[MS-CTA]:
Claims Transformation Algorithm

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
12/16/2011	1.0	New	Released new document.
03/30/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	2.0	Major	Significantly changed the technical content.
08/08/2013	3.0	Major	Significantly changed the technical content.
11/14/2013	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/13/2014	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
05/15/2014	3.0	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

This document specifies the Claims Transformation Algorithm, which is an algorithm to transform claims based on rules written in the claims transformation rules language, which is defined in this document as well.<1>

Section 2 of this specification is normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Section 1.6 is also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

ABNF
single-valued claim
UTF-16

The following terms are specific to this document:

claims transformation: The process of converting one set of claims by analyzing and filtering the claims and by adding new claims in order to generate a new set of claims.

claims transformation rules language syntax: The context-free grammar expressed in ABNF that specifies the language used to describe the rules used in the Claims Transformation Algorithm.

input claims: The set of claims provided as input to the Claims Transformation Algorithm.

production: An individual ABNF rule in the claims transformation rules language.

production name: The name on the left side of the production.

tag: A production name or a terminal from the claims transformation rules language syntax that is used to identify a portion of the given transformation rules.

terminal: A basic element of the claims transformation rules language syntax.

transformation rules: A set of rules defined according to the claims transformation rules language syntax that specifies how claims are transformed when the Claims Transformation Algorithm is invoked.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[ISO/IEC-9899] International Organization for Standardization, "Programming Languages - C", ISO/IEC 9899:TC2, May 2005, http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

1.3 Overview

This document defines the Claims Transformation Algorithm, which enables parsing, filtering, issuance and transformation of a set of **input claims** based on the input transformation rules.

The **claims transformation rules language syntax** specified in this document defines the syntax for transformation rules.

The Claims Transformation Algorithm essentially is a programmable transformation of claims.

This algorithm can be summarized at a high level as follows: Validate the **transformation rules** using the claims transformation rules language syntax and transform the input claims using the transformation rules based on the claims transformation processing rules.

1.4 Relationship to Protocols and Other Algorithms

This algorithm does not depend on any other protocols or algorithms.

1.5 Applicability Statement

This algorithm is applicable when programmable **claims transformation** needs to be performed on claims.

1.6 Standards Assignments

None.

2 Algorithm Details

2.1 Claims Transformation Algorithm Details

The Claims Transformation Algorithm is illustrated in the following state machine diagram, which consists of the following states:

- Initialization: Initializing the internal state (section 2.1.3).
- Claims Transformation Rules Syntax Evaluation: Validating that the given transformation rules text conforms to the claims transformation rules language syntax and generating transformation rules (section 2.1.4.2).
- Claims Transformation Rules Processing: Transforming input claims to output claims using transformation rules (section 2.1.4.3).
- Collect Output: Collecting the output claims from the transformation process.

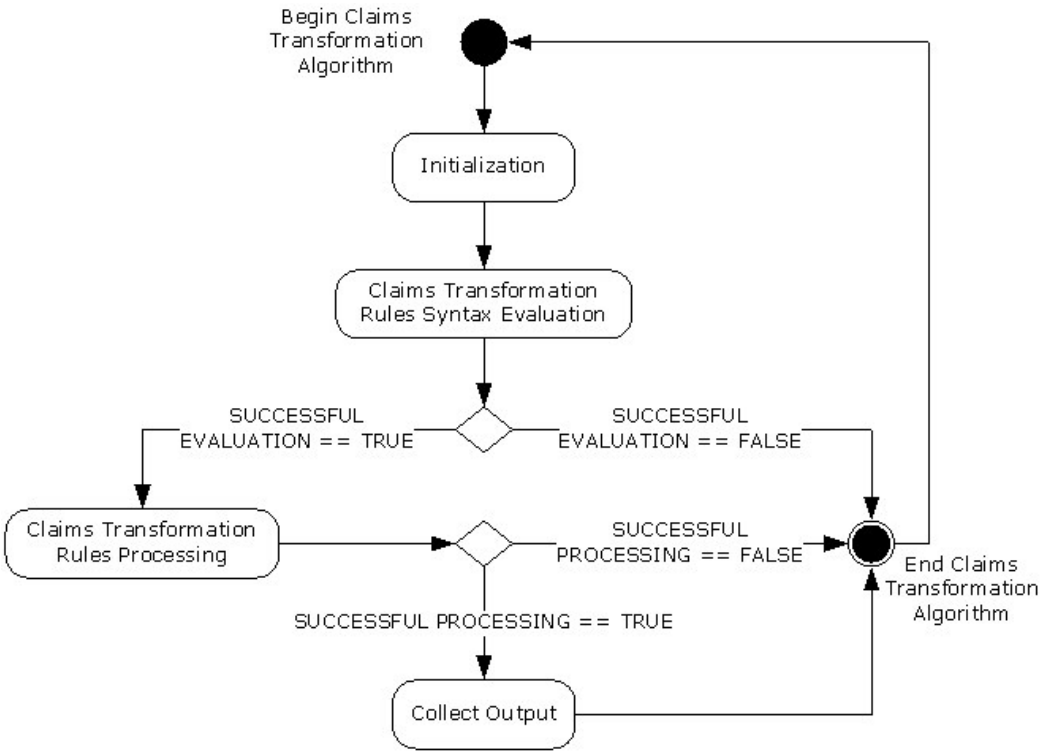

Figure 1: Claims Transformation Algorithm state machine

The Claims Transformation Algorithm depends only on the given input per invocation and does not use any other state for its functioning. It maintains state only on a per-invocation basis and only for the duration of the invocation and does not preserve state beyond that scope.

See the following sections for more details on the various states of the state machine.

2.1.1 Abstract Data Model

None.

2.1.2 Data Structures

- The following data structure definitions are applicable to the current document:

  Claim: A claim is defined as the 3-tuple of following values:

  - TYPE: The type or identifier of the claim, represented as a UTF-16 string.
  - VALUE_TYPE: The value type of the claim VALUE, represented as a UTF-16 string.
  - VALUE: A single claim value; its type depends on the VALUE_TYPE.

  This Claim is a **single-valued claim**.

  VALUE_TYPE: The VALUE_TYPE field in a claim MUST have one of the following **UTF-16** values or a case variation thereof:

  - "uint64"
  - "int64"
  - "string"
  - "boolean"

2.1.3 Initialization

The Claims Transformation Algorithm MUST be invoked by passing in the following parameters:

InputClaims: A set of zero or more claims (section 2.1.2) that need to be transformed.

InputTransformationRulesText: A set of transformation rules in UTF-16 format that define the transformation based on the language defined in Claims Transformation Rules Language Syntax (section 2.1.4.1).

The Claims Transformation Algorithm MUST generate the following output variables:

OutputClaims: This is a list of zero or more claims (section 2.1.2) returned by the Claims Transformation Algorithm when it finishes processing the given input.

ReturnValue: This variable holds the resulting value returned by this algorithm. The possible values are SUCCESS to indicate successful processing and FAILURE to indicate an error during the processing.

The Claims Transformation Algorithm MUST maintain state during processing in the following variables:

1. **InternalTransformationRules**: This is the representation of **InputTransformationRulesText** generated for Claims Transformation Rules Processing. This representation MUST contain the following:

   1. **InputTransformationRulesText**

   2. An ordered, hierarchical list of **tags** from the claims transformation rules language syntax that are arranged to match the given **InputTransformationRulesText** and the corresponding matching portion of **InputTransformationRulesText** for each tag.

2. **InternalEvaluationContext**: A list of claims on which the claims transformation rules processing operates.

3. **InternalOutputContext**: A list of claims that collects the output of claims transformation rules processing.

The Claims Transformation Algorithm MUST be initialized as follows:

1. **InternalTransformationRules** MUST be initialized by clearing it.

2. **InternalEvaluationContext** MUST be initialized by clearing it and then adding all **InputClaims** to it.

3. **InternalOutputContext** MUST be initialized by clearing it.

4. **OutputClaims** MUST be initialized by clearing it.

5. **ReturnValue** MUST be set to SUCCESS.

2.1.4 Processing Rules

The Claims Transformation Algorithm is invoked by a caller by providing InputClaims and the InputTransformationRulesText as indicated in Initialization (section 2.1.3). This algorithm continues processing until an error occurs or until successful completion.

The Claims Transformation Algorithm consists of the following processing steps.

1. Parse **InputTransformationRulesText** to validate the syntax against the claims transformation rules language syntax and generate **InternalTransformationRules** (section 2.1.4.2).

2. If evaluation in the previous step fails, set **ReturnValue** to FAILURE and **OutputClaims** to an empty list and exit this algorithm.

3. Perform processing steps detailed in Claims Transformation Rules Processing (section 2.1.4.3) on **InternalEvaluationContext** using **InternalTransformationRules**.

4. If an error occurs in the previous processing, set **ReturnValue** to FAILURE and **OutputClaims** to an empty list and exit this algorithm.

5. Set **ReturnValue** to SUCCESS, copy all the claims from the **InternalOutputContext** to **OutputClaims**, and exit this algorithm.

2.1.4.1 Claims Transformation Rules Language Syntax

- The claims transformation rules language is a context-free language defined following, using tokens and **ABNF**.

2.1.4.1.1 Language Terminals

The following table lists the complete set of terminal strings and associated language **terminals** used in the claims transformation rules language. These definitions MUST be treated as case insensitive. The terminal strings MUST be encoded in UTF-16.

String	Terminal

String	Terminal
"=>"	IMPLY
";"	SEMICOLON
":"	COLON
","	COMMA
"."	DOT
"["	O_SQ_BRACKET
"]"	C_SQ_BRACKET
"("	O_BRACKET
")"	C_BRACKET
"=="	EQ
"!="	NEQ
"=~"	REGEXP_MATCH
"!~"	REGEXP_NOT_MATCH
"="	ASSIGN
"&&"	AND
"issue"	ISSUE
"type"	TYPE
"value"	VALUE
"valuetype"	VALUE_TYPE
"claim"	CLAIM
"[_A-Za-z][_A-Za-z0-9]*"	IDENTIFIER
"\"[^\"\n]*\""	STRING
"uint64"	UINT64_TYPE
"int64"	INT64_TYPE
"string"	STRING_TYPE
"boolean"	BOOLEAN_TYPE
	NULL

2.1.4.1.2 Language Syntax

The claims transformation rules language is specified here in ABNF form. This definition uses the terminals specified in the previous section as well as new ABNF **productions** defined here. The rules MUST be encoded in UTF-16. The string comparisons MUST be treated as case insensitive.

```
Rule_set               = NULL
                         / Rules
Rules                  = Rule
                         / Rule Rules
Rule                   = Rule_body
Rule_body              = (Conditions IMPLY Rule_action SEMICOLON)
Conditions             = NULL
                         / Sel_condition_list
Sel_condition_list     = Sel_condition
                         / (Sel_condition_list AND Sel_condition)
Sel_condition          = Sel_condition_body
                         / (IDENTIFIER COLON Sel_condition_body)
Sel_condition_body     = O_SQ_BRACKET Opt_cond_list C_SQ_BRACKET
Opt_cond_list          = NULL
                         / Cond_list
Cond_list              = Cond
                         / (Cond_list COMMA Cond)
Cond                   = Value_cond
                         / Type_cond
Type_cond              = TYPE Cond_oper Literal_expr
Value_cond             = (Val_cond COMMA Val_type_cond)
                         /(Val_type_cond COMMA Val_cond)
Val_cond               = VALUE Cond_oper Literal_expr
Val_type_cond          = VALUE_TYPE Cond_oper Value_type_literal
Claim_prop             = TYPE
                         / VALUE
Cond_oper              = EQ
                         / NEQ
                         / REGEXP_MATCH
                         / REGEXP_NOT_MATCH
Literal_expr           = Literal
                         / Value_type_literal

Expr                   = Literal
                         / Value_type_expr
                         / (IDENTIFIER DOT Claim_prop)
Value_type_expr        = Value_type_literal
                         /(IDENTIFIER DOT VALUE_TYPE)
Value_type_literal     = INT64_TYPE
                         / UINT64_TYPE
                         / STRING_TYPE
                         / BOOLEAN_TYPE
Literal                = STRING
Rule_action            = ISSUE O_BRACKET Issue_params C_BRACKET
Issue_params           = Claim_copy
                         / Claim_new
Claim_copy             = CLAIM ASSIGN IDENTIFIER
Claim_new              = Claim_prop_assign_list
Claim_prop_assign_list = (Claim_value_assign COMMA Claim_type_assign)
                         /(Claim_type_assign COMMA Claim_value_assign)
Claim_value_assign     = (Claim_val_assign COMMA Claim_val_type_assign)
                         /(Claim_val_type_assign COMMA Claim_val_assign)
Claim_val_assign       = VALUE ASSIGN Expr
```

```
Claim_val_type_assign  = VALUE_TYPE ASSIGN Value_type_expr
Claim_type_assign      = TYPE ASSIGN Expr
```

2.1.4.2 Claims Transformation Rules Syntax Evaluation

Syntax evaluation MUST perform the following processing:

1. **InputTransformationRulesText** MUST be validated against the ABNF syntax definition of the claims transformation rules language to ensure conformity. Any failure MUST be considered an error, **ReturnValue** MUST be set to FAILURE, and the algorithm MUST exit.
2. The following validation MUST be performed on **InputTransformationRulesText**.
    1. Each *Sel_condition* in *Sel_condition_list* either MUST use an IDENTIFIER unique among all IDENTIFIERS in the *Sel_condition_list* or MUST use no IDENTIFIER.
    2. If *Rule_action* contains one or more IDENTIFIERs, then each of the IDENTIFIERs MUST have an identical matching IDENTIFIER in the *Condition* in the same *Rule*.
    3. If either of the preceding validation steps fails, it MUST be considered an error. **ReturnValue** MUST be set to FAILURE, and the algorithm MUST exit.
3. The **InternalTransformationRules** variable MUST be populated with **InputTransformationRulesText**.
4. The **InternalTransformationRules** variable MUST be populated in a depth-first fashion with tags (**production names** and terminals) and the matching portion of **InputTransformationRulesText**.

2.1.4.3 Claims Transformation Rules Processing

Claims transformation rules processing requires the **InternalTransformationRules** variable to be populated using **InputTransformationRulesText** and requires all other variables to be initialized (see section 2.1.3 and section 2.1.4.2).

Claims transformation rules processing uses an additional variable called **InternalMatchingClaimsList** to store temporary data during processing. Each **InternalMatchingClaimsList** is a list of claims that matches a Sel_condition (section 2.1.4.3.4). **InternalMatchingClaimsList**s are created dynamically on a per- Rule (section 2.1.4.3.2) basis.

The following state diagram illustrates the logical processing flow, with error handling excluded. Any error encountered during the claims transformation rules processing MUST set **ReturnValue** to FAILURE, and the processing MUST immediately continue from the Processing End (section 2.1.4.3.10) state.

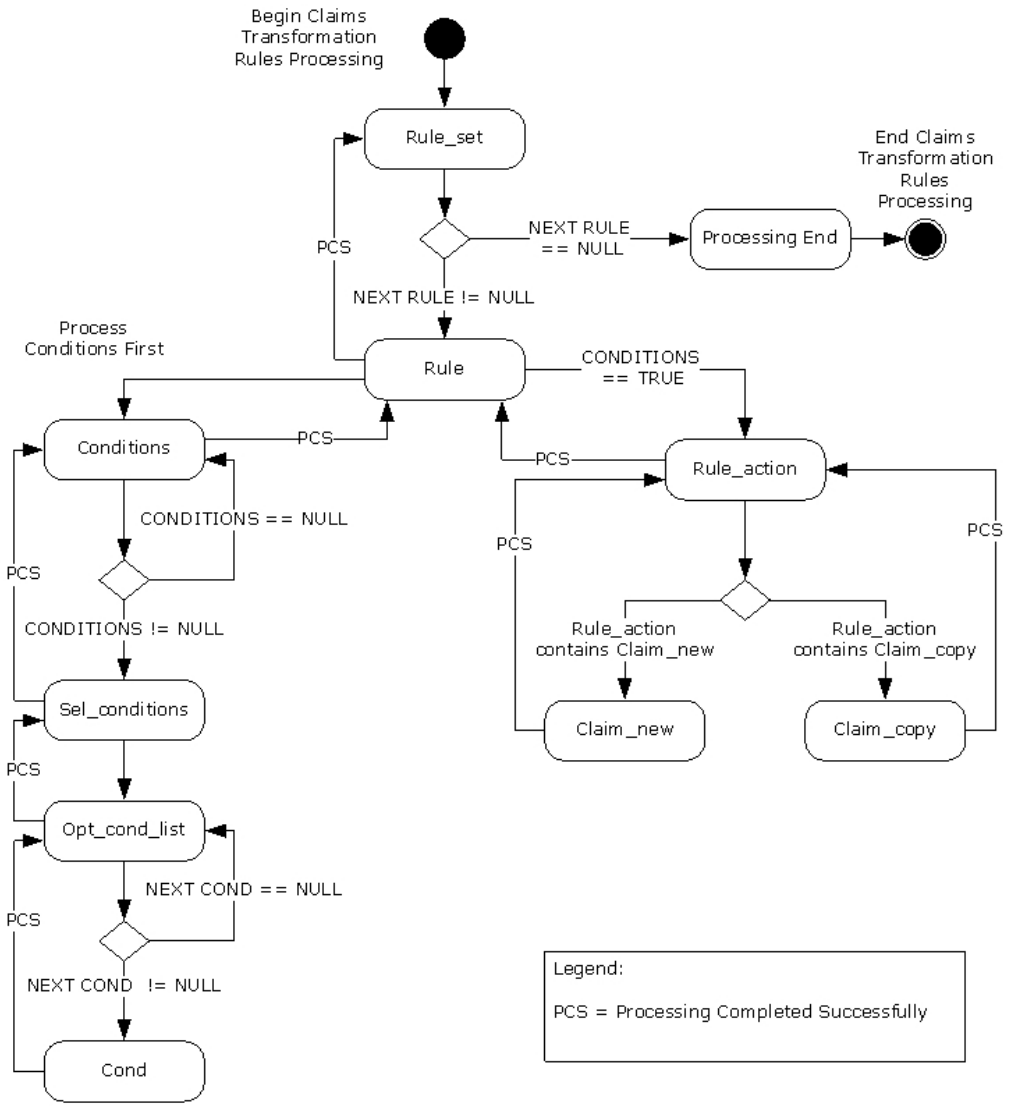

Figure 2: Claims transformation state machine

For the purposes of this section, processing is defined as **InternalTransformationRules** evaluation on the **InternalEvaluationContext** or **InternalTransformationRules** action taken using Matching Claims.

The processing MUST begin at the first tag in **InternalTransformationRules** and MUST proceed depth-first in the order in which the tags are placed.

The processing steps for the critical tags are specified in the following subsections. Those tags not listed MUST be treated as if they have no processing steps and MUST be ignored during processing.

2.1.4.3.1 Rule_set

1. Set ReturnValue to SUCCESS.
2. If the *Rule_set* is NULL, go to Processing End (section 2.1.4.3.10).

3. Process each *Rule* in the *Rule_set*.

4. Go to Processing End (section 2.1.4.3.10).

2.1.4.3.2 Rule

1. Processing a *Rule* MUST perform the necessary operations using the **InternalEvaluationContext** variable.

2. Create as many **InternalMatchingClaimsList** variables as there are *Sel_condition*s in this *Rule*, and initialize them by clearing them.

3. Process the *Conditions* tag in this *Rule*.

4. If the *Conditions* evaluates to TRUE, the *Rule_action* in this *Rule* MUST be processed using all the n-tuples of claims generated by the *Conditions.*

2.1.4.3.3 Conditions

1. This processing step MUST evaluate to TRUE or FALSE.

2. When *Conditions* evaluates to TRUE, a list of zero or more matching n-tuples of claims, where n is the number of *Sel_condition*s in the *Conditions,* MUST be returned.

3. If the *Conditions* is NULL, the processing of this production must stop and the evaluation result MUST be returned as TRUE with no entries in the matching n-tuples.

4. The following processing applies when *Conditions* is not NULL:

    1. When all *Sel_condition*s in the *Conditions* evaluate to TRUE, the *Conditions* MUST evaluate to TRUE; else the *Conditions* MUST evaluate to FALSE.

    2. Each *Sel_condition* MUST evaluate to TRUE when at least one claim in the **InternalEvaluationContext** matches it.

    3. The process of matching each *Sel_condition* MUST determine all claims in the **InternalEvaluationContext** that match it. The resulting list of matching claims MUST be stored in the **InternalMatchingClaimsList** corresponding to that *Sel_condition*.

    4. If *Conditions* evaluates to TRUE, there MUST exist an n-tuple of claims from the **InternalEvaluationContext** that matches each of the constituent "n" *Sel_condition*s. The n-tuple can contain duplicate claims; that is, one claim can match one or more *Sel_condition*s.

    5. Evaluation of *Conditions* MUST determine all possible unique n-tuples of claims from the **InternalEvaluationContext** that match each of the constituent "n" *Sel_condition*s.

    6. Return the list of n-tuples of claims.

2.1.4.3.4 Sel_condition

1. This processing step MUST fill one **InternalMatchingClaimsList** with zero or more claims from the **InternalEvaluationContext**. If an IDENTIFIER is used in the *Sel_condition*, the **InternalMatchingClaimsList** MUST be tagged by the string represented by the IDENTIFIER.

2. **InternalMatchingClaimsList** is filled by evaluating *Opt_cond_list.*

3. If the **InternalMatchingClaimsList** contains zero claims, the returned evaluation result MUST be FALSE; else it MUST be TRUE.

2.1.4.3.5 Opt_cond_list

1. If *Opt_cond_list* is NULL, the **InternalMatchingClaimsList** MUST be filled with all the claims in the **InternalEvaluationContext**. The processing of this production MUST stop, and **InternalMatchingClaimsList** must be returned as the evaluation result.

2. The following processing rules apply when *Opt_cond_list* is not NULL:

    1. The following processing MUST start from the first claim in the **InternalEvaluationContext,** and all the claims MUST be processed.

    2. If all the *Conds* in this *Opt_cond_list* evaluate to TRUE for a claim in the **InternalEvaluationContext**, the claim MUST be added to the **InternalMatchingClaimsList**.

Return the **InternalMatchingClaimsList** as the evaluation result.

2.1.4.3.6 Cond

1. This processing step MUST return TRUE if a given claim matches the current *Cond*, and FALSE otherwise.

2. The TYPE, VALUE, and VALUE_TYPE in a Cond MUST be replaced by the current claim's TYPE, VALUE, and VALUE_TYPE, respectively (section 2.1.2). The current claim's TYPE and VALUE_TYPE MUST always be treated as STRING_TYPE. The current claim's VALUE MUST be interpreted based on its VALUE_TYPE.

3. The right side of *Cond_oper* in the *Cond* MUST be convertible to the same type as the operand on the left side of the *Cond_oper*; otherwise, the *Cond* MUST return the evaluation result as FALSE. Converting STRING_TYPE variables to other types MUST be performed as specified in [ISO/IEC-9899] section 7.20.1.4.

4. The *Cond_oper* in the *Cond* MUST be interpreted based upon the type of the operand on the left side of the *Cond_oper*, as shown in the following table.

	INT64_TYPE	**UINT64_TYPE**	**BOOLEAN_TYPE**	**STRING_TYPE**
EQ	Signed integer equality comparison.	Unsigned integer equality comparison.	BOOLEAN equality comparison. Unsigned integers MUST be interpreted as BOOLEAN values as follows: 0 == FALSE (!0) == TRUE	Case-insensitive, NULL terminated Unicode-string comparison, excluding terminating NULLs for equality.
NEQ	Negation of EQ comparison.	Negation of EQ comparison.	Negation of EQ comparison.	Negation of EQ comparison.
REGEXP_MATCH	Not valid.	Not valid.	Not valid.	Regular

	INT64_TYPE	UINT64_TYPE	BOOLEAN_TYPE	STRING_TYPE
				expression match of NULL terminated Unicode strings.
REGEXP_NOT_MATCH	Not valid.	Not valid.	Not valid.	Negation of REGEXP_MATCH.

5. If the current processing encounters a *Cond_oper* and the type combination is identified as "Not Valid" in the preceding table, the processing MUST return the result of the evaluation as FALSE.

6. Return the result of the evaluation of *Cond*, comparing the operands based on interpretation of the *Cond_oper* from the preceding table.

2.1.4.3.7 Rule_action

1. Successful processing of this step MUST result in creation of one or more claims.

2. *Rule_action* acts on each of the n-tuples generated by *Conditions* in the same *Rule*.

3. If this *Rule_action* contains a *Claim_copy* sub-tag, *Claim_copy* (section 2.1.4.3.8) MUST be processed using the matching n-tuples as input and the resulting claims collected as output.

4. If this *Rule_action* contains a *Claim_new* sub-tag, *Claim_new* (section 2.1.4.3.9) MUST be processed using the matching n-tuples as input and the resulting claims collected as output.

The above processing MUST generate one or more claims. The generated claims MUST be appended to the **InternalEvaluationContext** and the **InternalOutputContext**.

2.1.4.3.8 Claim_copy

1. This processing step MUST create one claim per matching n-tuple.

2. The new claim MUST be a copy of the claim in the matching n-tuple indicated by the IDENTIFIER reference.

2.1.4.3.9 Claim_new

1. Successful processing of this step MUST create one or more claims.

2. If no matching n-tuples are presented to this processing step, the contained assignments MUST have only *Literals* and MUST NOT have any IDENTIFIER references. In this case, only one claim is generated.

3. If matching n-tuples are presented, this processing step MUST create one claim per matching n-tuple, using *Literals* and/or IDENTIFIER references to the matching n-tuple.

4. Assignments to TYPE, VALUE, and VALUE_TYPE MUST be interpreted as assignments to TYPE, VALUE and VALUE_TYPE, respectively, of each of the newly created Claims; see section 2.1.2.

5. If the *Expr* on the right side of the *ASSIGN* is a *Literal*, it MUST be interpreted based on the type on the left side of *ASSIGN.* When the left side of the *Expr* is not STRING_TYPE, the *Literal* MUST be converted in accordance with the rules specified in [ISO/IEC-9899] section 7.20.1.4. If the right side of the *Assign* is not a *Literal,* type conversion MUST NOT be performed.

6. Each newly created claim MUST adhere to the definition in section 2.1.2; else it MUST be considered invalid.

7. If any type mismatches or errors in type conversions are encountered by *ASSIGN,* or if an invalid claim is generated, processing MUST stop, and a processing error MUST be indicated.

2.1.4.3.10 Processing End

- If **ReturnValue** is set to SUCCESS, copy the claims in the **InternalOutputContext** to **OutputClaims** and exit the algorithm.
- If **ReturnValue** is set to FAILURE, clear **OutputClaims** and exit the algorithm.

3 Algorithm Examples

This section contains some examples of the Claims Transformation Algorithm.

3.1 Processing "Allow All Claims" Rule

```
Input:

InputTransformationRulesText: C1:[]=> ISSUE(Claim=C1);
InputClaims:  {(TYPE = "type1", VALUE = 5, VALUE_TYPE = "int64"),
                (TYPE = "type2", VALUE = "example", VALUE_TYPE = "string") }
Output:
OutputClaims: {(TYPE = "type1", VALUE = 5, VALUE_TYPE = "int64"),
                (TYPE = "type2", VALUE = "example", VALUE_TYPE = "string") }
ReturnValue: SUCCESS.
```

3.2 Processing "Deny Some Claims" Rule

```
Input:

InputTransformationRulesText: C1:[type != "Type1"] => ISSUE (Claim = C1);
InputClaims:  {(TYPE = "type1", VALUE = 5, VALUE_TYPE = "uint64"),
                (TYPE = "type2", VALUE = "example", VALUE_TYPE = "string"),
                (TYPE = "type3", VALUE = -33, VALUE_TYPE = "int64")}
Output:
OutputClaims: { (TYPE = "type2", VALUE = "example", VALUE_TYPE = "string"),
               (TYPE = "type3", VALUE = -33, VALUE_TYPE = "int64")}
ReturnValue: SUCCESS.
```

3.3 Processing "Issue always" Rule

```
Input:

InputTransformationRulesText: => ISSUE (type="type1", VALUE=false, VALUE_TYPE="boolean");
InputClaims:  {}

Output:
OutputClaims: {(TYPE = "type1", VALUE = false, VALUE_TYPE = "boolean")}

ReturnValue: SUCCESS.
```

3.4 Processing an Invalid Rule

```
Input:
InputTransformationRulesText: C1:[type] => ISSUE (Claim = C1);
InputClaims:  {(TYPE = "type1", VALUE = 5, VALUE_TYPE = "uint64"),
                (TYPE = "type2", VALUE = "example", VALUE_TYPE = "string")}

Output:
OutputClaims: {}
ReturnValue: FAILURE.
```

4 Security

4.1 Security Considerations for Implementers

None.

4.2 Index of Security Parameters

None.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows Server 2012 operating system
- Windows Server 2012 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1: When no operating system version information is specified, information in this document applies to Windows Server 2012 and Windows Server 2012 R2.

6 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

7 Index