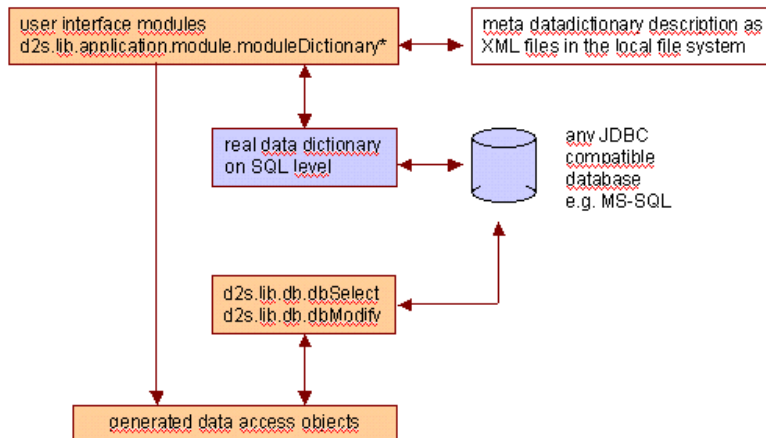


Data Dictionary / Metaschema

The datadictionary becomes necessary to target the following topics:

- Type safety
- Database independence
- Client distinction



The information of the E4S datadictionary is overlaid to the real data dictionary.

Definitions are stored in XML structures outside the database.

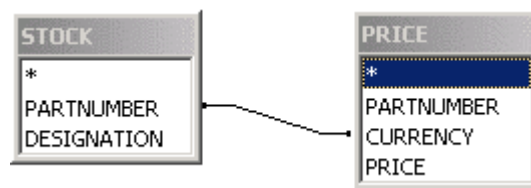
Modifications will automatically be synchronized to the real datadictionary of the DBMS.

Datatype definitions

One common problem in software development we want to address with this functionality is that in case you use a JDBC based access to the database, most datafields are somewhat character or numeric fields, but beside the column name there is no deeper meaning and interpretation transparent to the application. Generated database access classes address this, but also leave some areas for potential errors.

Let's assume, we have two tables:

A table *stock* which includes partnumber and designation and a table *price* which includes partnumber, price and currency symbol.



The column partnumber is used in both tables.

We could use (build or generate somehow) simple classes to access those data elements:

```

public class STOCK
{
    public String m_PARTNUMBER;
    public String m_DESIGNATION;

    public String getPARTNUMBER()
    {
        return m_PARTNUMBER;
    }
}
  
```

```

public class PRICE
{
    public String m_PARTNUMBER;
    public String m_CURRENCY;
    public float m_PRICE;

    public String getPARTNUMBER()
    {
        return m_PARTNUMBER;
    }
}
  
```

```

public void setPARTNUMBER( String p )
{
    m_PARTNUMBER = p;
}

public String getDESIGNATION()
{
    return m_DESIGNATION;
}

public void setDESIGNATION( String d )
{
    m_DESIGNATION = d;
}
}

}

public void setPARTNUMBER( String p )
{
    m_PARTNUMBER = p;
}

// t.b.c. ...
}

```

On the first view, this gives you security. But there is a bad chance to mix up things:

```

String designation = "Laser Printer";
String partnumber = "10001";

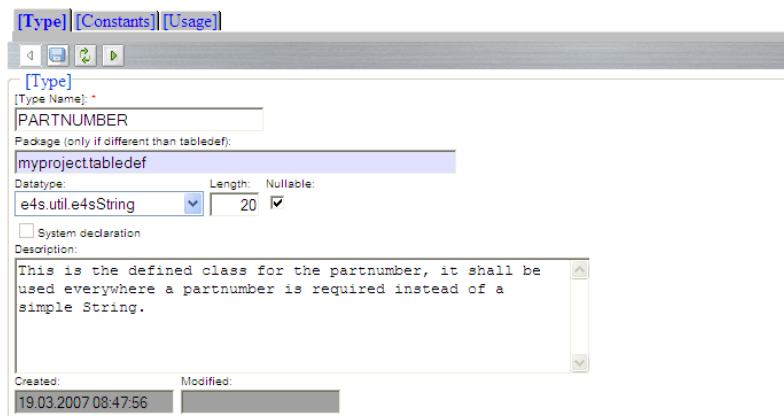
STOCK stock = new STOCK();
stock.setDESIGNATION(partnumber); // wrong !!
stock.setPARTNUMBER(designation); // wrong !!

```

In real world, things become more complicated rather than this example.

The concept of E4S defines *meaningful datatypes*, e.g. you can create a class called "PARTNUMBER" which is of type String and has a defined length.

E4S now creates a Java source (PARTNUMBER.java) file which can be included into your project for the datatype PARTNUMBER



```

public class STOCK
{
    public PARTNUMBER m_PARTNUMBER;
    public String m_DESIGNATION;

    public PARTNUMBER getPARTNUMBER()
    {
        return m_PARTNUMBER;
    }

    public void setPARTNUMBER(PARTNUMBER p )
    {
        m_PARTNUMBER = p;
    }

    public String getDESIGNATION()
    {
        return m_DESIGNATION;
    }

    public void setDESIGNATION( String d )
    {
        m_DESIGNATION = d;
    }
}

```

Now this new datatype can be used in your data dictionary table definitions, and wherever a partnumber is required, the interface takes care about the datatype and avoids assignment of simple datatypes such as String or other defined datatypes.

At this point, you can ensure that your database schema is consistent with your application's code as long as you do all changes using the E4S datadictionary and re-compile your sourcecode afterwards.

}

Benefits from this scenario:

- Database schema is consistent with application code.
- Database integrity matches (same datatype and fieldlength ensured when column is used in different tables).
- Referential integrity checks become easier and possible
- Referential delete become easier and possible
- Changing of index-key elements is possible over a database in some automated ways
- Violation of field length or nullable detected on value assignment early and not later on database transactions.

There are some RDBMS systems which can deal with named datatypes, but almost SQL based standards do not support this. It also raises up compatibility issues and more important we think, that this type declaration is the bridge between the RDBMS and the application and therefore definition makes sense in the E4S environment.

Table definitions

The integrated data dictionary allows you to create database tables that are originally stored in the DBMS but are also transparent to your Java application by automatically generated Java class files.

There are two basic, general classes for database access which are likely the rudimentary JDBC standards:

e4s.db.dbSelect	Read access to database tables (SQL Select, Metaschema)
e4s.db.dbModify	Write access to database tables (SQL-Insert, SQL-Update)

Based on this classes, you can define tables which result into two generated classes inherited one of dbSelect and one of dbModify. Additionally, some data encapsulating class will be generated.

Table Columns Index References Info Data

Table

Table Name *
STOCK

Package (only if different than tabledef)
[mypackage.tabledef]

Description
Table definition for spare parttr stock

Generate Java Class
 Table uses mandant distinction
 Table can have free field definitions

Created
10.03.2005 12:15:26

Modified

Define the database table. On definition, you can specify if a Java class shall be generated and if client distinction shall be used.

Client distinction means, that each data row becomes appended a client identification which will also be part of any index. Each access, reading or writing, takes automatic care to handle the client associated with the user logged in.

Table Columns Index References Info Data

Columns

[Table Name]
STOCK

[Column Name]	Datatype	Size	A.Incr.	Description	Delete
PARTNUMBER	PARTNUMBER	10	<input type="checkbox"/>	Partnumber	<input type="checkbox"/>
DESIGNATION	STRING	60	<input type="checkbox"/>	Designation	<input type="checkbox"/>
ONSTOCK	INTEGER	10	<input type="checkbox"/>	Quantity on Stock	<input type="checkbox"/>
LISTED	DATE	10	<input type="checkbox"/>	Date first listed	<input type="checkbox"/>
WEIGHT	FLOAT	10	<input type="checkbox"/>	Weight for one piece	<input type="checkbox"/>
ORIGINAL_CNTRY	CountryCode	10	<input type="checkbox"/>	Original Country of this product	<input type="checkbox"/>
	STRING	10	<input type="checkbox"/>		<input type="checkbox"/>
	STRING	10	<input type="checkbox"/>		<input type="checkbox"/>
	STRING	10	<input type="checkbox"/>		<input type="checkbox"/>
	STRING	10	<input type="checkbox"/>		<input type="checkbox"/>

Define the columns (fields) of the table. At this point, you can use the pre-defined meaningful datatypes (→ 0). You also can define fields based on some simple datatypes:

- STRING (< 256 chars)
- TEXT
- INTEGER
- TEXT
- DATE
- BOOLEAN

Finally, you can add or modify your index definitions.

Table Columns Index References Info Data

Index

[Table Name]
STOCK

#1	#2	#3	#4	#5	[Column Name]	Datatype	Description
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PARTNUMBER	PARTNUMBER	Partnumber
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	DESIGNATION	STRING	Designation
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ONSTOCK	INTEGER	Quantity on Stock
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	LISTED	DATE	Date first listed
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	WEIGHT	FLOAT	Weight for one piece
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ORIGINAL_CNTRY	CountryCode	Original Country of this product

[Alias #1] *
MAIN [Alias #2] *
[Alias #3] *
[Alias #4] *
[Alias #5] *

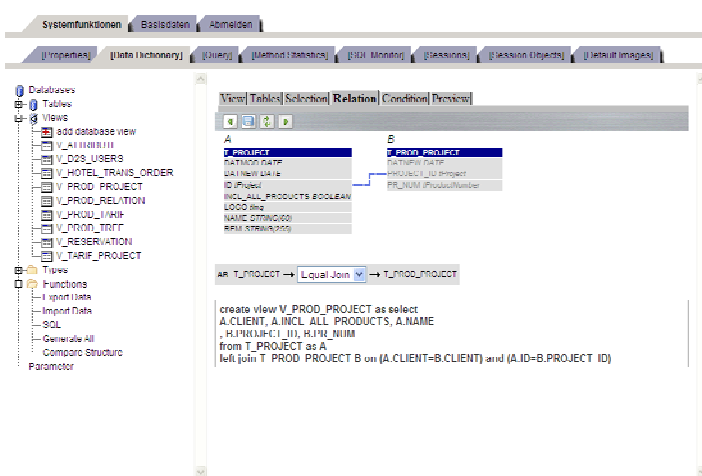
Datatypes

Due to compatibility reasons, we keep this very simple. This simlicism also can help to avoid failures and speed up development.

Datatypes			
Basic Datatypes	STRING	String	(<= 255 chars)
	TEXT	String	(larger text objects)

	INTEGER long DATE java.util.Date FLOAT float BOOLEAN boolean
Built-In Special Datatypes	Language – a two character ISO language code, will be treated as language selection during input CountryCode – a two character ISO country code, will be treated as country selection during input
Pre-Defined Datatypes	Either STRING, INTEGER, FLOAT or DATE but depending on the current datatype definition made in the dictionary

View Definitions



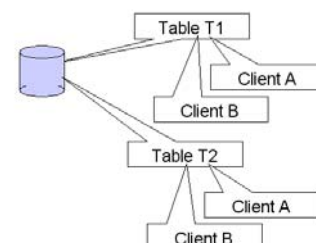
As a consequence of a data dictionary that allows the definition of tables and make them transparent to the application as Java objects, E4S also supports the definition of views.

Client Distinction

Client distinction is a useful feature when building applications that run in an application service business environment.

In commercial, organizational and technical terms, a self-contained unit in a system with separate database records is called Client.

Client distinction means, that each data row becomes appended a client identification which will also be part of any index. Each access, reading or writing, takes care to handle the client-ID associated with the user logged in.



It becomes important, to know the client at runtime, so this involves some of the login and security issues within E4S. Together with some other information (user, language, dateformat, ..) this is covered in the so called application-objects.

e4s.application.applObject_Intf	Information about the logged in user at runtime
---	---

e4s.application.applObject	This or an similar object must be created and registered to E4S for each new session, usually this will be done by the E4S login functionmodule.
--	--

Freefield Definition

After compilation of an application and deployment it is possible to define client based freefield definitions related to a core database table. Each freefield has a unique name and can be accessed similar to a database column but is not part of the database schema. This makes an application flexible, where standard solutions are built but project specific customisation is required.

Synchronisation

To synchronize different installations of the same application, there are two mechanism to synchronize tables: a manual mechanism shows differences and generates SQL statements for altering or an automated mechanism on table level on the first time access.

Code Example

Code examples of generated files can be found here:
<http://www.element4solution.com/javadoc/resources/codeexamples/>

See <http://www.element4solution.com> for details
©2007 door2solution software gmbH