

# Policy Expression

As discussed in the [previous section](#), a `policy` is an enforceable representation of some business or legal logic.

Tranquil Data policies are expressed in the [OASIS XACML 3.0](#) standard, or a simplified sub-set, as rules over context. Stable identifiers are used to reference categories of attributes, and a simple query-substitution syntax supports meta-data injection. A small number of functions, in addition to the standard set, are provided to simplify common business logic.

## Policy Syntax

The eXtensible Access Control Markup Language is an enterprise authorization open standard. It defines a syntax for encoding policy, a set of flexible evaluation semantics, and a model for decomposing attribute and policy management, decision-making, and enforcement. Tranquil Data uses all of these capabilities to make it simple to build rich, powerful rules based on context.

All datatypes, functions, and combining algorithms defined in the 3.0 specification are supported. All standard runtime capabilities, with the exception of XPath queries, are also supported.


## XML Standard Syntax


Tranquil Data has full support for the standard XML syntax. XML-encoded policies must be defined within the 3.0 standard namespace, like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
...

```

The value of the top-level `PolicyId` or `PolicySetId` attribute is the `context` entity name.


 Policy References




In the current release evaluation always uses the most recent policies associated with a `domain` or `peer`. Similarly, for policies referenced via `PolicyIdReference` or `PolicySetIdReference`, the most recent version of the referenced policy is used even if a version constraint is present. This will be addressed in a future release.

## JSON Syntax

Tranquil Data supports a subset of XACML in a JSON-encoding, capable of expressing most common types of business logic. It is designed to help focus users on the core of what they need to express. Both JSON and XML policies can be provisioned in the same deployment, and may reference each other, so that the appropriate encoding may be used for any given application. The syntax definition follows.

 Evolving Feature



The JSON definition provided here is an early version, designed to help with adoption and solicit feedback, but it is expected to change in future versions of the product.

### PolicyRoot

A JSON-encoded policy is an object with the following structure:

Field Name	Type	Required	Default
name	string	✓	
description	string		
version	<a href="#">semver</a>	✓	
priority	<code>permit</code> , <code>deny</code> , or <code>first</code>		<code>permit</code>
references	array of string		
policies	array of <a href="#">Embedded Policy</a>		

The value for `name` must be a valid URI, and is the name of this `entity`. The value for `priority` is `permit` if evaluation should stop as soon as a referenced policy has returned `permit`, `deny` if

evaluation should stop as soon as a referenced policy has returned `deny`, or `first` if evaluation should stop as soon as a referenced policy returns either `permit` or `deny`.

The value for `references` is an array of policy names, which may name one of the embedded policies included in `policies` or a policy that has been separately provisioned. An error is returned if a referenced policy needs to be evaluated but is not available.

### EmbeddedPolicy

Field Name	Type	Required	Default
name	string	✓	
description	string		
combiner	<code>or</code> , <code>or</code> and		<code>or</code>
effect	<code>permit</code> , or <code>deny</code>		<code>permit</code>
attributesMustBePresent	boolean		<code>false</code>
conditions	array of <a href="#">Expression</a>		

The value for `name` must be a valid URI. The value for `combiner` is `and` if all conditions must be `true`, or `or` if only one condition must be `true`. The value `effect` is returned if combining the conditions results in `true`, otherwise the policy is considered not applicable.

If `attributesMustBePresent` is `true` and any attribute value cannot be resolved then this policy returns an error. An error will also be returned if there is any error while evaluating the conditions.

### Expression

Field Name	Type	Required	Default
function	string	✓	
inputs	<a href="#">Input String</a> , or a mixed array of <a href="#">Expression</a> and <a href="#">Input String</a>		

The value for `function` is any standard function, or one of the functions [defined below](#).

### Input String

An Input String encodes the category, datatype, and identifier for an attribute in a single string value:

```
CATEGORY[ . (DATATYPE) ] :: IDENTIFIER
```

If the datatype is absent, the default value is `http://www.w3.org/2001/XMLSchema#string` . For instance, these two definitions reference the same attribute:

```
"some-category::some-attribute"  
"some-category.(http://www.w3.org/2001/XMLSchema#string)::some-attribute"
```

If the value of `CATEGORY` is `value` then this input is treated as an `AttributeValue` . Otherwise, this input is treated as an `AttributeDesignator` that is used to lookup values at runtime.

**AttributeValue**

This `input string` returns the string value `apple` :

```
"value::apple"
```

**AttributeDesignator**

This `input string` resolves all string values associated with the attribute named `fruit` in the category named `trees` :

```
"trees::fruit"
```

**Input Types**

Shorthand versions of the standard datatypes are provided to make input strings simpler:

Full DataType Identifier	Shorthand Identifier
<code>http://www.w3.org/2001/XMLSchema#string</code>	<code>string</code>
<code>http://www.w3.org/2001/XMLSchema#boolean</code>	<code>bool</code>
<code>http://www.w3.org/2001/XMLSchema#integer</code>	<code>int</code>
<code>http://www.w3.org/2001/XMLSchema#double</code>	<code>double</code>
<code>http://www.w3.org/2001/XMLSchema#time</code>	<code>time</code>
<code>http://www.w3.org/2001/XMLSchema#date</code>	<code>date</code>
<code>http://www.w3.org/2001/XMLSchema#dateTime</code>	<code>datetime</code>

Full Data Type Identifier	Shorthand Identifier
<code>http://www.w3.org/2001/XMLSchema#anyURI</code>	<code>uri</code> or <code>anyURI</code>
<code>http://www.w3.org/2001/XMLSchema#hexBinary</code>	<code>hex</code>
<code>http://www.w3.org/2001/XMLSchema#base64Binary</code>	<code>base64</code>
<code>http://www.w3.org/2001/XMLSchema#dayTimeDuration</code>	<code>daytime</code>
<code>http://www.w3.org/2001/XMLSchema#yearMonthDuration</code>	<code>yearmonth</code>
<code>urn:oasis:names:tc:xacml:1.0:data-type:x500Name</code>	<code>x500</code>
<code>urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name</code>	<code>email</code>
<code>urn:oasis:names:tc:xacml:2.0:data-type:ipAddress</code>	<code>address</code>
<code>urn:oasis:names:tc:xacml:2.0:data-type:dnsName</code>	<code>dns</code>

## Category Identifiers

In the XACML evaluation model, when an attribute value is resolved it is associated with a *category*. The standard allows for arbitrary categories. In a Tranquil Data policy, there are specific categories defined to simplify how context and environment values are used.

## Context Categories

The following categories define how *context values* are referenced in policy with Attribute Designators.

Category Identifier	Description
<code>urn:tranquildata:category:record</code>	context about the current record being read or written
<code>urn:tranquildata:category:session</code>	context about the current session
<code>urn:tranquildata:category:peer</code>	context about the local peer where policy evaluation is happening
<code>urn:tranquildata:category:origin</code>	context about the peer where the current record was created
<code>urn:tranquildata:category:subject:query</code>	context about the subject querying a datastore
<code>urn:tranquildata:category:subject:resource</code>	context about the subject associated with the current record

Category Identifier	Description
urn:tranquildata:category:group	context about a specific named group

The `record`, `session`, `peer`, and `origin` categories each expect a single property name. The `subject` categories both expect a [subject query](#) as the attribute identifier. The `group` category expects a [group query](#) as the attribute identifier.

As an example, here is the query subject's name being resolved by an Attribute Designator (XML) or an Input String (JSON):

XML

```
<AttributeDesignator Category="urn:tranquildata:category:subject:query"
  AttributeId="property:name"
  DataType="http://www.w3.org/2001/XMLSchema#string" />
```

JSON

```
"urn:tranquildata:category:subject:query::property:name"
```

Environment Categories

The following categories define how *environment values* are referenced by policies.

Category Identifier	Description
urn:tranquildata:category:metadata	any values provided as request metadata
urn:oasis:names:tc:xacml:3.0:attribute-category:environment	standard identifiers defined in the XACML specification

Request Metadata is the set of attributes provided to evaluation outside of context. These can come from a combination of properties on export configuration, headers in a query, and values from the caller's security token. Arbitrary values may be injected into evaluation as request metadata, but typically these are values like the name or group of a caller, the purpose of the request, or the locale where the result will be interpreted.

The XACML standard defines the [environment category](#). It provides, under the `urn:oasis:names:tc:xacml:1.0:environment` namespace, attributes named `current-time`, `current-date`, and `current-dateTime`. These return, respectively, values of type `http://www.w3.org/2001/XMLSchema#time`, `http://www.w3.org/2001/XMLSchema#date`, and

`http://www.w3.org/2001/XMLSchema#dateTime` . These values will always be stable over the evaluation of a policy.

As an example, here is the current date being resolved by an Attribute Designator (XML) or an Input String (JSON):

XML

```
<AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:environment"
  AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
  DataType="http://www.w3.org/2001/XMLSchema#date" />
```

JSON

```
"urn:oasis:names:tc:xacml:3.0:attribute-category:environment.
(date)::urn:oasis:names:tc:xacml:1.0:environment:current-date"
```

Evaluation Attributes

The following attributes are injected into the environment by the engine for use in policy:

Attribute Identifier	Type	Category	Description
<code>urn:tranquildata:attribute:categories:requested</code>	bag of string	metadata	all category identifiers that cover the fields of the requested record
<code>urn:tranquildata:attribute:score</code>	int	record or session	based on the category identifier, either the score for the requested record or the sum of all record scores in the session
<code>urn:tranquildata:attribute:resource-subjects</code>	bag of string	session	all subject identifiers for records that have been allowed in the current session
<code>urn:tranquildata:attribute:retention</code>	yearmonth	record	the retention period for the requested record
<code>urn:tranquildata:attribute:expired</code>	bool	record	whether the requested record is considered to have "expired" based on its retention

Obligations and Advice

Policy may return Obligations and Advice (collectively referred to in Tranquil Data as "Qualifiers") the instruct the engine to take specific action as a result of the decision. To notify the engine of these qualifiers, use `urn:tranquildata:obligation` for the `ObligationId` or `urn:tranquildata:advice` for the `AdviceId`.

If a policy returns an Advice with the `AttributeId` `urn:tranquildata:attribute:message:summary` it is interpreted as a summary that explains why a given decision was made. The string form of this will be included in the decision trace, and made available to other components that return details via API.

If a policy returns an Obligation with the `AttributeId`

`urn:tranquildata:attribute:categories:allowed` it is interpreted as the set of category identifiers that should be allowed for the request. This set is included in-context and in the change graph. If the engine is running in `redact` mode, then this will also be used to remove any fields from the record that are not associated with an allowed category.

## Identifier Substitution

Anywhere within an Attribute Designator's Identifier field a substitution may be defined, and substitutions may be nested. Substitutions take the form of `$(TYPE.IDENTIFIER)` and are used to form an identifier based on attributes available in the evaluation context. Valid values for `TYPE` are `metadata`, `resource`, and `group`.

If the `TYPE` is `metadata` then `IDENTIFIER` should simply be the identifier of an attribute in the metadata category.

If the `TYPE` is `resource` then this substitution is supplying a property of the request subject or one of their relationships. To substitute a property of the request subject, the value of `IDENTIFIER` should simply be the name of that property (which may include the standard property `name` to return the name of the subject). To substitute a property from one of the subject's relationships, the value of `IDENTIFIER` should be of the form `RELATIONSHIP-TYPE.PROPERTY`. If the subject has any relationship of type `RELATIONSHIP-TYPE` then this substitution will look for the requested property on that relationship.

If the `TYPE` is `group` then this substitution is supplying a property of a group or one of its members.

If an attribute contains a substitution with invalid syntax, if a required subject or group is unknown, or if the substitution resolves multiple values then an error is returned from the associated Attribute Designator. If the requested property is absent or empty then the associated Attribute Designator immediately returns an empty bag. Otherwise, the string form of the resolved property is inserted



into the Attribute Identifier string. When all substitutions in the identifier string have been resolved, the associated Attribute Designator uses the identifier to resolve the attribute.

As an example, suppose requestors are authenticated with tokens that include the property "id". This can be substituted into an Attribute Designator:

XML

```
<AttributeDesignator Category="urn:tranquildata:category:subject:resource"
  AttributeId="type:family:target:$(metadata.id):property:is-guardian"
  DataType="http://www.w3.org/2001/XMLSchema#boolean" />
```

JSON

```
"urn:tranquildata:category:subject:resource.
(bool)::type:family:target:$(metadata.id):property:is-guardian"
```

If the required metadata value is not present then an empty bag of boolean is returned. Otherwise, the value for "id" is substituted into the resource subject query, and the engine will attempt to resolve whether the subject has a "family" relationship with the identified requestor, and whether that relationship has the property "is-guardian".

Functions

In addition to supporting the standard XACML 3.0 functions, Tranquil Data defines the following:

Function Identifier	Input	Return
urn:tranquildata:function:consistent	two or more attributes of the same type	http://www.w3.org/2001/XMLSchema#boolean
urn:tranquildata:function:contains	a single attribute of any type	http://www.w3.org/2001/XMLSchema#boolean
urn:tranquildata:function:absent	a single attribute of any type	http://www.w3.org/2001/XMLSchema#boolean

The `consistent` function tests that a set of attributes do not result in inconsistent context state. It reeturns true if none of the inputs exists, if only one exists, or if all inputs have the same value. This function is typically used to test that any attributes of a record under evaluation are *consistent* with past record values captured in the current session (see the example below).

The `contains` and `absent` functions are related tests for the existence of a given attribute. They are typically used to check if a given property, relationship, or member exists in-context.

## Example

Here is a policy that enforces the common requirement for tenant isolation by ensuring that all records in a session are associated with the same user based on a field-value from the record.

### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  PolicyId="tenant-isolation" Version="1.0"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
algorithm:permit-overrides">
  <Description>enforce tenant isolation on the user property</Description>
  <Rule RuleId="isolate-on-user" Effect="Permit">
    <Target/>
    <Condition>
      <Apply FunctionId="urn:tranquildata:function:consistent">
        <AttributeDesignator Category="urn:tranquildata:category:record"
AttributeId="user"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        <AttributeDesignator Category="urn:tranquildata:category:session"
AttributeId="user"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
    </Condition>
  </Rule>
</Policy>
```

### JSON

```
{
  "name": "tenant-isolation",
  "description": "enforce tenant isolation on the user property",
  "version": "1.0",
  "priority": "permit",
  "references": [
    "isolate-on-user"
  ],
  "policies": [
    {
      "name": "isolate-on-user",
      "effect": "permit",
      "conditions": [
        {
          "function": "urn:tranquildata:function:consistent",
          "inputs": [
            "urn:tranquildata:category:record::user",
            "urn:tranquildata:category:session::user"
          ]
        }
      ]
    }
  ]
}
```

```
}  
  ]  
    }  
      ]
```