SEEBURGER
BUSINESS INTEGRATION

Compliance Check Workshop

ETS Workshop

Consulting Mapping Management | SEEBURGER AG

Introduction of participants

- Name
- Role/Tasks
- Experience with EDI, EAI and B2B
  Experience with SEEBURGER software
- Expectation of this workshop

Agenda

1st day

- Part 1: GUI & First Steps
- Part 2: Overlays
- Part 3: Assertions I
- Part 4: Assertions II

2nd day

- Part 5: Assertions III/Overlay Stacking
- Part 6: Export, deploy and configuration in BIS
- Part 7: Additional features

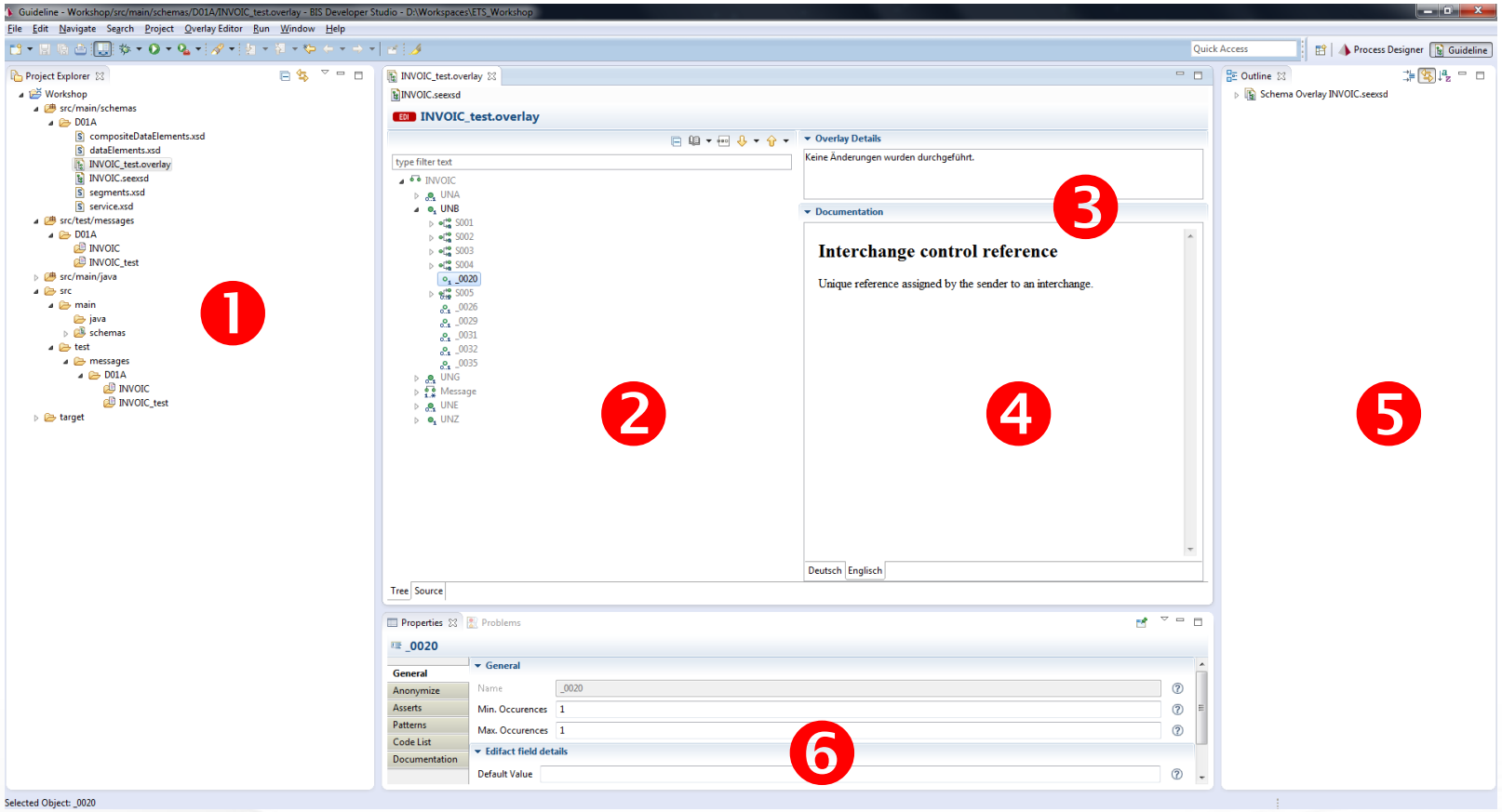
Agenda Part 1: GUI & First Steps

- Introduction to the ETS user interface
- Creation of a new ETS project
- Structure of an ETS project
- Import of message structures

Basics: ETS

- E.T.S = Enterprise Transformation Suite
- Compliance Check: Validation of EDI messages before sending or processing
- Integration into SEEBURGER DevStudio
- Usage of common standards like
  - XPath
  - Java

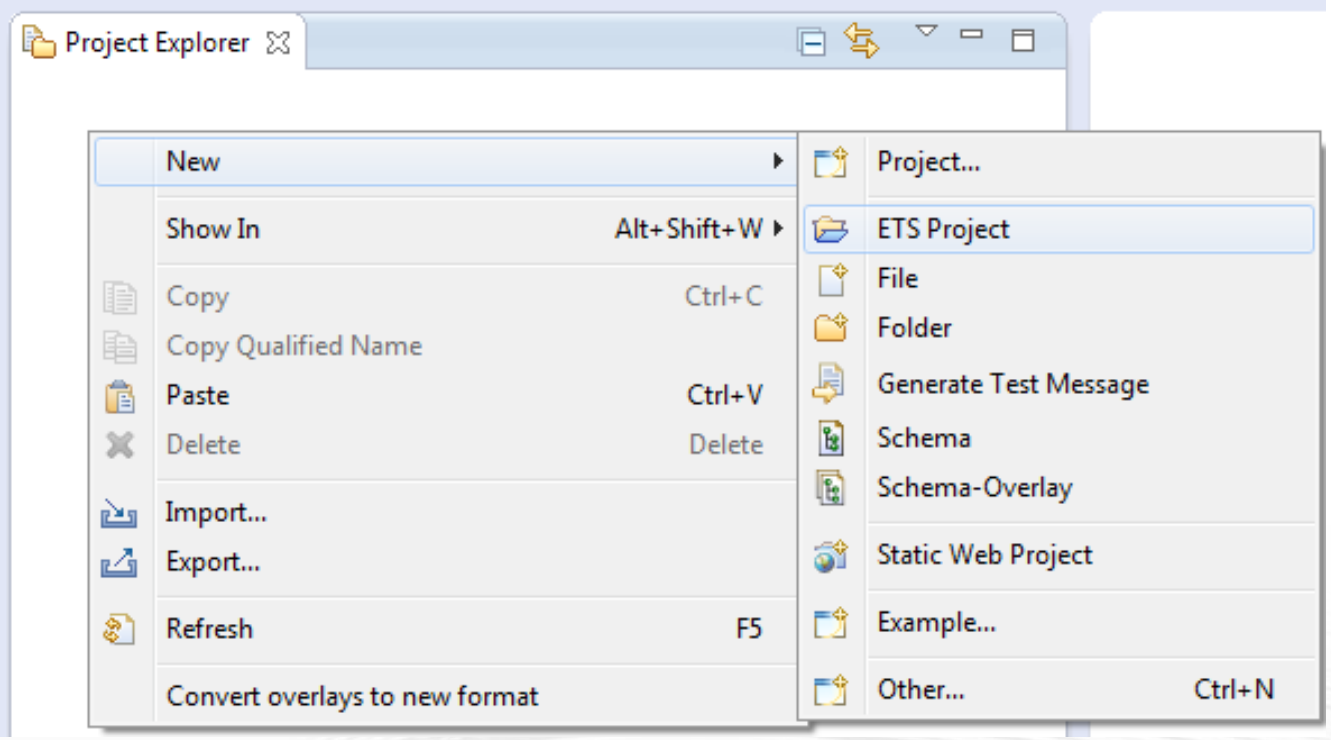
ETS user interface (Default)



1. Project-Explorer
2. Schema-Explorer
3. Overlay Details
4. Element Documentation
5. Schema/Message Outline
6. Overlay Properties

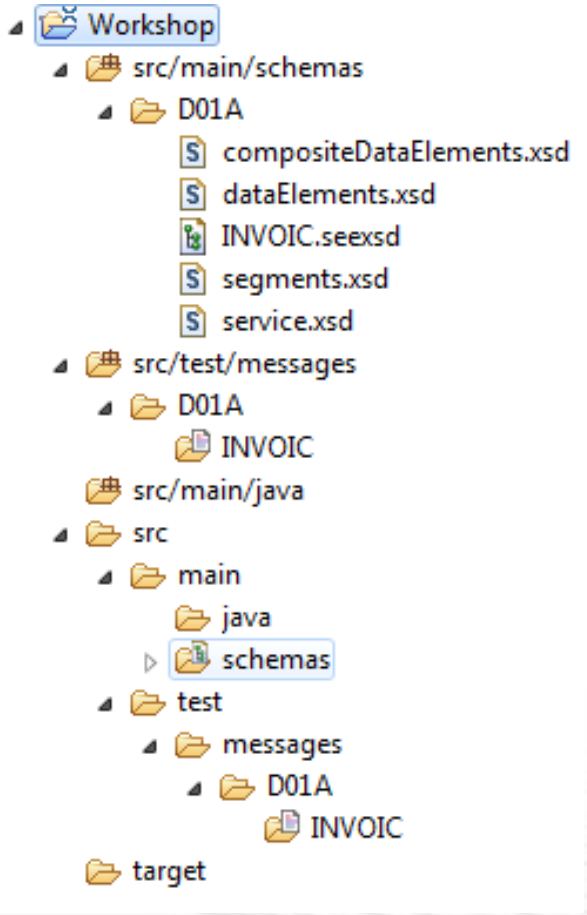
Create a new ETS project

- Open the context menu with right click in the project explorer
- *Select New/ETS* Project



Structure of an ETS projekt

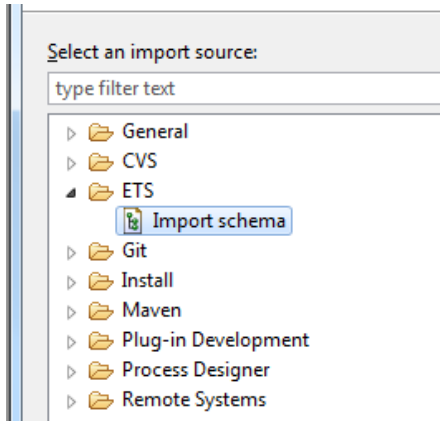
- Src/main/schemas/* contains the schemas and overlays
- Src/test/messages/* contains the test files
  - Same folder structure as src/main/schema
- Src/main/java contains own java classes
  - Create and use own java functions
- Src/* contains all previous mentioned folder
  Similar to an internal file explorer



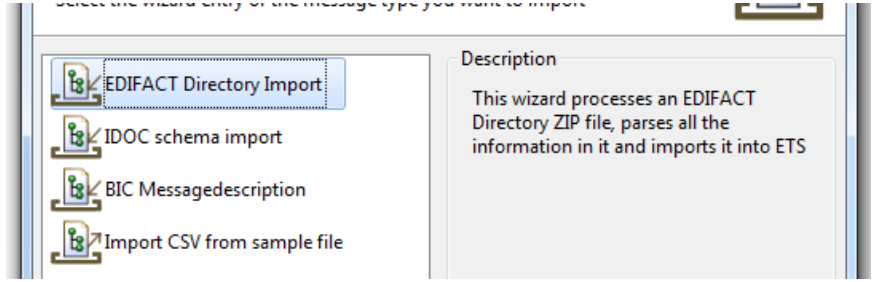
Import of message structures – UNECE/EDIFACT

- Open the context menu and click on Import
- *Select ETS/Import schema*

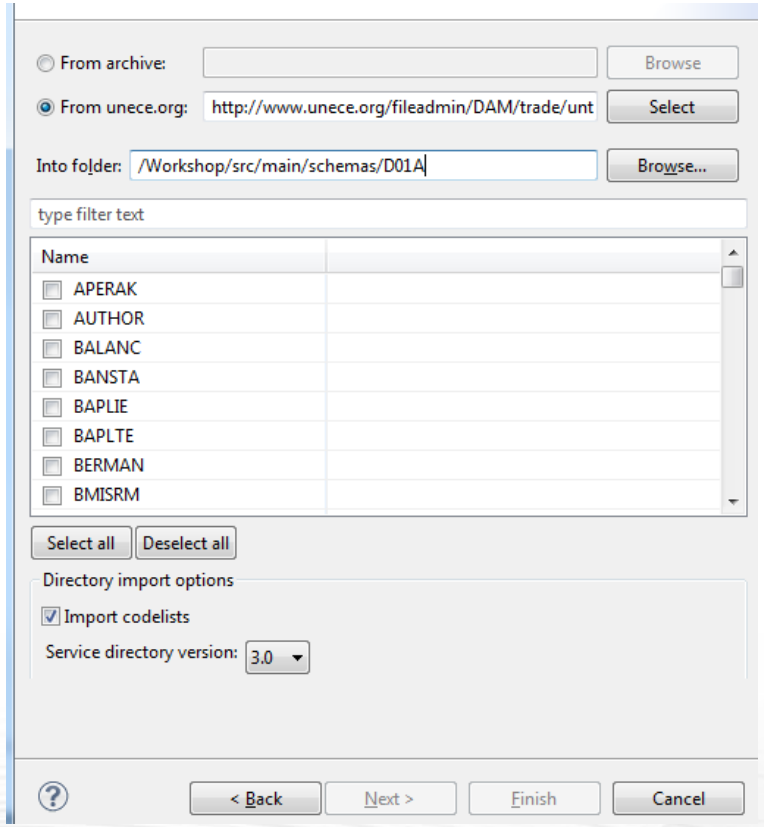
1



2



3



Characteristics of UNECE import

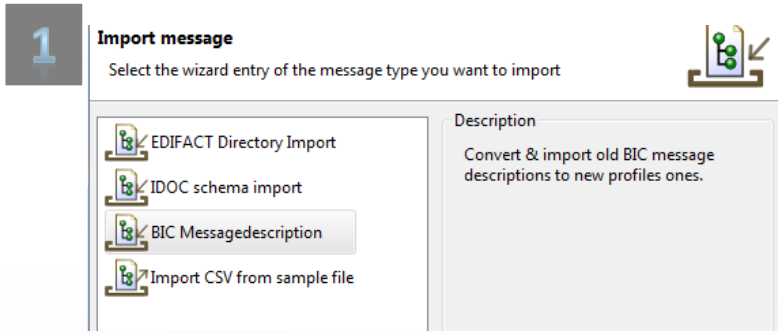
- Separation of a message structure into several XSD
  - Definition of data elements and segments
  - Can be different for each release version => use dedicated folder for each release
- Contains standardized code list, if selected while import (default setting)
- Syntax Level can be chosen
- Internet connection to the UNECE server is required (may be blocked by a firewall)

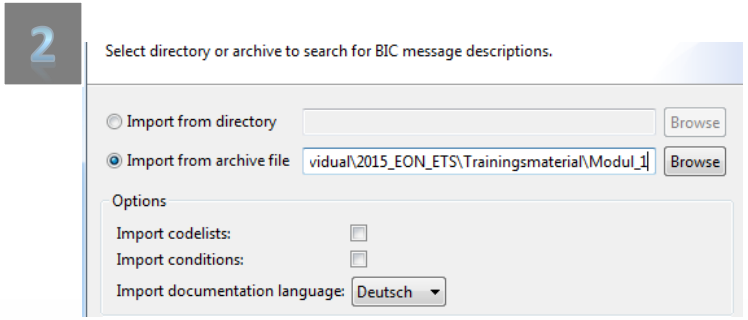
Import of message structure - XML

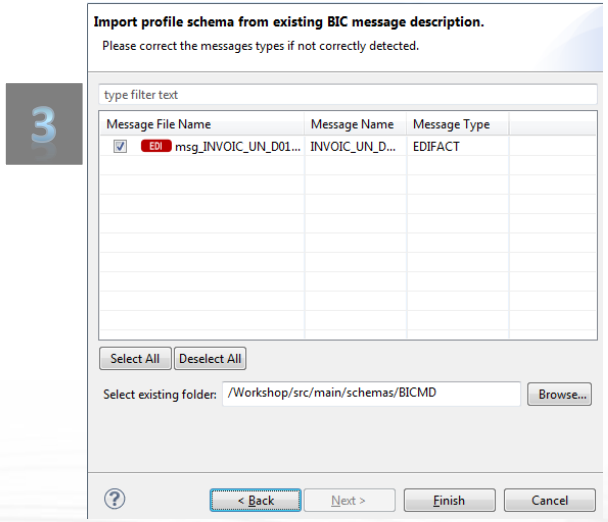
- Simply copy the XSD-files into the ETS and use it
  - Folder: src/main/schemas
  - Hint: create a subfolder
- Overlays can also be created on XSD-Files
- Folder are created automatically for SEEXSD files
  - Folder: src/test/messages/*
  - More information in part 2

Import of message structure - BIC

- Import by context menu (like UNECE-Import)
- If you try to import a mapping (BICMD file) there will be an error
  - The error is for the mapping code, that can‘t be imported into the ETS
  - The message structure(s) can be imported anyway







Part 1: Exercise

First steps with the ETS

Agenda Part 2: Overlays

- Basics: What is an Overlay?
- Creation of a new Overlay
- Modification to a message structure
- Changes to code lists
- Testing

Basics: Overlay

- The difference between “Schema-Overlay” and “Overlay”
  - Schema-Overlays define all changes to the original message description
    - Avoiding of redundancies
    - Schema-Overlays are files named *.overlay
  - Overlays are all changes to a data element/segment/etc.
    - Manual checks are also done here
- Overlays: Changes/checks can be done at different locations
  - Data elements/Fields
  - Segments/Records/Groups/Nodes
  - Logical level (e.g. Message)
  - Virtual nodes, e.g. Segment groups or manually defined virtual segment characteristics

Basics: Overlay

Schema:

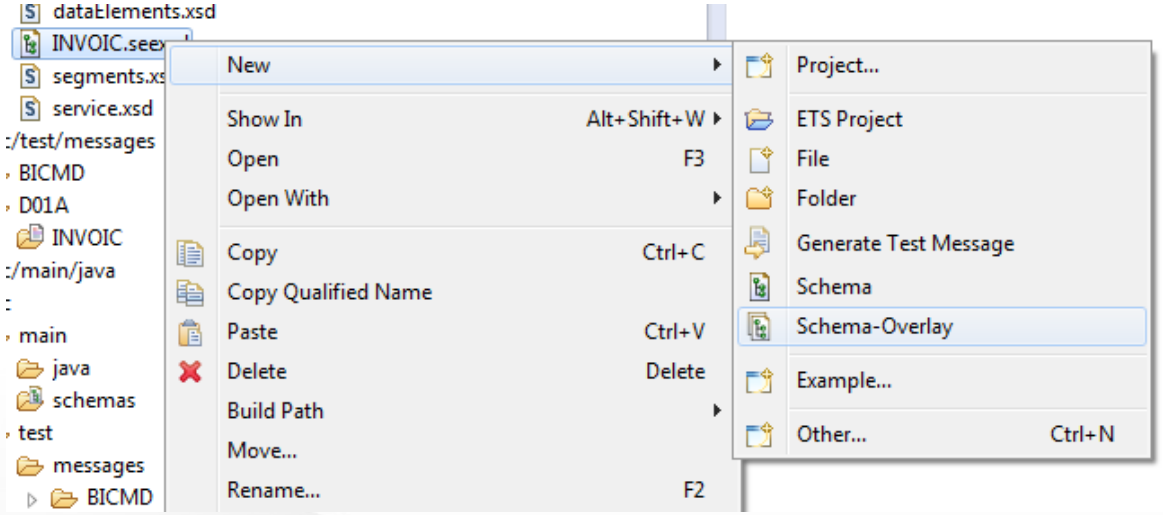
- Message Description, XSD, ...
- Defines the general syntax (e.g. allowed segments)

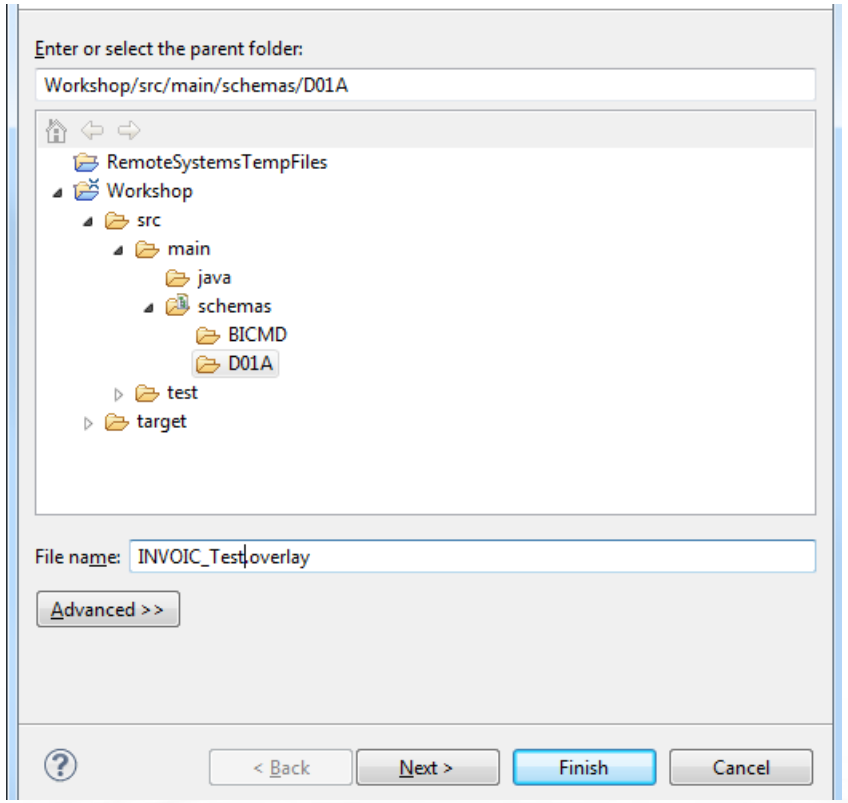
Schema-Overlay:

- "Refinement" of the schema
- Contains devations to the schema
  → "Overlays" (within the schema-overlay)
- e.g. checks against EDI guideline

Create a new Schema-Overlay

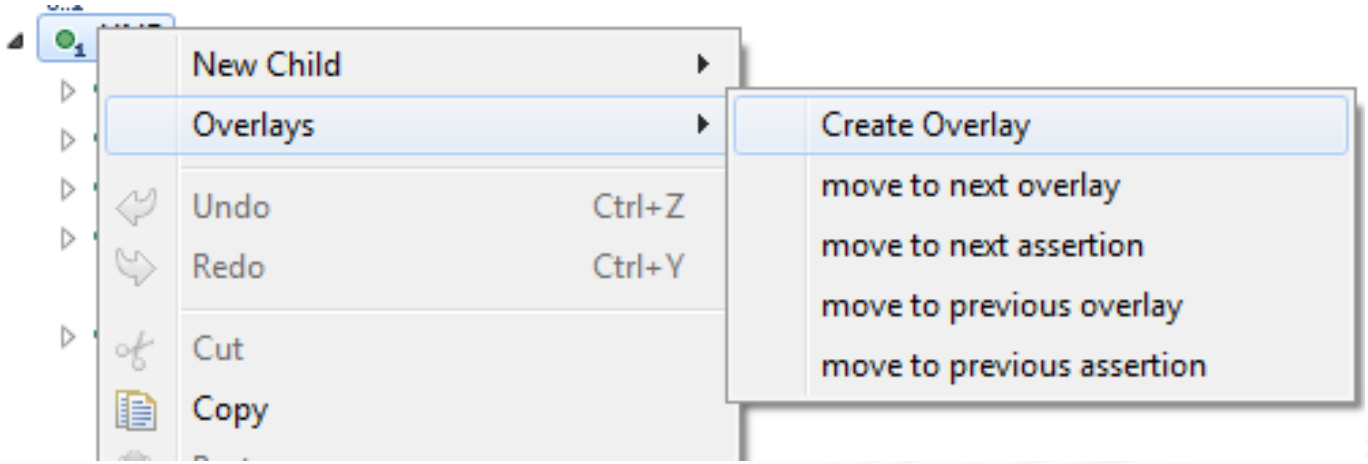
- Right click on the (SEE)XSD
- Select *New/Schema-Overlay* in the context menu





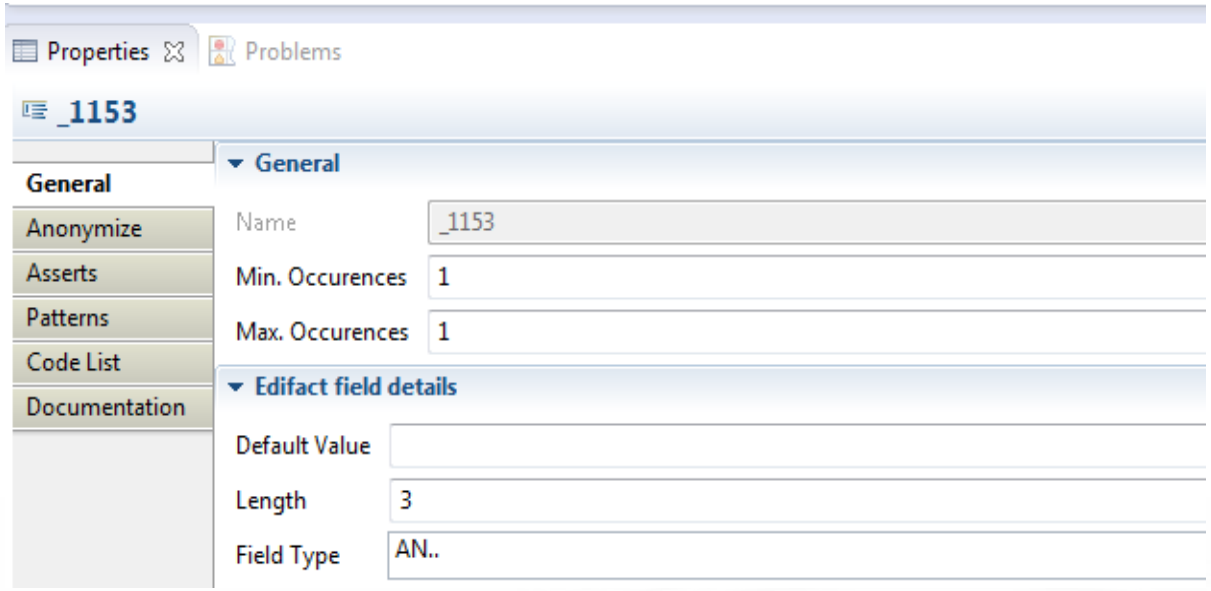
Create a new Overlay

- Right click to the data element
- Select Overlays/Create Overlay in the context menu
- Alternative: Double click on the data element to create an Overlay quickly
- **Important**: Changes can only made on elements where Overlays are created on



Modification to a message structure

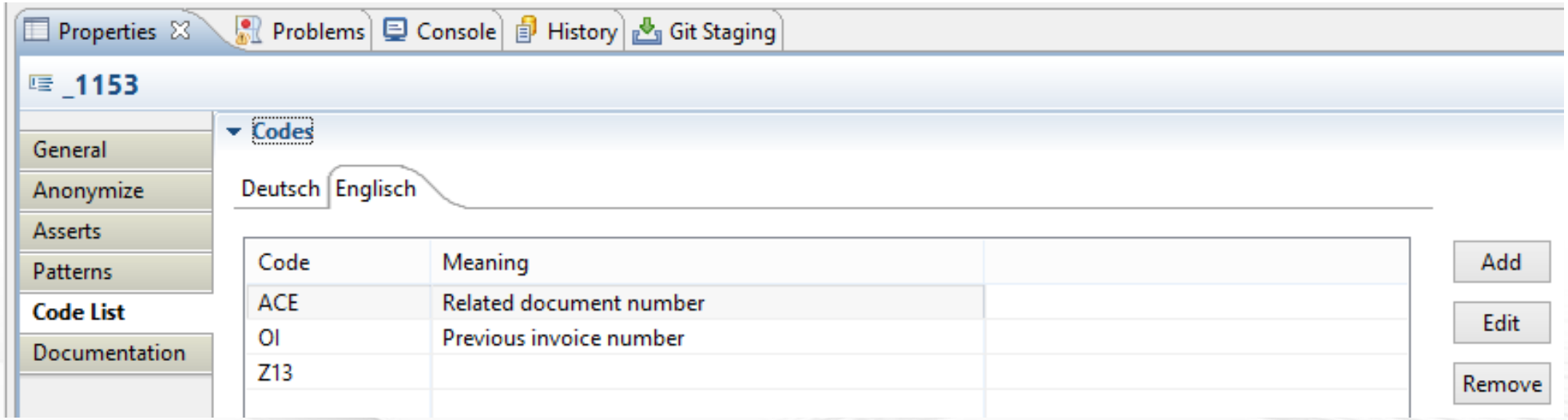
- Properties-Tab General
  - Min./max. Repetition of data elements/segments/etc.
- For data elements only
  - Length
  - Field type



Hint: By default, only changes that make the schema more restrictive are allowed. To allow any change, configure under Window > Preferences > ETS > Overlays

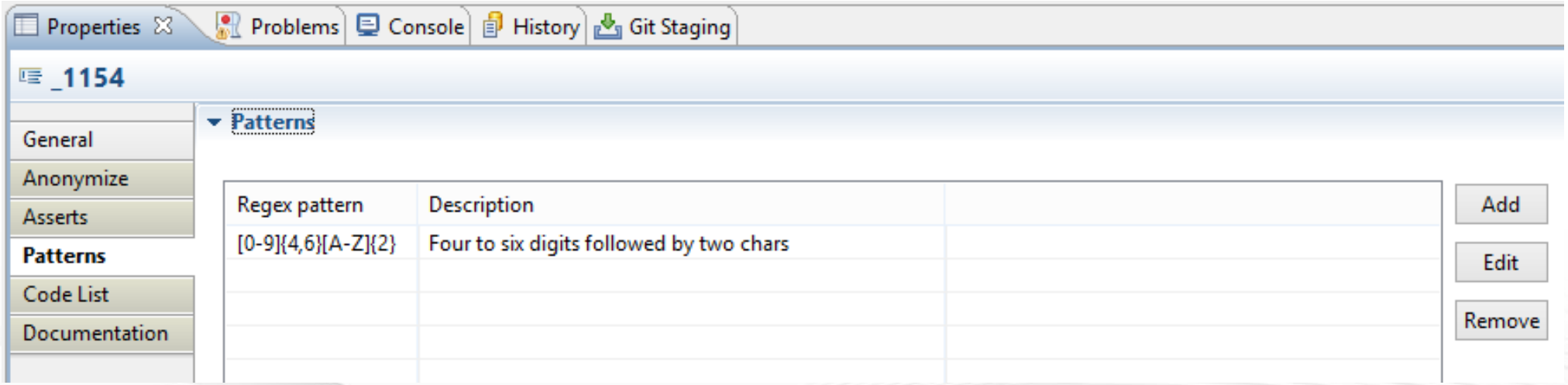
Changes to code lists

- Properties-Tab Code List
- List of possible values for the data element
- There can be a description for each value
  - Possible to use several languages



Regular expression

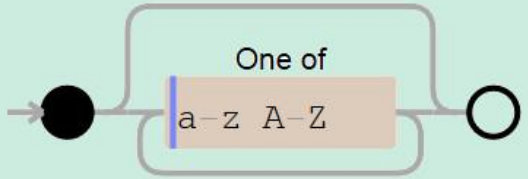
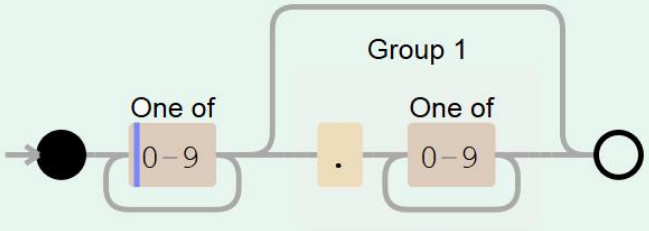
- Properties-Tab Pattern
- Complex checks for data elements
- Different expressions for one data element possible



Regular expressions

Characters (examples)	
[a-z]	All lower case letters
[A-Z]	All upper case letters
[0-9]	Numerical values 0-9
.	Any character
\	Masking character, Example: \. means . (Dot)
\d, \w	Shortcuts e.g.: \d: digits (0-9), \w: Letters, digits and Underscore
Operators	
?	Leading expression is optional
+	Expression must occur minimum 1x (equal to {1,})
*	Maximum occurrence unbounded (equal to {0,})
{n}	Expression must occur n times
{min,}	The leading expression must occur at least min times
{min,max}	The expression must occur at least min times, maximum occurrence max times
|	Alternative

Examples: Regular expressions

Example RegEx	Explanation	Visualisation
[a-zA-Z]* Abc abc1	Undefined number of upper-case and lower-case letters ⇒correct ⇒incorrect	
[0-9]+(\.[0-9]+)? 123.456 .890 123.	Numeric value with optional deziaml point (no length limitation) ⇒correct ⇒incorrect ⇒correct	
\d{1,4} 123 12345	Numeric value with min-length one digit max-length four digits ⇒correct ⇒incorrect	

Hint: Check of the complete string with ^ as prefix and $ as suffix (otherwise a correct substring is enough): ^RegulärerAusdruck$

Tutorial: http://regexone.com/
Online RegEx Test: http://www.regex101.com/
Online RegEx Visualizer: https://www.debuggex.com/

Testing

- Local tests in ETS are possible
  - Instant validation of test file while opening it
  - Compliance Check Run
    - Possible Output: CONTRL, APERAK, SEEAPPACK, HTML-Report
- Correlation of test file to Overlay via folder

Validation of test file while opening it

- Open a file by double click
- Errors are shown at the segment (Mouse over)
- The Problems-Tab contains all errors

```
UNA:+.? '
UNB+UNOC:3+S009114300005:502+4042322100002:14+
"06B" ist nicht Teil der Codeliste für Feld _0054
BGM+380+PRN231160316783+9'
DTM+137:20120627:102'
```

Properties | Problems

2 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (2 items)				
'000418287663' entspricht nicht dem angebenen Format '	INVOIC_D06A...	/Workshop/src/test...	Unknown	Problem
"06B" ist nicht Teil der Codeliste für Feld _0054	INVOIC_D06A...	/Workshop/src/test...	Unknown	Problem

Compliance Check Run

- Open the context menu for the test file you want to check
- Click Run As/Compliance Check
- The report files are created in the folder target/test/messages/* and opened instantly

Validation Report (SEEAPPACK)

Validated Message (INVOIC)		SEEAPPACK-Details	
Message Type	INVOIC	Message Type	SEEAPPACK
Version	UN D.06A	Version	-- --.--
Association Code	2.6e	Association Code	--
Interchange Reference	000418287663	Interchange Reference	DEFAULT_VALUE
Category of Transaction	458	Category of Transaction	--
Creation Date	27.06.2012 - 23:54	Creation Date	17.10.2017 - 16:46
Delivery Date	--		

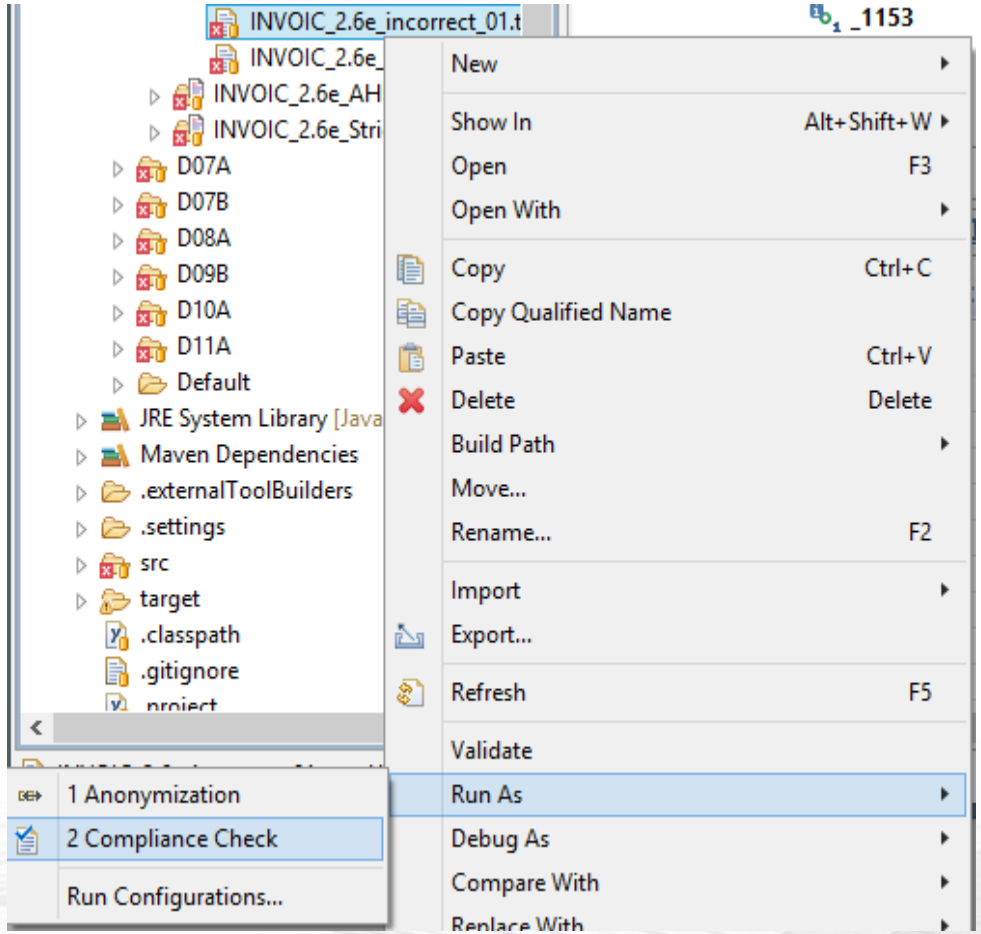
Sender (Validated Message)		Receiver (Validated Message)	
Sender ID	9870114300005	Receiver ID	4042322100002

Status

NOT COMPLIANT

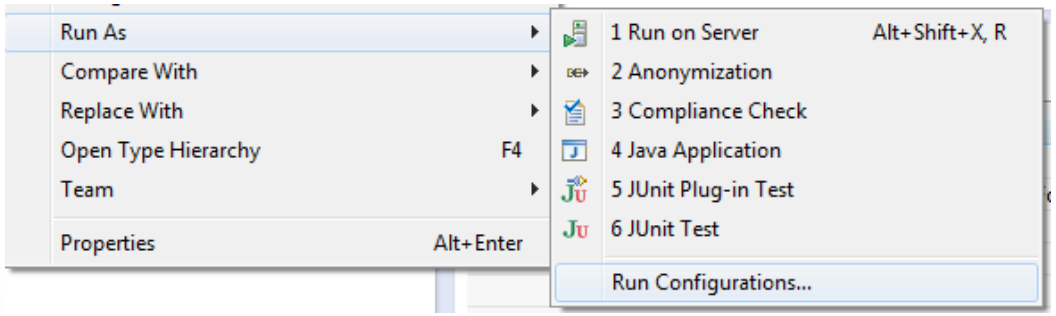
Summary

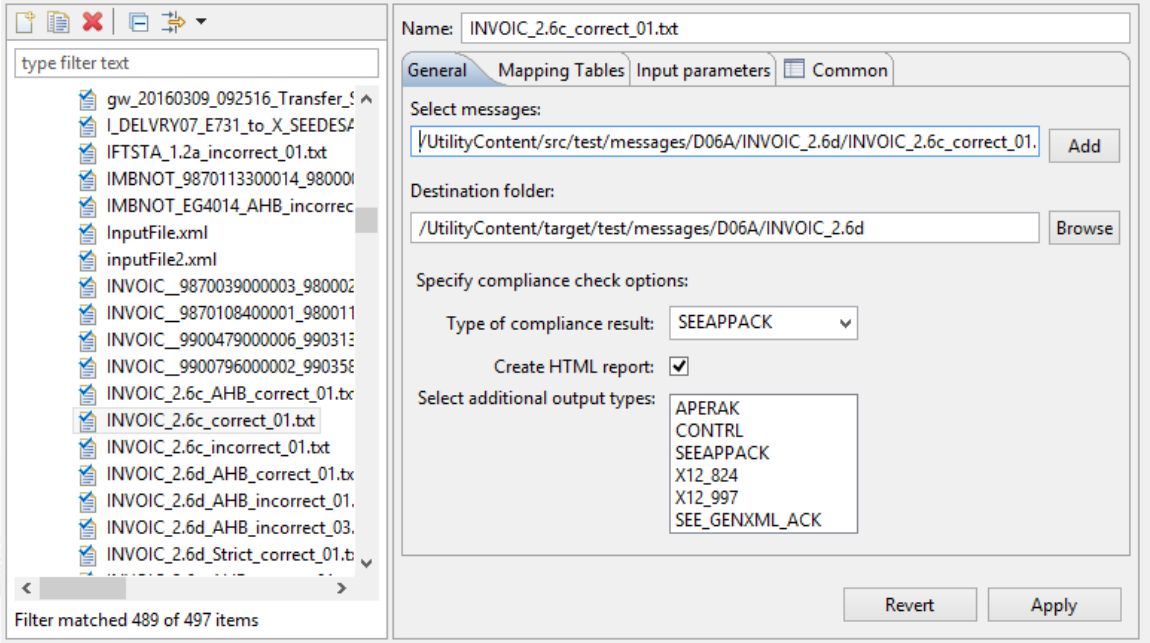
Error Code	Error Message	Count
12	Invalid value	4



Compliance Check Run

- Configuration for the local Compliance Check Run can be done under Run Configurations
- Select Run As/Run Configuration
- Response message format, special input parameters
- More information to Run Configurations is in the ETS help chapter 5.6.2





Part 2: Exercise

Overlays and structure adjustments

Agenda Part 3: Assertions I

- Assertion types
- Assertion structure
- XPath
- Positioning of assertions
- Shortcuts for functions

Assertion types

- Assertion
  - Checks which have to be validated as correct or incorrect
- Positive check (validated as correct)
  - Keyword: Assert
  - Expected result: Boolean true
- Negative check (validated as incorrect)
  - Keyword: fail
  - Expected result : Boolean false

Assertion structure

Keyword(expression)report(„error message", "error code“)

- Keyword
  - Assert for positive checks
  - Fail for negative checks
- Expression
  - Statement that has to be true/false
  - E.g. code list checks
- Error message
  - This text is displayed in the report, when is
- Error code
  - special code for the error

Template:

assert (expression) report("message", "errorCode")

Explanation:

If the expression result is false, an error with the given message and error code is reported.

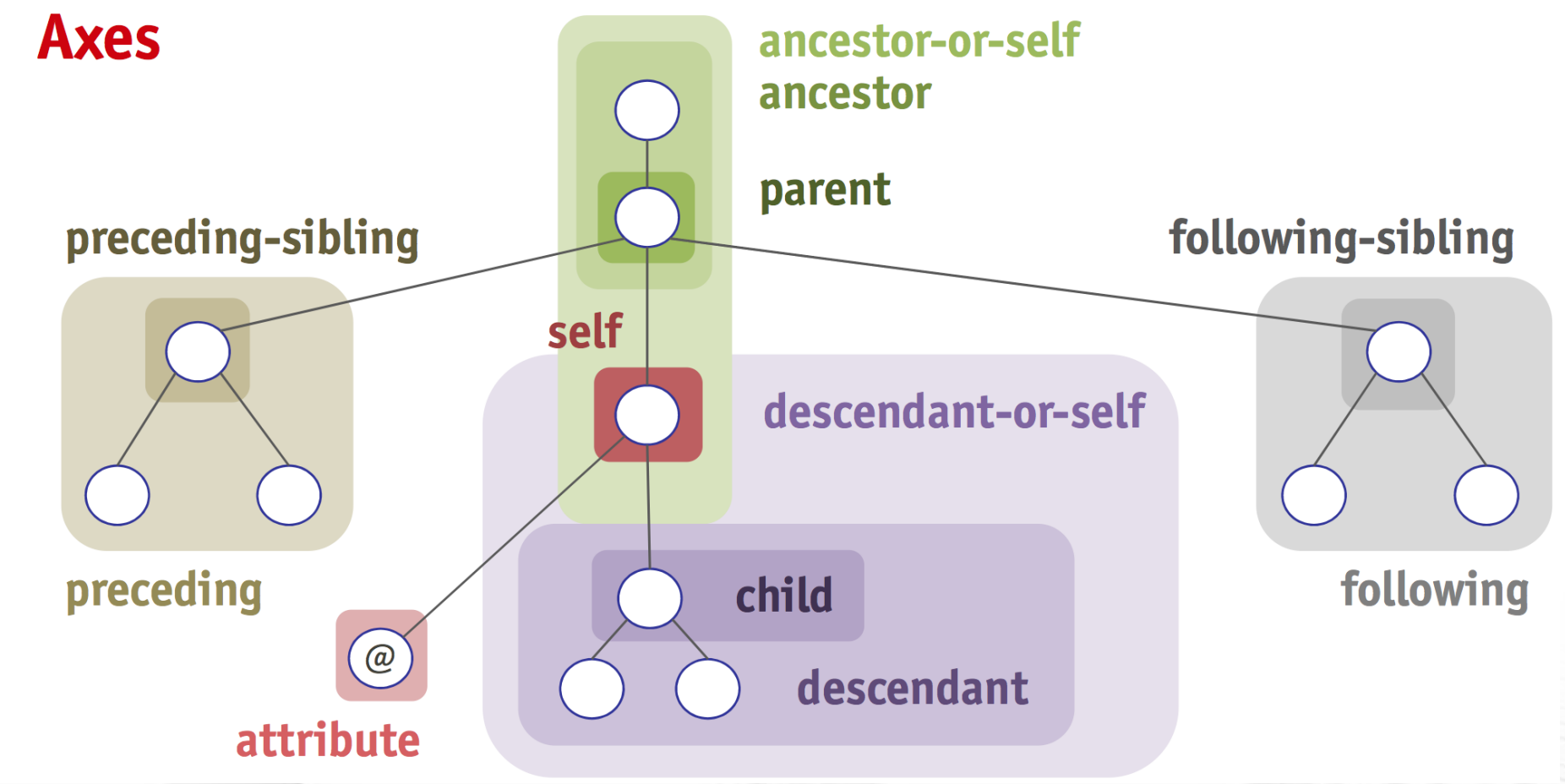
Example:

assert(_4713 = '1') report("Value of field 4713 must be 1.", "E42");

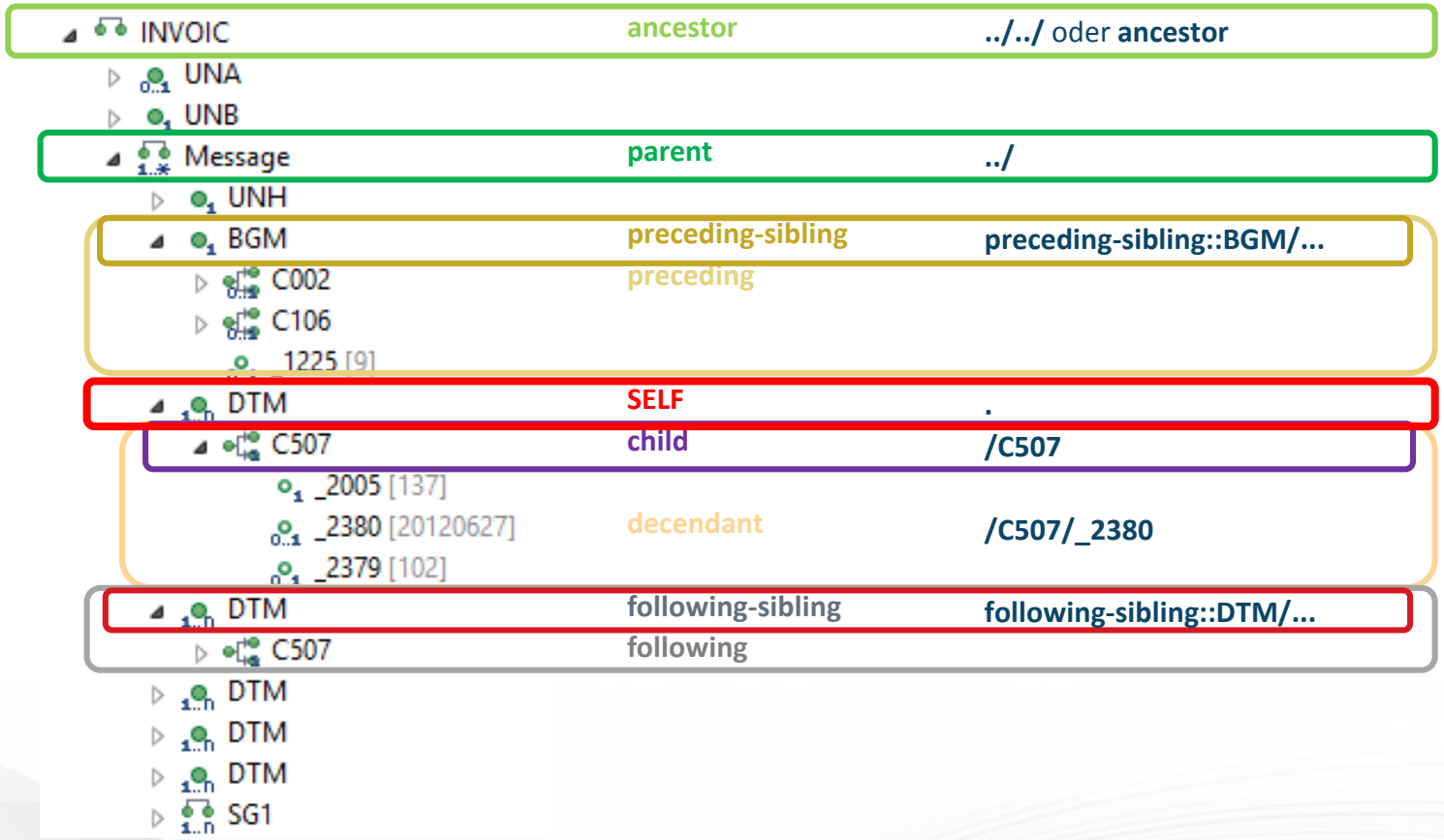
XPath

- Query language for XML documents
  - All message types are saved as XML within the ETS
- Definition by the W3 consortium
  - Global standard
- Navigation with axis, nodes, etc.
  - Ancestor/Descendant/Parent
  - Following/Preceding
  - Support of short notations
  - etc.
- Numeric element names
  - In XPath the element names must not be numeric
  - **Hint**: EDIFACT / ANSI data elements have numeric names, so a leading underscore is added

XPath: Nodes and axes



XPath: Nodes and axes



Xpath: Filters and lists

- Filters
  - Depiction with [filter]
  - Z.B. DTM/C507[_2005=,9']/2380

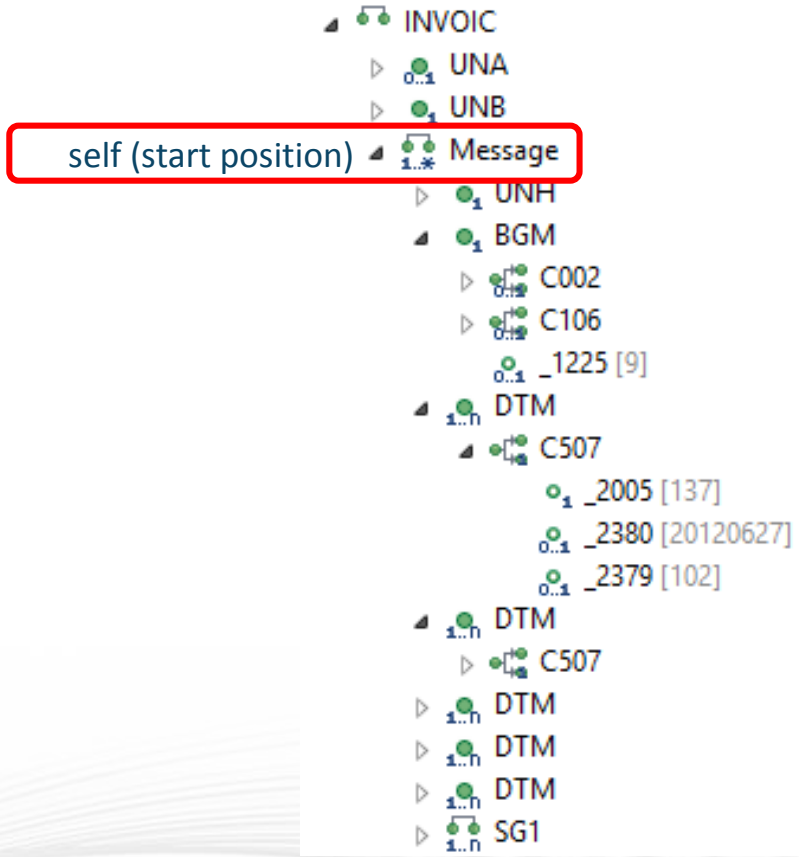
- Lists
  - Depiction (v1,v2,…)
  - Z.B. DTM/C507/_2005 =(,137','Z08')

- Combination of Filter und List
  - Filter on multiple values
  - DTM/C507[_2005=(,9, ,137')]/2380

Xpath: Filters and Lists

```
UNB+UNOC:3+9870114300005:502+4042322100002:14+120627:2354+14182872123663'
UNH+92907+INVOIC:D:06A:UN'
BGM+380+PRN231160316783+9'
DTM+137:20120627:102'
DTM+9:20120630:102'
DTM+155:20120802:102'
```

- **/BGM/_1225**
  → {„9“}
- **../UNB/_0020**
  → {„14182872123663“}
- **/DTM/C507/_2005**
  → {„137“, „9“, „155“,...}
- **/DTM/C507/_2380**
  → {„20120627“, „20120630“, „20120802
- **DTM[C507/_2005=‘137‘]/C507/_2380**
  → {„20120627“}
- **DTM[C507/_2005=(‘137‘,‘9‘)]/C507/_2380**
  → {„20120627“, „20120630“}



Assertion examples

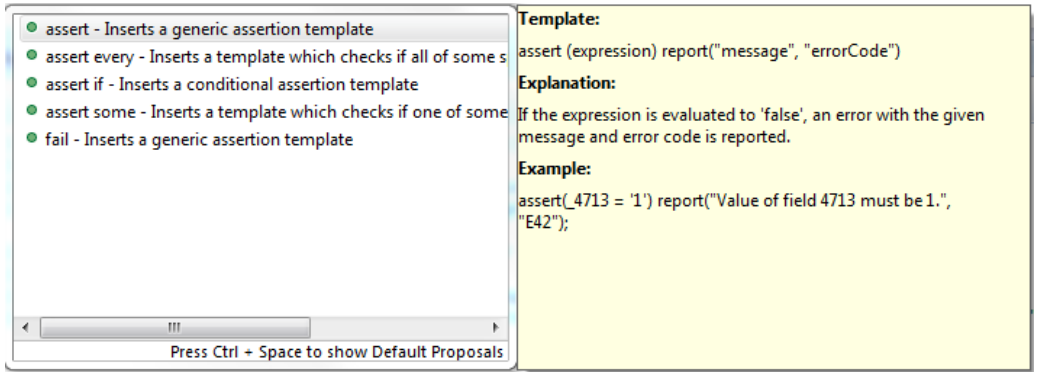
```
(:In DTM+137, 2379 must be '203':)
assert(DTM/C507[_2005 = '137']/2380 = '203') report(„Qualifier is not part of the code list", "12")

(:there must be one SG2.NAD with _3035 = 'MS':)
assert(SG2/NAD[_3035= "MS"]) report ("Segment is missing (NAD+MS)", "13")

(:There must be one IDE:)
assert(SG4/IDE) report("Segment is missing (IDE)", "13")

(:Only one UNH is allowed :)
fail(preceding-sibling::Message[UNH]) report(„There are too many segments (UNH)", "16")
```

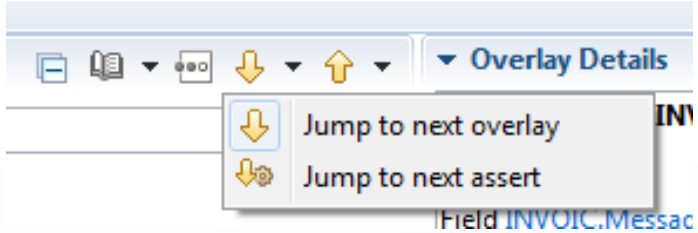
Positioning of assertions

- Assertions can be used nearly everywhere
  - Assertions can be simplified by using a good position
  - All necessary information can be collected by XPath
- Context is important
  - e.g. item checks on header level
- XPath addresses are relative to their position
- Tip:
  - Existence checks make most sense at higher (parent) level („**Every message** must contain a Segment XYZ“)
  - Content checks make most sense close to or on the respective field („**The field** XYZ must be numeric“)

Useful Things

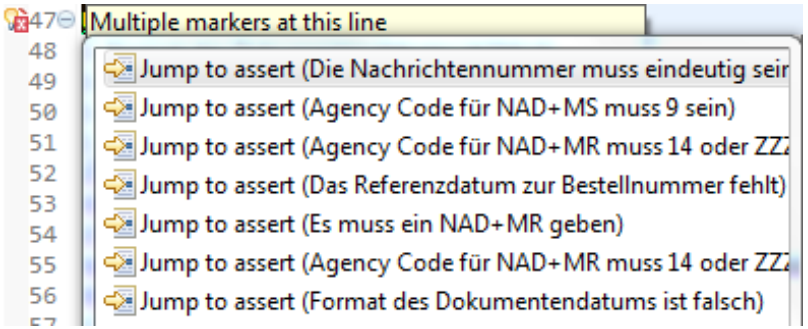
- Auto completion
  - Shortcut: Ctrl+ Space
  - Pattern for Assertions
  - Functions, paths, etc.
- Next/previous Overlay
  - Go to next/previous Overlay
  - Shortcut: Ctrl + arrow up/down
- Next/previous Assert
  - Go to next/previous Overlay containing an Assert
  - Shortcut: Ctrl+ Shift + arrow up/down



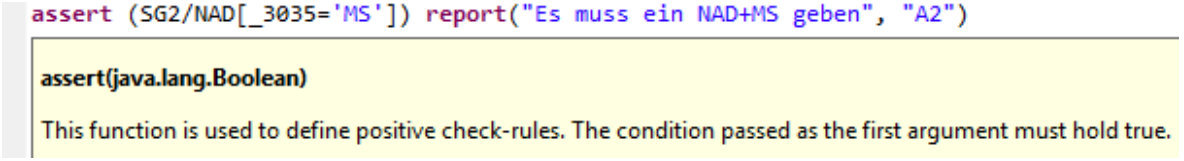


Useful Things

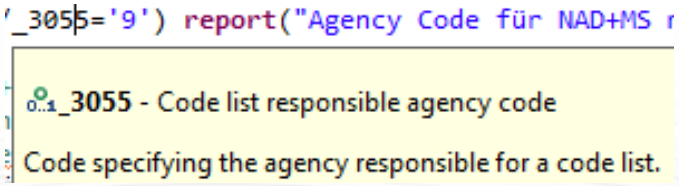
- Jump to assert
  - In message editor
  - Jump to Assertion in Schema-Overlay



- ETS-Help
  - Integrated into Dev-Studio help



- Mouse over
  - Tooltip to functions, data elements, etc.



Useful Things

- Comments
  - Helps to understand the following assertion
  - Display: `(:This is a comment:)`

- Marking of bracket pairs
  - The counterpart of the selected bracket is marked

```
Format 102 oder 20
79=('102','203')) r
```

Part 3: Exercise

First checks

Agenda Part 4: Assertions II

- Conditions within Assertions
- Conditions outside of Assertions
- Variables
- Function calls

Conditions (in Assertions)

if(Condition) then expression_1 else expression_2

- Changes the behaviour of the check while execute it
  - (De-)Activation of the check
  - Alternative check
- Else-clause is optional

Template:

assert (if (condition) then expression1 else expression2) report("message", "errorCode")

Explanation:

A conditional expression consists of a condition, a 'then' clause, and an optional 'else' clause. If the condition is satisfied, the 'then' clause will be applied. Otherwise, the 'else' clause (if present) will be applied.

Example:

assert(if(_4713 = '1') then c = '0' else c = '3') report("Wrong value of child node c.", "E42");

```
(:Delivery note number RFF+DQ needs a delivery note date :)
assert ( if (RFF/C506/_1153='DQ')
  then DTM/C507/_2005='171'
) report("delivery note date is missing", "1402")
```

- Comparison
  - <, >, =, >=, <=, etc.

Conditions (outside of Assertions)

if(condition) {Assertion(s)}

- Changes the behaviour of the check while execute it

```
if(BGM/C002/_1001='380'){
  (:there must be a MOA+79 in SG50 :)
  assert (SG50/MOA/C516/_5025='79') report("there must be a MOA+79 in SG50", "13")
}
```

- Can prevent redundancies of identical conditions

```
(:there must be a MOA+79 in SG50 :)
assert ( if (BGM/C002/_1001='380') then SG50/MOA/C516/_5025='79') report("
```

Variables

let $VarialeName := value

- Content of the variable can be set only once
  - No (unintended) overwriting
- Variables are only available in child nodes
  - following hierarchic level
- No different declaration for different types
  - Variable can contain numeric, alphanumeric, etc. values
- Optimising of Assertions
  - Complicate/long path is simplified with the use of the variable

```
let $DocumentDateOrDefault := if(DTM/C507[_2005='137']/_2380 and DTM/C507[_2005='137']/_2379 = '102')
                                then DTM/C507[_2005='137']/_2380
                              else "000000"
```

Function calls

- Functions can be used nearly everywhere
  - Assertion, condition, error message, etc.
- Many XPath functions are usable
  - e.g. string-length

- Call in assertions
  - Example: assert(isFilled(C507/_2380)) report(“2380 not filled!“,“13“)

- Storing the result in a variable
  - Example: let $dateTime := asEdifactTime(C507/_2380, C507/_2379)

Functions - examples

- **isFilled(field)**
  - Returns true if the data element/variable is filled, otherwise false
  - isFilled(DTM/C507[_2005='137']/_2379) => true/false
- **string-length(string)**
  - Returns the length of the string
  - string-length(„String“) => result: „6“
- **matches(string content, string pattern)**
  - Checks given value with a regular expression pattern
  - matches(C506/_1154,'^[1-9]([0-9]+)?$') => true/false
- **concat(string, string, …)**
  - Combines all parts to one string
  - concat(„This “,“is “,“an “,“example“) => „This is an example“

Functions - examples

- **getInputParameter(string parameterName)**
  - Reads in compliance component parameter
  - getInputParameter(„Param_1“)
- **getInputParameterOrDefault(string parameterName, string defaultValue)**
  - Reads in compliance component parameter, if empty the default value is used
  - getInputParameterOrDefault(„Param_1“,“Not_Found“)
- **isInputParameterSet(string parameterName)**
  - Checks if the parameter is set
  - isInputParameterSet(„Param_1“) => true/false
- **isValidEdifactTime(string date, string format qualifier)**
  - Validates the given date string with the EDIFACT format code
  - isValidEdifactTime(., "101") => true/false

Functions - examples

- **asEdifactTime(string date, string format qualifier)**
  - Converts a date string to a date object according given EDIFACT format code
  - asEdifactTime(C507/_2380,C507/_2379)
- **parseDate(string date, string date pattern)**
  - Converts a date string to a date object according to the format string
  - parseDate(head/messageDate,“yyyyMMddHHmmss“)
- **current-dateTime()**
  - Returns a date object of the current date/time
  - current-dateTime()
- **before(date object, date object)**
  - Compare two date objects, the first date has to be previous to the second
  - before($Date_1,$Date_2) => true/false

Functions - examples

- **lower-Case(string)**
  - Returns given string as small letter value
  - lower-Case(„ABC“) => „abc“
- **upper-Case(string)**
  - Returns given string as capital letter value
  - upper-Case(„Abc“) => „ABC“
- **number(numeric string)**
  - Converts a numeric string to a mathematical number
  - Needed for numeric operations e.g. calculations
  - Cuts leading zeros and zeros after the decimal separator
  - number(„0123.456000“) => 123.456

Functions - examples

- **not(expression)**
  - Inverts the Boolean value
  - not(true()) => false
- **false()**
  - Returns the Boolean value false
  - false()
- **true()**
  - Returns the Boolean value true
  - true()

More functions and detailed documentation can be found in the Guideline Tool Help Contents: Guideline Designer > 5 Schema Overlays > 5.5 Assertions > 5.5.4 Function Reference

Part 4: Exercise

Assertions with conditions and function calls

Agenda Part 5:
Assertions III / Overlaystacking

- Parameter
- Naming
- Overlay stacking

Parameter

with(„Parameter name“, Parameter value)

- Additional values per assertion
- Predefined parameter names
  - Values for EDI feedback message (e.g. CONTRL)
  - e.g. error location
- Individual parameter names
  - Additional information to HTML report

```
(:A message must contain a NAD+BY:)
@named("Message_NAD_BY_Check")
assert (SG2/NAD[_3035='BY'])
report("SG3 NAD+BY is missing", "13")
with("DESCRIPTION","Buyer")
```

General Error	(Message Reference: 000001)
Error	**SG3 NAD+BY is missing**
Code	13
	Parameter
	DESCRIPTION Buyer
Position	Segment: 2 | Message Reference: 000001
Path	ORDERS/Message

Predefined parameters

- CONTENT
  - Content of the data element
- CONTEXT
  - Whole segment with delimiter. Hint: needs content() function

```
(:Date/Time Format must match the format qualifier :)
@named("DTM_2380_Format_Check")
assert (if(isFilled(C507/_2380) and isFilled(C507/_2379))
  then isValidEdifactTime(C507/_2380, C507/_2379))
  report(concat("Date format for DTM+",C507/_2005," does not match the format qualifier ",C507/_2379), "12")
  with("CONTEXT",content(.))
  with("CONTENT",C507/_2380)
```

General Error	(Message Reference: 000001)
Error	**Date format for DTM+137 does not match the format qualifier 102**
Code	12
Context:	DTM+137:20170229:102
Content:	20170229
Position	Segment: 3 | Message Reference: 000001
Path	ORDERS/Message/DTM

Special EDIFACT parameters

Needed for generating a complete EDIFACT CONTRL from Checks

- ELEMENT
  - Index location of the data element, in context of the segment
- SUB_ELEMENT
  - Index location of the data element, in context of the containing data element container
- SEGMENT_COUNTER
  - Segment counter of the refered segment in context of the message

Report Level

Can be set after an assertion and all parameters

- error (Standard)
- warning
- info

```
(:UNB date must be valid:)
assert (isValidEdifactTime(., "101"))
report ("Date format of 0017 not correct", "12")
level warning
```

Summary

Warning Code	Warning Message
12	Date format of 0017 not correct

Validation Report

Warning	
Warning	**Date format of 0017 not correct**
Code	12
Position	Segment: 2
Error Location	5:1
Path	INVOIC/UNB/S004/_0017

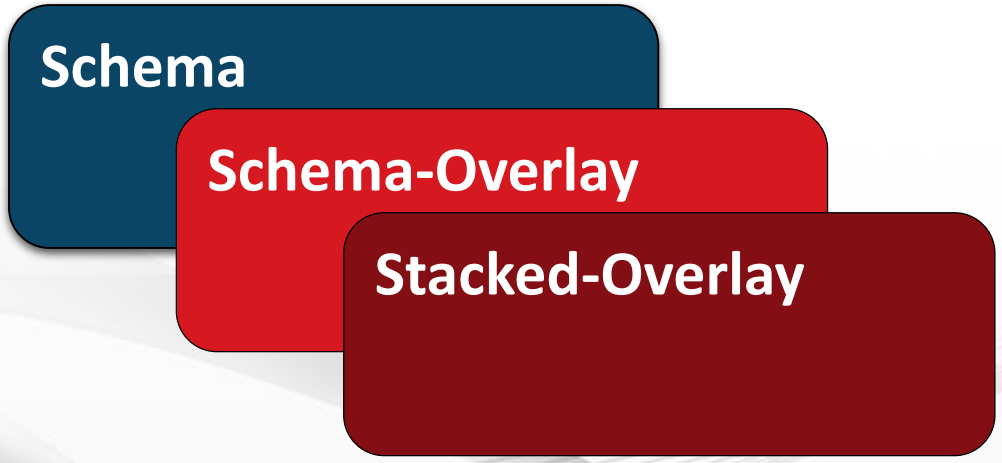
Naming
@Named(„Name_Assertion“)

- Set a name to an assertion or an if-block
- Must be unique per segment/data element
- Allows to refer to an assertion
  - Necessary for overlay stacking

```
@named("Message_Check_1")
(:there must be an order Number :)
assert (isFilled(SG1/RFF/C506[_1153='ON']/_1154)
) report("The order number is missing", "13")
with("My_own_param_1","First own parameter")
```

Overlay stacking

- Allows overwriting/enhancing of a existing (base) Overlay
- The stacked overlay inherits the assertions from the base Overlay
  - Changes in the base Overlay also affects the stacked Overlay
  - Overwritten Assertions are not actualized automatically!
- The creation of the stacked Overlay works the same way as described in Part 2
  - Basis for stacked Overlay is another Schema Overlay instead a message Schema



Overwriting of Assertions
@Override(„Name_Assertion“)

- Replaces the original Assertion or if-block
- Deactivation with an empty Override statement
  - There is a warning which can be ignored
- Original Assertion is shown in mouse over popup
- It’s impossible to overwrite variables
  - Variables can also be used in stacked Overlay

```
@override("Message_Check_1"){}
```

```
@override("Message_Check_1")
(:if BGM+380, then there must be an order Number :)
assert (if(BGM/C002/_1001='380') then isFilled(SG1/RFF/C506[_1153='ON']/_1154)
) report("The order number is missing", "13")
with("My_own_param_1","First own parameter")
```

```
@override("Message_Check_1")
@named("Message_Check_1") (:there must be an order Number :) assert (isFilled(SG1/RFF/C506
[_1153='ON']/_1154) ) report("The order number is missing", "13") with("My_own_param_1","First own
parameter") with("CONTENT",SG1/RFF/C506[_1153='ON']/_1154) with("CONTEXT",content(SG1/RFF
[C506/_1153='ON'])) with("ELEMENT",2) with("SUB_ELEMENT",1) with("SEGMENT_COUNTER",SG1/RFF)
Press 'F2' for focus
```

Part 5: Exercise

Additional parameter and Overlay stacking

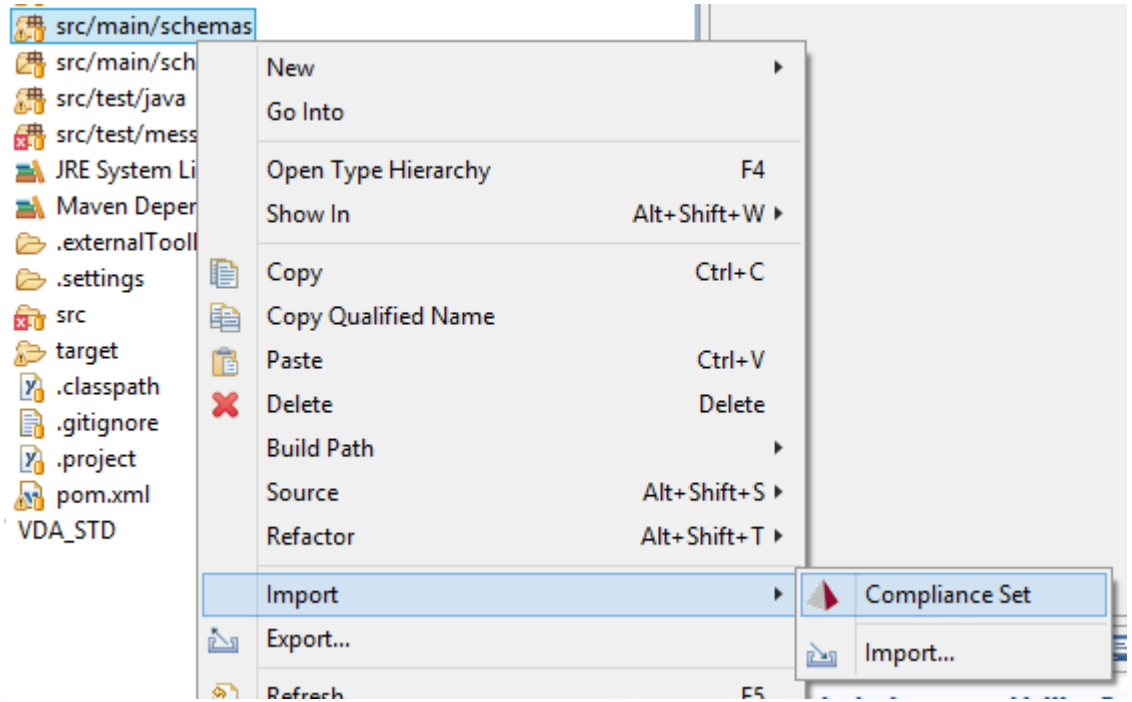
Agenda Part 6:

Export, deploy and configuration in BIS

- Import of existing Overlays
- Export of own Overlays
- Deployment of Compliance Sets in BIS
- Configuration in BIS

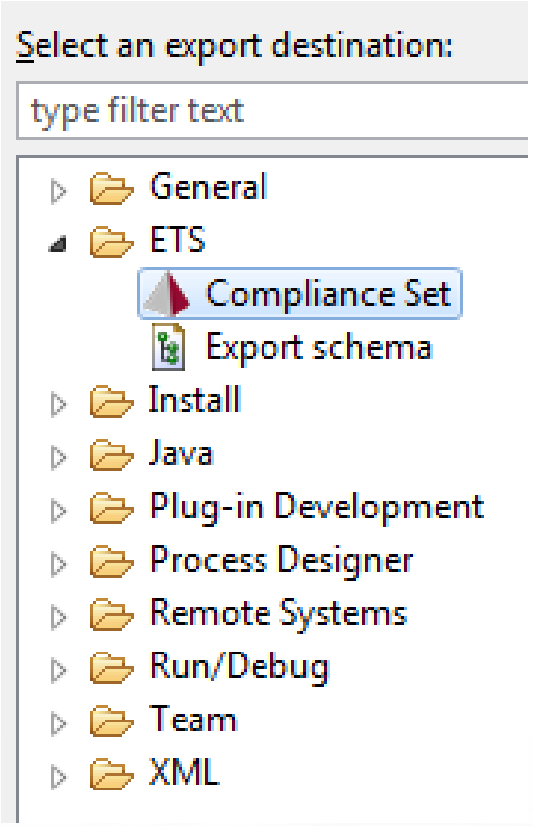
Import

- Import with the operating system
  - Drag & Drop
  - Copy & Paste
  - Folders have to be created manually
- Planned: Import function for Compliance sets
- Compliance Set can be unzipped
  - Compliance.jar =>Compliance.zip
  - Unzipping of the Overlays



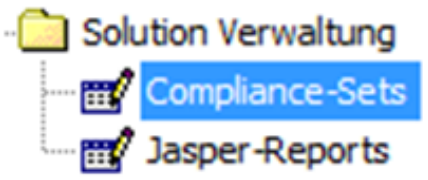
Export

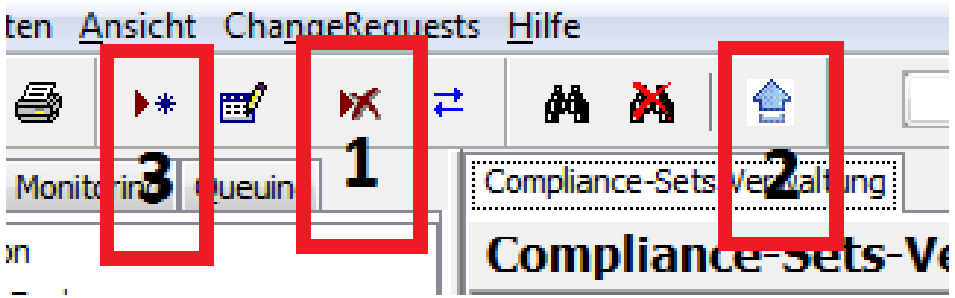
- Single Overlays
  - Drag & Drop
  - Copy & Paste
- Export of the project as Compliance Set
  - context menu => Export
  - ETS/Compliance Set



Deploy on BIS

- Configuration/Solution Management/Compliance Set
  - Undeploy of the old Compliance Set (1)
  - Upload of the new Compliance Set (2)
  - Deploy new Compliance Set (3)
- The Compliance Set name must be the same if you want to update an existing set!





Configuration in BIS

- Configuration in the Entity
- Component „Business Validation“
- Compliance Run settings (excerpt):
  - Compliance set*:
    Selection of the required Compliance Set (name of the .jar file, without file extension)
  - Compliance schema*:
    Complete path to the Overlay or Schema within the Compliance Set jar (regard subfolders)

Business validation	
Compliance set:	MSH_Compliance
Compliance schema:	D01B/DESADV_D01B.overlay
Processing mode:	Reject if errors
Reply mode:	Reply on errors
Create transaction details:	
Create segment/element details:	
Response message type:	SEEAPPACK
Response message interchange counter nan	
Response message message number:	
Reply process message type:	Application error and acknowledgment message
Output report:	HTML report
Queue:	<auto>
log message:	executed acSSC-BusinessCheck with TrackID: $[TRACKID]

* Necessary setting, manual input

Configuration in BIS

- Additional settings (excerpt):
  - Processing Mode:
    Behavior of the component on compliance errors or warnings
  - Reply mode:
    Controls the creation of the acknowledgement message on errors or warnings
  - Response message type
    Set the format of the acknowledgement message (SEEAPPACK, CONTRL, 997, etc.)
  - HTML Report:
    (De-) Activate the creation of the HTML report

Business validation

Compliance set:	MSH_Compliance
Compliance schema:	D01B/DESADV_D01B.overlay
Processing mode:	Reject if errors
Reply mode:	Reply on errors
Create transaction details:	
Create segment/element details:	
Response message type:	SEEAPPACK
Response message interchange counter nan	
Response message message number:	
Reply process message type:	Application error and acknowledgment message
Output report:	HTML report
Queue:	<auto>
log message:	executed acSSC-BusinessCheck with TrackID: $[TRACKID]

Questions?

Copyright