translation that would result from following the instructions in the remainder of this section.

A WebResourcePermission and a WebUserDataPermission[3] object must be added to the excluded policy statements for each distinct `url-pattern` occurring in the `security-constraint` elements that contain an `auth-constraint` naming no roles (i.e an excluding `auth-constraint`). The permissions must be constructed using the qualified (as defined in "Qualified URL Pattern Names") pattern as their name and with actions obtained by combining (as defined in "Combining HTTP Methods") the collections containing the pattern and occurring in a constraint with an excluding `auth-constraint`. The constructed permissions must be added to the excluded policy statements by calling the `addToExcludedPolicy` method on the `PolicyConfiguration` object.

A WebResourcePermission must be added to the corresponding role for each distinct combination in the cross-product of `url-pattern` and `role-name` occurring in the `security-constraint` elements that contain an `auth-constraint` naming roles. When an `auth-constraint` names the reserved `role-name`, "*", all of the patterns in the containing `security-constraint` must be combined with all of the roles defined in the web application. Each WebResourcePermission object must be constructed using the qualified pattern as its name and with actions defined by combining (as defined in "Combining HTTP Methods") the collections containing the pattern and occurring in a constraint that names (or implies via "*") the role to which the permission is being added. The resulting permissions must be added to the corresponding roles by calling the `addToRole` method on the `PolicyConfiguration` object.

A WebResourcePermission must be added to the unchecked policy statements for each distinct `url-pattern` occurring in the `security-constraint` elements that do not contain an `auth-constraint`. Each WebResourcePermission object must be constructed using the qualified pattern as its name and with actions defined by combining (as defined in "Combining HTTP Methods") the collections containing the pattern and occurring in a `security-constraint` without an `auth-constraint`. The resulting permissions must be added to the unchecked policy statements by calling the `addToUncheckedPolicy` method on the `PolicyConfiguration` object.

A WebUserDataPermission must be added to the unchecked policy statements for each distinct combination of `url-pattern` and acceptable connection type

[3] The WebUserDataPermission objects allow a container to determine when to reject a request before redirection if it would ultimately be rejected as the result of an excluding `auth-constraint`.

resulting from the processing of the `security-constraint` elements that do not contain an excluding `auth-constraint`. The mapping of security-constraint to acceptable connection type must be as defined in "Mapping Transport Guarantee to Connection Type". Each WebUserDataPermission object must be constructed using the qualified pattern as its name and with actions defined by appending[4] a representation of the acceptable connection type to the HTTP method specification obtained by combining (as defined in "Combining HTTP Methods) the collections containing the pattern and occurring in a `security-constraint` that maps to the connection type and that does not contain an excluding `auth-constraint`. The resulting permissions must be added to the unchecked policy statements by calling the `addToUncheckedPolicy` method on the `PolicyConfiguration` object.

A WebResourcePermission and a WebUserDataPermission must be added to the unchecked policy statements for each `url-pattern` in the deployment descriptor and the default pattern, "/", that is not combined by the `web-resource-collection` elements of the deployment descriptor with every possible HTTP method value[5]. The permission objects must be constructed using the qualified pattern as their name and with actions represented by an HTTP method specification that identifies all of the HTTP methods that do not occur in combination with the pattern. The resulting permissions must be added to the unchecked policy statements by calling the `addToUncheckedPolicy` method on the `PolicyConfiguration` object. The form of the HTTP method specification used in the permission construction depends on the representation of the methods that occurred in combination with the pattern. If the methods that occurred are represented by an HttpMethodExceptionList as defined in "HTTP Method Exception List"), the permissions must be constructed using an HTTPMethodList naming of all of the HTTP methods named in the exception list. Conversely, if the methods that occurred are represented by an HTTPMethodList, the permissions must be constructed using an HTTPMethodExceptionList naming all of the HTTP methods that occurred with the pattern.

---

[4] The value null should be used as the actions value in the construction of a WebUserDataPermission when both the HTTP method specification, and the representation of the acceptable connection type may be represented by null. If only one of the action components may be represented by null the other should be used as the actions value.

[5] The set of all possible HTTP methods is non-enumerable and contains the traditional HTTP methods (i.e., DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE) and any method conforming to the "extension-method" syntax defined in IETF RFC 2616 "Hypertext Transfer Protocol -- HTTP/1.1". A null or the emptyString HTTP method specification is used to this set.

Qualified URL Pattern Names

The URL pattern qualification described in this section serves to capture the best-matching semantics of the Servlet constraint model in the qualified names such that the WebResourcePermission and WebUserDataPermission objects can be tested using the standard Java SE permission evaluation logic.

The WebResourcePermission and WebUserDataPermission objects resulting from the translation of a Servlet deployment descriptor must be constructed with a name produced by qualifying the URL pattern. The rules for qualifying a URL pattern are dependent on the rules for determining if one URL pattern matches another as defined in Section 3.1.3.3, “Servlet URL-Pattern Matching Rules”, and are described as follows:

- If the pattern is a path prefix pattern, it must be qualified by every path-prefix pattern in the deployment descriptor matched by and different from the pattern being qualified. The pattern must also be qualified by every exact pattern appearing in the deployment descriptor that is matched by the pattern being qualified.
- If the pattern is an extension pattern, it must be qualified by every path-prefix pattern appearing in the deployment descriptor and every exact pattern in the deployment descriptor that is matched by the pattern being qualified.
- If the pattern is the default pattern, "/", it must be qualified by every other pattern except the default pattern appearing in the deployment descriptor.
- If the pattern is an exact pattern, its qualified form must not contain any qualifying patterns.

URL patterns are qualified by appending to their String representation, a colon separated representation of the list of patterns that qualify the pattern. Duplicates must not be included in the list of qualifying patterns, and any qualifying pattern matched by another qualifying pattern may[6] be dropped from the list.

```
QualifyingPatternList ::=
    empty string | colon QualifyingPattern |
    QualifyingPatternList colon QualifyingPattern

QualifiedPattern ::= Pattern QualifyingPatternList
```

6. Qualifying patterns implied by another qualifying pattern may be dropped because the use of the reduced list to qualify a pattern will yield a URLPatternSpec “equal” to the URLPatternSpec produced by qualifying the pattern with the full list (for example, /a/*:/a/b:/a/b/*:/a/b/c/* is equal to /a/*:/a/b/*).

All colon characters occurring within Pattern and QualifyingPattern elements must be transformed to escaped encoding[7] prior to inclusion of the corresponding element in the QualifiedPattern.

Any pattern, qualified by a pattern that matches it, is overridden and made irrelevant (in the translation) by the qualifying pattern. Specifically, all extension patterns and the default pattern are made irrelevant by the presence of the path prefix pattern "/*" in a deployment descriptor. Patterns qualified by the "/*" pattern violate the URLPatternSpec constraints of WebResourcePermission and WebUserDataPermission names and must be rejected by the corresponding permission constructors.

Combining HTTP Methods

The section defines the rules for combining HTTP method names occurring in `web-resource-collection` elements that apply to a common `url-pattern`. The rules are commutative and associative and are as follows:

- Lists of `http-method` elements combine to yield a list of `http-method` elements containing the union (without duplicates) of the `http-method` elements that occur in the individual lists.
- Lists of `http-method-omission` elements combine to yield a list containing only the `http-method-omission` elements that occur in all of the individual lists (i.e., the intersection).
- A list of `http-method-omission` elements combines with a list of `http-method` elements to yield the list of `http-method-omission` elements minus any elements whose method name occurs in the `http-method` list.
- An empty list (of `http-method` and `http-method-omission` elements) represents the set of all possible HTTP methods, including when it results from combination according to the rules described in this section. An empty list combines with any other list to yield the empty list.

When these combining rules are applied to a list of collections, the result is always either an empty list, a non-empty list of `http-method` elements, or a non-empty list of `http-method-omission` elements. When the result is an empty list, the corresponding actions value is the null (or the empty string) value. When the result is a non-empty list of `http-method` elements the corresponding actions value is a comma separated list of the HTTP method names occurring in the `http-method` elements of the list. When the result is a non-empty list of

[7]. See Section B.22, "Colons Within path-segment of Request URI for details.

`http-method-omission` elements the corresponding actions value is an HTTP method exception list (as defined in “HTTP Method Exception List”) containing the HTTP method names occurring in the `http-method-omission` elements of the list. The following table contains the three alternative combination results and their corresponding actions values.

TABLE 3-1 HTTP Method Combination to Actions Correspondence

Combination Result	*Actions Value*
empty list	null or empty string
list of http-method elements	HttpMethodList (e.g., “GET,POST”)
list of http-method-omission elements	HttpMethodExceptionList (e.g.,”!PUT,DELETE”)

HTTP Method Exception List

An HTTP method exception list is used to represent, by set difference, a non-enumerable subset of the set of all possible HTTP methods. An exception list represents the subset of the complete set of HTTP methods formed by subtracting the methods named in the exception list from the complete set.

An exception list is distinguished by its first character, which must be the exclaimation point (i.e., “!”) character. A comma separated list of one or more HTTP method names must follow the exclaimation point. The syntax of an HTTP method list is formally defined as follows:

```
ExtensionMethod ::= any token as defined by IETF RFC 2616
    (i.e., 1*[any CHAR except CTLS or separators as defined in RFC 2616])

HTTPMethod ::= "GET" | "POST" | "PUT" | "DELETE" | "HEAD" |
    "OPTIONS" | "TRACE" | ExtensionMethod

HTTPMethodList ::= HTTPMethod | HTTPMethodList comma HTTPMethod

HTTPMethodExceptionList ::= exclaimationPoint HTTPMethodList
```

Mapping Transport Guarantee to Connection Type

A `transport-guarantee` (in a user-data-constraint) of NONE, or a `security-constraint` without a `user-data-constraint`, indicates that the associated URL patterns and HTTP methods may be accessed over any (including an unprotected) transport. A `transport-guarantee` of INTEGRAL indicates that acceptable connections are those deemed by the container to be integrity protected. A `transport-guarantee` of