. The attributes are:

- name – the name is set, when you create the fields inside the structure editor.
- uuid – field reference number.
- type – gives you information about the field type;
    - 1 – Boolean
    - 2 – Long Integer
    - 3 – Integer
    - 5 - Integer 64 bits
    - 6 – Real
    - 7 – Float
    - 8 – Date
    - 10 - Alpha
    - 18 – BLOB
    - 20 – Picture
- Limiting_length – This attribute is available for “alpha” fields and limits the number of characters.
- unique – This attribute is displayed inside the “field” tag only if the “Unique” option is checked in the “Inspector” window. The value of this parameter is set to “true”.
- never_null – This attribute is available when the “Map NULL values to blank values” is checked in the “Inspector” window.

**index_ref**

This is available on indexed fields. The only attribute for this field is the uuid number. This element is a child element of the “field” element.

**field_extra**

This is a sibling of the “index_ref” element. The attributes are the settings in the “Data Entry Controls” section in the “Inspector” window.

**tip**

This element is  a child element of the <field_extra>. It contains the text entered into the “Help Tip” box.

**table_extra**

This is a child of the table element. This element contains <comment> and <editor_table_info>. The <comment> tag contains the style definition and the comment text from the “Comments” window. This element will show as a child of the <field_extra> element (sibling with <tag>) as well, if you have entered comments for some of the table fields.

**editor_table_info**

This contains information about the color and coordinates of the table inside the “Structure Editor” window.

**relation**

This is a child of the root element. The attributes are:

- uuid – reference number.
- name_Nto1 and name_1toN – This is the name of the relation.
- auto_load_Nto1 and auto_load_1toN – The value “true” or “false” depends on if the “Auto Relate One” and “Auto One to Many” are checked for this relation.
- stage – is equal to “1” when the relation is set, or “0” if you want to disable this relation.

**related_field**

This tag has only one attribute, “kind”. It has two possible values: “source” and “destination”.

**field_ref and table_ref**

This Both tags contain as attributes the references for the field and the table. These are the field uuid and table uuid numbers.

**relation_extra**

This contains information for the “Auto wild card support”.

**editor_relation_info**

This  tag carries the information about the display of the relation.

**index**

This element is a child of the root element. The attributes are:

- unique_keys – has the value true or false if the unique box has been checked in the “inspector” window
- index reference number (uuid)
- type – value is the type of the index set for the field with the reference number located in the child elements <field_ref> and <table_ref>. Also if the index has a name it will show in the “name” attribute.

**base_extra**

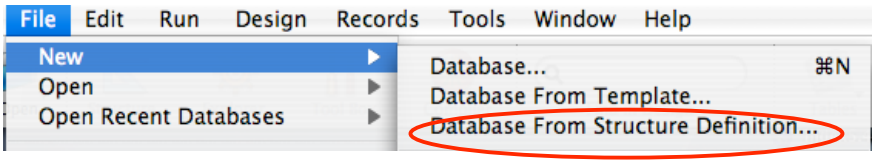
This tag contains information about the package name, structure file name, data file name.

**journal_file**

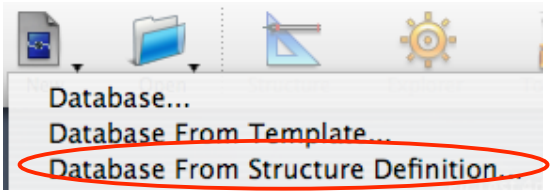
This is a child of the <base_extra> tag and it is displayed only if the “log file” check box has been checked in the “Preferences”, “Backup” theme.

## Importing a Structure Definition

New database can be created from a structure definition.
Open 4D and from the “File” menu select “New” and “Database from Structure Definition”.




The alternative way is from the “New” icon.



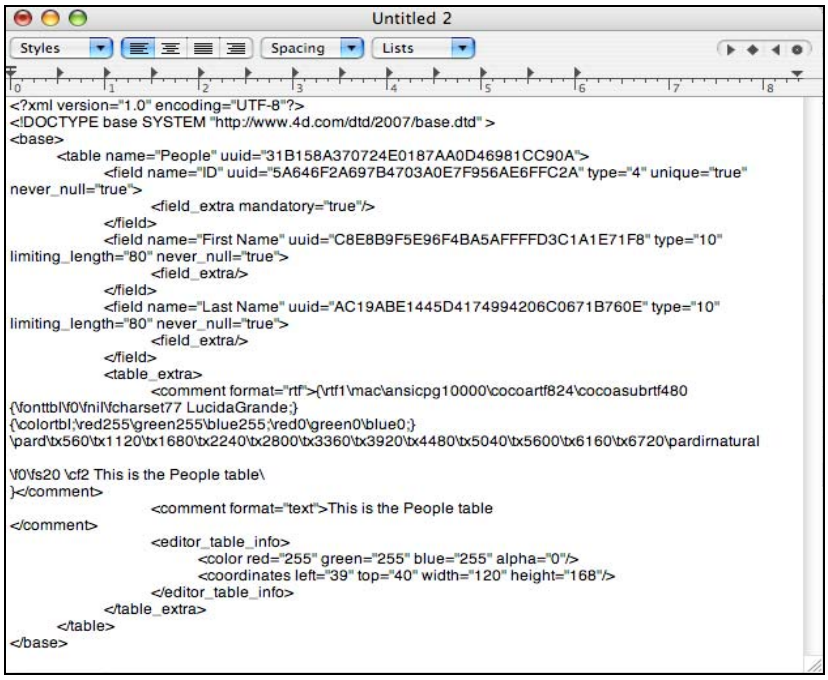

“Open” dialog window appears where you specify the location of the XML file.

***Note:*** *The location of the Structure definition file does need to be on the local disk. It can be located in a different location like different drive or network partition.*

## Moving Structure Objects

You can use Copy and Paste commands from Edit menu, or with Contextual click, to copy objects from the Structure Editor. When you copy, 4D places the XML structure of the copied object in the clipboard.

Here is an example of pasting a table into a text editor.




In example above, I copied the “People” table and pasted it into TextEdit.

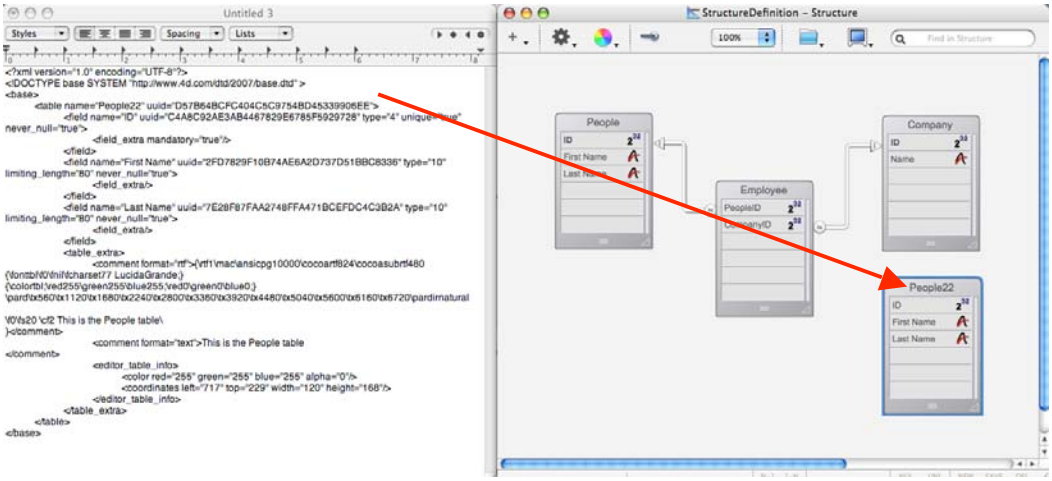
This allows you to modify table, field, index and link properties directly inside the text editor. Also, you can create table templates and use them when creating a new database or an already open database.

The UUID (Universal Unique Identifier), as the name implies, is a unique number assigned automatically from 4D to all 4D objects. Also this number is used as an elements reference number in the XML structure.

If you paste the XML structure inside the Structure Editor, 4D gives a different “UUID” number to the newly created structure objects.

The next example, I just renamed the table to “People22” in the text editor, copied to XML, and pasted the table back to the Structure Editor. The newly created table “People22” has a different table number and a different UUID number. The UUID numbers of the fields are different as well.

Here is what it looks like:




Compare with the previous example.

## Conclusion

This Technical Note described how to create a Structure Definition file and as a second step, to create a new database from this file. The Structure definition is in the XML format, and it can use the full potential of the XML language.

Before describing the Structure Definition supported in 4D v11 SQL, we took a quick tour inside the XML language. This is not a full presentation of the language, but just the minimum, which is needed to follow the syntax and the construction of the Structure Definition file.

This new feature of 4D will give 4D developers handy tools to create a brand new database structure from an old one or just move tables and fields, along with their attributes and properties between two databases.