<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Relationship Between the DAC, OS-MAC, and ABAC Policies</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Applied, Inherited, and Associated ABAC Security Policies</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Controlled Objects and Operations</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Decision Request, Policy Evaluation, and Outcome</td>
<td>5</td>
</tr>
<tr>
<td>1.4.1 Operational Behavior Given an Outcome</td>
<td>6</td>
</tr>
<tr>
<td>1.5 Obligations – Actions Taken Upon Policy Decision</td>
<td>6</td>
</tr>
<tr>
<td>1.6 Security Markup Language (RXSML) Overview</td>
<td>6</td>
</tr>
<tr>
<td>1.6.1 Attribute Markup Values</td>
<td>7</td>
</tr>
<tr>
<td>1.6.1.1 Attribute Bags</td>
<td>7</td>
</tr>
<tr>
<td>1.6.1.2 Context Attributes</td>
<td>8</td>
</tr>
<tr>
<td>1.6.1.3 Row Fields as Context Attributes</td>
<td>8</td>
</tr>
<tr>
<td>1.6.1.4 Examples</td>
<td>9</td>
</tr>
<tr>
<td>1.6.2 Functions</td>
<td>9</td>
</tr>
<tr>
<td>1.6.2.1 Examples</td>
<td>10</td>
</tr>
<tr>
<td>1.6.3 Target</td>
<td>10</td>
</tr>
<tr>
<td>1.6.3.1 Target Evaluation</td>
<td>11</td>
</tr>
<tr>
<td>1.6.3.2 Examples</td>
<td>14</td>
</tr>
<tr>
<td>1.6.4 Rule</td>
<td>17</td>
</tr>
<tr>
<td>1.6.4.1 Rule Evaluation</td>
<td>17</td>
</tr>
<tr>
<td>1.6.4.2 Example</td>
<td>18</td>
</tr>
<tr>
<td>1.6.5 Policy</td>
<td>18</td>
</tr>
<tr>
<td>1.6.5.1 Policy Evaluation</td>
<td>19</td>
</tr>
<tr>
<td>1.6.5.2 Example</td>
<td>21</td>
</tr>
<tr>
<td>1.6.6 Policy Set</td>
<td>23</td>
</tr>
<tr>
<td>1.6.6.1 Policy Set Evaluation</td>
<td>23</td>
</tr>
<tr>
<td>1.6.6.2 Example</td>
<td>24</td>
</tr>
<tr>
<td>2 Security Markup Language (RXSML) Policy Files</td>
<td>26</td>
</tr>
<tr>
<td>2.1 Creating and Editing RXSML Policy Files</td>
<td>26</td>
</tr>
<tr>
<td>2.2 Adding RXSML Policy Files to the Trusted Policy Repository</td>
<td>26</td>
</tr>
<tr>
<td>2.3 Applying RXSML Files to RDBMS Objects</td>
<td>27</td>
</tr>
<tr>
<td>2.4 The rxpolman Command</td>
<td>28</td>
</tr>
<tr>
<td>3 Security Markup Language (RXSML)</td>
<td>31</td>
</tr>
<tr>
<td>3.1 XML Elements</td>
<td>31</td>
</tr>
</tbody>
</table>
3.1.1 <PolicySet> .......................................................... 31
3.1.2 <Description> ...................................................... 32
3.1.3 <Target> ............................................................ 32
3.1.4 <Subjects> .......................................................... 33
3.1.5 <Subject> ........................................................... 33
3.1.6 <SubjectMatch> .................................................... 34
3.1.7 <Resources> ......................................................... 34
3.1.8 <Resource> ........................................................ 34
3.1.9 <ResourceMatch> .................................................. 35
3.1.10 <Actions> .......................................................... 35
3.1.11 <Action> ........................................................... 35
3.1.12 <ActionMatch> ..................................................... 36
3.1.13 <Environments> ................................................... 36
3.1.14 <Environment> .................................................... 36
3.1.15 <EnvironmentMatch> ............................................. 37
3.1.16 <PolicySetIdReference> ......................................... 37
3.1.17 <PolicyIdReference> .............................................. 37
3.1.18 <Policy> ........................................................... 38
3.1.19 <Rule> .............................................................. 39
3.1.20 <VariableDefinition> ............................................. 40
3.1.21 <VariableReference> .............................................. 41
3.1.22 <Condition> ......................................................... 41
3.1.23 <Apply> ............................................................. 41
3.1.24 <Function> ........................................................ 42
3.1.25 <SubjectAttributeDesignator> ................................. 43
3.1.26 <ResourceAttributeDesignator> ............................... 43
3.1.27 <ActionAttributeDesignator> .................................. 44
3.1.28 <EnvironmentAttributeDesignator> .......................... 44
3.1.29 <AttributeSelector> .............................................. 45
3.1.30 <AttributeValue> .................................................. 46
3.1.31 <Obligations> ....................................................... 46
3.1.32 <Obligation> ......................................................... 46
3.1.33 <DataTypeDesignator> ........................................... 47
3.1.34 <ImportColumnSelector> ...................................... 47
3.1.35 <ImportFieldSelector> ......................................... 48
3.1.36 <ErrorCode> ....................................................... 49

3.2 Functions .................................................................. 49
3.2.1 equal ................................................................... 50
3.2.2 not-equal ............................................................. 50
3.2.3 greater-than ........................................................ 50
3.2.4 greater-than-or-equal .............................................. 50
3.2.5 less-than .............................................................. 50
3.2.6 less-than-or-equal .................................................. 51
3.3  Data Types ................................................................. 62
  3.3.1  Relationship Between RXSML and SQL Data Types .......................... 62
    3.3.1.1 Converting from SQL to RXSML Data Types .................................. 63
    3.3.1.2 Converting from RXSML to SQL data types ..................................... 64
  3.3.2  string ............................................................... 67
3.3.3 boolean.................................................................................................................. 67
3.3.4 integer...................................................................................................................... 68
3.3.5 double...................................................................................................................... 68
3.3.6 decimal...................................................................................................................... 68
3.3.7 date........................................................................................................................... 68
3.3.8 time........................................................................................................................... 68
3.3.9 dateTime..................................................................................................................... 69
3.3.10 dayTimeDuration .................................................................................................... 69
3.3.11 yearMonthDuration................................................................................................. 70
3.3.12 rfc822Name ......................................................................................................... 70
3.3.13 hexBinary .............................................................................................................. 70
3.3.14 base64Binary........................................................................................................... 70
3.3.15 label ......................................................................................................................... 71
3.3.16 ipAddress ............................................................................................................... 71
3.3.17 dnsName ............................................................................................................... 71

3.4 Context Attributes....................................................................................................... 72
3.4.1 Subject Attributes .................................................................................................... 72
  3.4.1.1 subject-id .......................................................................................................... 72
  3.4.1.2 subject-name ...................................................................................................... 72
  3.4.1.3 group-id ........................................................................................................... 73
  3.4.1.4 group-name ...................................................................................................... 73
  3.4.1.5 session-start-time ............................................................................................... 73
  3.4.1.6 session-start-date ............................................................................................... 73
  3.4.1.7 session-start-dateTime ....................................................................................... 73
  3.4.1.8 ip-address ......................................................................................................... 73
  3.4.1.9 dns-name ......................................................................................................... 73
  3.4.1.10 session-label ................................................................................................... 73
  3.4.1.11 application-name ............................................................................................. 74
  3.4.1.12 application-user-name .................................................................................... 74
  3.4.1.13 application-user-id ......................................................................................... 74
3.4.2 Resource Attributes .................................................................................................. 74
  3.4.2.1 resource-label ................................................................................................... 74
  3.4.2.2 resource-name .................................................................................................. 75
  3.4.2.3 row-label ......................................................................................................... 75
  3.4.2.4 table-label ...................................................................................................... 75
  3.4.2.5 view-label ....................................................................................................... 76
  3.4.2.6 schema-label ................................................................................................... 76
  3.4.2.7 catalog-label .................................................................................................... 76
  3.4.2.8 database-label ................................................................................................. 76
  3.4.2.9 column-name .................................................................................................. 76
  3.4.2.10 table-name .................................................................................................... 76
  3.4.2.11 view-name ..................................................................................................... 77
  3.4.2.12 schema-name ................................................................................................ 77
  3.4.2.13 catalog-name ................................................................................................. 77
  3.4.2.14 database-name ............................................................................................... 78
3.4.3 Action Attributes......................................................................................................... 78
  3.4.3.1 action-id ........................................................................................................... 78
  3.4.3.2 action-type ....................................................................................................... 78
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.4.1</td>
<td><code>current-time</code></td>
<td>78</td>
</tr>
<tr>
<td>3.4.4.2</td>
<td><code>current-date</code></td>
<td>79</td>
</tr>
<tr>
<td>3.4.4.3</td>
<td><code>current-dateTime</code></td>
<td>79</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Selecting Fields of the Current Row</td>
<td>79</td>
</tr>
<tr>
<td>3.4.6</td>
<td>Importing Column Fields</td>
<td>80</td>
</tr>
<tr>
<td>3.5</td>
<td>Obligations</td>
<td>82</td>
</tr>
<tr>
<td>3.5.1</td>
<td><code>set-field</code></td>
<td>82</td>
</tr>
<tr>
<td>3.5.2</td>
<td><code>set-error-code</code></td>
<td>86</td>
</tr>
<tr>
<td>3.5.3</td>
<td><code>audit</code></td>
<td>87</td>
</tr>
<tr>
<td>3.6</td>
<td>Rule and Policy Combining Algorithms</td>
<td>89</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Rule Combining Algorithms</td>
<td>89</td>
</tr>
<tr>
<td>3.6.1.1</td>
<td><code>deny-overrides rule-combining algorithm</code></td>
<td>89</td>
</tr>
<tr>
<td>3.6.1.2</td>
<td><code>ordered-deny-overrides rule-combining algorithm</code></td>
<td>90</td>
</tr>
<tr>
<td>3.6.1.3</td>
<td><code>permit-overrides rule-combining algorithm</code></td>
<td>90</td>
</tr>
<tr>
<td>3.6.1.4</td>
<td><code>ordered-permit-overrides rule-combining algorithm</code></td>
<td>91</td>
</tr>
<tr>
<td>3.6.1.5</td>
<td><code>first-applicable rule-combining algorithm</code></td>
<td>91</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Policy Combining Algorithms</td>
<td>91</td>
</tr>
<tr>
<td>3.6.2.1</td>
<td><code>deny-overrides policy-combining algorithm</code></td>
<td>91</td>
</tr>
<tr>
<td>3.6.2.2</td>
<td><code>ordered-deny-overrides policy-combining algorithm</code></td>
<td>92</td>
</tr>
<tr>
<td>3.6.2.3</td>
<td><code>permit-overrides policy-combining algorithm</code></td>
<td>92</td>
</tr>
<tr>
<td>3.6.2.4</td>
<td><code>ordered-permit-overrides policy-combining algorithm</code></td>
<td>92</td>
</tr>
<tr>
<td>3.6.2.5</td>
<td><code>only-one-applicable policy-combining algorithm</code></td>
<td>93</td>
</tr>
<tr>
<td>3.6.2.6</td>
<td><code>first-applicable policy-combining algorithm</code></td>
<td>93</td>
</tr>
<tr>
<td>3.7</td>
<td>Indeterminate Execution and System Error Handling</td>
<td>93</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Indeterminate Execution</td>
<td>94</td>
</tr>
<tr>
<td>3.7.2</td>
<td>System Error</td>
<td>94</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

The Trusted Rubix Security Policy Manager (SPM) is a mechanism to enforce flexible and dynamic Attribute Based Access Control (ABAC) security policies during the operation of the TR Relation Database Management System (RDBMS). Security policies are created using the XML based Trusted RUBIX Security Markup Language (RXSML). The RXSML language allows policy creation and execution using a host of context attributes and functions to manipulate them. The RXSML language also allows actions to be executed based upon the outcome of the security policy execution. Policies may be configured to release information across any domain defined by the Mandatory Access Control (MAC) security policy of the host operating system (OS-MAC). For example, with an SELinux host, an SPM policy may be configured for very specific cases to permit DBMS operations that the operating system’s Multilevel Secure (MLS) or Type Enforcement (TE) policies would otherwise deny.

The RXSML language is based upon the policy language of the OASIS XACML 2.0 standard. The attributes used to write policy logic are:

- typed (e.g., string, integer) and are categorized as subject attributes (e.g., subject name, subject IP address),
- resource attributes (e.g., object name, object label, row values),
- action attributes (e.g., operation, operation category), and environment attributes (e.g., system date and time).

The functions used to manipulate attribute values are categorized as:

- logic functions (e.g., and, or),
- comparative functions (e.g., equal, greater than),
- conversion functions (e.g., cast, convert to lower case),
- group-of-values functions (e.g., testing if a value is in a group of values), and
- set functions (e.g., intersection, union).

Access control logic code is organized into rules, policies, and sets of policies and algorithms may be specified to define how they interact with each other. Policies and policy sets may be referenced by name allowing for the elegant, modular design of complex policy logic and the reuse of policy logic without code duplication. Policies are assigned to RDBMS objects and may be specified to protect a single object or an entire subtree of objects. Policies may also be configured to automatically protect newly created objects.

The Trusted RUBIX RDBMS integrates the host operating system’s Mandatory Access Control policies (OS-MAC) to control access to database objects. Examples of an OS-MAC policy include Multilevel Security (OS-MAC MLS) and Type Enforcement (OS-MAC TE) of the SELinux Red Hat Enterprise Linux (RHEL) operating system. SPM policies may be configured to override the underlying OS-MAC policy (i.e., a releasability policy) or to further restrict operations beyond the OS-MAC policy (i.e., a refining policy). Objects that have no ABAC policy associated with them are by default protected by the underlying OS-MAC security policy.

In addition to permitting or denying a database operation on an object, RXSML may be used to define actions that are conditionally taken based upon policy decisions. Possible actions are to set the value of a specific row field during a row-based operation, to change the default behavior when an operation is denied, and to write a customizable audit record.
1.1 Relationship Between the DAC, OS-MAC, and ABAC Policies

**Trusted Rubix** has three distinct security policies:

- the Discretionary Access Control policy (DAC),
- the Attribute Based Access Control policy (ABAC),
- and the operating system's Mandatory Access Control policy (OS-MAC).

The DAC policy is characterized by an access control list (ACL) that is used to permit or deny operations by users on an object. The policy is discretionary in that the ACL is configurable by the owner of the object. Access or the ability to grant and revoke access may be removed or passed to other users by the owner. The DAC policy is configured through SQL commands by users connecting through typical database client programs. Changes to the DAC policy are transactionally controlled by the RDBMS system.

The DAC security policy is completely independent of the OS-MAC and ABAC security policies. That is, it must always allow access to an object for an operation to execute regardless of the behavior of the ABAC and the OS-MAC policies. Also, the DAC policy can in no way override the ABAC or OS-MAC policies.

There are two types of OS-MAC policies, Multilevel Security (OS-MAC MLS) and Type Enforcement (OS-MAC TE). All **Trusted Rubix** platforms support the OS-MAC MLS policy. Only SELinux enabled platforms (e.g., Red Hat Enterprise Linux) support the OS-MAC TE policy.

The OS-MAC MLS policy is characterized by automatically associating security labels with objects and user sessions and then arbitrating access based upon those labels. The relationship between the labels is statically defined within the trusted operating system as a partially ordered list. When comparing two labels one may strictly dominate (be greater than), be strictly dominated by (be less than), be equal to, or be incomparable with the other. When an object is created it is automatically labeled with the user’s session label. Update and delete operations are limited to objects whose label equals the session label. Read operations are limited to objects whose label is strictly dominated by or equal to the session label. When a read operation occurs and the session label does not strictly dominate or equal the object’s label the object is “filtered” from the user’s view. The OS-MAC policy is mandatory in that the labeling of objects and the label based restrictions on operations is always in effect. Normal users have no ability modify or disable the policy. Administrative users may have the ability to supersede the policy in tightly controlled ways for certain operations according to their set of authorizations. The OS-MAC policy is configured once during system initialization.

The OS-MAC TE policy is a component of SELinux. SELinux is a security policy enforcement mechanism integrated into the Linux operating system. It is based upon the Flask security model. The SELinux security model assigns every Linux object (file, directory, socket, process, etc.) an object class and a set of operations, also called permissions, on the object class. The model assigns a type to each instantiated subject and object. The type assigned to each instantiated subject (e.g., process) is generally referred to as a domain or a domain type. The heart of the SELinux mechanism is a set of rules that define which operations a subject with a specific domain may perform given the target object's class and type. The enforcement of these rules is known as Type Enforcement (TE). The sum of these rules basically constitutes a large access control list (ACL) for object class operations, domains, and object types. Each element in the ACL would indicate, with a permit or deny, if a subject with the given domain is able to perform the operation upon an object with the given object class and type. SELinux denies an operation unless there is a specific TE rule permitting it. In addition to TE rules that allow or deny an operation, rules also exist that determine how types are assigned to subjects and objects (i.e., how subjects and objects are labeled).
The ABAC policy is characterized by policy decisions based upon XML access control logic that uses context attributes as inputs. There is a rich set of context attributes, logical language constructs, and attribute manipulation functions available. Discrete policies may be constructed using the RXSML language and associated with individual RDBMS objects or groups of objects. Each policy may be unique allowing for very different policy behavior for each object. The ABAC policy is non-discretionary in that the owner of an object has no special abilities with regard to the policy behavior. Also, there are no authorizations that allow an administrative user to supersede the ABAC policy. If it is desired to give an administrative user special access, that logic would be written into the policy itself using the RXSML language. The ABAC policy also allows actions to be performed, such as auditing, based upon policy logic. The ABAC policy is configured by security administrators who create, edit, and assign security policies to objects using administrative tools. Policies may be changed in real time by a security administrator.

The ABAC and OS-MAC policies interact with each other in special ways. An ABAC policy may be configured to override the OS-MAC policy. In this case the OS-MAC policy is disabled for that particular operation and the ABAC policy is relied upon to control access. This allows policy to be constructed that allows information flows beyond those allowed by the OS-MAC policy. These are known as releasability policies. A special function (MAC-check) is provided to easily calculate and duplicate the behavior of the OS-MAC policy. If a policy is not configured to override the OS-MAC policy then the ABAC may further restrict operations beyond the OS-MAC policy. This is known as a refining policy and is the default policy behavior. In this case both the ABAC policy and the OS-MAC policy must allow an operation for it to be permitted. During read based operations (e.g., object open and row selects) the ABAC policy operates upon the set of objects that have been filtered by the OS-MAC policy. That is, objects that have been filtered by the OS-MAC policy will appear not to exist when the ABAC policy is evaluated. If an object has no associated ABAC policy then the OS-MAC policy is automatically enforced.

1.2 Applied, Inherited, and Associated ABAC Security Policies

The Trusted Rubix RDBMS organizes its database object in an inverted tree structure. The database is the root of the tree and contains catalogs. Catalogs contain schemata. Schemata contain tables and views. Other database objects exist but are not relevant to this discussion.

ABAC security policies consist of operating system files containing Trusted RUBIX Security Markup Language (RXSML) XML code. The RXSML code defines which operations are permissible according to the defined rules. A single ABAC policy may be applied to any number of named databases, catalogs, schemata, tables, or views. Applying a policy to RDBMS objects is accomplished using the rxpolman administrative command. While a single ABAC policy may be applied to multiple RDBMS objects, only a single ABAC policy may be assigned to a given object. That is, there is a one-to-many relationship between assigned ABAC security policies and RDBMS objects.

A policy that has been applied to a database, catalog, or schema may be inherited by subordinate objects that have no applied policy. Specification of the inheritability of a policy is achieved using the XML attribute `Scope`, which may be set to Node or Subtree. Node means the policy is not inheritable and Subtree means it may be inherited by the subtree rooted at the object the policy is assigned to. A RDBMS object that has no applied policy will inherit the security policy of the closest ancestor containing a Scope attribute of Subtree. A RDBMS object may only inherit a single ABAC policy and will never inherit a policy if it has a policy assigned to it.
The policy that is evaluated when operations occur on a RDBMS object is called the associated policy for the given object. The policy associated with a RDBMS object is the applied policy, if it exists; otherwise, it is the inherited policy, if it exists. If there is no applied or inherited policy the object has no associated policy. At most there may be one ABAC security policy associated with an object while a single ABAC security policy may be associated with multiple RDBMS objects. That is, there is a one-to-many relationship between associated ABAC security policies and RDBMS objects. It should be noted that it is possible that an object has no associated ABAC security policy, in which case only the DAC and MAC security policies will be enforced. That is, when no associated ABAC security policy is found for an object the operation implicitly passes the ABAC security check.

The purpose in the distinction between applied and inherited policy is to allow a single policy to be associated with an entire subtree of objects without the need to maintain multiple policy files. It also allows default policy to be defined for new objects before the new objects are created. The result is a reduced administrative burden on the policy administrator and an elegant, natural way to associate ABAC security policy with subsets of the database objects.

It is possible to duplicate the logic of assigned and associated policies using the RXSML target construct. The target may be used to define a set of objects for which the policy should be evaluated. For example, it may be possible to create a single inheritable policy that has ABAC security rules for all objects in the database. This policy may then be assigned to the root database object. While the security behavior may be identical to creating individual policies specific to RDBMS objects, it is generally recommended to apply individual policies to specific objects where possible for increased performance and simplicity of policy logic.

### 1.3 Controlled Objects and Operations

The ABAC security policy is checked when a controlled operation is initiated on a controlled RDBMS object. At that time the associated policy is evaluated against the current context. If there is no associated policy for an object at the time an operation is initiated then only the DAC and MAC security policies are checked. That is, when no associated ABAC security policy is found for an object the operation implicitly passes the ABAC security check. However, if there is an associated policy for an object then it must explicitly allow (according to the policy rules) all operations that are expected to pass the ABAC security check.

The following table lists the set of controlled database objects and their associated controlled operations.

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>OPERATION</th>
<th>OBJECT</th>
<th>OPERATION</th>
<th>OBJECT</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>row-select</td>
<td>View</td>
<td>view-create</td>
<td>Catalog</td>
<td>catalog-open</td>
</tr>
<tr>
<td></td>
<td>row-insert</td>
<td></td>
<td>view-drop</td>
<td></td>
<td>catalog-create</td>
</tr>
<tr>
<td></td>
<td>row-update</td>
<td></td>
<td>view-open</td>
<td></td>
<td>catalog-drop</td>
</tr>
<tr>
<td></td>
<td>row-delete</td>
<td></td>
<td>schema-open</td>
<td></td>
<td>database-open</td>
</tr>
<tr>
<td></td>
<td>index-create</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>index-drop</td>
<td>Schema</td>
<td>schema-create</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>table-open</td>
<td></td>
<td>schema-drop</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>table-create</td>
<td></td>
<td></td>
<td></td>
<td>database-drop</td>
</tr>
</tbody>
</table>
In order for an SQL operation to succeed, all required controlled operations must be permitted according to the security policy. For instance, assume a SQL SELECT operation is submitted on a view identified as `db.cat1.sch1.vw` and that the view references table `db.cat2.sch2.tab`. In order for the SELECT to succeed the following operations must be allowed according to the policies associated with each object:

- `database-open` on `db`
- `catalog-open` on `db.cat1` and `db.cat2`
- `schema-open` on `db.cat1.sch1` and `db.cat2.sch2`
- `view-open` on `db.cat1.sch1.vw`
- `table-open` on `db.cat2.sch2.tab`
- `row-select` for each row of `db.cat2.sch2.tab` accessed by the SELECT operation

Policy written to restrict access to rows within a table will also restrict access to those rows for any view that references the table.

Any operation on a database, catalog, schema, table, or view object requires that the object first be opened. Therefore, policy control on the `-open` operations provides an elegant and efficient way to restrict access to objects or an entire subtree of objects.

During row based operations, the policy is evaluated for each row accessed. That is, a separate decision request is made for each row. This allows the field values of the individual rows to be used in reaching policy decisions.

Operations are classified by type as `read`, `create`, `update`, and `drop` and policy rules may be written according to the operation’s type. This allows policy rules to be easily created that arbitrate access for an entire class of operations. For instance, a user can be given read-only access to objects by explicitly allowing an operation if its type is `read`.

1.4 Decision Request, Policy Evaluation, and Outcome

When a controlled operation is initiated on a controlled object a decision request is made.

A decision request is the process of querying the SPM’s policy engine to make an access control decision for the current operation on a RDBMS object. When a decision request is made the policy associated with the current RDBMS object is located. If no such policy exists then the decision request implicitly allows the operation. The associated policy is evaluated against the current operational context and an outcome is reached.

Policy evaluation is the process by which the policy logic is applied using the current context (i.e., attribute values) to reach an access control outcome.

The outcome of a decision request determines if the operation is allowed or is disallowed. The outcome may be Permit, Deny, Not Applicable, or Indeterminate.

An outcome of Not Applicable indicates that an associated policy exists for the object but the policy logic indicates there are no applicable rules for the current context. An outcome of Indeterminate indicates the
policy logic was not able to execute due to programmatic errors. In both cases the policy code should be modified to prevent such outcomes.

A decision request is made for all controlled operations. For row-based operations a decision request is made for each row that is accessed. This allows the attribute values (e.g., field values) of each row to be used in the policy evaluation logic on a row-by-row basis.

1.4.1 Operational Behavior Given an Outcome

The outcome of a decision request may be Permit, Deny, Not Applicable, or Indeterminate. Only an outcome of Permit allows the operation to proceed. For all operations an outcome of Not Applicable and Indeterminate result in the operation being disallowed and an appropriate SQL error code being returned.

An outcome of Deny results in behavior that is specific to the operation being performed. For the row-select operation the current row is filtered from the result set. That is, the select operation proceeds as though the row did not exist. For all other operations, the operation is disallowed and an appropriate SQL error code is returned. This default behavior for an outcome of Deny may be modified using the set-error-code obligation. For instance, the set-error-code obligation may be used to allow the table-open operation to proceed as though the object did not exist when the policy evaluation results in an outcome of Deny. This prevents the existence of the table object being inferred from the SQL error code.

1.5 Obligations – Actions Taken Upon Policy Decision

In addition to RXSML policies being used to control access to objects, they may be used to perform actions based upon the policy outcome. These actions are called obligations. Whether an obligation’s action is executed is dependent upon the outcome of the policy and the configuration of the obligation. An obligation may be configured to execute on an outcome of Permit or Deny. Generally, if the policy containing the obligation evaluates to the specified outcome then the obligation’s action is executed.

RXSML supports three obligations:

→ set-field,    → set-error-code, and    → audit.

The set-field obligation allows the value of a specific row field to be dynamically set during a row based operation. The set-error-code obligation allows changes of the default behavior when an operation is denied. The audit obligation allows a highly customizable audit record to be written to the audit trail.

1.6 Security Markup Language (RXSML) Overview

The Trusted Rubix Security Markup Language (RXSML) is an XML language used to construct Attribute Based Access Control (ABAC) security policies. RXSML is based upon the policy language of the OASIS XACML 2.0 standard. The RXSML language allows highly customized ABAC security policies to be constructed to arbitrate access to RDBMS objects. The policies may further refine or override the underlying Multi-level Security (MLS) policy.

Conceptually, policies written with the RXSML language apply logical security rules to attribute values that represent the execution context. The context attributes are categorized as subject attributes, resource attributes, action attributes, and environment attributes. As policy evaluations are performed (i.e., as controlled database operations are executed), context attributes are retrieved and used as inputs to the policy logic.
The evaluated policy logic produces an outcome of Permit, Deny, or Not Applicable. An outcome of Permit allows the operation to proceed while all other outcomes result in the operation being denied. An outcome of Not Applicable results when the specified target of the policy does not match the context of the operation being performed.

Logical rules are grouped into policies. Policies and policy sets may be grouped into higher-order policy sets. Policies and policy sets may be referenced by identifier, allowing modular policy construction without duplicating physical policy code.

1.6.1 Attribute Values

Attribute values are typed units of data within the RXSML language. They represent the data that policy logic uses to reach a policy decision outcome. Attribute values may originate as:

- literal values
- runtime context attributes
- or values of database row fields

All attribute values are typed and their type defines legal syntax and allowable operations. Attribute values may be cast between types using the cast function. Attribute values may be implicitly cast during the execution of certain functions where possible. The data types of the result of functions are automatically calculated and need not be specified. Functions exist to compare, transform, mathematically manipulate, perform set operations upon, and perform bag operations upon attribute values.

The supported data types are:

- string → boolean → dateTime → label
- decimal → date → yearMonthDuration → hexBinary
- double → time → dayTimeDuration → base64Binary
- integer

1.6.1.1 Attribute Bags

Attribute values exist within a construct called a bag. A bag is a possibly empty collection of like-typed attribute values. Duplicate values may exist within a bag. The RXSML language makes no distinction between a bag with a single value and a single attribute value instance. Many functions (e.g., equal function) expect single valued attribute bags as parameters and care must be taken when constructing RXSML policies. Passing multi-valued bags into such functions results in an Indeterminate outcome. Special bag functions and set-theory based functions exist to perform operations on multi-valued bags.

The bag function may be used to construct multi-valued bags and the bag-size function may be used to extract the size of a bag. The one-and-only function may be used to determine if a bag contains a single value. Most context attributes are guaranteed to be single-valued bags and no policy logic is needed to check for a single value when passing such context attributes into functions that require single-valued bags.
1.6.1.2 Context Attributes

Context attributes represent the state of the system at the time of a decision request. The data type of a context attribute is fixed based upon its nature. Context attributes are categorized as subject attributes, resource attributes, action attributes, and environment attributes.

- **Subject context attributes** are attributes associated with the subject performing an operation and are accessed using the `<SubjectAttributeDesignator>` element. Supported subject context attributes are:
  - ↑ subject-id
  - ↑ subject-name
  - ↑ group-id
  - ↑ group-name
  - ↑ session-start-time
  - ↑ session-start-date
  - ↑ session-start-dateTime
  - ↑ ip-address
  - ↑ dns-name
  - ↑ session-label
  - ↑ application-label
  - ↑ application-user-name
  - ↑ application-user-id

- **Resource context attributes** are attributes associated with the RDBMS objects being acted upon by the current operation and are accessed using the `<ResourceAttributeDesignator>` element. Supported resource context attributes are:
  - ↑ resource-label
  - ↑ resource-name
  - ↑ row-label
  - ↑ table-label
  - ↑ view-label
  - ↑ schema-label
  - ↑ catalog-label
  - ↑ database-label
  - ↑ view-name
  - ↑ column-name
  - ↑ schema-name
  - ↑ catalog-name
  - ↑ database-name
  - ↑ table-name

- **Action context attributes** are attributes associated with the current operation and are accessed using the `<ActionAttributeDesignator>` element. Supported action context attributes are:
  - ↑ action-id
  - ↑ action-type

- **Environment context attributes** are attributes associated with the execution environment and are accessed using the `<EnvironmentAttributeDesignator>` element. Supported environment context attributes are:
  - ↑ current-time
  - ↑ current-date
  - ↑ current-dateTime

Not all context attributes are available during all operations. For instance, the `view-name` resource attribute is not available during the `schema-open` operation. Appropriate care must be taken during the construction of RXSML policies. The RXSML language allows either an empty bag or an Indeterminate outcome when a request for a non-existing context attribute is made.

1.6.1.3 Row Fields as Context Attributes

RXSML allows the value of row fields to be extracted as context attributes. The fields to be accessed are specified using a five part name of the form:

```
database_name.catalog_name.schema_name.table_name.column_name
```

The desired RXSML data type of the attribute value must be specified. The row field will be converted from the SQL data type into the specified RXSML data type, if possible; otherwise, an Indeterminate outcome is returned.

A field value of the row currently being operated upon (during row-based operations) may be accessed using the `<AttributeSelector>` element. A possibly multi-valued bag containing the field values of any
column in the database may be accessed (referred to as importing) during any operation using the `<ImportColumnSelector>` element and may be filtered using the `<ImportFieldSelector>` element.

### 1.6.1.4 Examples

The following are examples of literal attribute values:

```
1. <AttributeValue DataType="string">database-open</AttributeValue>
2. <AttributeValue DataType="label">TS</AttributeValue>
3. <AttributeValue DataType="integer">-9785</AttributeValue>
4. <AttributeValue DataType="dayTimeDuration">P50DT10H5M50.411103S</AttributeValue>
```

The following are examples of accessing context attributes:

```
1. <SubjectAttributeDesignator AttributeId="subject-name"/>
2. <ResourceAttributeDesignator AttributeId="database-label"/>
3. <ActionAttributeDesignator AttributeId="action-id"/>
4. <EnvironmentAttributeDesignator AttributeId="current-date"/>
```

The following is an example of accessing the field from the owner column of the current row:

```
1. <AttributeSelector DatabasePath="db.cat.sch.tab.owner" DataType="string"/>
```

The following is an example of accessing values of the ipaddr column of the `db.cat.sch.iptab` table. The values are filtered according to the value in the user column, which must equal the subject-name subject attribute:

```
1. <ImportColumnSelector DatabasePath="db.cat.sch.iptab.ipaddr" DataType="ipAddress">
2.  <Apply FunctionId="equal">
3.   <ImportFieldSelector DatabasePath="db.cat.sch.iptab.user" DataType="string"/>
4.  </Apply>
5. </ImportColumnSelector>
```

### 1.6.2 Functions

RXSML provides a variety of functions to manipulate attribute values. Functions exist to compare, transform, mathematically manipulate, perform set operations upon, and perform bag operations upon attribute values. Functions generally accept some number of typed arguments and produce a typed result.
The primary method for invoking a function is using the `<Apply>` element where the function is specified as the `FunctionId` attribute. Additionally, the context attribute matching elements (e.g., `<SubjectMatch>`) invoke one of the matching functions and the `map` function may be used to invoke a specified function over a bag of values.

Functions, called via the `<Apply>` element, evaluate to a typed bag of values or to `Indeterminate`. Each function type defines its behavior, the data type and number of values in its arguments, and the data type and number of values in its result. Many functions require their arguments to be single-valued bags. In general, the data type of the result is automatically calculated based upon the data types of the input arguments. Most functions take a static number of arguments, generally one, two, or three. Some functions take an unlimited number of arguments. For example the `sum` function produces the numerical sum of any number of arguments.

### 1.6.2.1 Examples

The following is an example of using the `equal` function that produces a boolean result representing equality between the `subject-name` context attribute and a literal attribute value:

```xml
<Apply FunctionId="equal">
  <AttributeValue DataType="string">smith</AttributeValue>
  <SubjectAttributeDesignator AttributeId="subject-name"/>
</Apply>
```

The following is an example of using the `concatenate` function to produce a single string that is the concatenation of the `subject-id` and `ip-address` context attributes, using `:'` as a separator:

```xml
<Apply FunctionId="concatenate">
  <SubjectAttributeDesignator AttributeId="subject-id"/>
  <AttributeValue DataType="string">:</AttributeValue>
  <SubjectAttributeDesignator AttributeId="ip-address"/>
</Apply>
```

The following is an example of using the `bag` function to produce a three-valued bag of literal string values:

```xml
<Apply FunctionId="bag">
  <AttributeValue DataType="string">smith</AttributeValue>
  <AttributeValue DataType="string">taylor</AttributeValue>
  <AttributeValue DataType="string">white</AttributeValue>
</Apply>
```

### 1.6.3 Target

The `<Target>` element is always parented by a `<PolicySet>`, `<Policy>`, or `<Rule>` element and defines the context for which the parent element applies. Specifically, it defines a set of subjects, resources,
actions, and environments for which the parent <PolicySet>, <Policy>, or <Rule> element will be used in fulfilling a decision request. If the current context does not match the target then the remainder of the parent element will not be evaluated in satisfying the decision request and the parent element will evaluate to Not Applicable. The <Target> element may evaluate to Match, NoMatch, or Indeterminate.

The <Target> element is used to define subjects, resources, actions, and environment that may be easily indexed. This allows indexes to be built that provide fast matching between a decision request and associated policy. The method of specifying the matching set of attributes is therefore restricted to simplistic comparisons. The set of functions that may be used within a target are known as matching functions and are a subset of the total set of functions provided by the RXSML language. Valid matching functions are:

- \( \rightarrow \text{equal} \), \( \rightarrow \text{greater-than} \), \( \rightarrow \text{less-than} \), \( \rightarrow \text{regexp-match} \),
- \( \rightarrow \text{not-equal} \), \( \rightarrow \text{greater-than-or-equal} \), \( \rightarrow \text{less-than-or-equal} \), \( \rightarrow \text{dnsName-match} \), and \( \rightarrow \text{ipAddress-match} \).

It should be noted that the logic of the <Target> element may instead be encapsulated within the <Condition> of the <Rule> elements; however, where possible, it is advisable to abstract the target of the policy from the rules of the policy by utilizing the <Target> element. It should also be noted that using the rxpolman command to apply policies to RDBMS objects along with configuring the Scope attribute effectively sets the policy’s target resources. Applying policies using the rxpolman command is preferable to using <Target> element logic, where possible, due to its increased performance and reduced policy complexity.

### 1.6.3.1 Target Evaluation

The <Target> element may evaluate to Match, NoMatch, or Indeterminate. For the parent <PolicySet>, <Policy>, or <Rule> element to be applicable to the decision request, the <Target> element must evaluate to Match for the current context.

The <Target> element consists of a conjunctive sequence of <Subjects>, <Resources>, <Actions>, and <Environments> elements. For the <Target> element to evaluate to Match, each child element must evaluate to Match. If a child element does not exist for a context attribute category then the <Target> implicitly matches all possible context attributes from that category. If there are no child elements at all (i.e., the <Target> element is empty) then the <Target> implicitly matches all contexts. The method for specifying the set of subjects within a target is described below. The methods for specifying the resources, actions, and environments are similar.

The subjects are specified within a <Subjects> element. The <Subjects> element may evaluate to Match, NoMatch, or Indeterminate. The <Subjects> element contains a disjunctive sequence of <Subject> elements. For the <Subjects> element to evaluate to Match at least one of its <Subject> elements must evaluate to Match.

The <Subject> element may evaluate to Match, NoMatch, or Indeterminate. The <Subject> element contains a conjunctive sequence of <SubjectMatch> elements. For <Subject> to evaluate to Match each <SubjectMatch> element must evaluate to Match.

The <SubjectMatch> element may evaluate to Match, NoMatch, or Indeterminate. The <SubjectMatch> element contains a matching specification for a single subject context attribute. Each matching specification consists of a matching function given as an XML attribute, an <AttributeValue> child element, and a <SubjectAttributeDesignator> child element. The subject context attribute may be a multi-valued bag in which case the <SubjectMatch> evaluates to Match if any value in the bag matches the
specification. The evaluation is performed by applying the matching function to the literal value specified by the `<AttributeValue>` element and each value in the subject context attribute bag returned by the `<SubjectAttributeDesignator>` element. If any matching function execution returns true then the `<SubjectMatch>` evaluates to `Match`.

The following truth table defines the `<Target>` element evaluation logic.

<table>
<thead>
<tr>
<th>&lt;Subjects&gt;</th>
<th>&lt;Resources&gt;</th>
<th>&lt;Actions&gt;</th>
<th>&lt;Environments&gt;</th>
<th>&lt;Target&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>Match</td>
<td>Match</td>
<td>Match</td>
<td>Match</td>
</tr>
<tr>
<td>Nomatch</td>
<td>Match or NoMatch</td>
<td>Match or NoMatch</td>
<td>Match or NoMatch</td>
<td>NoMatch</td>
</tr>
<tr>
<td>Match or NoMatch</td>
<td>NoMatch</td>
<td>Match or NoMatch</td>
<td>Match or NoMatch</td>
<td>NoMatch</td>
</tr>
<tr>
<td>Match or NoMatch</td>
<td>Match or NoMatch</td>
<td>NoMatch</td>
<td>Match or NoMatch</td>
<td>NoMatch</td>
</tr>
<tr>
<td>Match or NoMatch</td>
<td>Match or NoMatch</td>
<td>NoMatch</td>
<td>Match or NoMatch</td>
<td>NoMatch</td>
</tr>
<tr>
<td>Indeterminate</td>
<td>Don't care</td>
<td>Don't care</td>
<td>Don't care</td>
<td>Indeterminate</td>
</tr>
<tr>
<td>Don't care</td>
<td>Indeterminate</td>
<td>Don't care</td>
<td>Don't care</td>
<td>Indeterminate</td>
</tr>
<tr>
<td>Don't care</td>
<td>Indeterminate</td>
<td>Don't care</td>
<td>Indeterminate</td>
<td>Indeterminate</td>
</tr>
<tr>
<td>Don't care</td>
<td>Don't care</td>
<td>Don't care</td>
<td>Indeterminate</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>

The following truth table defines the `<Subjects>` element evaluation logic. The evaluation logic for the `<Resources>`, `<Actions>`, and `<Environments>` elements are similar.

<table>
<thead>
<tr>
<th>&lt;Subject&gt;</th>
<th>&lt;Subjects&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>At least one Match</td>
<td>Match</td>
</tr>
<tr>
<td>All Nomatch</td>
<td>NoMatch</td>
</tr>
<tr>
<td>At least one Indeterminate and none are Match</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>

The following truth table defines the `<Subject>` element evaluation logic. The evaluation logic for the `<Resource>`, `<Action>`, and `<Environment>` elements are similar.

<table>
<thead>
<tr>
<th>&lt;SubjectMatch&gt;</th>
<th>&lt;Subject&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Match</td>
<td>Match</td>
</tr>
<tr>
<td>At least one Nomatch</td>
<td>NoMatch</td>
</tr>
<tr>
<td>At least one Indeterminate and none are NoMatch</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>
The following truth table defines the `<SubjectMatch>` element evaluation logic. The evaluation logic for the `<ResourceMatch>`, `<ActionMatch>`, and `<EnvironmentMatch>` elements are similar.

### `<SubjectMatch>` EVALUATION LOGIC

<table>
<thead>
<tr>
<th>Matching Function Execution</th>
<th><code>&lt;SubjectMatch&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>At least one value in bag results in <code>true</code></td>
<td><code>Match</code></td>
</tr>
<tr>
<td>All values in bag result in <code>false</code></td>
<td><code>NoMatch</code></td>
</tr>
<tr>
<td>At least one value in bag results in <code>Indeterminate</code> and none result in <code>true</code></td>
<td><code>Indeterminate</code></td>
</tr>
</tbody>
</table>
1.6.3.2 Examples

The following is an example of a target specifying all four context attribute classes:
<Target>
  <Subjects>
    <Subject>
      <SubjectMatch MatchId="equal">
        <AttributeValue DataType="string">smith</AttributeValue>
        <SubjectAttributeDesignator AttributeId="subject-name"/>
      </SubjectMatch>
    </Subject>
  </Subjects>
  <Resources>
    <Resource>
      <ResourceMatch MatchId="equal">
        <AttributeValue DataType="string">d.cat.sch.t</AttributeValue>
        <ResourceAttributeDesignator AttributeId="table-name"/>
      </ResourceMatch>
    </Resource>
  </Resources>
  <Actions>
    <Action>
      <ActionMatch MatchId="equal">
        <AttributeValue DataType="string">row-select</AttributeValue>
        <ActionAttributeDesignator AttributeId="action-id"/>
      </ActionMatch>
    </Action>
  </Actions>
  <Environments>
    <Environment>
      <EnvironmentMatch MatchId="less-than">
        <AttributeValue DataType="time">18:00:0</AttributeValue>
        <EnvironmentAttributeDesignator AttributeId="current-time"/>
      </EnvironmentMatch>
    </Environment>
  </Environments>
</Target>
The following is an example of a target specifying only subjects (matching the user *smith* in group *admin* or the user *taylor* in group *admin*) and implicitly matching all resources, actions, and environments:

```xml
<Target>
  <Subjects>
    <Subject>
      <SubjectMatch MatchId="equal">
        <AttributeValue DataId="string">smith</AttributeValue>
        <SubjectAttributeDesignator AttributeId="subject-name"/>
      </SubjectMatch>
    </Subject>
    <Subject>
      <SubjectMatch MatchId="equal">
        <AttributeValue DataId="string">admin</AttributeValue>
        <SubjectAttributeDesignator AttributeId="group-name"/>
      </SubjectMatch>
    </Subject>
    <Subject>
      <SubjectMatch MatchId="equal">
        <AttributeValue DataId="string">taylor</AttributeValue>
        <SubjectAttributeDesignator AttributeId="subject-name"/>
      </SubjectMatch>
    </Subject>
  </Subjects>
</Target>
```
The following is an example of an empty target implicitly matching all contexts:

```
1  <Target/>
```

### 1.6.4 Rule

The rule is the most elemental construct that may produce a decision outcome. A `<Rule>` element may evaluate to Permit, Deny, Not Applicable, or Indeterminate. The `<Rule>` element may exist only within a `<Policy>` element. The main components of the `<Rule>` element are the target, effect, and condition.

The `<Target>` element of a `<Rule>` defines the context for which the rule applies. Specifically, it defines a set of subjects, resources, actions, and environments for which the rule will be considered in calculating the decision outcome of the parent `<Policy>` element. The `<Target>` element of a rule may evaluate to Match, NoMatch, or Indeterminate. If the current context does not match the target (i.e., the `<Target>` evaluates to NoMatch) then the remainder of the rule will not be evaluated and the rule will immediately evaluate to Not Applicable.

The Effect is an XML attribute of the `<Rule>` element and defines the outcome for the rule if the `<Condition>` element evaluates to true. The Effect may be Permit or Deny. If the `<Condition>` is evaluated to true the outcome of the `<Rule>` is equal to its Effect. If the `<Condition>` evaluates to false then the outcome is Not Applicable.

If it exists, the `<Condition>` element contains a predicate that represents the logic of the `<Rule>`. The `<Condition>` may evaluate to true, false, or Indeterminate. The outcome of the `<Condition>` controls if the `<Rule>` evaluates to its Effect (if `<Condition>` is true) or to Not Applicable (if `<Condition>` is false). The `<Condition>` element contains a single child element that must evaluate to a boolean data type, typically the `<Apply>` element with a boolean function. If `<Condition>` does not exist then it implicitly evaluates to true.

#### 1.6.4.1 Rule Evaluation

When a `<Rule>` is evaluated within a `<Policy>` element the `<Target>` is first evaluated.

If the `<Target>` evaluates to Match, then the `<Condition>` of the `<Rule>` will then be evaluated; otherwise, if the `<Target>` evaluates to NoMatch, the `<Condition>` is not evaluated, and the `<Rule>` evaluates to Not Applicable.

If `<Condition>` does not exist then it implicitly evaluates to true.

If the `<Condition>` evaluates to true then the `<Rule>` evaluates to its Effect. If the `<Condition>` evaluates to false then the `<Rule>` evaluates to Not Applicable.

The `<Rule>` will evaluate to Indeterminate if the `<Target>` evaluates to Indeterminate or if the `<Target>` evaluates to Match and the `<Condition>` evaluates to Indeterminate. The following truth table defines the `<Rule>` evaluation logic.

<table>
<thead>
<tr>
<th><code>&lt;Target&gt;</code></th>
<th><code>&lt;Condition&gt;</code></th>
<th><code>&lt;Rule&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>True</td>
<td>Equal to the Effect</td>
</tr>
<tr>
<td>Match</td>
<td>False</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Match</td>
<td>Indeterminate</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>
1.6.4.2 Example

The following is an example of a <Rule> element that evaluates to Permit if the user smith performs any action where the session label is greater than or equal to (i.e., dominates) the object label; otherwise, it will evaluate to Not Applicable.

```
1 <Rule RuleId="SmithDominatesRule" Effect="Permit">
2   <Target>
3     <Subjects>
4       <Subject>
5         <SubjectMatch MatchId="equal">
6           <AttributeValue
7               DataType="string">smith</AttributeValue>
8           <SubjectAttributeDesignator
9               AttributeId="subject-name"/>
10         </SubjectMatch>
11       </Subject>
12     </Subjects>
13   </Target>
14   <Condition>
15     <Apply FunctionId="greater-than-or-equal">
16       <SubjectAttributeDesignator AttributeId="session-label"/>
17       <ResourceAttributeDesignator AttributeId="object-label"/>
18     </Apply>
19   </Condition>
20 </Rule>
```

1.6.5 Policy

The <Policy> element is a top-level element and is the most basic element that may be associated with RDBMS objects. The main components of a <Policy> element are the target, variable definitions and references, a set of rules, a combining algorithm for those rules, and a set of obligations. Conceptually, policies represent a cohesive set of access control rules and the way those rules relate to each other along with associated actions (i.e., obligations) to take as decision requests are made. The <Policy> element may be referenced using its identifier within <Policy Set> elements allowing for the modular construction of complex ABAC policies.
The `<Target>` element of a `<Policy>` defines the context for which the policy applies. Specifically, it defines a set of subjects, resources, actions, and environments for which the policy will be considered in calculating the decision outcome. The `<Target>` element may evaluate to `Match`, `NoMatch`, or `Indeterminate`. If the current context does not match the target (i.e., the `<Target>` evaluates to `NoMatch`) then the remainder of the policy will not be evaluated and will immediately evaluate to `Not Applicable`.

Variable definitions and references allow sections of RXSML code that produce a typed result (expressions) to be defined and then referenced by identifier within a `<Policy>` element. A variable definition is constructed using the `<VariableDefinition>` element. A variable definition reference is made using the `<VariableReference>` element. Typically, the top-level element of variable definitions would be an `<Apply>` element or a context attribute designator. The references to variables are typically made from within `<Rule>` elements. Variable definitions may be nested and references to them may be made multiple times. Using variable definitions and references has the advantage of modularizing the RXSML code. It also may increase performance as the variable definition’s code may only need to be evaluated once regardless of the number of references to it.

The rules of a policy consist of a sequence of `<Rule>` elements. The rules represent the logic used to reach a decision outcome. Each `<Rule>` contains the logic that, if true, will cause the `<Rule>` to evaluate to its `Effect` (either `Permit` or `Deny`) and if false will evaluate to `Not Applicable`. The policy decision outcome is calculated from the outcome of the set of rules according to the policy’s rule combining algorithm.

The rule combining algorithm is used to calculate the policy’s decision outcome from the outcome of the set of rules. The rule combining algorithm is specified as the `RuleCombiningAlgId` attribute of the `<Policy>` element. Valid rule combining algorithms are:

\[
\begin{align*}
& \rightarrow \text{deny-overrides} & \rightarrow \text{ordered-deny-overrides} \\
& \rightarrow \text{permit-overrides} & \rightarrow \text{ordered-permit-overrides} \\
& \rightarrow \text{first-applicable}
\end{align*}
\]

Obligations are actions that are required to be taken when a decision request is made. Obligations may be contained within a `<Policy>` element and will be executed when the decision outcome of the obligation’s parent `<Policy>` and the top-level `<PolicySet>` (if any) matches the specified `FulfillOn` attribute of the `<Obligation>` element. Obligations exist to set the value of a specific row field during a row-based operation (set-field), to change the default behavior when an operation is denied (set-error-code), and to write a customizable audit record (audit).

### 1.6.5.1 Policy Evaluation

When a `<Policy>` is evaluated its `<Target>` is evaluated first. The `<Target>` element may evaluate to `Match`, `NoMatch`, or `Indeterminate`. If the `<Target>` evaluates to `Match` then the remainder of the `<Policy>` is evaluated. If the `<Target>` evaluates to `NoMatch` then the `<Policy>` immediately evaluates to `Not Applicable`.

When the sequence of `<Rule>` elements are evaluated they may produce an outcome of `Permit`, `Deny`, `Not Applicable`, or `Indeterminate`. The outcome of the `<Policy>` is then calculated based upon the specified rule combining algorithm, which may be:

\[
\begin{align*}
& \rightarrow \text{deny-overrides}, & \rightarrow \text{ordered-deny-overrides}, \text{ and} \\
& \rightarrow \text{permit-overrides}, & \rightarrow \text{ordered-permit-overrides}. \\
& \rightarrow \text{first-applicable},
\end{align*}
\]
In general, if a `<Rule>` need not be evaluated to determine the final outcome of the `<Policy>` it is not evaluated.

If a `<Rule>` contains `<VariableReference>` elements then the execution of the `<Rule>` is the same as though the `<VariableReference>` elements were replaced with the code within the corresponding `<VariableDefinition>` elements. Variable references are expressions meaning that they evaluate to a typed bag of values.

After the decision outcome of the `<Policy>` has been reached, any existing set of obligations is examined. Any `<Obligation>` elements that have a `FulfillOn` attribute that matches the decision outcome of the `<Policy>` are passed to the parent `<PolicySet>` element (if it exists) of the `<Policy>`. After the final policy decision has been reached by the top-level `<Policy>` or `<PolicySet>` element, all bundled obligations will be executed whose `FulfillOn` attribute equals the final policy decision.

The following truth table defines the evaluate logic for the `<Policy>` element.
### 6.0.5.2 Example

The following example the `<Policy>` allows the user `smith` (connecting from any IP) and a user in the group `admin` connecting from IP 127.0.0.1 (`localhost`) to perform a read operation if their session label does not dominate the object label. It provides MLS override permission for read operations from such subjects.

If the MLS override is permitted then it will audit the decision outcome.

If the subject does not match the targets, the action is not a read operation, or the session label dominates the object label the `<Policy>` evaluates to `Not Applicable` and no audit is performed.

- The `<Policy>` has a rule combining algorithm of *ordered-permit-override* which will cause an outcome of `Permit` if any rule evaluates to `Permit`. The `<Rule>` elements are guaranteed to be evaluated in order and execution stops as soon as a `Permit` is received.
- The `<Target>` of lines 2-7 ensure the current operation is a read operation; otherwise, the remainder of the `<Policy>` will not be evaluated and will immediately have an outcome of `Not Applicable`.
- The `<VariableDefinition>` of lines 9-16 will evaluate to true if the session label does not dominate the object label. It is referenced in both of the following `<Rule>` elements.
- The `<Rule>` of lines 18-28 targets a subject with a user name of `smith`. If the subject matches the target and the `<Condition>` is true then the `<Rule>` will evaluate to its `Effect`, in this case `Permit`. The `<Condition>` simply has a reference to the variable and will evaluate to `true` if the session label does not dominate the object label.
- The `<Rule>` of lines 30-44 targets a subject with a group name of `admin` connecting from 127.0.0.1. If the subject matches the target and the `<Condition>` is true then the `<Rule>` will evaluate to its `Effect`, in this case `Permit`. The `<Condition>` simply has a reference to the variable and will evaluate to `true` if the session label does not dominate the object label.
- The `<Obligation>` of lines 46-48 will execute when its `Effect` equals the `<Policy>`. For this case they must both be `Permit`. In addition to the common audit data, the audit record will contain a string that indicates an MLS read override has occurred. Note that the audit record will only be written if the MLS read override has been granted and the final decision outcome is `Permit`.

<table>
<thead>
<tr>
<th><code>&lt;Target&gt;</code></th>
<th><code>&lt;Rule&gt;</code>’s</th>
<th><code>&lt;Policy&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>At least one evaluates to its <code>Effect</code></td>
<td>Specified by the rule combining algorithm</td>
</tr>
<tr>
<td>Match</td>
<td>All rules evaluate to <code>Not Applicable</code></td>
<td><code>Not Applicable</code></td>
</tr>
<tr>
<td>Match</td>
<td>At least one rule evaluates to <code>Indeterminate</code></td>
<td>Specified by the rule combining algorithm</td>
</tr>
<tr>
<td>NoMatch</td>
<td>Don’t care</td>
<td><code>Not Applicable</code></td>
</tr>
<tr>
<td>Indeterminate</td>
<td>Don’t care</td>
<td><code>Indeterminate</code></td>
</tr>
</tbody>
</table>
<Policy PolicyId="MLSReadOverridePolicy" RuleCombiningAlgId="ordered-permit-overrides">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="equal">
          <AttributeValue DataType="string">read</AttributeValue>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>

  <VariableDefinition VariableId="NotDominates">
    <Apply FunctionId="not">
      <Apply FunctionId="greater-than-or-equal">
        <SubjectAttributeDesignator AttributeId="session-label"/>
        <ResourceAttributeDesignator AttributeId="object-label"/>
      </Apply>
    </Apply>
  </VariableDefinition>

  <Rule RuleId="SmithMLSReadOverrideRule" Effect="Permit">
    <Target>
      <Subjects><Subject>
        <SubjectMatch MatchId="equal">
          <AttributeValue DataType="string">smith</AttributeValue>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Condition>
      <VariableReference VariableId="NotDominates"/>
    </Condition>
  </Rule>

  <Rule RuleId="AdminMLSReadOverride" Effect="Permit">
    <Target>
      <Subjects><Subject>
        <SubjectMatch MatchId="equal">
          <AttributeValue DataType="string">admin</AttributeValue>
        </SubjectMatch>
      </Subject><SubjectMatch MatchId="equal">
        <AttributeValue DataType="ipAddress">127.0.0.1</AttributeValue>
      </SubjectMatch>
    </Subjects>
    <Condition>
      <VariableReference VariableId="NotDominates"/>
    </Condition>
  </Rule>
</Policy>

<Obligations> <Obligation FulfillOn="Permit" ObligationId="audit"/> </Obligations>
1.6.6 Policy Set

A policy set is a coherent collection of policies that are combined using a policy combining algorithm along with associated actions (i.e., obligations) to take as decision requests are made. The `<PolicySet>` is a top-level element and may be associated with RDBMS objects. The main components of the `<PolicySet>` element are:

- the target,
- policies,
- policy references,
- policy sets,
- policy set references,
- a policy combining algorithm, and
- obligations.

The `<Target>` element of a `<PolicySet>` defines the context for which it applies. Specifically, it defines a set of subjects, resources, actions, and environments for which `<PolicySet>` will be considered in calculating the decision outcome. The `<Target>` element may evaluate to Match, NoMatch, or Indeterminate. If the current context does not match the target (i.e., the `<Target>` evaluates to NoMatch) then the remainder of the `<PolicySet>` will not be evaluated and will immediately evaluate to Not Applicable.

The `<PolicySet>` element does not directly contain access control rules. Instead, its evaluation behavior is controlled by the policies that are contained within it. These may exist explicitly, using `<Policy>` and `<PolicySet>` elements or by reference, using the `<PolicyIdReference>` and `<PolicySetIdReference>` elements. The decision outcome of a `<PolicySet>` is calculated from the outcomes of these elements using the policy combining algorithm.

The policy combining algorithm is used to calculate the `<PolicySet>` element’s decision outcome from the outcome of the set of child policies. The policy combining algorithm is specified as the PolicyCombiningAlgId attribute of the `<PolicySet>` element. Valid policy combining algorithms are:

- deny-overrides,
- permit-overrides,
- first-applicable,
- only-one-applicable,
- ordered-deny-overrides, and
- ordered-permit-overrides.

Obligations are actions that are required to be taken when a decision request is made. Obligations may be contained within a `<PolicySet>` element and will be executed when the decision outcome of the obligation’s parent `<PolicySet>` and the top-level `<PolicySet>` (if any) matches the specified FulfillsOn attribute of the `<Obligation>` element. Obligations exist to set the value of a specific row field during a row-based operation (set-field), to change the default behavior when an operation is denied (set-error-code), and to write a customizable audit record (audit).

1.6.6.1 Policy Set Evaluation

When a `<PolicySet>` is evaluated its `<Target>` is evaluated first. The `<Target>` element may evaluate to Match, NoMatch, or Indeterminate.

If the `<Target>` evaluates to Match then the remainder of the `<PolicySet>` is evaluated.

If the `<Target>` evaluates to NoMatch then the `<PolicySet>` immediately evaluates to Not Applicable.

When each policy is evaluated it may produce an outcome of Permit, Deny, Not Applicable, or Indeterminate. The outcome of the `<PolicySet>` is then calculated based upon the specified policy combining algorithm, which may be deny-overrides, permit-overrides, first-applicable, only-one-applicable, ordered-deny-overrides, and ordered-permit-overrides. In general, if a policy need not be evaluated to determine the final outcome of the `<PolicySet>` it is not evaluated.
If a policy is included by reference (using the `<PolicyIdReference>` or `<PolicySetIdReference>` elements) then the execution of the `<PolicySet>` is the same as though the references were replaced with the code within the corresponding policy.

After the decision outcome of the `<PolicySet>` has been reached, any existing set of obligations is examined. Any `<Obligation>` elements that have a FulfillOn attribute that matches the decision outcome of the `<PolicySet>` are passed to the parent `<PolicySet>` element (if it exists) of the `<PolicySet>`. After the final policy decision has been reached by the top-level `<PolicySet>` element, all bundled obligations will be executed whose FulfillOn attribute equals the final policy decision.

The following truth table defines the evaluate logic for the `<PolicySet>` element.

<table>
<thead>
<tr>
<th><code>&lt;Target&gt;</code></th>
<th>Child Policies</th>
<th><code>&lt;PolicySet&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>At least one policy evaluates to Permit or Deny</td>
<td>Specified by the policy combining algorithm</td>
</tr>
<tr>
<td>Match</td>
<td>All policies evaluate to Not Applicable</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Match</td>
<td>At least one policy evaluates to Indeterminate</td>
<td>Specified by the policy combining algorithm</td>
</tr>
<tr>
<td>NoMatch</td>
<td>Don’t care</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Indeterminate</td>
<td>Don’t care</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>

1.6.6.2 Example

The following `<PolicySet>` element example defines policy for all RDBMS read-based operations (selects and opens). The read-based operation is allowed if it passes the basic OS-MAC check or if it is submitted by subjects allowed to override the OS-MAC policy. If MLS override is allowed an audit record is written.

On line 1 the policy combining algorithm is specified as *ordered-permit-overrides* which will evaluate the enclosed policies in order. If a Permit outcome is received then evaluation stops and the `<PolicySet>` immediately evaluates to Permit. The `<PolicySet>` will evaluate to Deny only if a policy evaluates to Deny and all other policies evaluate to Not Applicable.

The `<Target>` on lines 2-7 limits the applicability of the `<PolicySet>` to read-based operations. For all other operations the `<PolicySet>` will immediately evaluate to Not Applicable.

The `<Policy>` on lines 9-17 performs a basic OS-MAC check (i.e., session label dominates object label). The `<Policy>` will evaluate to Permit if the operation would be allowed based upon the OS-MAC policy; otherwise, it will evaluate to Not Applicable. The special *MAC-check* function is used (line 14) which returns true if the current operation would be allowed on the current object using the OS-MAC policy; otherwise, it returns false. Note that this `<Policy>` only makes a decision if the operation were permitted according to the OS-MAC policy and makes no decision (i.e., Not Applicable) if the OS-MAC policy would deny operation.

The `<PolicyIdReference>` on line 19 references the `<Policy>` of the previous example in this document. The *MLSReadOverridePolicy* policy will allow OS-MAC override on a read-based operation if it is performed by the user smith (connecting from any IP) or a user in the group admin connecting from IP 127.0.0.1 (localhost). It will also create an audit record if OS-MAC override is granted. Note that if OS-MAC override is not granted the `<PolicyIdReference>` will evaluate to Not Applicable and no audit
record will be produced.

The <Policy> of line 21-26 explicitly denies the operation. Note that is has no <Condition> element which means the condition is always implicitly true. Also, it has an empty <Target> which means it applies to all contexts. Therefore, if this <Policy> is evaluated it will always have an outcome of Deny. Within our <PolicySet> this <Policy> will only be evaluated if the previous two polices do not evaluate to Permit.

```
<PolicySet PolicySetId="MLSReadPolicySet" PolicyCombiningAlgId="ordered-permit-overrides">
  <Target> <Actions> <Action>
    <ActionMatch MatchId="equal">
      <AttributeValue DataType="string">read</AttributeValue>
      <ActionAttributeDesignator AttributeId="action-type"/>
    </ActionMatch>
  </Action> </Actions> </Target>

  <Policy PolicyId="MLSBasicReadPolicy" RuleCombiningAlgId="ordered-permit-overrides">
    <Target/>
    <Condition>
      <Apply FunctionId="MAC-check"/>
    </Condition>
  </Policy>

  <PolicyIdReference PolicySetId="MLSReadOverridePolicy"/>

  <Policy PolicyId="DenyPolicy" RuleCombiningAlgId="ordered-deny-overrides">
    <Target/>
    <Rule PolicyId="DenyRule" Effect="Deny">
      <Target/>
    </Rule>
  </Policy>
</PolicySet>
```
2 SECURITY MARKUP LANGUAGE (RXSML) POLICY FILES

Trusted Rubix Security Markup Language (RXSML) is contained within XML-based operating system text files with ‘.rxsml’ extensions. Each RXSML policy file is characterized by its top-level <Policy> or <PolicySet> element. The top-level element defines the RXSML policy file’s name, if it has MAC-override privileges, and its scope of applicability. The RXSML policy file is the construct that is assigned to database objects. The “life cycle” of RXSML policy files is as follows:

1. Create and edit the RXSML file. This may be done with the Trusted Rubix Policy Editor, a generic XML editor, or a basic text editor. This may be performed on any platform by any user.
2. Review the RXSML file for correctness. This is performed on the server platform by the TR Security Administrator.
3. Add the RXSML file to the Trusted Policy Repository. This is performed on the server platform by the TR Security Administrator.
4. Apply the RXSML policy file to the appropriate database objects. This is performed on the server platform by the TR Security Administrator.

Errors in the RXSML language are minimized by using the TR Policy Editor with the associated XML schema. However, not all RXSML language errors are found during the editing stage. Additional errors checks are made when the RXSML policy file is added to the Trusted Policy Repository. Other errors are only determinable during execution and result in Indeterminate outcomes. In all cases error messages are produced and the RXSML code should be modified to remove any errors.

2.1 Creating and Editing RXSML Policy Files

The RXSML policy file is an XML-based operating system text file. It may be created and edited using the TR Policy Editor, a generic XML editor, or a basic text editor. This may be performed by any user on any platform that supports the desired editor. That is, no special authorizations are required to create and edit a RXSML policy file. Authorizations are required to add the RXSML policy file to the Trusted Policy Repository. Once an RXSML policy file has been created and edited it should be moved to the TR server platform for review by the TR Security administrator.

It is advised, but not required, that the TR Policy Editor be used to create and edit RXSML policy files. The TR Policy Editor is a specially configured version of the open-source, java XML editor Jaxe. More information on the Jaxe XML editor project including available source code can be found at http://jaxe.sourceforge.net. The TR Policy Editor is configured to provide context sensitive editing menus specific to the XML schema associated with the RXSML language. If a generic XML editor is used to create and edit SMPL policy files an XML schema file (RXSMLSchema.xsd) is provided to check syntax.

2.2 Adding RXSML Policy Files to the Trusted Policy Repository

Once an RXSML policy file has been created, edited, and moved to the server platform it must be reviewed by the Trusted Rubix Security Administrator. This is achieved by opening the RXSML policy file in the operating system trusted editor and visually verifying the RXSML code for correctness in syntax and functionality.
Once the RXSML policy file has been verified it may be added to the Trusted Policy Repository using the rxpolman administrative command. This requires the rubix.admin.policy.create authorization. When the RXSML policy file is added to the repository it is parsed and additional error checks are made. Any errors produce appropriate messages. If any errors are reported they should be fixed prior to reattempting to add the RXSML policy file. When the RXSML policy file is stored within the repository it is placed in an operating system file with a name equal to the PolicySetId attribute (if the top-level element is <PolicySet>) or PolicyId attribute (if the top-level element is <Policy>).

The Trusted Policy Repository is an isolated and protected directory that contains verified RXSML policy files. The Trusted Policy Repository stores RXSML policy configuration files in the RUBIXHOME/etc/security/policies directory. Individual files are named for their top-level PolicyId or PolicySetId values. Only RXSML policy files present in the repository may be assigned to database objects. When a Trusted Rubix database server is instantiated all RXSML policy files are read, converted to binary form, and cached for future decision requests.

### 2.3 Applying RXSML Files to RDBMS Objects

After an RXSML policy file has been added to the Trusted Policy Repository it may be applied to database objects. Applying an RXSML policy file to a target object causes the top-level <PolicySet> or <Policy> to be enforced for that object and optionally part of the subtree of database objects rooted by the target object. RXSML policy files are applied to database objects using the rxpolman command. The operation is limited to administrators with the rubix.admin.policy.apply authorization.

RXSML policy files may be applied to databases, catalogs, schemata, tables, and views. Within a database, these objects are structured as an inverted four level tree. As shown in the following diagram, the root of the tree is the database, the second level are catalogs, the third level are schemata, and the fourth level are tables and views.

```
Database
    └── Catalog1
        └── Schema1
            └── Table1
        └── Schema2
            └── View1
    └── Catalog2
        └── Schema1
            └── Table1
        └── Schema2
            └── View1
```

Each database object may have at most one RXSML policy file applied to it. When an RXSML policy file is applied to a database object it will be enforced for only that object if the top-level <PolicySet> or <Policy> element’s Scope attribute is Node (the default). If the top-level element’s Scope attribute is Subtree then the RXSML policy file will be inherited by descendent objects with no RXSML policy file assignment. If a database object has no applied or inherited RXSML policy file then there is no associated RXSML policy file and the default MAC security policy is automatically enforced.
The RXSML policy file associated with an object will be enforced for operations on that object. A database object may only have a single RXSML policy file associated with it. The rules for determining which RXSML policy file is associated with a database object are:

1. If an RXSML policy file has been directly applied to an object then that RXSML policy file is associated with the object; otherwise,
2. The RXSML policy file with a Scope attribute of Subtree directly assigned to the closest ancestor in the object’s path, if any, is associated with the object; otherwise,
3. There is no RXSML policy file associated with the object and the default MAC security policy is enforced.

As an example, assume RXSML policy file policy1.rxsm, with the Scope attribute set to Subtree, is assigned to the object Database of the proceeding diagram. If no other policy files are assigned to objects then policy1.rxsm will be inherited by every object in the database. Note that new objects created in the database will automatically inherit policy1.rxsm.

Now assume we also assign policy2.rxsm, with the Scope attribute set to Subtree, to the object Database.Catalog2. This will cause the entire subtree rooted by Database.Catalog2 to be associated with policy2.rxsm while all other database objects will be associated with policy1.rxsm.

Finally assume we apply policy3.rxsm, with the Scope attribute set to Node, to object Database.Catalog2.Schema2. This will result in Database.Catalog2.Schema2 having the policy3.rxsm associated with it; the other objects will remain unchanged. The following diagram shows the results of applying policy1.rxsm (red), policy2.rxsm (blue), and policy3.rxsm (yellow) as previously discussed.

![Diagram of RXSML policy enforcement](image)

### 2.4 The rxpolman Command

The rxpolman administrative command is used to manipulate RXSML policy files. It may be used to:

- Add an RXSML policy file to the Trusted Policy Repository
- Delete an RXSML policy file from the Trusted Policy Repository
- Update an RXSML policy file in the Trusted Policy Repository
- Extract an RXSML policy file from the Trusted Policy Repository
→ List the RXSML policy files in the Trusted Policy Repository
→ Apply an RXSML policy file to a database object
→ Remove the application of an RXSML policy file from a database object
→ List assignments of RXSML policy files to database objects

The following shows the synopsis for the rxpolman command:

```
rxpolman
-1 [-o dbname] (list policies or assignments)
-c filepath   (create new policy)
-u filepath   (update policy)
-d policy     (delete policy)
-e policy     (extract policy)
-a policy -o db[.cat][.sch][.tab] (apply policy to DBMS object)
```

Using the `-l` option without the `--o dbname` option causes all RXSML policy files in the Trusted Policy Repository to be listed. The RXSML policy files are named the same as their top-level element’s PolicySetId or PolicyId attribute. The `rubix.policy.list` authorization is required for this action.

Using the `-l` option with the `--o dbname` option lists all RXSML Policy file database object assignments for the `dbname` database. The `rubix.policy.list` authorization is required for this action.

Using the `--c filepath` option will create a new RXSML policy file in the Trusted Policy Repository. The policy file to be added must be specified by `filepath`. The syntax for the policy file will be checked. Error messages will be produced and the operation will fail if any errors are found. The RXSML policy file will be named the same as the top-level element’s PolicySetId or PolicyId attribute. All PolicySetId and PolicyId attribute values (both top-level and non top-level) must be unique after the new policy file has been added. The `rubix.policy.create` authorization is required for this action.

Using the `--u filepath` option will update an existing RXSML policy file in the Trusted Policy Repository. The new version of the policy file must be specified by `filepath`. The syntax for the policy file will be checked. Error message will be produced and the operation will fail if any errors are found. The RXSML policy file that will be updated will be named the same as the top-level element’s PolicySetId or PolicyId attribute of the file specified by `filepath`. All PolicySetId and PolicyId attribute values (both top-level and non top-level) must be unique after the policy file has been updated. The `rubix.policy.update` authorization is required for this action.

Using the `--d policy` option will delete an existing RXSML policy file from the Trusted Policy Repository. The policy’s name, the value of the top-level element’s PolicySetId or PolicyId attribute, must be specified by `policy`. The policy file must not be applied to any database object or the operation will fail. The `rubix.policy.delete` authorization is required for this action.

Using the `--e policy` option will extract an existing RXSML policy file from the Trusted Policy Repository into the user’s current working directory. The policy’s name, the value of the top-level element’s PolicySetId or PolicyId attribute, must be specified by `policy`. The policy file will be created in the user’s current directory. The `rubix.policy.extract` authorization is required for this action.
Using the `-a policy -o db[.cat][.sch][.tab]` option will apply an RXSML policy file that exists in the Trusted Policy Repository to a database object. The policy’s name, the value of the top-level element’s `PolicySetId` or `PolicyId` attribute, must be specified by `policy`. If `policy` is given as `NULL` then any policy applied to the database object will be unapplied. The database object must be specified by `db[.cat][.sch][.tab]`. If the database object already has policy applied to it the new apply will replace it. The database object need not exist to allow for policy to be defined before an object is added to the database. The `rubix.policy.apply` authorization is required for this action.
3 SECURITY MARKUP LANGUAGE (RXSML)

3.1 XML Elements

3.1.1 <PolicySet>

The <PolicySet> element is a named, top-level element in the RXSML policy schema. Conceptually the policy set is a combined group of child policies along with an associated set of obligations. Child policies may be included within a <PolicySet> element either explicitly (using the <PolicySet> or <Policy> elements) or by reference to their name using the <PolicyIdReference> and <PolicySetIdReference> elements. Policies included within a <PolicySet> element are combined according to the required PolicyCombiningAlgId attribute.

The <PolicySet> element may be evaluated and when evaluated has an outcome of Permit, Deny, Indeterminate, or Not Applicable.

If the <PolicySet> is evaluated the <Target> is evaluated first.
If the <Target> evaluates to Match the remainder of the <PolicySet> is evaluated; otherwise, the <PolicySet> evaluates to Not Applicable (if the <Target> evaluated to No Match) or Indeterminate (if the <Target> evaluated to Indeterminate).
If the remainder of the <PolicySet> is evaluated (that is, if the <Target> evaluated to Match) the final <PolicySet> outcome is calculated by evaluating the child policies, combining their outcomes according to the PolicyCombiningAlgId attribute.
If, during evaluation, a child policy need not be evaluated to determine the final <PolicySet> outcome, that policy is not evaluated.

If the <PolicySet> is evaluated and the final <PolicySet> outcome matches the FulfillOn attribute of any obligations, those obligations are combined with any obligations of evaluated child policies whose FulfillOn attribute matched their outcome. The entire set of obligations is passed to any parent <PolicySet>. Each obligation in the set whose FulfillOn attribute matches the final top-level outcome is executed in the order that its associated policy was evaluated.

The <PolicySet> element may contain the following attributes:

→ PolicySetId [Required; RubixName type]
   A string policy identifier. It may be up to 50 characters long and must be unique across all policies (<PolicySet> and <Policy> elements) in the Trusted Policy Repository.

→ PolicyCombiningAlgId [Required; PolicyCombiningAlgIdType type]
   The identifier of the policy-combining algorithm by which the <PolicySet> will combine child policies. Valid values are
   - deny-overrides,  first-applicable,  ordered-denys-overrides, and
   - permit-overrides,  only-one-applicable,  ordered-permit-overrides.

→ PolicyScope [Optional; default = Node; ScopeType type]
   The identifier of the scope of the policy. Valid values are Node and Subtree. If the value is Node the policy will be applied only to the database object to which it has been explicitly assigned. If the value is Subtree the policy will be applied to the database object it has been explicitly
assigned to and all descendant database objects. The PolicyScope attribute is only meaningful if the <PolicySet> is the top-level element in an assigned policy.

→ **MacOverride** [Optional; default = false; boolean type]

Boolean value indicating whether the Mandatory Access Control (MAC) security policy of the TRUSTED RUBIX database security kernel is enforced prior to enforcing the <PolicySet> policy and all child policies. If the value is false the MAC policy will be enforced. In this mode the <PolicySet> policy is a refining policy and may only act to further refine the MAC security policy. In this case the MAC security policy will enforce Bell-LaPadula rules, perform polyinstantiation of unique objects, and set error codes as suitable to restrict improper information flows. If the value is true then the <PolicySet> policy is a releasability policy and the MAC policy is effectively turned off. In this case all security checks are performed by the <PolicySet> policy. The MacOverride attribute is only meaningful if the <PolicySet> is the top-level element in an assigned policy.

The <PolicySet> element may contain the following elements:

→ **<Description>** [Optional]

A free-form text description of the policy set. This element is ignored during evaluation.

→ **<Target>** [Required]

The <Target> element defines the applicability of <PolicySet> to a set of decision requests for operations on the database objects the policy is associated with. The <Target> element is evaluated and if it evaluates to Match the remainder of the <PolicySet> is evaluated. An empty <Target> means <PolicySet> is applicable to all decision requests for operations on the database objects the policy is associated with and that produce a Match in the <Target> of any parent <PolicySet>.

→ **<PolicySet>** [Any Number]

An explicitly defined <PolicySet> that is a child to this <PolicySet>.

→ **<Policy>** [Any Number]

An explicitly defined <Policy> that is a child to this <PolicySet>.

→ **<PolicySetIdReference>** [Any Number]

The name of a <PolicySet> (PolicySetId attribute of the referred policy) that is to be a child to this <PolicySet>.

→ **<PolicyIdReference>** [Any Number]

The name of a <Policy> (PolicyId attribute of the referred policy) that is to be a child to this <PolicySet>.

→ **<Obligations>** [Optional]

Contains the set of <Obligation> elements that are to be executed if the top-level outcome and the current <PolicySet> outcome equal the FulfillOn attribute of the <Obligation>. The <PolicySet> must be evaluated for the <Obligation> to have the potential to be executed.

### 3.1.2 <Description>

The <Description> element contains a free-form text description of the <PolicySet>, <Policy> or <Rule> element. This element is ignored during evaluation.

### 3.1.3 <Target>

The <Target> element defines the applicability of the <PolicySet>, <Policy>, or <Rule> elements to a set of decision requests for operations on the database object to which the policy is associated. The
<Target> element is required for the <PolicySet> and <Policy> elements and is optional for the <Rule> element. The <Target> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate.

The <Target> element consists of a sequence of <Subjects>, <Resources>, <Actions>, and <Environments> elements. For the parent of the <Target> element to be applicable to the decision request, the <Target> element must evaluate to Match for the given database context. For the <Target> element to evaluate to Match, each child element must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The <Target> element may contain the following elements:

→ <Subjects> [Optional]
Matching specification for the subject attributes in the database context. If this element is missing, then the target matches all subjects.

→ <Resources> [Optional]
Matching specification for the resource attributes in the database context. If this element is missing, then the target matches all resources.

→ <Actions> [Optional]
Matching specification for the action attributes in the database context. If this element is missing, then the target matches all actions.

→ <Environments> [Optional]
Matching specification for the environment attributes in the database context. If this element is missing, then the target matches all environments.

3.1.4 <Subjects>

The <Subjects> element contains a disjunctive sequence of <Subject> elements. The <Subjects> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For <Subjects> to evaluate to Match at least one of the <Subject> elements must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The <Subjects> element contains the following elements:

→ <Subject> [One to Many, Required]
Conjunctive sequence of matching specifications for a set of subject context attributes.

3.1.5 <Subject>

The <Subject> element contains a conjunctive sequence of <SubjectMatch> elements. The <Subject> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For <Subject> to evaluate to Match each <SubjectMatch> element must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The <Subject> element contains the following elements:

→ <SubjectMatch> [One to Many]
A matching specification for a single subject context attribute. The subject context attribute may be a multi-valued bag in which case the <SubjectMatch> evaluates to Match if any value in the bag matches the specification.
3.1.6 <SubjectMatch>

The <SubjectMatch> element contains a matching specification for a single subject context attribute. The <SubjectMatch> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. The subject context attribute may be a multi-valued bag in which case the <SubjectMatch> evaluates to Match if any value in the bag matches the specification. The evaluation is performed by applying the matching function to the literal value specified by the <AttributeValue> element and each value in the subject context attribute bag returned by the <SubjectAttributeDesignator> element. If any matching function execution returns true then the <SubjectMatch> evaluates to Match.

The <SubjectMatch> element contains the following attribute:

\[ \text{MatchId} \] [Required; MatchIdType type]

- Specifies a matching function. Valid values are:
- **equal**, **greater-than**, **less-than**, **greater-than-or-equal**, **less-than-or-equal**, **regexp-match**, **dnsName-match**, and **ipAddress-match**.

The <SubjectMatch> element contains the following elements:

- **<AttributeValue>** [Required]
  - A literal attribute value.
- **<SubjectAttributeDesignator>** [Required]
  - Identifies a subject context attribute bag of values. If the attribute is not available because the current operation does not support it then either Indeterminate or a zero-sized bag is returned, depending on the value of the MustBePresent element attribute.

3.1.7 <Resources>

The <Resources> element contains a disjunctive sequence of <Resource> elements. The <Resources> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For <Resources> to evaluate to Match at least one of the <Resource> elements must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The <Resources> element contains the following elements:

- **<Resource>** [One to Many, Required]
  - Conjunctive sequence of matching specifications for a set of resource context attributes.

3.1.8 <Resource>

The <Resource> element contains a conjunctive sequence of <ResourceMatch> elements. The <Resource> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For <Resource> to evaluate to Match each <ResourceMatch> element must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The <Resource> element contains the following elements:

- **<ResourceMatch>** [One to Many]
  - A matching specification for a single resource context attribute. The resource context attribute
may be a multi-valued bag in which case the `<ResourceMatch>` evaluates to Match if any value in the bag matches the specification.

### 3.1.9 `<ResourceMatch>`

The `<ResourceMatch>` element contains a matching specification for a single resource context attribute. The `<ResourceMatch>` element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. The resource context attribute may be a multi-valued bag in which case the `<ResourceMatch>` evaluates to Match if any value in the bag matches the specification. The evaluation is performed by applying the matching function to the literal value specified by the `<AttributeValue>` element and each value in the resource context attribute bag returned by the `<ResourceAttributeDesignator>` or `<AttributeSelector>` element. If any matching function execution returns true then the `<ResourceMatch>` evaluates to Match.

The `<ResourceMatch>` element contains the following attribute:

- `<MatchId>` [Required; `MatchIdType` type]
  - Specifies a matching function. Valid values are:
    - `equal`, `greater-than`, `less-than`,
    - `not-equal`, `greater-than-or-equal`, `less-than-or-equal`, and `regexp-match`.

The `<ResourceMatch>` element contains the following elements:

- `<AttributeValue>` [Required]
  - A literal attribute value.
- `<ResourceAttributeDesignator>` [Required choice]
  - Identifies a resource context attribute bag of values. If the attribute is not available because the current operation does not support it then either Indeterminate or a zero-sized bag is returned, depending on the value of the `MustBePresent` element attribute.
- `<AttributeSelector>` [Required choice; `RubixPath` type]
  - Identifies a field in the current row being operated upon. If the current operation is not acting upon a row then either Indeterminate or a zero-sized bag is returned, depending on the value of the `MustBePresent` element attribute.

### 3.1.10 `<Actions>`

The `<Actions>` element contains a disjunctive sequence of `<Action>` elements. The `<Actions>` element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For `<Actions>` to evaluate to Match at least one of the `<Action>` elements must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The `<Actions>` element contains the following elements:

- `<Action>` [One to Many, Required]
  - Conjunctive sequence of matching specifications for a set of action context attributes.

### 3.1.11 `<Action>`

The `<Action>` element contains a conjunctive sequence of `<ActionMatch>` elements. The `<Action>` element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For
<Action> to evaluate to Match each <ActionMatch> element must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The <Action> element contains the following elements:

→ <ActionMatch> [One to Many]
   A matching specification for a single action context attribute.

### 3.1.12 <ActionMatch>

The <ActionMatch> element contains a matching specification for a single action context attribute. The <ActionMatch> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. The evaluation is performed by applying the matching function to the literal value specified by the <AttributeValue> element and the value in the action context attribute bag returned by the <SubjectAttributeDesignator> element. If the matching function execution returns true then the <ActionMatch> evaluates to Match.

The <ActionMatch> element contains the following attribute:

→ MatchId [Required; MatchIdType type]
   Specifies a matching function. Valid values are:
   \[\uparrow \text{equal}, \uparrow \text{greater-than}, \uparrow \text{less-than}, \uparrow \text{regexp-match}.
   \uparrow \text{not-equal}, \uparrow \text{greater-than-or-equal}, \uparrow \text{less-than-or-equal}, \text{and}\]

The <ActionMatch> element contains the following elements:

→ <AttributeValue> [Required]
   A literal attribute value.

→ <ActionAttributeDesignator> [Required]
   Identifies an action context attribute bag containing a single value.

### 3.1.13 <Environments>

The <Environments> element contains a disjunctive sequence of <Environment> elements. The <Environments> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For <Environments> to evaluate to Match at least one of the <Environment> elements must evaluate to Match. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The <Environments> element contains the following elements:

→ <Environment> [One to Many, Required]
   Conjunctive sequence of matching specifications for a set of environment context attributes.

### 3.1.14 <Environment>

The <Environment> element contains a conjunctive sequence of <EnvironmentMatch> elements. The <Environment> element may be evaluated and when evaluated has an outcome of Match, NoMatch, or Indeterminate. For <Environment> to evaluate to Match each <EnvironmentMatch> element must
evaluate to *Match*. If, during evaluation, a child element need not be evaluated to determine the final outcome, that element is not evaluated.

The `<Environment>` element contains the following elements:

- `<EnvironmentMatch>` [One to Many]
  - A matching specification for a single environment context attribute.

### 3.1.15 `<EnvironmentMatch>`

The `<EnvironmentMatch>` element contains a matching specification for a single environment context attribute. The `<EnvironmentMatch>` element may be evaluated and when evaluated has an outcome of *Match*, *NoMatch*, or *Indeterminate*. The evaluation is performed by applying the matching function to the literal value specified by the `<AttributeValue>` element and the value in the environment context attribute bag returned by the `<EnvironmentAttributeDesignator>` element. If the matching function execution returns *true* then the `<EnvironmentMatch>` evaluates to *Match*.

The `<EnvironmentMatch>` element contains the following attribute:

- `<MatchId>` [Required; `MatchIdType` type]
  - Specifies a matching function. Valid values are:
    - `equal`, `greater-than`, `less-than`, `regexp-match`, `not-equal`, `greater-than-or-equal`, `less-than-or-equal`, and

The `<EnvironmentMatch>` element contains the following elements:

- `<AttributeValue>` [Required]
  - A literal attribute value.
- `<EnvironmentAttributeDesignator>` [Required]
  - Identifies an environment context attribute bag containing a single value.

### 3.1.16 `<PolicySetIdReference>`

The `<PolicySetIdReference>` element is used to reference a `<PolicySet>` element by id. The `<PolicySetIdReference>` contains the id as string data within the element. The id is a string of type `RubixName` with a maximum length of 50 characters. A `<PolicySet>` element with a matching `PolicySetId` attribute must exist in the Trusted Policy Repository prior to evaluation of the parent `<PolicySet>`. Conceptually the referenced `<PolicySet>` replaces the `<PolicySetIdReference>` element in the parent `<PolicySet>` at the location of the `<PolicySetIdReference>` element during evaluation. The `<PolicySetIdReference>` element allows `<PolicySet>` elements to be referenced multiple times while having a single point of maintenance. Therefore, altering the referenced `<PolicySet>` element changes the evaluation behavior of all parent `<PolicySet>` elements that reference it. Care must be taken not to have a cycle in chained `<PolicySetIdReference>` elements.

### 3.1.17 `<PolicyIdReference>`

The `<PolicyIdReference>` element is used to reference a `<Policy>` element by id. The `<PolicyIdReference>` contains the id as string data within the element. The id is a string of type `RubixName` with a maximum length of 50 characters. A `<Policy>` element with a matching `PolicyId` attribute must exist in the Trusted Policy Repository prior to evaluation of the parent `<PolicySet>`. Conceptually the referenced `<Policy>` replaces the `<PolicyIdReference>` element in the parent
<PolicySet> at the location of the <PolicyIdReference> element during evaluation. The <PolicyIdReference> element allows <Policy> elements to be referenced multiple times while having a single point of maintenance. Therefore, altering the referenced <Policy> element changes the evaluation behavior of all parent <PolicySet> elements that reference it.

3.1.18 <Policy>

The <Policy> element is a named, top-level element in the RXSML policy schema. Conceptually the policy is a combined group of child rules along with an associated set of obligations. <Rule> elements included within a <Policy> element are combined according to the required RuleCombiningAlgId attribute.

The <Policy> element may be evaluated and when evaluated has an outcome of Permit, Deny, Indeterminate, or Not Applicable.

If the <Policy> is evaluated the <Target> is evaluated first.
If the <Target> evaluates to Match the remainder of the <Policy> is evaluated; otherwise, the <Policy> evaluates to Not Applicable (if the <Target> evaluated to No Match) or Indeterminate (if the <Target> evaluated to Indeterminate).
If the remainder of the <Policy> is evaluated (that is, if the <Target> evaluated to Match) the final <Policy> outcome is calculated by evaluating the child rules, combining their outcomes according to the RuleCombiningAlgId attribute.
If, during evaluation, a child rule need not be evaluated to determine the final <Policy> outcome, that rule is not evaluated.
If the <Policy> is evaluated and the final <Policy> outcome matches the FulfillOn attribute of any obligations, those obligations are passed to any parent <PolicySet>. Each obligation in the set whose FulfillOn attribute matches the final top-level outcome is executed in the order that its associated policy was evaluated.

The <Policy> element may contain the following attributes:

→ PolicyId [Required; RubixName type]
  A string policy identifier. It may be up to 50 characters long and must be unique across all policies (<PolicySet> and <Policy> elements) in the Trusted Policy Repository.

→ RuleCombiningAlgId [Required; RuleCombiningAlgIdType type]
  The identifier of the rule-combining algorithm by which the <Policy> will combine child rules. Valid values are:
  ↑ deny-overrides, ↑ ordereddeny-overrides,
  ↑ permit-overrides, and
  ↑ first-applicable, ↑ ordered-permit-overrides.

→ PolicyScope [Optional; default = Node; ScopeType type]
  The identifier of the scope of the policy. Valid values are Node and Subtree. If the value is Node the policy will be applied only to the database object to which it has been explicitly assigned. If the value is Subtree the policy will be applied to the database object it has been explicitly assigned to and all descendant database objects. The PolicyScope attribute is only meaningful if the <Policy> is the top-level element in an assigned policy.
→ **MacOverride** [Optional; default = false; boolean type]

Boolean value indicating whether the Mandatory Access Control (MAC) security policy of the

**Trusted Rubix** database security kernel is enforced prior to enforcing the `<Policy>` and all child rules. If the value is false the MAC policy will be enforced. In this mode the `<Policy>` policy is a refining policy and may only act to further refine the MAC security policy. In this case the MAC security policy will enforce Bell-LaPadula rules, perform polyinstantiation of unique objects, and set error codes as suitable to restrict improper information flows. If the value is true then the `<Policy>` policy is a releasability policy and the MAC policy is effectively turned off. In this case all security checks are performed by the `<Policy>` policy. The MacOverride attribute is only meaningful if the `<Policy>` is the top-level element in an assigned policy.

The `<Policy>` element may contain the following elements:

→ **<Description>** [Optional]

A free-form text description of the policy. This element is ignored during evaluation.

→ **<Target>** [Required]

The `<Target>` element defines the applicability of `<Policy>` to a set of decision requests for operations on the database objects to which the policy is associated. The `<Target>` element is evaluated and if it evaluates to **Match** the remainder of the `<Policy>` is evaluated. An empty `<Target>` means `<Policy>` is applicable to all decision requests for operations on the database objects to which the policy is associated and that produce a **Match** in the `<Target>` of any parent `<PolicySet>`.

→ **<VariableDefinition>** [Any Number]

Common function definitions that can be referenced from anywhere in a rule where an expression can be found.

→ **<Rule>** [Any Number]

A sequence of rules that are combined according to the `RuleCombiningAlgorithm` attribute. Rules whose `<Target>` elements match the decision request are evaluated if needed to determine the `<Policy>` element’s final outcome. Rules whose `<Target>` elements do not match the decision request are ignored.

→ **<Obligations>** [Optional]

Contains the set of `<Obligation>` elements that are to be executed if the top-level outcome and the current `<Policy>` outcome equal the `FulfillOn` attribute of the `<Obligation>`. The `<Policy>` must be evaluated for the `<Obligation>` to have the potential to be executed.

### 3.1.19 `<Rule>`

The `<Rule>` element defines the individual rules of the parent `<Policy>`. Each `<Rule>` element may be evaluated and when evaluated has an outcome of **Permit**, **Deny**, **Indeterminate**, or **Not Applicable**. If the `<Rule>` is evaluated the `<Target>` is evaluated first.

If the `<Target>` evaluates to **Match** the `<Condition>` of the `<Rule>` is evaluated, if it exists.

If the `<Target>` does not evaluate to **Match** the `<Rule>` evaluates to **Not Applicable** (if the `<Target>` evaluated to **No Match** or **Indeterminate** (if the `<Target>` evaluated to **Indeterminate**).

If the entire `<Rule>` is evaluated (that is, if the `<Target>` evaluated to **Match**) the `<Rule>` outcome is determined by evaluating the `<Condition>`, if it exists.

If `<Condition>` does not exist than `<Rule>` implicitly evaluates to **Effect**.

If `<Condition>` does exist then `<Rule>` evaluates to **Effect** if `<Condition>` evaluates to **true**; `<Rule>` evaluates to **Not Applicable** if `<Condition>` evaluates to **false**; and `<Rule>` evaluates to **Indeterminate** if `<Condition>` evaluates to **Indeterminate**.
The `<Rule>` element contains the following attributes:

- **RuleId** [Required; RubixName type]
  A string rule identifier. It may be up to 50 characters long and must be unique across all other `RuleId` attributes in the parent `<Policy>`.

- **Effect** [Required; EffectType type]
  Specifies the value the `<Rule>` evaluates to if the `<Condition>` explicitly or implicitly evaluates to `true`. Valid values of this attribute are `Permit` and `Deny`.

The `<Rule>` element may contain the following elements:

- **<Description>** [Optional]
  A free-form text description of the rule. This element is ignored during evaluation.

- **<Target>** [Optional]
  The `<Target>` element defines the applicability of `<Rule>` to a set of decision requests for operations on the database objects to which the policy is associated. The `<Target>` element is evaluated and if it evaluates to `Match` the remainder of the `<Rule>` is evaluated. An empty or non-existing `<Target>` means `<Rule>` is applicable to all decision requests for operations on the database objects to which the policy is associated and that produce a `Match` in the `<Target>` of the parent `<Policy>`.

- **<Condition>** [Optional]
  A predicate expression that must evaluate to `true` for the rule to evaluate to its `Effect` value. If an existing `<Condition>` evaluates to `false` the `<Rule>` evaluates to `Not Applicable`. The predicate consists of a single expression that returns a boolean value. A non-existing `<Condition>` element results in the `<Rule>` implicitly evaluating to its `Effect`.

### 3.1.20 `<VariableDefinition>`

The `<VariableDefinition>` element may be used to define a value that can be referenced by `<VariableReference>` elements within a single `<Policy>`. The value may be of any valid data type and is defined by the enclosed `<Expression>` complex type. There may be multiple references to each `<VariableDefinition>` element. Using the `<VariableDefinition>` and `<VariableReference>` elements provides the advantage of having a single point of maintenance for expressions that are frequently used. It may also provide a performance improvement as the `<VariableDefinition>` need to be evaluated only once for a given `<Policy>`, regardless of the number of times it is referenced.

The `<VariableDefinition>` element contains the following attribute:

- **VariableId** [Required; RubixName type]
  The name of the variable definition. It may be up to 50 characters long and must be unique across all other `<VariableDefinition>` elements within the parent `<Policy>`.

The `<VariableDefinition>` element contains the following element:

- **<Expression>** [Required]
  Any element of `ExpressionType` complex type. The following elements are in the `<Expression>` element substitution group:

  - `<Apply>`
  - `<AttributeSelector>`
  - `<AttributeValue>`
  - `<VariableReference>`
  - `<ImportColumnSelector>`
  - `<SubjectAttributeDesignator>`
  - `<ResourceAttributeDesignator>`
  - `<ActionCodeDesignator>`
  - `<ErrorCode>`, and
  - `<Function>`.
3.1.21 <VariableReference>

The <VariableReference> element is used to reference a value defined by a <VariableDefinition> element within the parent <Policy>. The <VariableReference> element may be evaluated and when evaluated has an outcome of Indeterminate or specific typed value(s). The type of the values is determined by the enclosed <Expression> element of the referenced <VariableDefinition>. The <VariableReference> element refers to the <VariableDefinition> element by having equal VariableId attributes. Multiple <VariableReference> elements may refer to the same <VariableDefinition> element. The <VariableReference> element is a member of the <Expression> element substitution group and may appear any place an <Expression> element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. Using the <VariableDefinition> and <VariableReference> elements provides the advantage of having a single point of maintenance for expressions that are frequently used. It may also provide a performance improvement as the <VariableDefinition> need to be evaluated only once for a given <Policy>, regardless of the number of times it is referenced.

The <VariableReference> element contains the following attribute:

- **VariableId** [Required; RubixName type]
  The name used to refer to the value defined in a <VariableDefinition> element. It may be up to 50 characters long and the referenced <VariableDefinition> element must exist in the parent <Policy>.

3.1.22 <Condition>

The <Condition> element is a Boolean function over subject, resource, action and environment attributes or functions of attributes. The <Condition> element may be evaluated and when evaluated has an outcome of Indeterminate, true, or false.

The <Condition> contains the following element:

- **Expression** [Required; restricted to boolean return values]
  Any element of ExpressionType complex type that returns a single boolean data type. The following elements are in the <Expression> element substitution group and may return single boolean values:
    - <Apply>, <AttributeValue>, and <VariableReference>.

3.1.23 <Apply>

The <Apply> element specifies the application of a function to its arguments. It duplicates the semantics of a function call in that it applies a function to a group of arguments and returns a value. The <Apply> element may be evaluated and when evaluated has an outcome of Indeterminate or specific typed values. The type of the values is dependent upon the function that is being applied. The function being applied is specified by the FunctionId attribute. The <Apply> element can be applied to any combination of the members of the <Expression> element substitution group that is consistent with the required arguments for the function specified by the FunctionId attribute. The <Apply> element is a member of the
<Expression> element substitution group and may appear any place an <Expression> element occurs in
the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the
parent element.

The <Apply> element contains the following attributes and elements:

→ **FunctionId** [Required; FunctionIdType type]
  The identifier of the function to be applied to the arguments. Valid values are:
  
  ↑ equal, ↑ subtract, ↑ and, ↑ all-of-any,
  ↑ greater-than, ↑ multiply, ↑ or, ↑ any-of-all,
  ↑ greater-than-or-equal, ↑ divide, ↑ not, ↑ all-of-all,
  ↑ less-than, ↑ abs, ↑ n-of, ↑ map,
  ↑ less-than-or-equal, ↑ round, ↑ one-and-only, ↑ intersection,
  ↑ regexp-match, ↑ floor, ↑ bag-size, ↑ at-least-one-
  ↑ rfc822Name-match, ↑ string-normalize-
  ↑ dnsName-match, ↑ string-normalize-
  ↑ ipAddress-match, ↑ to-lower-case, ↑ member-of,
  ↑ MAC-check, ↑ cast, ↑ bag,
  ↑ add, ↑ concatenate, ↑ union,

The <Apply> element may contain the following elements:

→ **Expression** [Optional]
  Arguments to the function, which may include other functions. The number and type of required
  arguments are specific to the function specified by the FunctionId attribute. The following
  elements are in the <Expression> element substitution group:
  
  ↑ <Apply>
  ↑ <AttributeSelector>
  ↑ <AttributeValue>
  ↑ <VariableReference>
  ↑ <ImportColumnSelector>
  ↑ <ImportFieldSelector>
  ↑ <SubjectAttributeDesignator>
  ↑ <ResourceAttributeDesignator>
  ↑ <ActionCodeDesignator>
  ↑ <EnvironmentAttributeDesignator>
  ↑ <DataTypeDesignator>

3.1.24 <Function>

The <Function> element is used to identify a two argument predicate function as an argument to a
higher-order bag function (any-of, all-of, any-of-any, all-of-any, any-of-all, all-of-all). The
.DataTypeDesignator> element is a member of the <Expression> element substitution group but may
only occur as an argument to a higher-order bag function.

The <Function> element contains the following attribute:

→ **FunctionId** [Required; Restricted FunctionIdType type]
  The identifier of the predicate function. Valid predicate functions are:
  
  ↑ equal, ↑ less-than, ↑ rfc822Name-match,
  ↑ greater-than, ↑ less-than-or-equal, ↑ dnsName-match, and
  ↑ greater-than-or-equal, ↑ regexp-match, ↑ ipAddress-match.
3.1.25<SubjectAttributeDesignator>

The <SubjectAttributeDesignator> element retrieves a bag of values for a named subject context attribute specified by the AttributeId attribute. The <SubjectAttributeDesignator> element may be evaluated and when evaluated has an outcome of Indeterminate or a bag of specific typed values. The <SubjectAttributeDesignator> element is a member of the <Expression> element substitution group and may appear any place an <Expression> element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The type of the values is dependent upon the subject context attribute specified by the AttributeId attribute. In the event that the specified subject context attribute is not available in the context of the decision request, the MustBePresent attribute governs whether this element returns an empty bag or Indeterminate. Currently all subject context attributes are available for all database operations; therefore, the MustBePresent attribute is not meaningful. However, in the future this may change.

The <SubjectAttributeDesignator> may contain the following attributes:

→ **AttributeId** [Required; SubjectAttributeIdType type]
  This attribute specifies the subject context attribute. Valid values are:
  
  ‣ subject-id,
  ‣ subject-name,
  ‣ group-id,
  ‣ group-name,
  ‣ session-start-time,
  ‣ session-start-date,
  ‣ session-start-dateTime,
  ‣ application-name,
  ‣ application-user-name,
  ‣ ip-address,
  ‣ dns-name,
  ‣ application-user-id,
  ‣ session-label.

→ **MustBePresent** [Optional; boolean type; default is false]
  This attribute governs whether the element returns Indeterminate (MustBePresent is true) or an empty bag (MustBePresent is false) if the specified subject context attribute is not available in the context of the decision request. Currently all subject context attributes are available for all database operations; therefore, the MustBePresent attribute is not meaningful. However, in the future this may change.

3.1.26<ResourceAttributeDesignator>

The <ResourceAttributeDesignator> element retrieves a bag of values for a named resource context attribute specified by the AttributeId attribute. The <ResourceAttributeDesignator> element may be evaluated and when evaluated has an outcome of Indeterminate or a bag of specific typed values. The <ResourceAttributeDesignator> element is a member of the <Expression> element substitution group and may appear any place an <Expression> element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The type of the values is dependent upon the resource context attribute specified by the AttributeId attribute. In the event that the specified resource context attribute is not available in the context of the decision request, the MustBePresent attribute governs whether this element returns an empty bag or Indeterminate. As an example, the row-label resource context attribute is not available in the context of a decision request for a database-open operation.

The <ResourceAttributeDesignator> may contain the following attributes:

→ **AttributeId** [Required; ResourceAttributeIdType type]
  This attribute specifies the resource context attribute. Valid values are:
  
  ‣ resource-label,
  ‣ resource-name,
  ‣ row-label,
  ‣ schema-label,
  ‣ catalog-label,
  ‣ database-label,
  ‣ view-name,
  ‣ schema-name,
  ‣ catalog-name,


3.1.27 <ActionAttributeDesignator>

The <ActionAttributeDesignator> element retrieves a bag of values for a named action context attribute specified by the AttributeId attribute. The <ActionAttributeDesignator> element may be evaluated and when evaluated has an outcome of Indeterminate or a bag of specific typed values. The <ActionAttributeDesignator> element is a member of the <Expression> element substitution group and may appear anywhere an <Expression> element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The type of the values is dependent upon the action context attribute specified by the AttributeId attribute. In the event that the specified action context attribute is not available in the context of the decision request, the MustBePresent attribute governs whether this element returns an empty bag or Indeterminate. Currently all action context attributes are available for all database operations; therefore, the MustBePresent attribute is not meaningful. However, in the future this may change.

The <ActionAttributeDesignator> may contain the following attributes:

- **AttributeId** [Required; ActionAttributeIdType type]
  This attribute specifies the action context attribute. Valid values are action-id and action-type.

- **MustBePresent** [Optional; boolean type; default is false]
  This attribute governs whether the element returns Indeterminate (MustBePresent is true) or an empty bag (MustBePresent is false) if the specified resource context attribute is not available in the context of the decision request.

3.1.28 <EnvironmentAttributeDesignator>

The <EnvironmentAttributeDesignator> element retrieves a bag of values for a named environment context attribute specified by the AttributeId attribute. The <EnvironmentAttributeDesignator> element may be evaluated and when evaluated has an outcome of Indeterminate or a bag of specific typed values. The <EnvironmentAttributeDesignator> element is a member of the <Expression> element substitution group and may appear anywhere an <Expression> element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The type of the values is dependent upon the environment context attribute specified by the AttributeId attribute. In the event that the specified environment context attribute is not available in the context of the decision request, the MustBePresent attribute governs whether this element returns an empty bag or Indeterminate. Currently all environment context attributes are available for all database operations; therefore, the MustBePresent attribute is not meaningful. However, in the future this may change.

The <EnvironmentAttributeDesignator> may contain the following attributes:
AttributeId [Required; EnvironmentAttributeIdType type]
This attribute specifies the environment context attribute. Valid values are current-time, current-date, and current-dateTime.

MustBePresent [Optional; boolean type; default is false]
This attribute governs whether the element returns Indeterminate (MustBePresent is true) or an empty bag (MustBePresent is false) if the specified environment context attribute is not available in the context of the decision request. Currently all environment context attributes are available for all database operations; therefore, the MustBePresent attribute is not meaningful. However, in the future this may change.

3.1.29 <AttributeSelector>
The <AttributeSelector> element retrieves a single field context attribute from the database row currently being executed upon. The <AttributeSelector> element may be evaluated and when evaluated has an outcome of Indeterminate or a typed value. The <AttributeSelector> element is a member of the <Expression> element substitution group and may appear any place an <Expression> element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The RequestContextPath attribute specifies which database field context attribute should be retrieved. In the event that the specified database field context attribute is not available in the context of the decision request, the MustBePresent attribute governs whether this element returns an empty bag or Indeterminate. The database field context attribute may not be present in the context of the decision request because the current database operation is not a row based operation or because the path given by the RequestContextPath attribute is not currently being operated upon. The data type of the resultant typed value is specified by the DataType attribute and must be chosen to be compatible with the SQL data type of the specified field.

The <AttributeSelector> element contains the following attributes:

RequestContextPath [Required; RubixPath type]
A string based database path to the field attribute being retrieved. The path is of the form database_name.catalog_name.schema_name.table_name.column_name where each name component is limited to 50 characters.

DataType [Required; DataTypeType type]
The data type of the value to be returned by the <AttributeSelector> element. The RXSML data type must be chosen to be compatible with the SQL data type of the database field being retrieved. If necessary, casting will occur when converting from the SQL data type into the RXSML data type. Valid values for the DataType attribute are:

- string,
- decimal,
- dateTimeDuration,
- rfc822Name,
- boolean,
- time,
- yearMonthDuration,
- ipAddress,
- integer,
- date,
- hexBinary,
- dnsName, and
- double,
- dateTime,
- base64Binary,
- label.

MustBePresent [Optional; boolean type; default is false]
This attribute governs whether the element returns Indeterminate (MustBePresent is true) or an empty bag (MustBePresent is false) if the specified database field attribute is not available in the context of the decision request.
3.1.30 <AttributeValue>

The `<AttributeValue>` element is used to specify a literal attribute value. The `<AttributeValue>` element may be evaluated and when evaluated has an outcome of Indeterminate or a typed value. The `<AttributeValue>` element is a member of the `<Expression>` element substitution group and may appear any place an `<Expression>` element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The value is specified by the text representation of the value as data within the `<AttributeValue>` element. The data type of the resultant typed value is specified by the `DataType` attribute.

The `<AttributeValue>` element contains the following attribute:

- **DataType** [Required; `DataTypeType` type]
  The data type of the value to be returned by the `<AttributeValue>` element. The text representation of the value must correspond to the data type of the value. Valid values for the `DataType` attribute are:
  - `string`, `decimal`, `dayTimeDuration`, `rfc822Name`, `ipAddress`, `dnsName`, `label`.
  - `boolean`, `time`, `yearMonthDuration`, `ipAddress`, `dnsName`.
  - `integer`, `date`, `hexBinary`, `base64Binary`.
  - `double`, `dateTime`.

3.1.31 <Obligations>

The `<Obligations>` element contains a set of `<Obligation>` elements.

The `<Obligations>` element contains the following element:

- **<Obligation>** [One to Many]
  A sequence of obligations.

3.1.32 <Obligation>

The `<Obligation>` element is used to specify an action that is preformed conditionally on the outcome effect of the parent `<PolicySet>` or `<Policy>` element. The `FulfillOn` attribute specifies the effect for which the obligation will be performed. The `ObligationId` attribute specifies the action performed. It also defines which `<Expression>` arguments are required as children to the `<Obligation>` element. An obligation’s action is performed if the parent `<Policy>` or `<PolicySet>` element is executed, the parent `<Policy>` or `<PolicySet>` element evaluates to an outcome effect equal to the effect specified by the `FulfillOn` attribute, and the final top-level outcome effect is equal to the effect specified by the `FulfillOn` attribute. Each such matching obligation from all evaluated `<Policy>` and `<PolicySet>` elements is executed in the order that their associated policy was evaluated. All obligations are executed and if any obligation execution is not successful for any reason the permission to perform the database operation is denied.

The `<Obligation>` element contains the following attributes:

- **ObligationId** [Required; `ObligationIdType` type]
  The action performed. Valid values of this attribute are `set-field`, `set-error-code`, and `audit`.
- **FulfillOn** [Required; `EffectType` type]
  The effect for which the obligation will be performed. Valid values of this attribute are `Permit` and `Deny`. 

© 2014 Infosystems Technology, Inc.
The \(<Obligation>\) element contains the following elements:

- \(<Expression>\) [Optional]
  Arguments to the obligation. The number and type of required arguments are specific to the obligation specified by the \(ObligationId\) attribute. The following elements are in the \(<Expression>\) element substitution group and may be used as arguments to an obligation:

  - \(<Apply>\)
  - \(<AttributeSelector>\)
  - \(<AttributeValue>\)
  - \(<SubjectAttributeDesignator>\)
  - \(<ResourceAttributeDesignator>\)
  - \(<ActionAttributeDesignator>\)
  - \(<EnvironmentAttributeDesignator>\)
  - \(<ErrorCode>\)

### 3.1.33 \(<DataTypeDesignator>\)

The \(<DataTypeDesignator>\) element is be used to specify a data type as an argument to the \(cast\) function. The \(<DataTypeDesignator>\) element is a member of the \(<Expression>\) element substitution group but may only occur as an argument to the \(cast\) function.

The \(<DataTypeDesignator>\) element contains the following attribute:

- \(DataType\) [Required; \(DataTypeType\) type]
  The data type to be used as an argument for the \(cast\) function. Valid values for the \(DataType\) attribute are:

  - \(string\)
  - \(decimal\)
  - \(time\)
  - \(date\)
  - \(dateTime\)
  - \(dayTimeDuration\)
  - \(yearMonthDuration\)
  - \(hexBinary\)
  - \(base64Binary\)
  - \(rfc822Name\)
  - \(ipAddress\)
  - \(dnsName\)
  - \(label\).

### 3.1.34 \(<ImportColumnSelector>\)

The \(<ImportColumnSelector>\) element retrieves a subset of any database table’s column fields as a bag of typed values. The \(<ImportColumnSelector>\) element may be evaluated and when evaluated has an outcome of \(Indeterminate\) or a bag of typed value. The \(<ImportColumnSelector>\) element is a member of the \(<Expression>\) element substitution group and may appear any place an \(<Expression>\) element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The \(RequestContextPath\) attribute specifies which table’s column fields should be retrieved. In the event that the specified table column does not exist the \(MustBePresent\) attribute governs whether this element returns an empty bag or \(Indeterminate\). The data type of the resultant bag of values is specified by the \(DataType\) attribute and must be chosen to be compatible with the SQL data type of the specified column. Nested occurrences of the \(<ImportColumnSelector>\) element are disallowed and will produce non-determinable results.

The \(<ImportColumnSelector>\) element allows the values (fields) of any database table column to be used in policy decisions. All relevant fields are retrieved to main memory upon evaluation of the \(<ImportColumnSelector>\) element. To limit the number of fields corresponding to a table column that are imported into the policy, a predicate expression that may operate on an \(<ImportFieldSelector>\) element is used to filter the number of fields. During evaluation of the \(<ImportColumnSelector>\) element each row in the specified table is iterated over and the predicate expression is applied. If the predicate evaluates to \(true\) for a given iteration then the corresponding field is added to the resultant bag of typed values. The \(<ImportFieldSelector>\) element may be used as an argument to the predicate expression to retrieve the value of any field in the table row of the current iteration.
The `<ImportColumnSelector>` element contains the following attributes:

- **RequestContextPath** [Required; RubixPath type]
  A string based database path specifying any database table column. The path is of the form `database_name.catalog_name.schema_name.table_name.column_name` where each name component is limited to 50 characters.

- **DataType** [Required; DataTypeType type]
  The data type of the bag of values to be returned by the `<ImportColumnSelector>` element. The RXSML data type must be chosen to be compatible with the SQL data type of the database column being retrieved. If necessary, casting will occur when converting from the SQL data type into the RXSML data type. Valid values for the `DataType` attribute are:
  - `string`
  - `boolean`
  - `integer`
  - `double`
  - `decimal`
  - `time`
  - `date`
  - `dateTime`
  - `dayTimeDuration`
  - `yearMonthDuration`
  - `hexBinary`
  - `base64Binary`
  - `rfc822Name`
  - `ipAddress`
  - `dnsName`
  - `label`.

- **MustBePresent** [Optional; boolean type; default is false]
  This attribute governs whether the element returns `Indeterminate` (`MustBePresent` is true) or an empty bag (`MustBePresent` is false) if the specified table column does not exist.

The `<ImportColumnSelector>` element contains the following element:

- **<Expression>** [Required; restricted to boolean return values]
  Any element of `ExpressionType` complex type that returns a single `boolean` data type. The expression is evaluated for each iteration through the database table and if the expression evaluates to `true` for the given iteration then the corresponding field is added to the resultant bag of values. It is anticipated, but not required, that the `<ImportFieldSelector>` will be used as an argument to the expression so as to filter which fields should be retrieved. The following elements are in the `<Expression>` element substitution group and may return single boolean values:
  - `<Apply>`
  - `<AttributeValue>`
  - `<VariableReference>`.

### 3.1.35 `<ImportFieldSelector>`

The `<ImportFieldSelector>` element is used to retrieve a single database field from the current row being iterated upon during the evaluation of a `<ImportColumnSelector>` element. The `<ImportFieldSelector>` element may be evaluated and when evaluated has an outcome of `Indeterminate` or a typed value. The `<ImportFieldSelector>` element is a member of the `<Expression>` element substitution group and may appear any place an `<Expression>` element occurs in the schema as long as the return value satisfies any restrictions (in data type and bag size) imposed by the parent element. The `<ImportFieldSelector>` is only meaningful if it is enclosed within an `<ImportColumnSelector>` element. It is anticipated that the `<ImportFieldSelector>` will be used as an argument to a function in filtering unwanted fields during the evaluation of the `<ImportColumnSelector>` element. The `RequestContextPath` attribute specifies which field should be retrieved from the current row being iterated upon. In the event that the specified table column does not exist in the current row being iterated upon during the evaluation of a `<ImportColumnSelector>` element or if the `<ImportFieldSelector>` is not enclosed within an `<ImportColumnSelector>` element the `MustBePresent` attribute governs whether this element returns an empty bag or `Indeterminate`. The data type of the resultant typed value is specified by the `DataType` attribute and must be chosen to be compatible with the SQL data type of the specified field.
The `<ImportFieldSelector>` element contains the following attributes:

- **RequestContextPath** [Required; RubixPath type]
  A string based database path specifying any database table field from the current row being iterated upon during the evaluation of a `<ImportColumnSelector>` element. The path is of the form `database_name.catalog_name.schema_name.table_name.column_name` where each name component is limited to 50 characters.

- **DataType** [Required; DataTypeType type]
  The data type of the value to be returned by the `<ImportFieldSelector>` element. The RXSML data type must be chosen to be compatible with the SQL data type of the database column being retrieved. If necessary, casting will occur when converting from the SQL data type into the RXSML data type. Valid values for the `DataType` attribute are:

  - `string`
  - `decimal`
  - `dayTimeDuration`
  - ` RFC822Name`
  - `double`
  - `time`
  - `yearMonthDuration`
  - `ipAddress`
  - `integer`
  - `date`
  - `hexBinary`
  - `dnsName`
  - ` MustBePresent` [Optional; boolean type; default is `false`]
    This attribute governs whether the element returns `Indeterminate` (MustBePresent is `true`) or an empty bag (MustBePresent is `false`) if the specified table field does not exist in the current row being iterated upon during the evaluation of a `<ImportColumnSelector>` element or if the `<ImportFieldSelector>` is not enclosed within an `<ImportColumnSelector>` element.

### 3.1.36 `<ErrorCode>`

The `<ErrorCode>` element is used to specify an error code as an argument to the `set-error-code` obligation. The `<ErrorCode>` element is a member of the `<Expression>` element substitution group but may only occur as an argument to the `set-error-code` obligation.

The `<ErrorCode>` element contains the following attribute:

- **ErrorCodeId** [Required; ErrorCodeIdType type]
  The error code to be used as an argument for the `set-error-code` obligation. Valid values for the `ErrorCodeId` attribute are `object-not-found` and `policy-violation`.

### 3.2 Functions

*Functions* are operations that take some number of arguments and evaluate to (or return) a typed value or bag of typed values. Function names are used as attributes to the `<Apply>`, `<SubjectMatch>`, `<ResourceMatch>`, `<ActionMatch>`, `<EnvironmentMatch>`, `<Function>` elements. In use, care must be taken that the arguments to each function are correct with respect to data type, bag size (single or multiple values), and number. Functions automatically produce resultant values of a type consistent with its argument types, given the functionality being performed. Restrictions on arguments and resultant data types are specified below for each function. Violations of the argument restrictions and run-time processing errors (e.g., mathematical overflow) result in the function evaluating to `Indeterminate`.
3.2.1 equal

The \textit{equal} function takes two single valued arguments of the same data type. It returns a \textit{boolean} typed value that is \textit{true} if the two arguments are equal; otherwise, it returns \textit{false}. All data types are acceptable as arguments.

3.2.2 not-equal

The \textit{equal} function takes two single valued arguments of the same data type. It returns a \textit{boolean} typed value that is \textit{true} if the two arguments are not equal according to the \textit{equal} function; otherwise, it returns \textit{false}. All data types are acceptable as arguments.

3.2.3 greater-than

The \textit{greater-than} function takes two single valued arguments of the same data type. It returns a \textit{boolean} typed value that is \textit{true} if the first argument is greater than the second argument; otherwise, it returns \textit{false}. Acceptable data types are:

\begin{itemize}
  \item string,
  \item decimal,
  \item double,
  \item integer,
  \item date,
  \item time,
  \item dateTime,
  \item yearMonthDuration,
  \item dayTimeDuration,
  \item label.
\end{itemize}

3.2.4 greater-than-or-equal

The \textit{greater-than-or-equal} function takes two single valued arguments of the same data type. It returns a \textit{boolean} typed value that is \textit{true} if the first argument is greater than or equal to the second argument; otherwise, it returns \textit{false}. Acceptable data types are:

\begin{itemize}
  \item string,
  \item decimal,
  \item double,
  \item integer,
  \item date,
  \item time,
  \item dateTime,
  \item yearMonthDuration,
  \item dayTimeDuration,
  \item label.
\end{itemize}

3.2.5 less-than

The \textit{less-than} function takes two single valued arguments of the same data type. It returns a \textit{boolean} typed value that is \textit{true} if the first argument is less than the second argument; otherwise, it returns \textit{false}. Acceptable data types are:

\begin{itemize}
  \item string,
  \item decimal,
  \item double,
  \item integer,
  \item date,
  \item time,
  \item dateTime,
  \item yearMonthDuration,
  \item dayTimeDuration,
  \item label.
\end{itemize}
3.2.6 less-than-or-equal

The less-than-or-equal function takes two single valued arguments of the same data type. It returns a boolean typed value that is true if the first argument is less than or equal to the second argument; otherwise, it returns false. Acceptable data types are:

- → string,    → date,    → yearMonthDuration,
- → decimal,   → time,    → dayTimeDuration
- → double,    → dateTime, and
- → integer,   → label.

3.2.7 time-in-range

The time-in-range function takes three single valued arguments of type time. It returns a boolean typed value that is true if the first argument is inclusively within the time range that starts with the second argument and ends with the third argument; otherwise it returns false. Regardless of its value, the third argument is interpreted as a time that is equal to, or later than by less than twenty-four hours, the second argument.

3.2.8 dnsName-match

The dnsName-match function takes two arguments of type dnsName. It returns a boolean typed value that is true if the first argument matches the second argument; otherwise, it returns false. The first argument matches the second argument if the set of names indicated by the first argument is a subset of the names indicated by the second argument and the entire port range indicated by the first argument is contained within port range indicated by the second argument. Any name component of the first or second argument may be equal to the '*' wildcard character which represents all possible values for that component. If the initial (left-most) component equals the wildcard character then it matches all values for that component (including no component) as well as all possible additional components that could be placed on the far-left. For example, *.rubix.com matches rubix.com, oak.green.rubix.com, and green.rubix.com.

3.2.9 ipAddress-match

The ipAddress-match function takes two arguments of type ipAddress. It returns a boolean typed value that is true if the first argument matches the second argument; otherwise, it returns false. The first argument matches the second argument if the set of network addresses indicated by the first argument is a subset of the network addresses indicated by the second argument and the entire port range indicated by the first argument is contained within the port range indicated by the second argument. Any component of the first or second argument may be equal to the '*' wildcard character which represents all possible values for that component.

3.2.10 MAC-check

The MAC-check function performs an operating system Mandatory Access Control (OS-MAC) security access check for the current operation. This includes both the Multilevel Security (OS-MAC MLS) and Type Enforcement (OS-MAC TE, SELinux only) policies. The MAC-check function has no arguments. It returns a boolean typed value that is true if the operation is allowed according to the MAC security policies; otherwise, it returns false. If both the OS-MAC MLS and OS-MAC TE policies exist, then both policies must permit the operation for the MAC-check to return true. The exact behavior of the MAC security policy
depends on the configuration of the MAC security policy of the trusted operating system hosting the
**Trusted Rubix** server. The MAC-check function also has a “built-in” obligation to set the error
code to a value appropriate for a failed MAC policy check, if the function returns false. The general
purpose of setting the error code is to hide the existence of objects with labels that are not dominated by
the session label.

It is important to note that the MAC policy check is only performed on the primary object for the operation
and not any parent objects. For instance, during the `table-open` operation a MAC check is performed
against the session label and the table label. It is not performed against the containing schema, catalog, or
database label. If MAC checks are required for those objects they either must be performed during the
corresponding `open` operation or explicitly checked by RXSML policy logic using the appropriate context
attributes.

The MAC-check function is only useful for policies that execute in MAC override mode. When policies do
not operate in MAC override mode a MAC check is performed before the RXSML policy is evaluated. The
following table summarizes the MLS behavior of the MAC-check function. The Type Enforcement (TE)
behaviour is dependent upon the specific SELinux policy configured for the system.

### **mac-check MLS Function Behavior Summary**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Primary Object</th>
<th>Session &amp; Object Label Relationship</th>
<th>Return</th>
<th>SQL Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>row-select</code></td>
<td><code>row</code></td>
<td>Session label dominates object label</td>
<td><code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>row-select</code></td>
<td><code>row</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Row does not exist</td>
</tr>
<tr>
<td><code>row-insert</code></td>
<td><code>table</code></td>
<td>Session label dominates object label</td>
<td><code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>row-insert</code></td>
<td><code>table</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Table does not exist</td>
</tr>
<tr>
<td><code>row-update</code></td>
<td><code>row</code></td>
<td>Session label equals object label</td>
<td><code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>row-update</code></td>
<td><code>row</code></td>
<td>Session label strictly dominates object label</td>
<td><code>false</code></td>
<td>Session label must equal object label</td>
</tr>
<tr>
<td><code>row-update</code></td>
<td><code>row</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Row does not exist</td>
</tr>
<tr>
<td><code>row-delete</code></td>
<td><code>row</code></td>
<td>Session label equals object label</td>
<td><code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>row-delete</code></td>
<td><code>row</code></td>
<td>Session label strictly dominates object label</td>
<td><code>false</code></td>
<td>Session label must equal object label</td>
</tr>
<tr>
<td><code>row-delete</code></td>
<td><code>row</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Row does not exist</td>
</tr>
<tr>
<td><code>index-create</code></td>
<td><code>table</code></td>
<td>Session label strictly dominates object label</td>
<td><code>false</code></td>
<td>Session label must equal object label</td>
</tr>
<tr>
<td><code>index-create</code></td>
<td><code>table</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Table does not exist</td>
</tr>
<tr>
<td><code>index-drop</code></td>
<td><code>table</code></td>
<td>Session label equals object label</td>
<td><code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>index-drop</code></td>
<td><code>table</code></td>
<td>Session label strictly dominates object label</td>
<td><code>false</code></td>
<td>Session label must equal object label</td>
</tr>
<tr>
<td><code>index-drop</code></td>
<td><code>table</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Table does not exist</td>
</tr>
<tr>
<td><code>table-open</code></td>
<td><code>table</code></td>
<td>Session label dominates object label</td>
<td><code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>table-open</code></td>
<td><code>table</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Table does not exist</td>
</tr>
<tr>
<td><code>table-create</code></td>
<td><code>schema</code></td>
<td>Session label dominates object label</td>
<td><code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>table-create</code></td>
<td><code>schema</code></td>
<td>Session label does not dominate object label</td>
<td><code>false</code></td>
<td>Schema does not exist</td>
</tr>
<tr>
<td><code>table-drop</code></td>
<td><code>table</code></td>
<td>Session label equals object label</td>
<td><code>true</code></td>
<td></td>
</tr>
</tbody>
</table>
### mac-check MLS FUNCTION BEHAVIOR SUMMARY

<table>
<thead>
<tr>
<th>Operation</th>
<th>Primary Object</th>
<th>Session &amp; Object Label Relationship</th>
<th>Return</th>
<th>SQL Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>table-drop</td>
<td>table</td>
<td>Session label strictly dominates object label</td>
<td>false</td>
<td>Session label must equal object label</td>
</tr>
<tr>
<td>table-drop</td>
<td>table</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>Table does not exist</td>
</tr>
<tr>
<td>table-alter</td>
<td>table</td>
<td>Session label equals object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>table-alter</td>
<td>table</td>
<td>Session label strictly dominates object label</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>table-alter</td>
<td>table</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>view-create</td>
<td>schema</td>
<td>Session label dominates object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>view-create</td>
<td>schema</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>Schema does not exist</td>
</tr>
<tr>
<td>view-drop</td>
<td>view</td>
<td>Session label equals object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>view-drop</td>
<td>view</td>
<td>Session label strictly dominates object label</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>view-drop</td>
<td>view</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>View does not exist</td>
</tr>
<tr>
<td>view-open</td>
<td>view</td>
<td>Session label dominates object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>schema-open</td>
<td>schema</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>View does not exist</td>
</tr>
<tr>
<td>schema-open</td>
<td>schema</td>
<td>Session label equals object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>schema-open</td>
<td>schema</td>
<td>Session label strictly dominates object label</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>schema-drop</td>
<td>schema</td>
<td>Session label dominates object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>schema-drop</td>
<td>schema</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>Schema does not exist</td>
</tr>
<tr>
<td>schema-drop</td>
<td>schema</td>
<td>Session label strictly dominates object label</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>catalog-open</td>
<td>catalog</td>
<td>Session label dominates object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>catalog-open</td>
<td>catalog</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>Catalog does not exist</td>
</tr>
<tr>
<td>catalog-create</td>
<td>database</td>
<td>Session label dominates object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>catalog-create</td>
<td>database</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>Database does not exist</td>
</tr>
<tr>
<td>catalog-drop</td>
<td>catalog</td>
<td>Session label equals object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>catalog-drop</td>
<td>catalog</td>
<td>Session label strictly dominates object label</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>database-open</td>
<td>database</td>
<td>Session label dominates object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>database-open</td>
<td>database</td>
<td>Session label does not dominate object label</td>
<td>false</td>
<td>Database does not exist</td>
</tr>
<tr>
<td>database-drop</td>
<td>database</td>
<td>Session label equals object label</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>database-drop</td>
<td>database</td>
<td>Session label strictly dominates object label</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>database-drop</td>
<td>database</td>
<td>Subject label does not dominate object label</td>
<td>false</td>
<td>Database does not exist</td>
</tr>
</tbody>
</table>
3.2.11 regexp-match

The `regexp-match` function takes two arguments. The first argument is of type `string` and must be in the form of a valid regular expression matching pattern. The second argument may be of any type. It returns a `boolean` typed value that is `true` if the regular expression matches the second argument after it has been converted to a `string` type; otherwise, it returns `false`. The `regexp-match` function automatically converts the second argument to its string representation (using the `cast` function) and then applies the regular expression using the `regcmp` and `regex` systems calls provided by the host operating system. For specifics of the behavior of regular expression matching please refer to the `man` pages of the `regcmp` and `regex` system calls. If the first argument is not a valid regular expression matching pattern then the function evaluates to `Indeterminate`.

3.2.12 sum

The `sum` function computes the mathematical sum of an unbounded number of numerical arguments. All of the arguments must be of the same type. Valid types for the arguments are:

- `→ double`, `→ yearMonthDuration`, and `→ dayTimeDuration`.
- `→ integer`, `→ decimal`.

The type of the return value is the same as the argument type. If mathematical overflow occurs during the addition then the function evaluates to `Indeterminate`.

3.2.13 add

The `add` function computes mathematical addition on two arguments. The two arguments must be compatible with each other. The type of the return value is automatically determined by the argument types. If mathematical overflow occurs during the addition then the function evaluates to `Indeterminate`. The following table shows which types may be used as arguments and the type of the return value.

<table>
<thead>
<tr>
<th>Argument Types</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer, integer</td>
<td>integer</td>
</tr>
<tr>
<td>double, double</td>
<td>double</td>
</tr>
<tr>
<td>decimal, decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>integer, double</td>
<td>double</td>
</tr>
<tr>
<td>integer, decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>double, decimal</td>
<td>double</td>
</tr>
<tr>
<td>yearMonthDuration, yearMonthDuration</td>
<td>yearMonthDuration</td>
</tr>
<tr>
<td>dayTimeDuration, dayTimeDuration</td>
<td>dayTimeDuration</td>
</tr>
<tr>
<td>date, yearMonthDuration</td>
<td>date</td>
</tr>
<tr>
<td>date, dayTimeDuration</td>
<td>date</td>
</tr>
<tr>
<td>(only the days component is added to the date)</td>
<td></td>
</tr>
<tr>
<td>time, dayTimeDuration</td>
<td>time</td>
</tr>
<tr>
<td>dateTime, yearMonthDuration</td>
<td>dateTime</td>
</tr>
</tbody>
</table>
### 3.2.14 subtract

The `subtract` function computes mathematical subtraction on two arguments. The second argument is subtracted from the first argument. The two arguments must be compatible with each other. The type of the return value is automatically determined by the argument types. If mathematical overflow occurs during the subtraction then the function evaluates to `Indeterminate`. The following table shows which types may be used as arguments and the type of the return value. If a `date`, `time`, or `dateTime` typed argument is used it must be the first argument.

<table>
<thead>
<tr>
<th>Argument Types</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dateTime</code>, <code>dayTimeDuration</code></td>
<td><code>dateTime</code></td>
</tr>
<tr>
<td><code>integer</code>, <code>integer</code></td>
<td><code>integer</code></td>
</tr>
<tr>
<td><code>double</code>, <code>double</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>decimal</code>, <code>decimal</code></td>
<td><code>decimal</code></td>
</tr>
<tr>
<td><code>integer</code>, <code>double</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>integer</code>, <code>decimal</code></td>
<td><code>decimal</code></td>
</tr>
<tr>
<td><code>double</code>, <code>decimal</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>yearMonthDuration</code>, <code>yearMonthDuration</code></td>
<td><code>yearMonthDuration</code></td>
</tr>
<tr>
<td><code>dayTimeDuration</code>, <code>dayTimeDuration</code></td>
<td><code>dayTimeDuration</code></td>
</tr>
<tr>
<td><code>date</code>, <code>yearMonthDuration</code></td>
<td><code>date</code></td>
</tr>
<tr>
<td><code>date</code>, <code>dayTimeDuration</code></td>
<td><code>date</code></td>
</tr>
<tr>
<td>(only days component is subtracted from the date)</td>
<td></td>
</tr>
<tr>
<td><code>time</code>, <code>dayTimeDuration</code></td>
<td><code>time</code></td>
</tr>
<tr>
<td><code>dateTime</code>, <code>yearMonthDuration</code></td>
<td><code>dateTime</code></td>
</tr>
<tr>
<td><code>dateTime</code>, <code>dayTimeDuration</code></td>
<td><code>dateTime</code></td>
</tr>
</tbody>
</table>

### 3.2.15 multiply

The `multiply` function computes mathematical multiplication on two numerical arguments. The two arguments must be compatible with each other. The type of the return value is automatically determined by the argument types. If mathematical overflow occurs during the multiplication then the function evaluates to `Indeterminate`. The following table shows which types may be used as arguments and the type of the return value.
### 3.2.16 divide

The *divide* function computes mathematical division on two numerical arguments. The first argument is divided by the second argument. The two arguments must be compatible with each other. The type of the return value is automatically determined by the argument types. If divide by zero occurs during the division then the function evaluates to *Indeterminate*. The following table shows which types may be used as arguments and the type of the return value.

<table>
<thead>
<tr>
<th>Argument Types</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer, integer</td>
<td>Integer</td>
</tr>
<tr>
<td>double, double</td>
<td>Double</td>
</tr>
<tr>
<td>decimal, decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>integer, double</td>
<td>Double</td>
</tr>
<tr>
<td>integer, decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>double, decimal</td>
<td>Double</td>
</tr>
<tr>
<td>integer, yearMonthDuration</td>
<td>yearMonthDuration</td>
</tr>
<tr>
<td>integer, dayTimeDuration</td>
<td>dayTimeDuration</td>
</tr>
</tbody>
</table>

### 3.2.17 mod

The *mod* function computes the mathematical modulus (the remainder of division) of two *integer* arguments. The return value is of type *integer* and contains the modulus of dividing the first argument by the second. If divide by zero occurs during the modulus then the function evaluates to *Indeterminate*.

### 3.2.18 abs

The *abs* function computes the absolute value of a single numerical argument. Valid types for the argument are:

- → *double*,
- → *yearMonthDuration*, and
- → *integer*,
- → *dayTimeDuration*.
The type of the return value is the same as the argument type.

### 3.2.19 round

The `round` function rounds a single `double` argument to its nearest integral value. Halfway cases are rounded away from zero. The return value is of type `double`.

### 3.2.20 floor

The `floor` function computes the floor of a single double argument. The floor is the largest integral value that is not greater than the input argument. The return value is of type `double`.

### 3.2.21 string-normalize-space

The `string-normalize-space` function converts a single `string` argument by removing all leading and trailing whitespace. White space is defined as the space, tab carriage-return, newline, vertical-tab, and form-feed characters. The return value is of type `string`.

### 3.2.22 string-normalize-to-lower-case

The `string-normalize-to-lower-case` function converts a single `string` argument to its lower case equivalent. Each character in the input argument is converted to its lower case equivalent. The return value is of type `string`.

### 3.2.23 concatenate

The `concatenate` function produces a single `string` typed return value that is the string-wise concatenation of the string representation (according to the `cast` function) of the values of an unbounded number of arguments. The arguments are evaluated and if the return values are not of type `string`, they are internally converted using the `cast` function prior to being concatenated.

### 3.2.24 cast

The `cast` function converts a single argument to a different data type. The function takes two arguments. The first argument is a `<DataTypeDesignator>` element that specifies the destination data type. The second argument is the value to be cast. The source and destination data types must be convertible; otherwise, the function evaluates to `Indeterminate`. Overflow will also cause the function to evaluate to `Indeterminate`. The following table defines which data type combinations are convertible along with any relevant comments.

<table>
<thead>
<tr>
<th>Source Type</th>
<th>Destination Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Any</td>
<td>The source string must be a valid string representation of the destination data type.</td>
</tr>
</tbody>
</table>
PERMISSIBLE RXSML DATA TYPE CONVERSIONS

<table>
<thead>
<tr>
<th>Source Type</th>
<th>Destination Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>String</td>
<td>Data types with case insensitive components will have those components converted to lower case. The string representation of a label will depend on the configuration of the MAC security policy of the host computer.</td>
</tr>
<tr>
<td>integer</td>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>decimal</td>
<td>Fractional component discarded. Overflow possible.</td>
</tr>
<tr>
<td>double</td>
<td>integer</td>
<td>Fractional component discarded. Overflow possible.</td>
</tr>
<tr>
<td>double</td>
<td>decimal</td>
<td>Overflow possible. Loss of precision possible.</td>
</tr>
<tr>
<td>decimal</td>
<td>integer</td>
<td>Fractional component discarded. Overflow possible.</td>
</tr>
<tr>
<td>decimal</td>
<td>double</td>
<td>Loss of precision possible.</td>
</tr>
</tbody>
</table>

3.2.25 or

The or function computes the logical ‘or’ of an unbounded number of boolean typed arguments. The function returns a boolean typed value that is true if any argument’s return value is true; otherwise, it returns false. If there are no arguments to the function it returns true. Arguments are evaluated in order and evaluation of the arguments stops as soon as it evaluates an argument that returns true.

3.2.26 and

The and function computes the logical ‘and’ of an unbounded number of boolean typed arguments. The function returns a boolean typed value that is true if every argument’s return value is true; otherwise, it returns false. If there are no arguments to the function it returns false. Arguments are evaluated in order and evaluation of the arguments stops as soon as it evaluates an argument that returns false.

3.2.27 n-of

The n-of function determines if at least ‘n’ boolean arguments evaluate to true. The first argument to the function is of type integer and specifies ‘n’. The remaining arguments are of type boolean and have an unbounded number. The function returns a boolean typed value that is true if at least ‘n’ argument’s return value are true; otherwise, it returns false. If ‘n’ is equal to zero the function returns true. If ‘n’ is less than zero or is greater than the number of boolean typed arguments then the function evaluates to Indeterminate. Arguments are evaluated in order and evaluation of the arguments stops as soon as it evaluates ‘n’ argument that return true.

3.2.28 not

The not function computes the logical ‘not’ of a single boolean typed argument. The function returns a boolean typed value that is true if the argument’s return value is false; otherwise, it returns true.
3.2.29 one-and-only

The one-and-only function determines if a bag of values contains one and only one value, and returns that value if it exists. If the bag does not contain one and only one value then the function evaluates to Indeterminate. The function has a single argument consisting of a bag of values. The return value will be of the same type as its argument.

3.2.30 bag-size

The bag-size function extracts the number of values in a bag of values. The function has a single argument consisting of a bag of values of any type. The function returns an integer that contains the size of the bag.

3.2.31 is-in

The is-in function determines if a single value exists in a bag of values. The first argument must be a single value with a type the same as the second argument. The second argument consists of a bag of values. The bag of values of the second argument is iterated over and the equal function is called using the first argument and the iterated value from the bag. If the equal function returns true evaluation stops and the function returns true. If no call to the equal function returns true then the is-in function returns false.

3.2.32 bag

The bag function produces a bag of values from an unbounded number of like type arguments. The function returns a single argument consisting of the bag of values with a type the same as the type of the arguments. If there are no arguments the function evaluates to Indeterminate.

3.2.33 any-of

The any-of function applies a Boolean function between a single value and a bag of values, and returns true if and only if the Boolean function returns true for at least one value in the bag. The function takes three arguments. The first argument is a <Function> element that specifies the Boolean function to apply. The second argument is the single value. The third argument is the bag of values. The types and values of the second and third arguments must satisfy the requirements of the Boolean function specified by the first argument. If the third argument contains zero values the function evaluates to Indeterminate. If the Boolean function returns true evaluation stops and the function returns true. Valid Boolean functions are:

- \(\rightarrow equal\),
- \(\rightarrow greater-than\),
- \(\rightarrow greater-than-or-equal\),
- \(\rightarrow less-than\),
- \(\rightarrow less-than-or-equal\),
- \(\rightarrow regex-match\),
- \(\rightarrow dnsName-match\),
- \(\rightarrow ipAddress-match\),
- \(\rightarrow and\),
- \(\rightarrow or\).

3.2.34 all-of

The all-of function applies a Boolean function between a single value and a bag of values, and returns true if and only if the Boolean function returns true for all values in the bag. The function takes three arguments. The first argument is a <Function> element that specifies the Boolean function to apply. The second argument is the single value. The third argument is the bag of values. The types and values of the
second and third arguments must satisfy the requirements of the Boolean function specified by the first argument. If the third argument contains zero values the function evaluates to Indeterminate. Valid Boolean functions are:

- \( \rightarrow \text{equal}, \)  \( \rightarrow \text{less-than-or-equal}, \)  \( \rightarrow \text{regex-match}, \)
- \( \rightarrow \text{greater-than}, \)  \( \rightarrow \text{dnsName-match}, \)  \( \rightarrow \text{and} \)
- \( \rightarrow \text{greater-than-or-equal}, \)  \( \rightarrow \text{ipAddress-match}, \)  \( \rightarrow \text{and} \)
- \( \rightarrow \text{less-than}, \)  \( \rightarrow \text{or}. \)

3.2.35 any-of-any

The \textit{any-of-any} function applies a Boolean function between each value in a first bag of values and each value in a second bag of values, and returns \textit{true} if and only if the Boolean function returns \textit{true} for at least one evaluation. The function takes three arguments. The first argument is a \(<\text{Function}>\) element that specifies the Boolean function to apply. The second argument is the first bag of values. The third argument is the second bag of values. The types and values of the second and third arguments must satisfy the requirements of the Boolean function specified by the first argument. If the second or third argument contains zero values the function evaluates to Indeterminate. Valid Boolean functions are:

- \( \rightarrow \text{equal}, \)  \( \rightarrow \text{less-than-or-equal}, \)  \( \rightarrow \text{regex-match}, \)
- \( \rightarrow \text{greater-than}, \)  \( \rightarrow \text{dnsName-match}, \)  \( \rightarrow \text{and} \)
- \( \rightarrow \text{greater-than-or-equal}, \)  \( \rightarrow \text{ipAddress-match}, \)  \( \rightarrow \text{or}. \)

3.2.36 all-of-any

The \textit{all-of-any} function applies a Boolean function between each value in a first bag of values and each value in a second bag of values, and returns \textit{true} if and only if the Boolean function returns \textit{true} for each value of the first bag combined with \textit{any} value of the second bag. The function takes three arguments. The first argument is the first bag of values. The third argument is the second bag of values. The types and values of the second and third arguments must satisfy the requirements of the Boolean function specified by the first argument. If the second or third argument contains zero values the function evaluates to Indeterminate. Valid Boolean functions are:

- \( \rightarrow \text{equal}, \)  \( \rightarrow \text{less-than-or-equal}, \)  \( \rightarrow \text{regex-match}, \)
- \( \rightarrow \text{greater-than}, \)  \( \rightarrow \text{dnsName-match}, \)  \( \rightarrow \text{and} \)
- \( \rightarrow \text{greater-than-or-equal}, \)  \( \rightarrow \text{ipAddress-match}, \)  \( \rightarrow \text{or}. \)

3.2.37 any-of-all

The \textit{any-of-all} function applies a Boolean function between each value in a first bag of values and each value in a second bag of values, and returns \textit{true} if and only if the Boolean function returns \textit{true} for \textit{any} value of the first bag combined with \textit{each} value of the second bag. The function takes three arguments. The first argument is a \(<\text{Function}>\) element that specifies the Boolean function to apply. The second argument is the first bag of values. The third argument is the second bag of values. The types and values of the second and third arguments must satisfy the requirements of the Boolean function specified by the
first argument. If the second or third argument contains zero values the function evaluates to \textit{Indeterminate}. Valid Boolean functions are:

\[
\begin{align*}
\rightarrow & \text{ equal}, & \rightarrow & \text{ less-than-or-equal}, & \rightarrow & \text{ regex-match}, \\
\rightarrow & \text{ greater-than}, & \rightarrow & \text{ dnsName-match}, & \rightarrow & \text{ and} \\
\rightarrow & \text{ greater-than-or-equal}, & \rightarrow & \text{ ipAddress-match}, & \rightarrow & \text{ and} \\
\rightarrow & \text{ less-than}, & \rightarrow & \text{ or}.
\end{align*}
\]

\subsection*{3.2.38 all-of-all}

The \textit{all-of-all} function applies a Boolean function between each value in a first bag of values and each value in a second bag of values, and returns \textit{true} if and only if the Boolean function returns \textit{true} for each value of the first bag combined with each value of the second bag. The function takes three arguments. The first argument is a \texttt{<Function>} element that specifies the Boolean function to apply. The second argument is the first bag of values. The third argument is the second bag of values. The types and values of the second and third arguments must satisfy the requirements of the Boolean function specified by the first argument. If the second or third argument contains zero values the function evaluates to \textit{Indeterminate}. Valid Boolean functions are:

\[
\begin{align*}
\rightarrow & \text{ equal}, & \rightarrow & \text{ less-than-or-equal}, & \rightarrow & \text{ regex-match}, \\
\rightarrow & \text{ greater-than}, & \rightarrow & \text{ dnsName-match}, & \rightarrow & \text{ and} \\
\rightarrow & \text{ greater-than-or-equal}, & \rightarrow & \text{ ipAddress-match}, & \rightarrow & \text{ and} \\
\rightarrow & \text{ less-than}, & \rightarrow & \text{ or}.
\end{align*}
\]

\subsection*{3.2.39 map}

The \textit{map} function converts a bag of values into another bag of values. The function iterates over the argument’s bag of values and applies a mapping function to each, producing the return bag of values. The data type of the return bag of values is dependent upon the mapping function used. The first argument is a \texttt{<Function>} element that specifies the mapping function. The second argument is the bag of values to be mapped. The remaining arguments are passed into the mapping function. The number of remaining arguments is dependent upon the mapping function used. Each value in the bag of values is iterated over and the mapping function is called with the iterated value and the remaining arguments. The return value of the mapping function is placed into the \textit{map} function’s return bag of values. Valid mapping functions are:

\[
\begin{align*}
\rightarrow & \text{ equal}, & \rightarrow & \text{ ipAddress-match}, & \rightarrow & \text{ add}, & \rightarrow & \text{ round}, \\
\rightarrow & \text{ greater-than}, & \rightarrow & \text{ regex-match}, & \rightarrow & \text{ subtract}, & \rightarrow & \text{ floor}, \\
\rightarrow & \text{ greater-than-or-equal}, & \rightarrow & \text{ and}, & \rightarrow & \text{ multiply}, & \rightarrow & \text{ string-normalize-space}, \\
\rightarrow & \text{ less-than}, & \rightarrow & \text{ or}, & \rightarrow & \text{ divide}, & \rightarrow & \text{ string-normalize-to-lower-case}, \\
\rightarrow & \text{ less-than-or-equal}, & \rightarrow & \text{ n-of}, & \rightarrow & \text{ mod}, & \rightarrow & \text{ and} \\
\rightarrow & \text{ dnsName-match}, & \rightarrow & \text{ time-in-range}, & \rightarrow & \text{ abs}, & \rightarrow & \text{ cast}.
\end{align*}
\]

\subsection*{3.2.40 intersection}

The \textit{intersection} function constructs the intersection of two bags of values. The function has two arguments which are the bags of values from which to construct the intersection. The data types of the two arguments must be the same. The return bag of values will have the same data type as the arguments.
The intersection consists of all value that are common (according to the *equal* function) between both bags of values with duplicate values removed.

### 3.2.41 at-least-one-member-of

The *at-least-one-member-of* function determines if at least one value in a first bag of values is contained in a second bag of values. The function takes two arguments, which are bags of like typed values. The function has a *boolean* return value that is *true* if and only if at least one value from the first argument’s bag is contained in the second argument’s bag using the *is-in* function; otherwise it returns *false*.

### 3.2.42 union

The *union* function constructs the union of two bags of values. The function has two arguments which are the bags of values from which to construct the union. The data types of the two arguments must be the same. The return bag of values will have the same data type as the arguments. The union consists of all values that exist in either of the bags of values with duplicate values removed (according to the *equal* function).

### 3.2.43 subset

The *union* function determines if a first bag of values is a subset of a second bag of values. The function has two arguments and the data types of the two arguments must be the same. The function returns a *boolean* value that is *true* if and only if the first argument’s bag of values is a subset of the second argument’s bag of values (according to the *equal* function), with duplicate values being removed; otherwise, it returns *false*.

### 3.2.44 set-equals

The *set-equals* function determines if a first bag of values is equal to a second bag of values. The function has two arguments and the data types of the two arguments must be the same. The function returns a *boolean* value that is *true* if and only if every value in the first bag equals a value in the second bag and every value in the second bag equals a value in the first bag (according to the *equal* function); otherwise, it returns *false*.

### 3.3 Data Types

The Security Markup Language (RXSML) has a set of data types that is distinct from the set of SQL data types. In general, these correspond to XML data types. The RXSML data types are usable in XML and suitable for policy evaluation. While the two sets of data types are distinct, they are similar and converting between the two sets is straightforward.

#### 3.3.1 Relationship Between RXSML and SQL Data Types

In order for the Security Policy Manager to make policy decisions based upon the values of database rows, SQL typed database fields must be converted into RXSML typed attribute values. This operation is performed when the `<ImportColumnSector>`, `<ImportFieldSelector>`, or `<AttributeSelector>` elements are evaluated. The source SQL data type is given by the corresponding SQL column specification and the destination RXSML data type is specified as an attribute of the element. If the conversion between the SQL
data type and the specified RXSML data type is permissible it is performed; otherwise, the element evaluates to Indeterminate.

In order for the Security Policy Manager to set an SQL typed field within a database row, RXSML typed attribute values must be converted into SQL typed database fields. This operation is performed when and <Obligation> element with an ObligationId attribute of set-field is executed. The source RXSML data type is specified by the RXSML element that originates the attribute value and the destination SQL data type is given by the corresponding SQL column specification. If the conversion between the RXSML data type and the specified SQL data type is permissible it is performed; otherwise, the obligation execution fails.

### 3.3.1.1 Converting from SQL to RXSML Data Types

Converting from an RXSML typed attribute value to an SQL typed database field is performed when the <ImportColumnSector>, <ImportFieldSelector>, or <AttributeSelector> elements are evaluated. The source SQL data type is given by the corresponding SQL column specification and the destination RXSML data type is specified as an attribute of the element. If the conversion between the SQL data type and the specified RXSML data type is permissible it is performed; otherwise, the element evaluates to Indeterminate. The following table contains permissible conversions and any relevant comments. If loss of precision or overflow is possible as a result of a conversion it is noted in the comments of the table.

<table>
<thead>
<tr>
<th>SQL Data Type</th>
<th>RXSML Data Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER</td>
<td>Any</td>
<td>The source string must match a valid literal string representation of the destination RXSML data type.</td>
</tr>
<tr>
<td>CHARACTER VARYING</td>
<td>Any</td>
<td>The source string must match a valid literal string representation of the destination RXSML data type.</td>
</tr>
<tr>
<td>DECIMAL &amp; NUMERIC</td>
<td>Decimal</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>DECIMAL &amp; NUMERIC</td>
<td>Double</td>
<td>Precision will be truncated to a maximum of 15 digits. All source values are permissible.</td>
</tr>
<tr>
<td>DECIMAL &amp; NUMERIC</td>
<td>Integer</td>
<td>Fractional portion of the SQL value is truncated. The truncated SQL value must be between -2,147,483,648 and 2,147,483,647 inclusive.</td>
</tr>
<tr>
<td>DECIMAL &amp; NUMERIC</td>
<td>Boolean</td>
<td>If the source value is zero the boolean is set to false; otherwise, it is set to true.</td>
</tr>
<tr>
<td>DOUBLE PRECISION, REAL, &amp; FLOAT</td>
<td>Decimal</td>
<td>Precision may be lost. The exponent of the source must be less than 50.</td>
</tr>
<tr>
<td>DOUBLE PRECISION, REAL, &amp; FLOAT</td>
<td>Double</td>
<td>All source values are permissible</td>
</tr>
<tr>
<td>DOUBLE PRECISION, REAL, &amp; FLOAT</td>
<td>Integer</td>
<td>Fractional portion of the SQL value is truncated. The truncated SQL value must be between -2,147,483,648 and 2,147,483,647 inclusive.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Decimal</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Double</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Boolean</td>
<td>If the source value is zero the boolean is set to false; otherwise, it is set to true.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Decimal</td>
<td>All source values are permissible.</td>
</tr>
</tbody>
</table>
### Permissible SQL to RXSML Data Type Conversions

<table>
<thead>
<tr>
<th>SQL Data Type</th>
<th>RXSML Data Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>Double</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Integer</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Boolean</td>
<td>If the source value is zero the boolean is set to false; otherwise, it is set to true.</td>
</tr>
<tr>
<td>DATE</td>
<td>Date</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>TIME</td>
<td>Time</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>dateTime</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL YEAR</td>
<td>yearMonthDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL MONTH</td>
<td>yearMonthDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>yearMonthDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL DAY</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL HOUR</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL MINUTE</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL SECOND</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL DAY TO HOUR</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL DAY TO MINUTE</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL HOUR TO MINUTE</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL HOUR TO SECOND</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>INTERVAL MINUTE TO SECOND</td>
<td>dayTimeDuration</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>LONG LABEL</td>
<td>label</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>SHORT LABEL</td>
<td>label</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>BIT</td>
<td>hexBinary</td>
<td>Source is zero bit padded to byte boundary.</td>
</tr>
<tr>
<td>BIT</td>
<td>base64Binary</td>
<td>Source is zero bit padded to byte boundary.</td>
</tr>
<tr>
<td>BIT VARYING</td>
<td>hexBinary</td>
<td>Source is zero bit padded to byte boundary.</td>
</tr>
<tr>
<td>BIT VARYING</td>
<td>base64Binary</td>
<td>Source is zero bit padded to byte boundary.</td>
</tr>
</tbody>
</table>

### 3.3.1.2 Converting from RXSML to SQL data types

Converting from an RXSML typed attribute value to an SQL typed database field is performed when and `<Obligation>` element with an `ObligationId` attribute of `set-field` is executed. The source RXSML data type is specified by the RXSML element that originates the attribute value and the destination SQL data type is given by the corresponding SQL column specification. If the conversion between the RXSML data type and the specified SQL data type is permissible it is performed; otherwise, the obligation execution fails. The following table contains permissible conversions and any relevant comments.
<table>
<thead>
<tr>
<th>RXSML Data Type</th>
<th>SQL Data Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>CHARACTER</td>
<td>The source string length must be less than or equal to the destination length. The source is space padded to the length of the destination.</td>
</tr>
<tr>
<td>string</td>
<td>CHARACTER VARYING</td>
<td>The source string length must be less than or equal to the destination length.</td>
</tr>
<tr>
<td>decimal</td>
<td>DECIMAL &amp; NUMERIC</td>
<td>The number of significant digits to the left of the decimal place of the source must be greater than or equal to precision – scale of the destination. If the number of significant digits to the right of the decimal place of the source is greater than the scale of the destination then precision will be lost.</td>
</tr>
<tr>
<td>double</td>
<td>DOUBLE PRECISION, REAL, &amp; FLOAT</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>integer</td>
<td>INTEGER</td>
<td>All source values are permissible.</td>
</tr>
<tr>
<td>integer</td>
<td>SMALLINT</td>
<td>The source value must be between -32,768 and 32,767 inclusive.</td>
</tr>
<tr>
<td>boolean</td>
<td>INTEGER</td>
<td>If the boolean value is true then the destination is set to 1; otherwise the destination is set to 0.</td>
</tr>
<tr>
<td>boolean</td>
<td>SMALLINT</td>
<td>If the boolean value is true then the destination is set to 1; otherwise the destination is set to 0.</td>
</tr>
<tr>
<td>date</td>
<td>DATE</td>
<td>The year of the source must be between 1 and 9999.</td>
</tr>
<tr>
<td>time</td>
<td>TIME</td>
<td>All source values are permissible. Precision of fractional seconds may be lost.</td>
</tr>
<tr>
<td>dateTime</td>
<td>TIMESTAMP</td>
<td>The year of the source must be between 1 and 9999. Precision of fractional seconds may be lost.</td>
</tr>
</tbody>
</table>
| yearMonthDuration | INTERVAL YEAR | dst_years = src_years  
The src_months component is discarded. The total dst_years must fall within the range of values allowed by the destination given its leading precision. |
| yearMonthDuration | INTERVAL MONTH | dst_months = src_years * 12 + src_months  
The total dst_months must fall within the range of values allowed by the destination given its leading precision. |
| yearMonthDuration | INTERVAL YEAR TO MONTH | dst_years = src_years  
dst_months = src_months  
The total dst_years must fall within the range of values allowed by the destination given its leading precision. |
| dayTimeDuration | INTERVAL DAY   | dst_days = src_days  
The total dst_days must fall within the range of values allowed by the destination given its leading precision. The remaining source components are discarded. |
### PERMISSIBLE RXSML TO SQL DATA TYPE CONVERSIONS

<table>
<thead>
<tr>
<th>RXSML Data Type</th>
<th>SQL Data Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>dayTimeDuration</td>
<td>INTERVAL HOUR</td>
<td>$dst_hours = src_days * 24 + src_hours$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The total $dst_hours$ must fall within the range of values allowed by the destination given its leading precision.</td>
</tr>
<tr>
<td>dayTimeDuration</td>
<td>INTERVAL MINUTE</td>
<td>$dst_minutes = src_days * 1440 + src_hours * 60 + src_minutes$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The total $dst_minutes$ must fall within the range of values allowed by the destination given its leading precision. The remaining source components are discarded.</td>
</tr>
<tr>
<td>dayTimeDuration</td>
<td>INTERVAL SECOND</td>
<td>$dst_seconds = src_days * 86400 + src_hours * 3600 + src_minutes * 60 + src_seconds$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The total $dst_seconds$ must fall within the range of values allowed by the destination given its leading precision. Precision of fractional seconds may be lost.</td>
</tr>
<tr>
<td>dayTimeDuration</td>
<td>INTERVAL DAY TO HOUR</td>
<td>$dst_days = src_days$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$dst_hours = src_hours$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The total $dst_days$ must fall within the range of values allowed by the destination given its leading precision. The remaining source components are discarded.</td>
</tr>
<tr>
<td>dayTimeDuration</td>
<td>INTERVAL DAY TO MINUTE</td>
<td>$dst_days = src_days$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$dst_hours = src_hours$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$dst_minutes = src_minutes$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The total $dst_days$ must fall within the range of values allowed by the destination given its leading precision. The remaining source components are discarded.</td>
</tr>
<tr>
<td>dayTimeDuration</td>
<td>INTERVAL DAY TO SECOND</td>
<td>$dst_days = src_days$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$dst_hours = src_hours$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$dst_minutes = src_minutes$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$dst_seconds = src_seconds$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The total $dst_days$ must fall within the range of values allowed by the destination given its leading precision. Precision of fractional seconds may be lost.</td>
</tr>
<tr>
<td>dayTimeDuration</td>
<td>INTERVAL HOUR TO MINUTE</td>
<td>$dst_hours = src_days * 24 + src_hours$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$dst_minutes = src_minutes$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The total $dst_hours$ must fall within the range of values allowed by the destination given its leading precision. The remaining source components are discarded.</td>
</tr>
</tbody>
</table>
PERMISSIBLE RXSML TO SQL DATA TYPE CONVERSIONS

<table>
<thead>
<tr>
<th>RXSML Data Type</th>
<th>SQL Data Type</th>
<th>Comments</th>
</tr>
</thead>
</table>
| dayTimeDuration | INTERVAL HOUR TO SECOND | $dst\_hours = src\_days \times 24 + src\_hours$
|                  |               | $dst\_minutes = src\_minutes$
|                  |               | $dst\_seconds = src\_seconds$
|                  |               | The total $dst\_hours$ must fall within the range of values allowed by the destination given its leading precision. Precision of fractional seconds may be lost. |
| dayTimeDuration | INTERVAL MINUTE TO SECOND | $dst\_minutes = src\_days \times 1440 + src\_hours \times 60 + src\_minutes$
|                  |               | $dst\_seconds = src\_seconds$
|                  |               | The total $dst\_minutes$ must fall within the range of values allowed by the destination given its leading precision. Precision of fractional seconds may be lost. |
| label           | LONG LABEL    | All source values are permissible. |
| label           | SHORT LABEL   | All source values are permissible. |
| hexBinary       | BIT           | Source is zero bit padded to the length of the destination. |
|                 |               | The length of the source must be less than or equal to the length of the destination. |
| hexBinary       | BIT VARYING   | The length of the source must be less than or equal to the length of the destination. |
| base64Binary    | BIT           | Source is zero bit padded to the length of the destination. The length of the source must be less than or equal to the length of the destination. |
| base64Binary    | BIT VARYING   | The length of the source must be less than or equal to the length of the destination. |

3.3.2 string

The string data type consists of a literal ASCII text string. There is no upper bound on its length.

Example:

```
<AttributeValue DataType="string">This is a string</AttributeValue>
```

3.3.3 boolean

The boolean data type represents a boolean value. The value may be either true or false.

Example:

```
<AttributeValue DataType="boolean">true</AttributeValue>
```
3.3.4 integer

The integer data type represents a numerical integer value. Valid values range from -2,147,483,648 through 2,147,483,647. The string representation is an optional sign followed by a sequence of digits with no thousands separator.

Example:

```
<AttributeValue DataType="integer">18765</AttributeValue>
<AttributeValue DataType="integer">-95</AttributeValue>
```

3.3.5 double

The double data type represents an approximate floating point value. Valid values range from -1.797693134862316e+308 through 1.797693134862316e+308. It has at most 15 digits of precision. The string representation may be in decimal notation or engineering notation.

Example:

```
<AttributeValue DataType="double">-1.234e45</AttributeValue>
<AttributeValue DataType="double">68.1234</AttributeValue>
```

3.3.6 decimal

The decimal data type represents an exact numeric decimal value. It has at most 50 digits of precision. The string representation is in decimal notation.

Example:

```
<AttributeValue DataType="decimal">68.1234</AttributeValue>
```

3.3.7 date

The date data type represents a specific day on the Gregorian calendar. It has the form:

```
yyyy-mm-dd
```

The value of yyyy (year) is constrained to the range 1-9999. The value of mm (month) is constrained to the range 1-12. The value of dd (day) is constrained to the range from 1 to the maximum number of days in the month given by mm, accounting for leap years.

Example:

```
<AttributeValue DataType="date">2008-11-28</AttributeValue>
```

3.3.8 time

The time data type represents a specific time of an unspecified date. It has the form:

```
hh:mm:ss.[fffffff]
```
The value of \( hh \) (hour) is constrained to the range 0-23. The value of \( mm \) (minute) is constrained to the range 0-59. The value of \( ss \) (second) is constrained to the range 0-59. The value of \( fffff \) (fraction of second) is constrained to the range 0-999999. Specification of the fraction of second is optional and defaults to 0.

**Example:**

```xml
<AttributeValue DataType="time">01:45:15.125</AttributeValue>
<AttributeValue DataType="time">15:30:00</AttributeValue>
```

### 3.3.9 dateTime

The *dateTime* data type represents a specific time on a specific date. It has the form:

```text
yyyy-mm-ddThh:mm:ss[fffff]
```

The value of \( yyyy \) (year) is constrained to the range 1-9999. The value of \( mm \) (month) is constrained to the range 1-12. The value of \( dd \) (day) is constrained to the range from 1 to the maximum number of days in the month given by \( mm \), accounting for leap years. ‘\( T \)’ is a separator indicating that time-of-day follows. The value of \( hh \) (hour) is constrained to the range 0-23. The value of \( mm \) (minute) is constrained to the range 0-59. The value of \( ss \) (second) is constrained to the range 0-59. The value of \( fffff \) (fraction of second) is constrained to the range 0-999999. Specification of the fraction of second is optional and defaults to 0.

**Example:**

```xml
<AttributeValue DataType="dateTime">2008-11-28T01:45:15.125</AttributeValue>
<AttributeValue DataType="dateTime">199-12-25T15:30:00</AttributeValue>
```

### 3.3.10 dayTimeDuration

The *dayTimeDuration* data type represents a span of time. It has components of days, hours, minutes, and seconds. It has the form:

```text
[-]PnDTnHnMn.nn.nnS
```

[-] represents an optional minus sign. ‘\( P \)’ is a token indicating that a duration follows. ‘\( nD \)’ represents ‘n’ number of days. ‘\( T \)’ is a separator indicating that an amount of time follows. ‘\( nH \)’ represents ‘n’ hours. ‘\( nM \)’ represents ‘n’ minutes. ‘\( nn.nnS \)’ represents ‘n’ whole seconds and ‘\( nn.nn \)’ fraction of a second. Fraction of a second is optional. All components are optional although at least one must be present. If ‘\( T \)’ is present then at least one component of time must be present. Any component that is omitted defaults to 0.

There are no restrictions placed on the values of each component other than being non-negative integers. Internally, the *dayTimeDuration* is normalized so that all components other than the days component contains values constrained as their corresponding component in the *dateTime*. After normalization the days component must be between 0 and 2,147,483,647.

**Example:**

```xml
<AttributeValue DataType="dayTimeDuration"> -P50DT10H5M50.234S </AttributeValue>
<AttributeValue DataType="dayTimeDuration">P100D</AttributeValue>
```
<AttributeValue DataType="dayTimeDuration ">PT5M10S</AttributeValue>

### 3.3.11 yearMonthDuration

The `yearMonthDuration` data type represents a span of time. It has components of days, hours, minutes, and seconds. It has the form:

```
[-]PnYnM
```

[-] represents an optional minus sign. ‘P’ is a token indicating that a duration follows. ‘nY’ represents ‘n’ number of years. ‘nM’ represents ‘n’ months. All components are optional although at least one must be present. Any component that is omitted defaults to 0.

There are no restrictions placed on the values of each component other than being non-negative integers. Internally, the `yearMonthDuration` is normalized so that all components other than the years component contains values constrained as their corresponding component in the `dateTime`. After normalization the years component must be between 0 and 2,147,483,647.

**Example:**

```
<AttributeValue DataType="yearMonthDuration ">-P50Y10M</AttributeValue>
<AttributeValue DataType="yearMonthDuration ">P123M</AttributeValue>
```

### 3.3.12 rfc822Name

The `rfc822Name` data type represents an email address. It has the form:

```
id@domain
```

‘id’ is a case sensitive use identifier. ‘@’ separates the id from the domain. ‘domain’ is a case insensitive domain name.

**Example:**

```
<AttributeValue DataType="rfc822Name ">billy@microsoft.com</AttributeValue>
<AttributeValue DataType="rfc822Name ">George@192.168.1.22</AttributeValue>
```

### 3.3.13 hexBinary

The `hexBinary` data type represents an unstructured array of bytes. The array of bytes is encoded to a string of hexadecimal digits (0-9; a-f). Digits must be given in groups of two. There is no upper bound on the length of the array.

**Example:**

```
<AttributeValue DataType="hexBinary ">ab0f49</AttributeValue>
```

### 3.3.14 base64Binary

The `base64Binary` data type represents an unstructured array of bytes. The array of bytes is encoded to a string of base64 digits (A–Z, a–z, 0–9, +, /). ‘=’ may be used as a special terminator indicating empty
bytes. Any non-base64 digits in the string are ignored. Digits must be given in groups of four. There is no upper bound on the length of the array.

**Example:**

```xml
<AttributeValue DataType="base64Binary">abcdABC</AttributeValue>
```

### 3.3.15 label

The *label* data type represents an MLS security label. Its string format is dependent on the configuration of the MLS security policy of the host computer. Typically, it consists of a classification (e.g., Top Secret) and a set of categories (Able). The classifications and categories may be given in long or short format. The string representation of the label must be recognizable by the MLS security mechanism of the underlying host computer.

**Example:**

```xml
<AttributeValue DataType="label">Secret Able Bravo</AttributeValue>
<AttributeValue DataType="label">S A B</AttributeValue>
```

### 3.3.16 ipAddress

The *ipAddress* data type represents an IPv4 network address, with port or port range. It has the form:

```
ddd.ddd.ddd.ddd[ "" :"" [ portrange ] ]
```

where *portrange* has the form:

```
portnumber | "" -"" portnumber | portnumber""[portnumber]
```

The address has four required components (*ddd*) each with a 1-3 digit number. Any component may also be represented by the wildcard character ‘*’. This character is used in the special matching function *ipAddress-match*. The ‘:’ indicates that the optional port or port range follows. A single *portnumber* indicates a single port. A leading ‘-’ indicates a range of ports starting at the minimum possible port. A trailing ‘-’ indicates a range of ports that ends at the maximum possible port. A ‘-’ enclosed in two *portnumbers* indicates an explicit port range.

**Example:**

```xml
<AttributeValue DataType="ipAddress">100.101.102.103</AttributeValue>
<AttributeValue DataType="ipAddress">100.101.102.4:-45</AttributeValue>
<AttributeValue DataType="ipAddress">100.101.102.4:30</AttributeValue>
<AttributeValue DataType="ipAddress">100.101.102.4:678-</AttributeValue>
<AttributeValue DataType="ipAddress">100.101.102.4:30-678</AttributeValue>
<AttributeValue DataType="ipAddress">100.*.102.*</AttributeValue>
```

### 3.3.17 dnsName

The *dnsName* data type represents a Domain Name Service (DNS) host name, with port or port range. It has the form:
where portrange has the form:

\[ \text{portnumber} | \text{portnumber} - | \text{portnumber} \]

The host name may have one or more string components (\(dns\)). Any component may also be represented by the wildcard character ‘*’. This character is used in the special matching function \(dnsName\)-match. The ‘:’ indicates that the optional port or port range follows. A single \(portnumber\) indicates a single port. A leading ‘-’ indicates a range of ports starting at the minimum possible port. A trailing ‘-’ indicates a range of ports that ends at the maximum possible port. A ‘-’ enclosed in two \(portnumbers\) indicates an explicit port range.

**Example:**

```xml
<AttributeValue DataType="dnsName">rubix.com</AttributeValue>
<AttributeValue DataType="dnsName">orange.rubix.com:45</AttributeValue>
<AttributeValue DataType="dnsName">green.rubix.com:30</AttributeValue>
<AttributeValue DataType="dnsName">*.rubix.com:678</AttributeValue>
<AttributeValue DataType="dnsName">rubix.com:30-678</AttributeValue>
```

### 3.4 Context Attributes

Context attributes are named attribute values reflecting the context in which a decision request is made. These values are automatically assigned during database operation and may be used to make security policy decisions. Context attributes are classified into attributes relating to the subject performing the database operation, the resource (database object) the database operation acts upon, the type of database operation, and the general environment. The value of any database row, whether it is currently being operated on or not, may also be used to make security policy decision. The process of retrieving named database row values that are not being operated upon by the current database operation is referred to as importing.

#### 3.4.1 Subject Attributes

Subject attributes are named attribute values associated with the subject performing the database operation. They may be retrieved using the \(<SubjectAttributeDesignator>\) element where the AttributeId attribute contains the name of the corresponding subject attribute.

**3.4.1.1 subject-id**

The \(subject-id\) subject attribute is the numerical user ID of the database subject performing the current database operation. The \(subject-id\) subject attribute always returns a single value of type \(integer\) and is available for all database operations.

**3.4.1.2 subject-name**

The \(subject-name\) subject attribute is the text user name of the database subject performing the current database operation. The \(subject-name\) subject attribute always returns a single value of type \(string\) and is available for all database operations.
3.4.1.3 group-id
The group-id subject attribute is the numerical group ID of the database subject performing the current database operation. The group-id subject attribute always returns a single value of type integer and is available for all database operations.

3.4.1.4 group-name
The group-name subject attribute is the text group name of the database subject performing the current database operation. The group-name subject attribute always returns a single value of type string and is available for all database operations.

3.4.1.5 session-start-time
The session-start-time subject attribute is the local time the database subject started the current database session. The session-start-time subject attribute always returns a single value of type time and is available for all database operations.

3.4.1.6 session-start-date
The session-start-date subject attribute is the local date the database subject started the current database session. The session-start-date subject attribute always returns a single value of type date and is available for all database operations.

3.4.1.7 session-start-dateTime
The session-start-dateTime subject attribute is the local date and time the database subject started the current database session. The session-start-dateTime subject attribute always returns a single value of type dateTime and is available for all database operations.

3.4.1.8 ip-address
The ip-address subject attribute is the IPv4 network address from which the database subject started the current database session. The value is retrieved from the socket connection of the database client. If the client resides on the same host as the server this value will always be 127.0.0.1. The ip-address subject attribute always returns a single value of type ipAddress and is available for all database operations.

3.4.1.9 dns-name
The dns-name subject attribute is the Domain Name Service (DNS) host name from which the database subject started the current database session. The value is retrieved by performing a host name lookup using the network address retrieved from the socket connection of the database client. If the client resides on the same host as the server this value will always be localhost. The dns-name subject attribute always returns a single value of type dnsName and is available for all database operations.

3.4.1.10 session-label
The session-label subject attribute is the database session label at the time the database operation was submitted. The session-label subject attribute always returns a single value of type label and is available for all database operations.
3.4.1.11 application-name

The application-name subject attribute is the text name of the current application. The application-name subject attribute always returns a single value of type string. It is only available if the current application has been set by the ALTER SESSION SET APPLICATION SQL command.

3.4.1.12 application-user-name

The application-user-name subject attribute is the text name of the currently authenticated application user. The application-user-name subject attribute always returns a single value of type string. It is only available if the current application user has been set by the AUTHENTICATE APPLICATION_USER or ALTER SESSION SET APPLICATION_USER SQL command.

3.4.1.13 application-user-id

The application-user-id subject attribute is the numerical ID of the currently authenticated application user. The application-user-id subject attribute always returns a single value of type integer. It is only available if the current application user has been set by the AUTHENTICATE APPLICATION_USER or ALTER SESSION SET APPLICATION_USER SQL command.

3.4.2 Resource Attributes

Resource attributes are named attribute values associated with the database object the database operation is acting upon. The resource attributes that are not the value of a database field may be retrieved using the <ResourceAttributeDesignator> element where the AttributeId attribute contains the name of the corresponding resource attribute.

The value of any database field in the current row being operated upon (if the current operation is a row based operation) may be retrieved using the <AttributeSelector> element where the RequestContextPath attribute refers to the database column using the five part naming convention “database_name.catalog_name.schema_name.table_name.column_name”.

3.4.2.1 resource-label

The resource-label resource attribute is the sensitivity label of the database object currently being operated upon. The resource-label resource attribute always returns a single value of type label and is available for all database operations. The following table details which object’s sensitivity label is returned for each database operation.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Label</th>
<th>Operation</th>
<th>Label</th>
<th>Operation</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-select</td>
<td>row-label</td>
<td>table-open</td>
<td>table-label</td>
<td>schema-open</td>
<td>schema-label</td>
</tr>
<tr>
<td>row-insert</td>
<td>row-label</td>
<td>table-create</td>
<td>table-label</td>
<td>schema-create</td>
<td>schema-label</td>
</tr>
<tr>
<td>row-update</td>
<td>row-label</td>
<td>table-drop</td>
<td>table-label</td>
<td>schema-drop</td>
<td>schema-label</td>
</tr>
<tr>
<td>row-delete</td>
<td>row-label</td>
<td>table-alter</td>
<td>table-label</td>
<td>catalog-open</td>
<td>catalog-label</td>
</tr>
<tr>
<td>index-create</td>
<td>table-label</td>
<td>view-create</td>
<td>view-label</td>
<td>catalog-create</td>
<td>catalog-label</td>
</tr>
<tr>
<td>index-drop</td>
<td>table-label</td>
<td>view-drop</td>
<td>view-label</td>
<td>catalog-drop</td>
<td>catalog-label</td>
</tr>
<tr>
<td></td>
<td></td>
<td>view-open</td>
<td>view-label</td>
<td>database-open</td>
<td>database-label</td>
</tr>
</tbody>
</table>
6.4.2.2 resource-name

The resource-name resource attribute is the fully specified name of the database object currently being operated upon. The resource-name resource attribute may return a single value or a multi-valued bag of type string and is available for all database operations. The following table details which object’s name is returned for each database operation and any comments.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-select</td>
<td>column-name(s)</td>
<td>Multi-value bag</td>
</tr>
<tr>
<td>row-insert</td>
<td>column-name(s)</td>
<td>Multi-value bag of columns names with non-null values being inserted.</td>
</tr>
<tr>
<td>row-update</td>
<td>column-name(s)</td>
<td>Multi-value bag of columns names actually being updated.</td>
</tr>
<tr>
<td>row-delete</td>
<td>column-name(s)</td>
<td>Multi-value bag of columns names.</td>
</tr>
<tr>
<td>index-create</td>
<td>table-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>index-drop</td>
<td>table-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>table-open</td>
<td>table-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>table-create</td>
<td>table-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>table-drop</td>
<td>table-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>table-alter</td>
<td>table-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>catalog-create</td>
<td>catalog-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>catalog-drop</td>
<td>catalog-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>database-open</td>
<td>database-name</td>
<td>Single value.</td>
</tr>
<tr>
<td>database-drop</td>
<td>database-name</td>
<td>Single value.</td>
</tr>
</tbody>
</table>

3.4.2.3 row-label

The row-label resource attribute is the sensitivity label of the database row currently being operated upon. The row-label resource attribute always returns a single value of type label. It is available for the row-select, row-insert, row-update, and row-delete operations.

3.4.2.4 table-label

The table-label resource attribute is the sensitivity label of the database table currently being operated upon. The table-label resource attribute always returns a single value of type label. It is available for the:

row-select, index-create, table-create,
→ row-insert,  → index-drop,  → table-drop,
→ row-update,  → table-open, and
→ row-delete,  → table-alter operations.

### 3.4.2.5 view-label

The **view-label** resource attribute is the sensitivity label of the database view currently being operated upon. The **view-label** resource attribute always returns a single value of type **label**. It is available for the **view-open, view-create, and view-drop** operations.

### 3.4.2.6 schema-label

The **schema-label** resource attribute is the sensitivity label of the database schema currently being operated upon. The **schema-label** resource attribute always returns a single value of type **label**. It is available for the:

→ row-select,  → index-create,  → table-drop,  → view-open,
→ row-insert,  → index-drop,  → table-alter,  → schema-open,
→ row-update,  → table-open,  → view-create,  → schema-create,
→ row-delete,  → table-create,  → view-drop,  → and schema-drop operations.

### 3.4.2.7 catalog-label

The **catalog-label** resource attribute is the sensitivity label of the database catalog currently being operated upon. The **catalog-label** resource attribute always returns a single value of type **label**. It is available for the:

→ row-select,  → index-drop,  → view-create,  → schema-drop,
→ row-insert,  → index-create,  → view-drop,  → catalog-open,
→ row-update,  → table-create,  → view-open,  → catalog-create,
→ row-delete,  → table-drop,  → schema-open, and
→ index-create,  → table-alter,  → schema-create,  → catalog-drop operations.

### 3.4.2.8 database-label

The **database-label** resource attribute is the sensitivity label of the database currently being operated upon. The **database-label** resource attribute always returns a single value of type **label**. It is available for the all operations.

### 3.4.2.9 column-name

The **column-name** resource attribute is a possibly multi-valued bag of fully specified column names of the database row currently being operated upon. Each name in the bag takes the form:

database_name.catalog_name.schema_name.table_name.column_name

The **column-name** resource attribute always returns a bag of values of type **string**. It is available for the **row-select, row-insert, row-update, and row-delete** operations. For the **row-select** operation only the subset of columns actually being selected are returned. For the **row-insert** operation only the subset of columns that have a non-null value are returned. For the **row-update** operation only the subset of columns that are being updated are returned. For the **row-delete** operation all columns are returned.
3.4.2.10  **table-name**

The *table-name* resource attribute is the fully specified name of the database table currently being operated upon. The name takes the form:

\[
\text{database_name.catalog_name.schema_name.table_name}
\]

The *table-name* resource attribute always returns a single value of type *string*. It is available for the:

- row-select,  
- index-create,  
- table-create,  
- row-insert,  
- index-drop,  
- table-drop,  
- row-update,  
- table-open,  
- and  
- row-delete,  
- table-alter operations.

3.4.2.11 **view-name**

The *view-name* resource attribute is the fully specified name of the database view currently being operated upon. The name takes the form:

\[
\text{database_name.catalog_name.schema_name.view_name}
\]

The *view-name* resource attribute always returns a single value of type *string*. It is available for the *view-open*, *view-create*, and *view-drop* operations.

3.4.2.12 **schema-name**

The *schema-name* resource attribute is the fully specified name of the database schema currently being operated upon. The name takes the form:

\[
\text{database_name.catalog_name.schema_name}
\]

The *schema-name* resource attribute always returns a single value of type *string*. It is available for the:

- row-select,  
- index-create,  
- table-drop,  
- view-open,  
- row-insert,  
- index-drop,  
- table-alter,  
- schema-open,  
- row-update,  
- table-open,  
- view-create,  
- schema-create,  
- and  
- row-delete,  
- table-open,  
- view-drop,  
- schema-drop operations.

3.4.2.13 **catalog-name**

The *catalog-name* resource attribute is the fully specified name of the database catalog currently being operated upon. The name takes the form:

\[
\text{database_name.catalog_name}
\]

The *catalog-name* resource attribute always returns a single value of type *string*. It is available for the:

- row-select,  
- index-drop,  
- view-create,  
- schema-drop,  
- row-insert,  
- table-open,  
- view-drop,  
- catalog-open,  
- row-update,  
- table-create,  
- view-open,  
- catalog-create,  
- row-delete,  
- table-drop,  
- schema-open,  
- and  
- index-create,  
- table-alter,  
- schema-create,  
- catalog-drop operations.
3.4.2.14 database-name

The database-name resource attribute is the name of the database currently being operated upon. The database-name resource attribute always returns a single value of type label. It is available for all operations.

3.4.3 Action Attributes

Action attributes are named attribute values associated with the current database operation. They may be retrieved using the <ActionAttributeDesignator> element where the AttributeId attribute contains the name of the corresponding action attribute.

3.4.3.1 action-id

The action-id action attribute is the identifier for the current database operation. The action-id action attribute always returns a single value of type string. It is available for all operations. It will have one of the following values:

- row-select,  →  table-open,  →  view-drop,  →  catalog-open,
- row-insert,  →  table-create,  →  view-open,  →  catalog-create,
- row-update,  →  table-drop,  →  schema-open,  →  catalog-drop,
- row-delete,  →  table-alter,  →  schema-create,  →  database-open,
- index-create,  →  view-create,  →  schema-drop,  →  database-drop.

3.4.3.2 action-type

The action-type action attribute is the type of the current database operation. The action-type action attribute always returns a single value of type string. It is available for all operations. It will have one of the following values: read, update, drop, create, open. The type of each database operation is given in the following table.

<table>
<thead>
<tr>
<th>Operation Type per Database Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-select</td>
</tr>
<tr>
<td>row-insert</td>
</tr>
<tr>
<td>row-update</td>
</tr>
<tr>
<td>row-delete</td>
</tr>
<tr>
<td>index-create</td>
</tr>
<tr>
<td>index-drop</td>
</tr>
<tr>
<td>table-open</td>
</tr>
</tbody>
</table>

3.4.4 Environment Attributes

Environment attributes are named attribute values associated with the general context of the decision request. They may be retrieved using the <EnvironmentAttributeDesignator> element where the AttributeId attribute contains the name of the corresponding environment attribute.
3.4.4.1 current-time

The current-time environment attribute is the local time of the start of the current decision request. The current-time environment attribute always returns a single value of type time. Multiple requests for current-time within the same decision request will always produce the same value. It is available for all operations.

3.4.4.2 current-date

The current-date environment attribute is the local date of the start of the current decision request. The current-date environment attribute always returns a single value of type date. Multiple requests for current-date within the same decision request will always produce the same value. It is available for all operations.

3.4.4.3 current-dateTime

The current-dateTime environment attribute is the local time and date of the start of the current decision request. The current-dateTime environment attribute always returns a single value of type dateTime. Multiple requests for current-dateTime within the same decision request will always produce the same value. It is available for all operations.

3.4.5 Selecting Fields of the Current Row

Selecting row fields refers to the process by which column values of the row currently being executed upon are accessed as resource attributes. Selecting allows the value of any field in the row being operated upon to be used to make policy decisions. Selecting a column field is only possible when the current database operation is a row based operation (row-select, row-insert, row-update, or row-delete). Selecting fields is accomplished with the <AttributeSelector> element. The field is specified as a fully specified column name.

While both selecting and importing may be used to access field values of rows that are currently being operated upon, selecting is the preferred method. It is faster as it directly accesses the row where importing requires a table open and the explicit reading of rows. Therefore, if selecting a field satisfies the requirements of the policy, it is recommended over importing. Generally, importing is required when multiple column values must be retrieved for a single decision request or the column being accessed is not part of a row currently being operated upon.

As an example, assume we wish to restrict access to each row of a table (db.cat.sch.tab) to the “owner” (user who inserted) the row. To accomplish this we add a column to our table called owner. We can then use the set-field obligation to ensure the value of the owner column is always set to the subject-name context attribute at the time the row was inserted. Then, during subsequent accesses to the table we use the following RXSML code snippet to detect if the current user is the “owner” of the row.

```
1 <Apply FunctionId="equal">
2   <SubjectAttributeDesignator AttributeId="subject-name"/>
3     <AttributeSelector RequestContextPath="db.cat.sch.tab.owner" DataType="string"/>
4   </Apply>
```
In the preceding RXSML code snippet the `equal` function is applied to the `subject-name` context attribute and the value of the `owner` column for the current row being operated upon. The `<Apply>` element will evaluate to `true` if the two are equal; otherwise, it will evaluate to `false`. This logic would then be used within a policy to permit access to the row only if the `equal` function element evaluated to `true`. Note that this assume the target (or some other policy logic) has been used to restrict the decision request to the `db.cat.sch.tab` table.

### 3.4.6 Importing Column Fields

*Importing* column fields refers to the process by which values of database columns which are not necessarily being operated upon by the current database operation are accessed as resource attributes. By importing database fields the value of any database row may be used to make policy decisions. Importing database fields is accomplished using the `<ImportColumnSelector>` and the `<ImportFieldSelector>` elements. The result of the import is a typed multi-valued bag of attributes.

The `<ImportColumnSelector>` element specified the database column that will be imported. All relevant fields are retrieved to main memory upon evaluation of the `<ImportColumnSelector>` element. To limit the number of fields that are imported into the policy, a predicate expression that may operate on an `<ImportFieldSelector>` element is used to select the desired fields. During evaluation of the `<ImportColumnSelector>` element each row in the specified table is iterated over and the predicate expression is applied. If the predicate evaluates to `true` for a given iteration then the corresponding field is added to the resultant bag of typed values. The `<ImportFieldSelector>` element may be used as an argument to the predicate expression to retrieve the value of any field in the table row of the current iteration.

As an example, assume we have a policy where we wish to specify which users may connect from which network addresses. Any user connecting from a non-specified network address must be denied access. To accomplish this we can initialize a table containing one column for the user's name and one column for the permitted network address from which they may connect. There is one row for each permissible user name / network address pair. The following shows such a table:

<table>
<thead>
<tr>
<th>user</th>
<th>ipaddr</th>
<th>user</th>
<th>ipaddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>warner</td>
<td>127.0.0.1</td>
<td>bush</td>
<td>100.101.108.103</td>
</tr>
<tr>
<td>warner</td>
<td>100.101.102.1</td>
<td>bush</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>smith</td>
<td>100.101.107.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The values of this table indicate that the user `warner` is allowed to connect from 127.0.0.1 and 100.101.102.1, the user `smith` is allowed to connect from 100.101.107.6, and the user `bush` is allowed to connect from 127.0.0.1 and 100.101.108.103. Connections from all other user name / network address pairs should be denied. Assume the fully specified paths to the two table columns are `db.cat.sch.iptab.user` and `db.cat.sch.iptab.ipaddr`. Both columns are of SQL type CHARACTER VARYING.

The following RXSML code snippet contains the logic to produce a boolean `true` or `false` if the current connection is permissible or not permissible. This code could be included in a policy with a target of the `action-id` context attribute with a value of `db-open`.

© 2014 Infosystems Technology, Inc.
For explanation we will start with the <ImportColumnSelector> element that spans line 3 to line 8. This element indicates that the db.cat.sch.iptab.ipaddr column is to be imported and converted from the SQL type CHARACTER VARYING to the RXSML type ipAddress.

The <ImportColumnSelector> element takes one child that must be a predicate element. In this case it is the <Apply> element with the equal function that spans line 4 to line 7. When the <ImportColumnSelector> element is evaluated, the rows in the iptab table will be iterated over. Each row will be read and the equal function applied. If the equal function evaluates to true then the ipaddr column value for that row will be added to the resultant bag of values.

For each iteration the equal function is applied with the <ImportFieldSelector> and <SubjectAttributeDesignator> elements as arguments. The <ImportFieldSelector> element indicates that the user column value for the currently iterated row is the first argument to the equal function. The <SubjectAttributeDesignator> element indicates that the second argument to the equal function is the subject-name subject attribute. Thus, if the current user’s name is equal to the value of the user column, the predicate evaluates to true and the corresponding ipaddr column value will be added to our bag of result values.

If the <ImportColumnSelector> element is evaluated and the subject-name is bush then the resultant bag of values will contain 100.101.108.103 and 127.0.0.1. If the subject-name is smith then the resultant bag of values will contain 100.101.107.6. If the subject-name is foobar (or any that has no row in the table) then the resultant bag of values will be empty. Thus, the bag of values will contain the network addresses that are permissible for the current user. In SQL terms, the use of <ImportColumnSelector> element here mimics the SQL statement “SELECT ipaddr WHERE user = subject-name”.

The bag of ipAddress typed values produced by the <ImportColumnSelector> element is the second argument to the is-in function specified by the <Apply> element that spans line 1 to line 9. The first argument is the ip-address subject attribute of line 2. The is-in function will return true if any value in the bag of the second argument is equal to the value of the first argument. Thus, the is-in function will return true if the subject’s network address is equal to any network address in the set of permissible values. Note that the case where a user does not have an entry in the iptab table will have an empty bag of values from the <ImportColumnSelector> element and the is-in function will return false.

It may be noted that the logic of restricting users to connecting from specific network addresses may be accomplished completely through literal values and RXSML logic. That is, the mapping between the user
and permissible network address need not be stored in a table it may be stored within the policy’s RXSML itself as literal values. The advantage of storing the information in a database table is that it abstracts the policy’s security logic from the storage of the policy’s security data. Changing the permissible user / network address mappings is simply a matter of updating the table and the policy’s RXSML does not change, thus reducing potential errors. It does, however, require that modifications to the database table holding the security data be restricted to authorized administrative users. Consequently, it is recommended that all such security data be isolated within a single database schema with strict RXSML policy and DAC policy protections.

3.5 Obligations

Obligations are actions that may conditionally accompany a decision request. Trusted Rubix RXSML supports three obligations:

→ to set a column field with an attribute value before a row-select, row-insert, or row-update is performed (called the set-field obligation);
→ to set the SQL error code when a policy decision denies an operation (called the set-error-code obligation);
→ and to perform security auditing (called the audit obligation).

A set of obligations (an <Obligations> element) may exist within a <Policy> or <PolicySet> element. Each obligation (an <Obligation> element) has a FulfillOn attribute that determines the condition under which the obligation will be executed. The FulfillOn attribute may be Permit or Deny. The obligation is executed if the following three conditions are true:

1. The parent <Policy> or <PolicySet> element is evaluated. The ordering of the elements within a <PolicySet>, the specified policy combining algorithm, and the outcome of the various <Policy> or <PolicySet> elements determines if a particular <Policy> or <PolicySet> element is evaluated.

2. The outcome of the parent <Policy> or <PolicySet> element is equal to the FulfillOn attribute of the obligation.

3. The final top-level outcome is equal to the FulfillOn attribute.

Obligation executions are ordered firstly according to the evaluation order of the parent policy and secondly according to the order that the obligations are specified within the policy. All obligations that satisfy the above three conditions for a particular decision request are executed, even if previous obligations evaluate to Indeterminate. If any obligation fails during execution an appropriate SQL error code is set and the decision request outcome denies the current operation.

If obligations are used within policy sets it is highly recommended that a deterministic policy-combining algorithm (ordered, only-one-applicable, or first-applicable) be used. A deterministic policy-combining algorithm allows the execution order of the policies (and thus the obligations) to be known for a given decision request. Using non-deterministic policy-combining algorithms in combination with obligations will result in non-deterministic obligation execution.

3.5.1 set-field

The set-field obligation allows a column field of the current row to be set with an attribute value during a row-select, row-insert, or row-update operation. Used during the row-select operation, this obligation allows a dynamic, run-time “cover value” to be assigned during the selecting of database tables. Used
during the *row-insert* and *row-update* operations it provides a mechanism to assure that a column value always holds a specified attribute value, such as the user name of the subject performing the operation. Typically, the value set during the *row-insert* or *row-update* operation would be used for subsequent policy decisions using the `<AttributeSelector>` element. Using *set-field* during the *row-update* operation will cause the specified field to be set (updated) for every SQL update operation on the row, irrespective of the set of columns the SQL update operation targets.

The *set-field* obligation for *row-select* must not be applied to columns that have an index defined. Doing so will produce unspecified results during select as the ordering of the index will be violated when the field is set with the literal value. Also, the *set-field* obligation for *row-update* should not be applied to columns that are the target of a cascaded update integrity constraint as the cascaded update integrity constraint may overwrite the results of a previous *set-field* operation, thus nullifying the desired effect of the policy.

The *set-field* obligation has two arguments. The first argument is of type *string* and specifies the five part database path of the column to be set. The path has the form:

\[\text{database\_name.catalog\_name.schema\_name.table\_name.column\_name}\]

The second argument is an expression that evaluates to a value used to set the column field. The RXSML data type of the value must be convertible to the SQL data type of the column. If the specified path is not current or the current operation is not *row-select*, *row-insert*, or *row-update* then the obligation evaluates successfully but takes no action. If the data type is not convertible or any child element evaluates to *Indeterminate* then the obligation evaluates to *Indeterminate*.

As an example, assume we have an SQL VARCHAR column named \textit{db.cat.sch.tab.org} that holds the US government organization a person is employed by. We wish to allow all users from within the US to view the column “as is” but require all non-US users to only see a generic “US Government Employee” value. Also assume that the *group-name* subject attribute specifies that a user is from the US by having a value of *USEmployee*. The following RXSML code snippet shows how the *set-field* obligation can be used accordingly. Assume the policy is only assigned to the \textit{db.cat.sch.tab} table.
The policy’s target spans lines 3 through 15. The <ActionMatch> starting on line 5 limits the application of our policy to row-select operations. The <SubjectMatch> starting on line 10 limits our policy to subjects whose group-name is not “USEmployee”. Note that we have no <ResourceMatch> to target the db.cat.sch.tab table because the policy is assigned to the db.cat.sch.tab table, giving us the implicit target logic. The result of the target will be that it is applicable only for non-US users performing a row-select on the db.cat.sch.tab table. For all other cases the policy will evaluate to Not Applicable and the obligation will not be executed.

The <Rule> on line 17 simply assigns the Permit effect to the policy evaluation. The <Rule> has no <Condition> or <Target> so the rule always has an outcome of its Effect, Permit in this case. The logic for the policy is contained entirely within the target. Therefore, if the target evaluates to Match the policy will evaluate to Permit.
The obligation spans lines 19 through 22. The *FulfillOn* attribute is *Permit*. It will be executed if the policy outcome is *Permit*. Note that our policy is also the top-level policy for the object; otherwise, the top-level policy would also need to evaluate to *Permit* for the obligation to be executed.

The first argument of the obligation specifies that the *db.cat.sch.tab.org* column is to be set. If the row-select does not include the *db.cat.sch.tab.org* column then the obligation will execute successfully but take no action. The second argument of the obligation specifies that the column is to be set to the literal value “US Government Employee.”

Assume the *db.cat.sch.tab* table holds the following rows:

```
<table>
<thead>
<tr>
<th>name</th>
<th>org</th>
<th>name</th>
<th>org</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael Hayden</td>
<td>CIA</td>
<td>George Bush</td>
<td>White House</td>
</tr>
<tr>
<td>Keith Alexander</td>
<td>NSA</td>
<td>John McCain</td>
<td>Senate</td>
</tr>
</tbody>
</table>
```

If a user with a group “USEmployee” issues the SQL command:

```
SELECT * FROM db.cat.sch.tab;
```

They will receive a result set of:

```
<table>
<thead>
<tr>
<th>name</th>
<th>org</th>
<th>name</th>
<th>org</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael Hayden</td>
<td>CIA</td>
<td>George Bush</td>
<td>White House</td>
</tr>
<tr>
<td>Keith Alexander</td>
<td>NSA</td>
<td>John McCain</td>
<td>Senate</td>
</tr>
</tbody>
</table>
```

If a user with a group not equal to “USEmployee” issues the SQL command:

```
SELECT * FROM db.cat.sch.tab;
```

They will receive a result set of:

```
<table>
<thead>
<tr>
<th>name</th>
<th>org</th>
<th>name</th>
<th>org</th>
</tr>
</thead>
</table>
```

In each case no matter how the user selects the rows they will always see the same values for the *org* column. That is, if they do a direct select on the table or a select a view that references the table, the obligation will still appropriately cover the value of the *org* column.

As a second example, assume it is required that a column always contains the user name of the user who inserted the row. We will call this user the “owner” of the row, even though SQL has no such concept. For our example the column that contains the owner of the row is *db.cat.sch.tab.owner*. The following RXSML code snippet shows how the *set-field* obligation can be used accordingly. Assume the policy is only assigned to the *db.cat.sch.tab* table.
The result of the target spanning lines 27 through 34 is that the policy is applicable for all row-insert operations. The rule of line 36 always sets the outcome to Permit. As in the previous example, all of the policy logic is in the target.

If the target evaluates to Match then the policy outcome is Permit and the obligation of lines 38 through line 41 will be executed. The obligation will set the owner column of the db.cat.tab table to the value of the subject-name attribute during the row-insert operation. Typically, the owner column would be accessed using the <AttributeSelector> element of another policy to constrain row access to the owner of the row. Also note that the row-update operation for the owner column needs to be constrained by another policy to maintain the invariant that the column always contains the owner of the row.

### 3.5.2 set-error-code

The set-error-code obligation allows specification of the SQL error that is set when an operation is denied due to policy violation, overriding the default SQL error code. The obligation has a single argument that must be the <ErrorCode> element. The desired error code is specified using the ErrorCodeId attribute and may have a value of object-not-found or policy-violation. A value of object-not-found may be used to hide the existence of an object. In this case an SQL error code suitable for the current database operation is set when the decision request outcome is Deny. The user cannot distinguish between a policy violation and a non-existing object. A value of policy-violation may be used to explicitly cause the operation to fail with an SQL error reflecting a security policy violation. The set-error-code obligation is only useful when
the outcome of policy evaluation is *Deny*. If the outcome is not *Deny* then the obligation executes successfully but no action is taken.

The default SQL error (when no set-error-code obligation is executed) for security policy denial is *object-not-found* for the *row-select* operation and *policy-violation* for all other operations. Setting the SQL error to *object-not-found* during a row select has the behavior of ignoring rows that cause a security policy violation, effectively filtering the rows. For example, if a user issues a “SELECT * FROM tab” command only the rows that pass the security policy check will be returned in the result set. However, if the *set-error-code* obligation is used to override the error code and set it to *policy-violation* then the “SELECT * FROM tab” command will fail with an error indicating the operation failed due to a security policy violation. For operations other than *row-select* the opposite behavior is found. The default error behavior is to indicate a policy violation occurred. This allows a user to distinguish between a policy violation and if the object does not exist, by observing the error code. If it is desired to hide the existence of the object then the *set-error-code* obligation may be used and an SQL error code indicating the object does not exist will be returned when the security policy violation occurs.

### 3.5.3 audit

The *audit* obligation allows security auditing according to the logic of a security policy. The *audit* obligation has one *optional* argument which must evaluate to a *string* type. As much information as possible is audited given any errors or *Indeterminate* evaluations that occur during the execution of the *audit* obligation. It is anticipated that the *concatenate* function will be used to construct a string that contains the desired attributes to be audited.

The audit record will have an event type of *sql_spm_event*. The audit record will contain the following common audit data in addition to any optional audit string provided as an argument:

- → subject user name, → operation outcome, → *action-id* action attribute,
- → subject group name, → timestamp, → *resource-name* resource attribute,
- → database name, → process ID, → *resource-label* resource attribute,
- → session label, → session ID, → and policy evaluation outcome.

As an example assume that we allow a user (named *smith*) to violate the MAC security policy to open a database (named *db*) but we require the operation be audited, including the client’s network address and domain name. The following RXSML code snippet contains a policy that will accomplish our requirements. Assume the policy is only applied to the *db* database.
The `<Policy>` element of lines 1-2, and specifically the MAC Override attribute being true, allow this policy to override the MAC security policy. This means that the policy may allow the `database-open` operation to occur even if the subject’s session label does not dominate the database’s object label. Note that this policy must be the top-level policy for the MAC override to take effect. In practice this policy would also need to allow `database-open` operations by other users where the MAC policy is not violated, presumably using the MAC-check function.
The `<Target>` element spanning lines 4-16 limits the applicability of the policy to the `database-open` operation (the `<ActionMatch>` of lines 6-9) being performed by the user `smith` (the `<SubjectMatch>` of lines 11-14). The target is also implicitly limited to the `db` database object because this is the only object the policy is assigned to.

The `<Rule>` on line 18 simply assigns the `Permit` effect to the policy evaluation. The `<Rule>` has no `<Condition>` or `<Target>` so the rule always has an outcome of its Effect, `Permit` in this case. The logic for the policy is contained entirely within the target. Therefore, if the target evaluates to `Match` the policy will evaluate to `Permit`.

If the target evaluates to `Match` then the policy outcome is `Permit` and the obligation of lines 20 through line 28 will be executed. The obligation will create an audit record with all common audit data plus the string returned by the `concatenate` function of lines 22-26. The `concatenate` function will concatenate the string representations of the values returned by its three arguments. The first argument, of lines 23, casts the `ip-address` subject context attribute as a `string`. The second argument, of line 24, is a literal string of `|---|` and is used as a separator between the network address and domain name. The third argument, of lines 25, is the `dns-name` subject context attribute cast as a `string`.

### 3.6 Rule and Policy Combining Algorithms

#### 3.6.1 Rule Combining Algorithms

##### 3.6.1.1 deny-overrides rule-combining algorithm

The intent of the `deny-overrides` rule-combining algorithm is to produce an outcome of `Deny` if any `<Rule>` element evaluates to `Deny`. If there is any potential for a `Deny` outcome but the outcome cannot be determined due to an `Indeterminate` outcome then the rule-combining outcome is `Indeterminate`. If there is no actual or potential `Deny` outcome and at least one outcome is `Permit` then the rule-combining outcome is `Permit`.

The following rules specify the behavior of the `deny-overrides` rule-combining algorithm of a `<Policy>` element:

→ For a rule-combination outcome of `Deny`:

↑ **Any** of the `<Rule>` elements must evaluate to `Deny`. This is true **regardless** of any outcome of `Permit`, `Not Applicable`, or `Indeterminate` that may occur during the evaluation of other `<Rule>` elements. If a `<Rule>` element is evaluated with an outcome of `Deny` then evaluation of other `<Rule>` elements is not performed as their evaluation would not affect the rule-combination outcome.

→ For a rule-combination outcome of `Permit`:

↑ **At least one** `<Rule>` element must evaluate to `Permit`. **All** other `<Rule>` elements must evaluate to one of the three following outcomes:

1. `Permit`
2. `Not Applicable`
3. `Indeterminate` **and** have a rule Effect of `Permit`.

→ For a rule-combination outcome of `Not Applicable`:

↑ **All** `<Rule>` elements must evaluate to `Not Applicable`.

→ For a rule-combination outcome of `Indeterminate`:

↑ **Any** of the following must be true:
1. Any `<Rule>` element evaluates to `Indeterminate` and all other `<Rule>` elements evaluate to `Indeterminate` or `Not Applicable`.

2. Any `<Rule>` element evaluates to `Indeterminate` with an Effect of `Deny` and no other `<Rule>` element evaluates to `Deny`.

3.6.1.2 ordered-deny-overrides rule-combining algorithm

The behavior of the ordered-deny-overrides rule-combining algorithm is identical to that of the deny-overrides rule-combining algorithm except that the order in which the `<Rule>` elements are evaluated is guaranteed to be the same as the order the `<Rule>` elements occur within the parent `<Policy>` element.

A change in the evaluation ordering of the `<Rule>` elements will not change the rule-combining outcome. However, the ordering may be used to increase performance by ensuring that more general rules are executed first, thus reducing the average lines of policy code needed to reach a decision request.

3.6.1.3 permit-overrides rule-combining algorithm

The intent of the permit-overrides rule-combining algorithm is to produce an outcome of `Permit` if any `<Rule>` element evaluates to `Permit`. If there is any potential for a `Permit` outcome but the outcome cannot be determined due to an `Indeterminate` outcome then the rule-combining outcome is `Indeterminate`. If there is no actual or potential `Permit` outcome and at least one outcome is `Deny` then the rule-combining outcome is `Deny`.

The following rules specify the behavior of the permit-overrides rule-combining algorithm of a `<Policy>` element:

→ For a rule-combination outcome of `Permit`:
  
  ↑ Any of the `<Rule>` elements must evaluate to `Permit`. This is true regardless of any outcome of `Deny`, `Not Applicable`, or `Indeterminate` that may occur during the evaluation of other `<Rule>` elements. If a `<Rule>` element is evaluated with an outcome of `Permit` then evaluation of other `<Rule>` elements is not performed as their evaluation would not affect the rule-combining outcome.

→ For a rule-combination outcome of `Deny`:
  
  ↑ At least one `<Rule>` element must evaluate to `Deny`. All other `<Rule>` elements must evaluate to one of the three following outcomes:

  1. `Deny`
  2. `Not Applicable`
  3. `Indeterminate` and have a rule Effect of `Deny`.

→ For a rule-combination outcome of `Not Applicable`:
  
  ↑ All `<Rule>` elements must evaluate to `Not Applicable`.

→ For a rule-combination outcome of `Indeterminate`:
  
  ↑ Any of the following must be true:

  1. Any `<Rule>` element evaluates to `Indeterminate` and all other `<Rule>` elements evaluate to `Indeterminate` or `Not Applicable`.
  2. Any `<Rule>` element evaluates to `Indeterminate` with an Effect of `Permit` and no other `<Rule>` element evaluates to `Permit`. 

© 2014 Infosystems Technology, Inc.
3.6.1.4 ordered-permit-overrides rule-combining algorithm

The behavior of the ordered-permit-overrides rule-combining algorithm is identical to that of the permit-overrides rule-combining algorithm except that the order in which the <Rule> elements are evaluated is guaranteed to be the same as the order the <Rule> elements occur within the parent <Policy> element.

A change in the evaluation ordering of the <Rule> elements will not change the rule-combining outcome. However, the ordering may be used to increase performance by ensuring that more general rules are executed first, thus reducing the average lines of policy code needed to reach a decision request.

3.6.1.5 first-applicable rule-combining algorithm

The intent of the first-applicable rule-combining algorithm is to cause the rule-combining outcome to be equal to the first applicable <Rule> element, according to its target and condition, when the <Rule> elements are executed in the order they occur in the parent <Policy> element. The rule-combining outcome is calculated by evaluating the <Rule> elements in order and choosing the first Permit, Deny, or Indeterminate result. The evaluation of <Rule> elements stops as soon as a rule-combining outcome is chosen. If all <Rule> elements evaluate to Not Applicable then the rule-combining outcome is Not Applicable.

3.6.2 Policy Combining Algorithms

3.6.2.1 deny-overrides policy-combining algorithm

The intent of the deny-overrides policy-combing algorithm is to produce an outcome of Deny if any <Policy>, <PolicySet>, <PolicyIdReference>, or <PolicySetIdReference> element evaluates to Deny. If there is any potential for a Deny outcome but the outcome cannot be determined due to an Indeterminate outcome then the policy-combining outcome is Indeterminate. If there is no actual or potential Deny outcome and at least one outcome is Permit then the policy-combining outcome is Permit.

The following rules specify the behavior of the deny-overrides policy-combining algorithm of a <PolicySet> element:

→ For a policy-combination outcome of Deny:
  ↑ Any of the <Policy>, <PolicySet>, <PolicyIdReference>, or <PolicySetIdReference> elements must evaluate to Deny. This is true regardless of any outcome of Permit, Not Applicable, or Indeterminate that may occur during the evaluation of other policy elements. If a policy element is evaluated with an outcome of Deny then evaluation of other policy elements is not performed as their evaluation would not affect the policy-combination outcome.

→ For a policy-combination outcome of Permit:
  ↑ At least one <Policy>, <PolicySet>, <PolicyIdReference>, or <PolicySetIdReference> element must evaluate to Permit. All other policy elements must evaluate to either Permit or Not Applicable.

→ For a policy-combination outcome of Not Applicable:
  ↑ All <Policy>, <PolicySet>, <PolicyIdReference>, or <PolicySetIdReference> elements must evaluate to Not Applicable.

→ For a policy-combination outcome of Indeterminate:
  ↑ Any <Policy>, <PolicySet>, <PolicyIdReference>, or <PolicySetIdReference> element must evaluate to Indeterminate and no other policy element may evaluate to Deny.
### 3.6.2.2 ordered-deny-overrides policy-combining algorithm

The behavior of the ordered-deny-overrides policy-combining algorithm is identical to that of the deny-overrides policy-combining algorithm except that the order in which the `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, and `<PolicySetIdReference>` elements are evaluated is guaranteed to be the same as the order the elements occur within the parent `<PolicySet>` element.

A change in the evaluation ordering of the policy elements will not change the policy-combining outcome. However, the ordering may be used to increase performance by ensuring that more general policies are executed first, thus reducing the average lines of policy code needed to reach a decision request. Also, ordering the policy evaluations will impact whether an obligation is executed during a given decision request. That is, if a Deny outcome is reached in policy evaluation then subsequent policy elements will not be evaluated and their corresponding obligations will not be executed. It is therefore highly recommended that a deterministic policy-combining algorithm (ordered, only-one-applicable, or first-applicable) be used if obligations are used within policy sets.

### 3.6.2.3 permit-overrides policy-combining algorithm

The intent of the permit-overrides policy-combining algorithm is to produce an outcome of Permit if any `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` element evaluates to Permit.

If there is any potential for a Permit outcome but the outcome cannot be determined due to an Indeterminate outcome then the policy-combining outcome is Indeterminate. If there is no actual or potential Permit outcome and at least one outcome is Deny then the policy-combining outcome is Deny.

The following rules specify the behavior of the permit-overrides policy-combining algorithm of a `<PolicySet>` element:

- For a policy-combination outcome of Permit:
  - Any of the `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` elements must evaluate to Permit. This is true regardless of any outcome of Deny, Not Applicable, or Indeterminate that may occur during the evaluation of other policy elements. If a policy element is evaluated with an outcome of Permit then evaluation of other policy elements is not performed as their evaluation would not affect the policy-combining outcome.

- For a policy-combination outcome of Deny:
  - At least one `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` element must evaluate to Deny. All other policy elements must evaluate to either Deny or Not Applicable.

- For a policy-combination outcome of Not Applicable:
  - All `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` elements must evaluate to Not Applicable.

- For a policy-combination outcome of Indeterminate:
  - Any `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` element must evaluate to Indeterminate and no other policy element may evaluate to Permit.

### 3.6.2.4 ordered-permit-overrides policy-combining algorithm

The behavior of the ordered-permit-overrides policy-combining algorithm is identical to that of the permit-overrides policy-combining algorithm except that the order in which the `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, and `<PolicySetIdReference>` elements are evaluated is guaranteed to be the same as the order the elements occur within the parent `<PolicySet>` element.
A change in the evaluation ordering of the policy elements will not change the policy-combining outcome. However, the ordering may be used to increase performance by ensuring that more general policies are executed first, thus reducing the average lines of policy code needed to reach a decision request. Also, ordering the policy evaluations will impact whether an obligation is executed during a given decision request. That is, if a Permit outcome is reached in policy evaluation then subsequent policy elements will not be evaluated and their corresponding obligations will not be executed. It is therefore highly recommended that a deterministic policy-combining algorithm (ordered, only-one-applicable, or first-applicable) be used if obligations are used within policy sets.

### 3.6.2.5 only-one-applicable policy-combining algorithm

The intent of the only-one-applicable policy-combining algorithm is to allow a single policy element to be applicable, according to its target, for a given decision request. The outcome of the decision request is the outcome of the applicable policy. The following rules specify the behavior of the only-one-applicable policy-combining algorithm of a `<PolicySet>` element:

- For a policy-combining outcome of Permit:
  - A single `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` element must evaluate to Permit and all other policy element must evaluate to Not Applicable.
- For a policy-combining outcome of Deny:
  - A single `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` element must evaluate to Deny and all other policy elements must evaluate to Not Applicable.
- For a policy-combining outcome of Indeterminate one of the following must be true:
  1. At least one `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` element must evaluate to Indeterminate.
  2. More than one policy element evaluates to Permit or Deny.
- For a policy-combining outcome of Not Applicable:
  - All `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` elements must evaluate to Not Applicable.

### 3.6.2.6 first-applicable policy-combining algorithm

The intent of the first-applicable policy-combining algorithm is to cause the policy-combining outcome to be equal to the first applicable policy element, according to its target, when the policy elements are executed in the order they occur in the parent `<PolicySet>` element. The policy-combining outcome is calculated by evaluating the policy elements in order and choosing the first Permit, Deny, or Indeterminate result. The evaluation of policy elements stops as soon as a policy-combining outcome is chosen. If all policy elements evaluate to Not Applicable then the policy-combining outcome is Not Applicable.

### 3.7 Indeterminate Execution and System Error Handling

There are two types of abnormal processing events that may occur during the evaluation of security policies. The first is indeterminate execution and the second is system error during execution.
3.7.1 Indeterminate Execution

Indeterminate execution is the result of a normal runtime exception that prevents a policy evaluation from reaching a definitive outcome. This may be the result of improper RXSML code or a runtime error specific to the context. Common examples of situations that result in indeterminate executions are:

→ Improper XML syntax that is not checked during editing or parsing.
→ Function arguments with improper number, type, or value.
→ Literal values that do not conform to their specified data type.
→ Mathematical exceptions such as overflow or divide by zero.
→ Referencing a context attribute that is not available for the current operation (e.g., referencing the column-name attribute during a table-open operation).
→ Referencing a database object by its path when the object is not currently being operated upon (e.g., referencing the db.cat.sch.tab.col column when the current object is the db.cat.sch.tab2.col column).
→ Transaction conflicts (e.g., reading an uncommitted value) during a column import.

If an RXSML element evaluation results in indeterminate execution the outcome is Indeterminate. The Indeterminate outcome is passed to the parent element and, in general but not always, causes the parent element to also evaluate to Indeterminate. The cause of the indeterminate execution is logged to the Trusted Rubix error log. Any top-level policy evaluation outcome of Indeterminate results in the operation being denied and an SQL error indicating indeterminate policy execution being returned to the user.

It is possible to rely on the SPM’s indeterminate execution behavior for normal policy evaluation; however, it is strongly discouraged. Any indeterminate behavior resulting from the RXSML code should be removed.

3.7.2 System Error

A system error is the result of any abnormal error originating from the underlying operating system or related library. System errors are considered fatal errors within TR. When a system error occurs the following occurs, if possible:

→ an SQL error code is sent to the user, → all work of the current transaction is rolled back,
→ a log message is produced in the TR error log, → the server process exits.

Therefore, if a system error occurs during the evaluation of a security policy its current database operation will not be performed.