

# Oakland Software Data Transformer

## Mapping Productivity

Francis Upton IV  
Oakland Software Incorporated  
5 September 2008

The Oakland Software Data Transformer (OSDT) applies standard software engineering principles like inheritance and automated regression testing to the practice of data transformation. Originally developed in conjunction with a major EDI managed service provider, its design goals are to support the rapid development and maintenance of thousands of transformation maps by personnel with only business analyst skills for the managed service environment.

This paper focuses only on the main benefits of the product for mapping productivity from a user interface point of view. It does not cover the broader capabilities of the product, runtime execution issues, performance issues, or implementation issues.

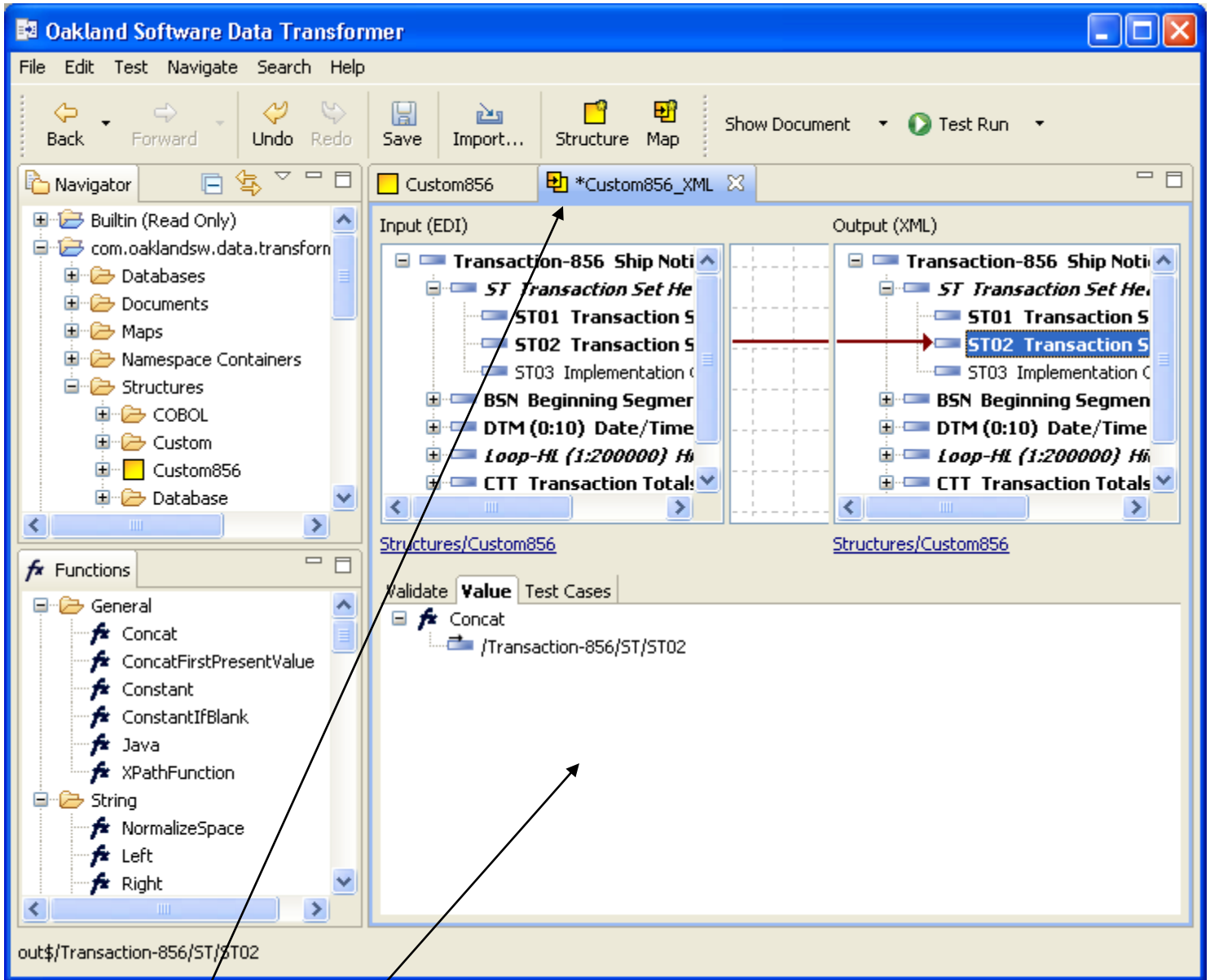
*A word about terminology:* A structure is the definition of a document type, it corresponds for example to an EDI transaction definition, XML Schema or DTD. A map is the relationship that converts an instance (document) of one structure to another. Maps relate exactly two structures.

The following technologies are used to meet these goals:

- **Automated regression testing** – Instant creation and execution of test cases that are recorded permanently with the map for future use to verify its correct functioning.
- **Integration of test/sample document management** – Extensive support is provided for including multiple applicable sample documents with structure definitions and having these available for execution during map development. Parts of maps can be executed with one click, down to a single map element. In addition, a view is available which highlights the selected data in the sample document.
- **Integration of specifications (Implementation Guides) and mapping** – Most of the information for how to do the mapping comes from the implementation guides or standards definitions. Implementation guides can be imported from tools like SpecBuilder and all of IG content is immediately available when doing mapping. In addition, notes about the mapping can be added.
- **Structure Inheritance** – Allows structures to be reused and composed for maximum maintainability and **migrations to newer versions** of standards or implementation guides.
- **Map Inheritance** – Allows portions of maps to be composed into other maps, allowing common mapping functionality to be used and allowing map customization from standard maps.
- **Automatic data type conversion** – A tremendous amount of time in typical mapping writing mapping instructions or code converting from one data type to another, particularly with dates. This conversion is handled completely automatically.
- **No programming** – The mapping instructions are simple functions, no variables, no coding. With powerful looping functions, the unique expression tree mechanism, and allowing maps to be chained together the most demanding mapping requirements can be met. Little special training is required for mapping personnel. This was validated in the product's initial development.
- **Dependency management** – Automatic management of the relationships between all mapping objects (structures, maps, etc) such that they can be freely moved and renamed.
- **Comprehensive search** – Allows searching across all mapping objects.
- **Unified definition of all structure types** (EDI, XML, database, flat) – The standard structure definition allows all of the above features for all structure types. It also hides the complexities of understanding XML Schema for example from the mapping users.
- **Built on Eclipse technology** – Eclipse is one of the leading open source Java development environments and is constantly adding new features to improve productivity. IBM, Sybase, WebMethods, BEA, and many other companies heavily support Eclipse. Since the OSDT is tightly integrated with Eclipse, OSDT improves “for free” as Eclipse improves.

## Transformer Overview

This presents an overview of the major parts of the transformer UI.



**Navigator** – Locates structures, map, and other objects that are contained in projects.

**Functions** – Used for mapping.

**Editor Area** – Contains one or more structure map or other editors. Currently the Custom856\_XML map is visible. An example of the structure editor is shown later.

**Expression Panel** – Contains the expressions used to do the mapping. In this case it shows that the output ST02 element is mapped to the input ST02 element.

## Automated Regression Testing

One of the main principles of software development, and particularly agile technologies like Scrum is that tests should be developed wherever possible and as close to (or before) the implementation as possible. The OSDT allows test cases to be developed while you are developing a map.

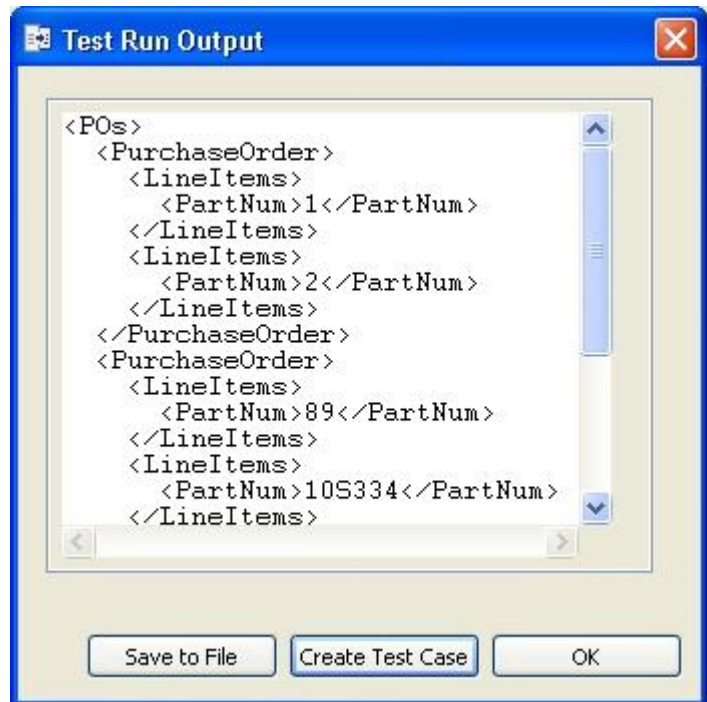
These test cases remain part of the map definition and can be executed at any time, either for the entire map or a portion of the map.

Creating and using test cases is simple. When you are creating a map, as you map each element (or small collection of elements), you can execute that element immediately to see the result of the mapping.

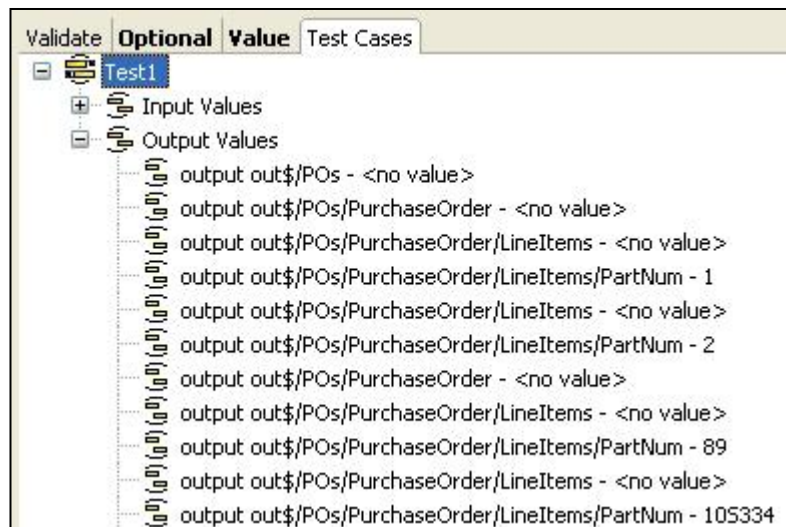
Here we see an example of **executing** the PartNum element in the map. It shows all instances of PartNum (and no other elements).

Press the Create Test Case button to automatically create a test case based on the inputs and outputs that were just executed.

Any number of test cases may be associated with any map elements.

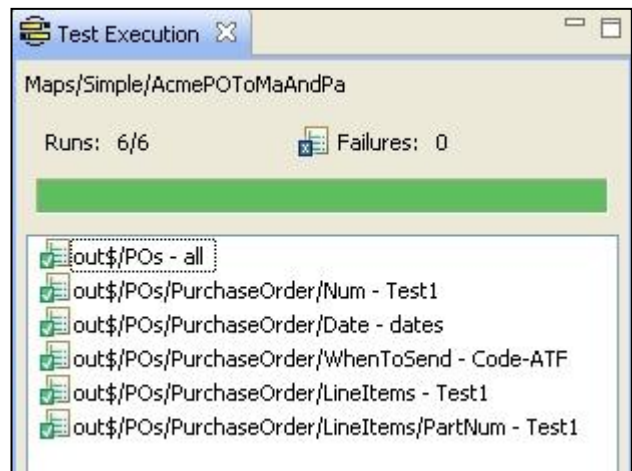


This shows the **created test case** for the PartNum element which is put in its Test Cases tab. From here you can examine or change the input or output values of the test case. You can also execute it.

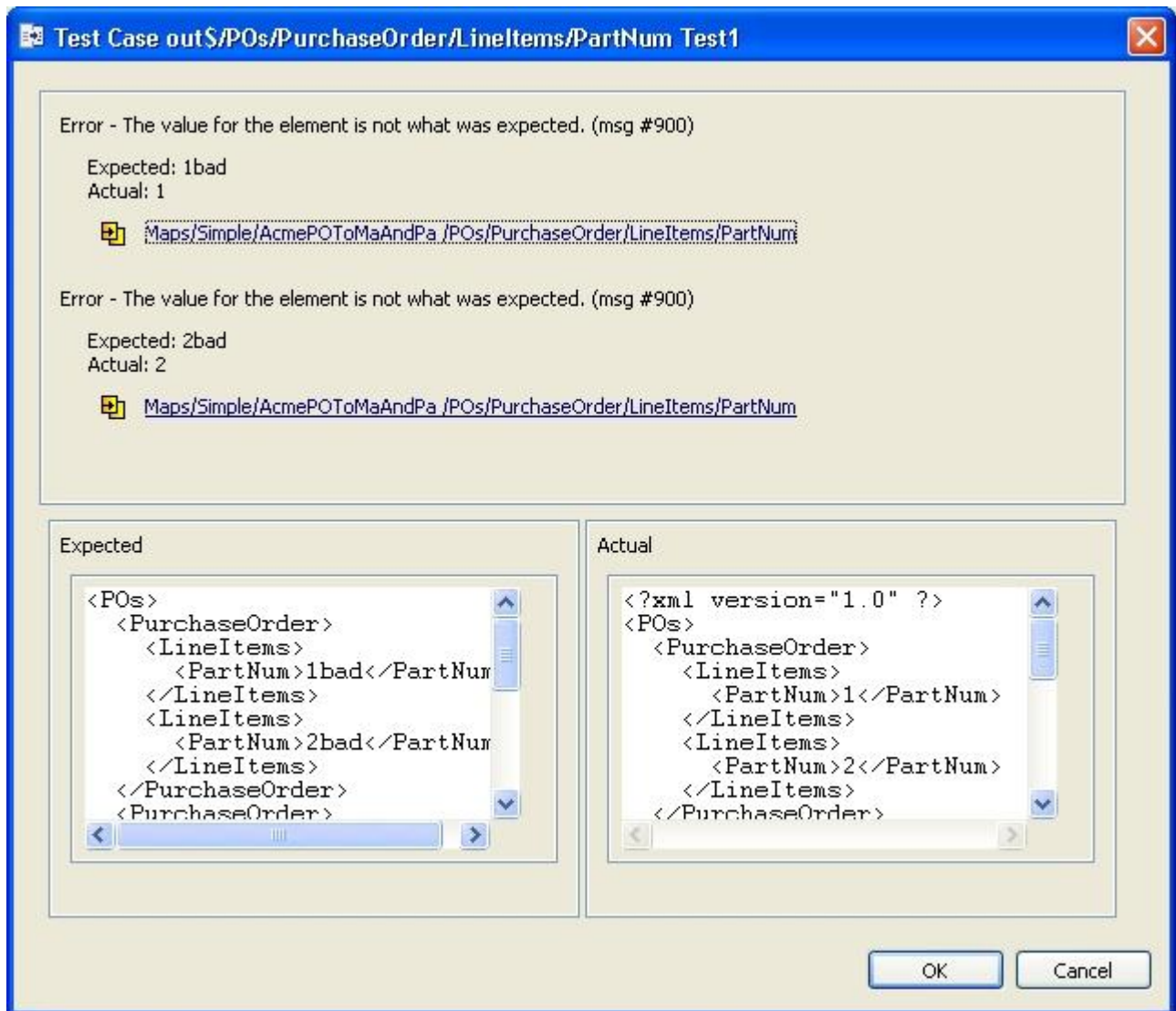


This shows the **execution** of all of the current test cases associated with the map, and that they all passed.

Note next to the map element name there is the name of the test case.



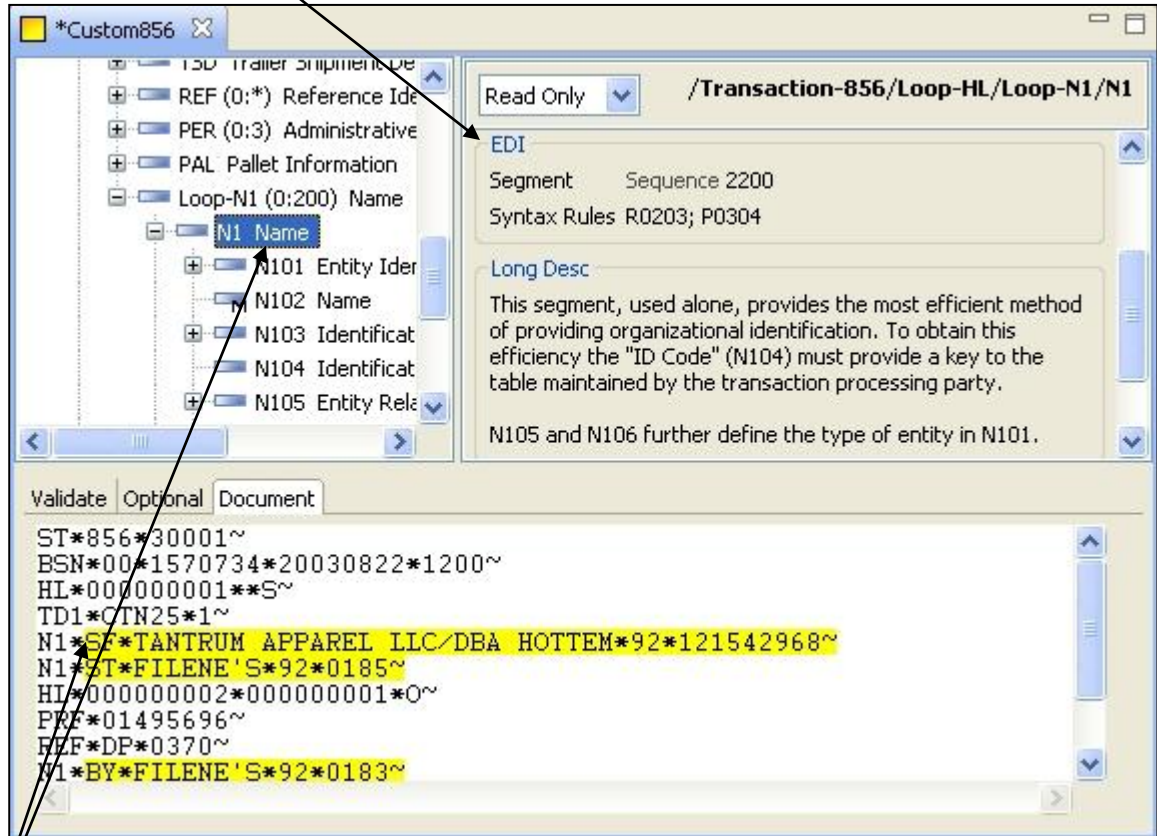
And here we see an example of a **failing test case**. This can be corrected by adjusting the test case, or by changing the mapping. Often if the mapping changed and resulted in a failing test case, it's trivial to simply replace the test case using the same means (above) where it was created.



## Sample Documents and Integration of Specifications

The figures below show the structure editor.

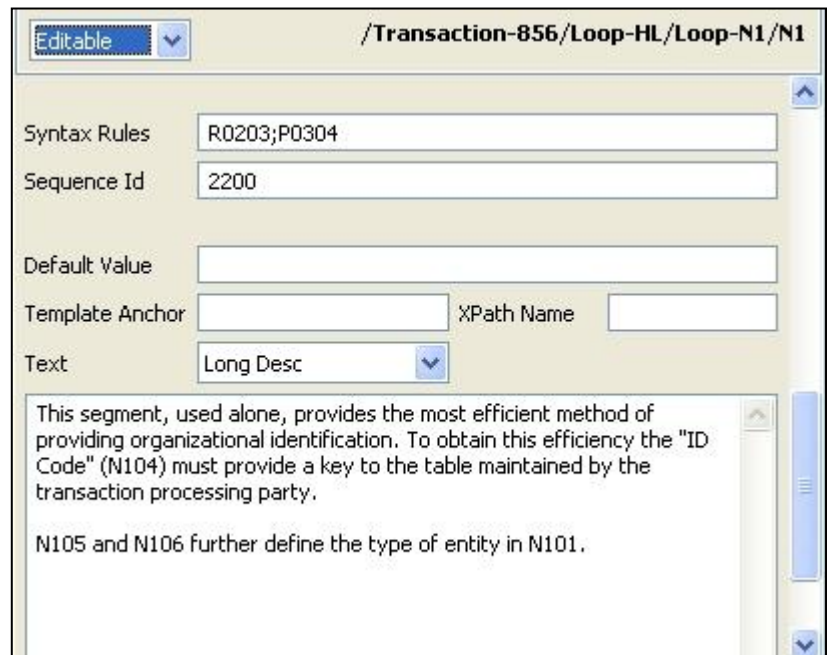
**Integration of Specifications** shows the full descriptive content of the IG/standard. This can also include user notes. For clarity this is presented in a more compact “read-only” mode. (Note that only a portion of the element’s properties is visible).



**Highlighting** is provided for the selected element in the **sample document**. For EDI segments are automatically broken on lines, regardless of the separator. The sample document stays with the structure and is thus automatically available to any maps that use the structure. Each structure may have any number of sample documents.

On the right is a view of the **editable** properties for the same element.

Any kind of user notes can also added here which can help document the mappings.

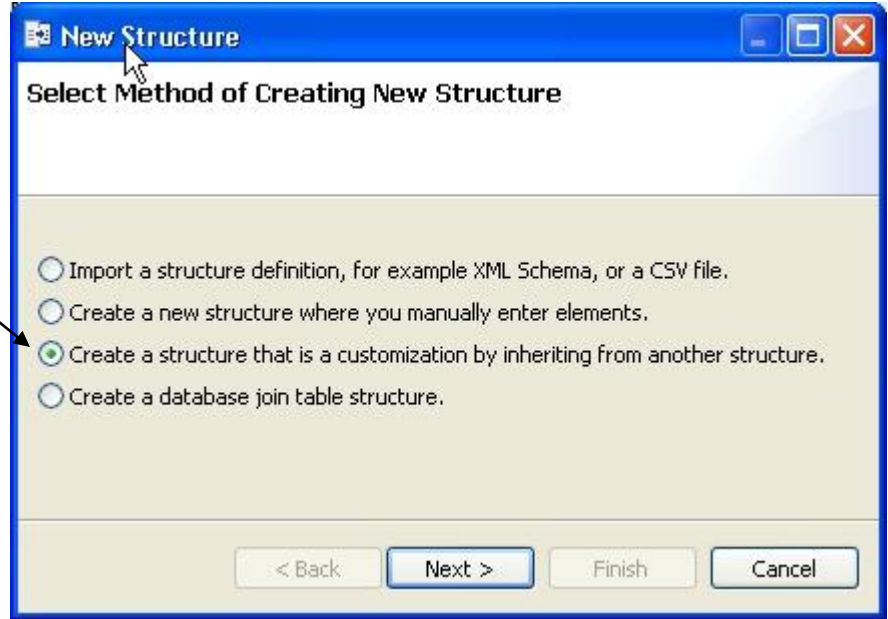


## Structure Inheritance

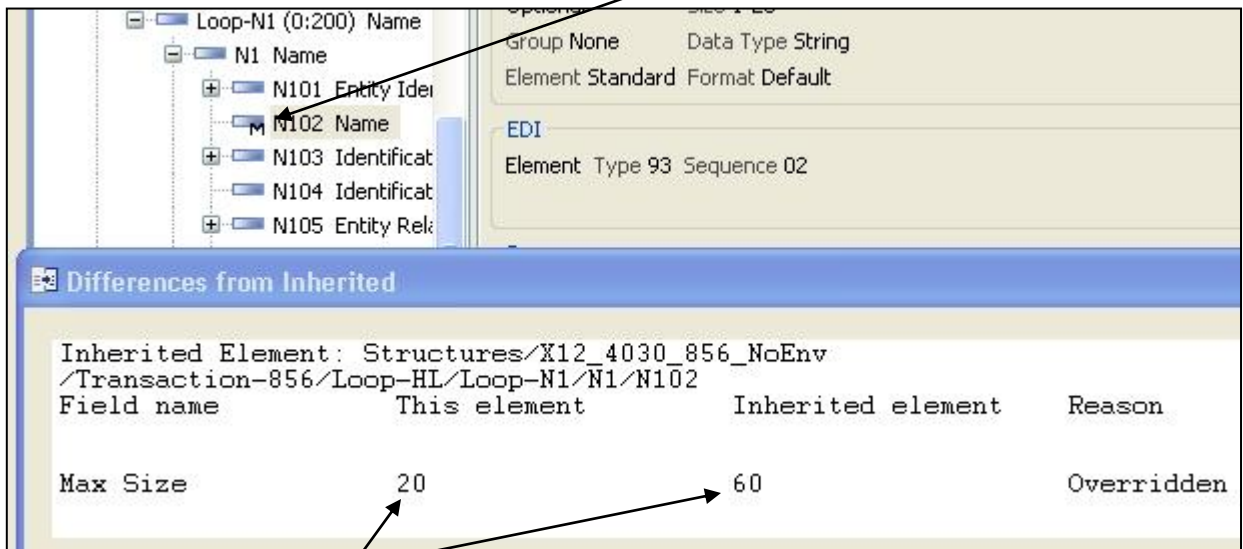
Inheritance allows you to **customize** a structure based on another.

Initially, the custom structure is completely identical to the structure from which it inherited (the *standard* structure). You customize the structure typically by deleting elements that are not used and possibly shortening element length values. You can change anything in the custom structure.

The benefit of inheritance is when the standard structure changes, those changes are automatically picked up by the custom structure.



You can also see an indication that the element has been **modified** from the standard.



At any time, you can see the **differences** between a modified element in the custom structure and the standard.

**Code values** are simply elements like any other in the structure editor. They also benefit from the normal inheritance rules.

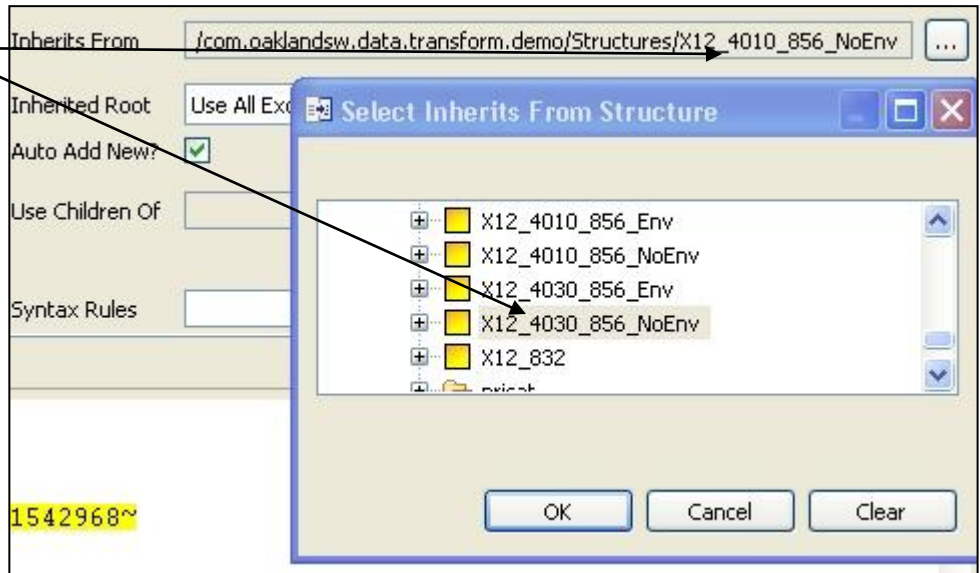
Typically when customizing you delete all unused values. A special “delete except selected” feature easily allows this.



## Version Upgrades

The inheritance mechanism easily allows **upgrading** to a new standard version, simply by changing the inherited structure.

Note that the version upgrade mechanism is not specific to EDI; it works with any type of structure.



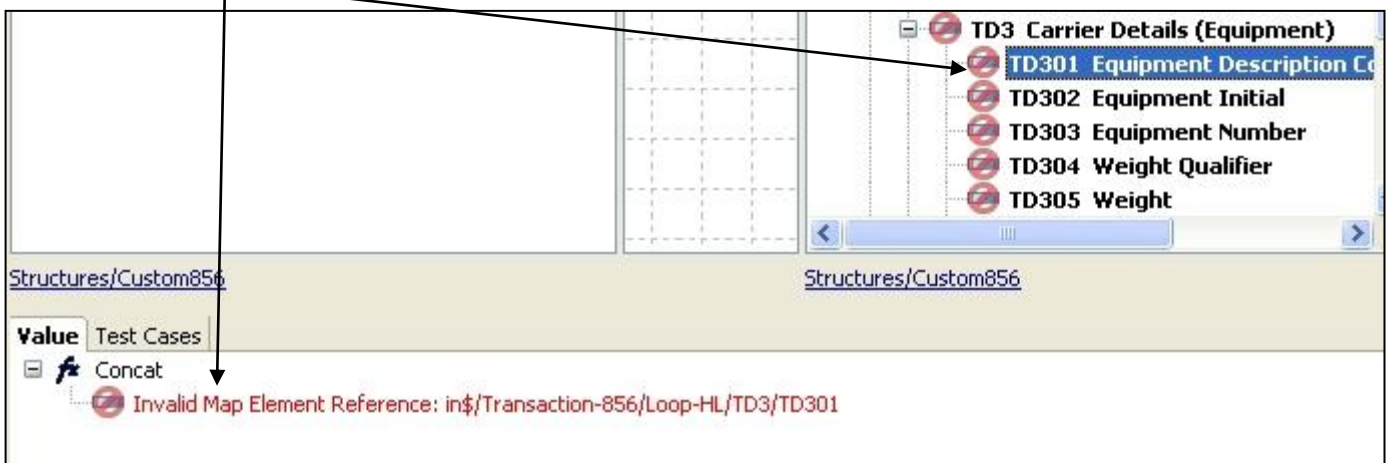
When a map using this structure is subsequently accessed, it will report these **validation errors** because of the version upgrade.

**Added** elements in the new version are shown.

**Deleted** elements are also indicated, but their mapping data remains in the map until you copy it elsewhere or delete it.



The map **indicates** the elements that are no longer valid.



## Data Type Conversion

A powerful timesaver is the fact that that OSDT allows the user to work with the semantic data type of an element, leaving the syntactic details to be handled automatically.

This excerpt from the element properties of the X12 DTM02 element shows the different possible date formats.

Elements have a **data type**, which defines their semantic range, like integer, string, date, etc.

Elements also have a **data format** that defines the specific format to use to express the type. Where possible the data format values are derived automatically from the standard, and in the case of EDIFACT EDI dynamically from the date format qualifier.

Name	DTM02				
Description	Date				
In fields below, specify -1 for unlimited					
Occurs Min/Max	0	1	Size Min/Max	8	8
Group Type	None		Start Offset	0	
Element Type	Standard		Column	0	
Data Type	Date		Data Format	Date CCYYMMDC	
Visible	<input checked="" type="checkbox"/>	Null	<input type="checkbox"/>	Decimal Places	
Initiator		Terminator			

If it's possible to automatically convert between two data types (like a string and an integer), it is done automatically.

## No Programming

To perform mapping functions, the OSDT relies on a functional programming model where expressions consist of functions that can have multiple arguments and return exactly one value. These are expressed graphically using expression trees.

Here is an example of a very simple **expression** to concatenate 3 values. It uses the Concat function that can take any number of arguments. Each argument was placed by dragging a map element from the input to the Concat function.

More complex expression trees are possible using a wide variety of mapping functions.

