The Authorization Token Layer Acquisition Service Specification

This OMG document replaces the draft adopted specification (ptc/2001-10-01). It is an OMG Final Adopted Specification, which has been approved by the OMG board and technical plenaries, and is currently in the finalization phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by May 2, 2002.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*; however, at the time of this writing there were no pending issues.

The FTF Recommendation and Report for this specification will be published on July 1, 2002. If you are reading this after that date, please download the available specification from the OMG formal specifications web page.

OMG Final Adopted Specification
October 2001

ATLAS: The Authorization Token Layer Acquisition Service 27

Note – from the OMG Technical Editor: Eventually, this specification will become part of the CORBA Core document. The chapter number is temporary and may change.

Contents

This chapter contains the following topics.

Topic	Page
Section I - Overview	
“Introduction”	27-2
“Relationship to Other OMG Modules”	27-2
“Existing Specifications”	27-2
Section II - ATLAS Specification	
“Introduction”	27-3
“Specification Scope”	27-4
“Reference Model for CSIv2 Authorization Interoperability”	27-5
“Reference Model for Spanning Authorization Domains”	27-6
“The ATLAS Module”	27-7
“The Target ATLAS Interoperability Profile”	27-10
“Locating the Target’s ATLAS”	27-11
“The Target ATLAS Interoperability Specification”	27-12

Topic	Page
“Security Concerns”	27-12
“IllegalTokenRequest Error Codes”	27-13
Appendix A - “References”	27-15
Appendix B - “Conformance Points”	27-16
Appendix C - “OMG IDL”	27-17

Section I - Overview

[1] This section discusses some of the design decisions that were made to provide the components to achieve full secure interoperability between clients and targets.

27.1 Introduction

[2] This document describes the service needed to acquire authorization tokens to access a target system using the newly adopted CSIv2 protocol. This design, mandated by the RFP, defines a single interface with which to a client acquires an authorization token for a particular token. This token may be pushed, using the CSIv2 protocol in order to gain access to a CORBA invocation on the target.

[3] This specification solves the problem of acquiring the privileges needed for a client to acquire a set of privileges the target will understand as the client need not understand the token that is retrieved from the ATLAS.

27.2 Relationship to Other OMG Modules

[4] This document refers to the following modules:

- module CSI
- module CSIIOP
- module Time
- module CosNaming
- module CosNamingExt

27.3 Existing Specifications

[5] This section describes the relationships of this specification with other existing CORBA specifications.

27.3.1 Use of Existing Specifications

[6] This specification is dependent on the ORB services, data structures, and semantic definitions defined in the following specifications:

- OMG Naming Service v1.1 (OMG TC Document orbos/99-10-11) [2]
- OMG Time Service (OMG TC Document formal/2000-06-26) [3]
- OMG Common Secure Interoperability Version 2 RFP Response (OMG TC Document orbos/2000-08-04)

27.3.1.1 The Name Service

[7] Dependencies on the Name Service specification include:

- use of the Name Services **CosNaming::NamingContext** interface,
- use of the definition of **CosNaming::NamingContextExt::StringName**, and
- use of the definition of **CosNaming::NamingContextExt::URLString**.

27.3.1.2 The Time Service

[8] Dependencies on the Time Service specification are limited to the **TimeBase::UtcT** database structure.

27.3.1.3 The CSIv2 Interoperably Specification

[9] Dependencies on the CSIv2 specification include:

- use of the **CSI::IdentityToken**,
- use of the **CSI::AuthorizationToken**,
- use of the **CSIIOP::ServiceConfiguration**

structures, and their related types.

Section II - ATLAS Specification

27.4 Introduction

[10] This section describes the Authorization Token Layer Acquisition Service (ATLAS). The Authorization Token Acquisition Service is a service by which a client's security service (CSS) acquires authorization tokens to deliver to a target's security service (TSS).

[11] An authorization token consists of information that is processed by a TSS for security purposes. For example, the TSS uses the information in the authorization token to grant or deny access to the target's resources on behalf of the client.

[12] Authorization tokens must contain privilege information that is scoped to the target's understanding of privileges to be effective. Privilege information must be scoped to the target's realm of understood privileges. For example, the privilege "doctor" at target Hospital A does not necessarily have the same meaning as "doctor" at target Hospital B, if any meaning at all. Alternatively, a single client, such as "Alice," may have the "doctor" privilege at Hospital A, but not at Hospital B.

[13] Privileges are defined by privilege authorities, which define the privilege scope. The previous example illustrates that two different entities define the privilege "doctor," as well as the mappings from clients to those privileges. Hospital A subscribes to one privilege authority, Hospital B subscribes to a different privilege authority. The hospitals have different privilege scopes.

[14] The CSS needs to deliver an authorization token that is within the target's privilege scope. The definition of a privilege scope consists of, but is not limited to, the following capabilities:

- Authorizing the client with privileges defined by a privilege authority that is understood by the target.
- Authorizing the target to be endorsed with the client's privileges or identity should it be necessary for the needs of both the client and target.

[15] To facilitate secure interoperability, the TSS indicates to a client the location of the specific ATLAS that defines the target's privilege scope. The CSS retrieves from that ATLAS an authorization token, and the CSS is guaranteed that the token is understood by the TSS.

[16] Many different targets may belong to the same privilege scope and therefore they may indicate the use of the same ATLAS. One client may use many of these targets. This specification defines the data structures and semantics with which TSS's convey the location of the target's ATLAS. The approach defined in this specification facilitates client caching of the authorization tokens based on the privilege scope.

27.5 Specification Scope

[17] An ATLAS only delivers authorization tokens for one privilege scope. A service that issues authorization tokens of a variety of different token formats and different scopes is outside of this specification.

[18] This specification only addresses retrieval of authorization tokens that clients deliver to targets for security purposes. Specification of the delivery of those tokens from the CSS to the TSS is left to a transport mechanism and is outside the scope of this specification. Also, definition of the mechanism by which the TSS transmits the location of its ATLAS to the CSS is left to a transport mechanism and is also outside the scope of this specification.

[19] Administration of privileges, privilege scopes, and token formats is outside the scope of this specification.

[20] This specification facilitates the notion of client caching of authorization tokens. This specification defines the caching semantics. However, the specific caching mechanism and the conditions on which the client chooses to cache authorization tokens is outside the scope of this specification.

[21] This specification defines the method by which a client locates an ATLAS, but only does so with respect to making an interoperable request on a specific target. The client may know beforehand the various ATLAS's in which it will be dealing. In that case, it may want to locate those ATLAS's and fill its cache with frequently used authorization tokens. Locating those ATLAS's for this purpose is outside the scope of this specification. However, it might be helpful to mention to the implementers that naming or trading services may be employed to do so.

27.6 Reference Model for CSIv2 Authorization Interoperability

[22] The following model illustrates authorization interoperability with the CSIv2 authorization layer that this specification supports.

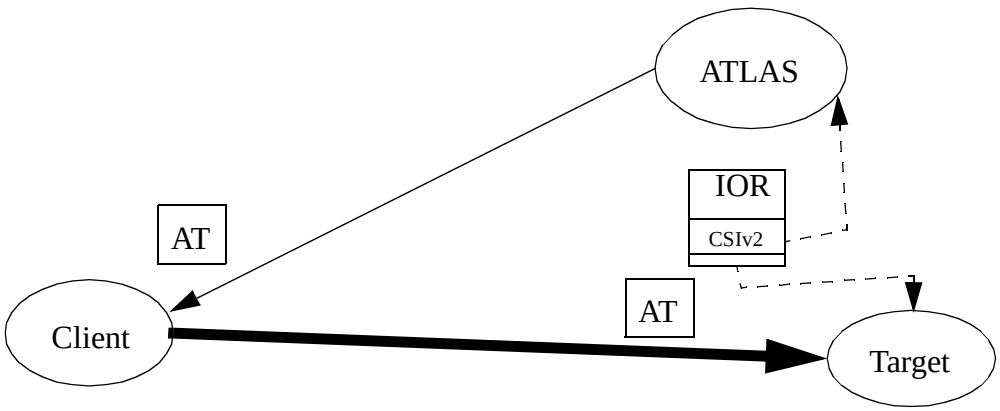

Figure 27-1

[23] In the above model, the Client has no prior agreements with the Target. The Client has no knowledge as to the format and authority of an Authorization Token (AT) that the Target will understand. The following scenario ensues:

1. The Client acquires the IOR of the Target.
2. The Client looks at the CSIv2 component in the Target's IOR and locates the ATLAS based on the information in the **CSIIOP::SAS_ContextSec:privilege_authorities** field.
3. The Client requests an Authorization Token from the Target's ATLAS based on its own authentication to the ATLAS.
4. The Client makes its intended CSIv2 protected invocation on the Target pushing the AT that was retrieved from the ATLAS in the authorization layer of the CSIv2 protocol.

5. Since the Target specified the ATLAS, the Target will understand the format and encoding of the AT that is produced by the ATLAS. The Client need not understand the format or encoding of the AT.

27.7 Reference Model for Spanning Authorization Domains

[24] The following model illustrates authorization interoperability with the CSIv2 authorization layer when Authorization Domains may be crossed.

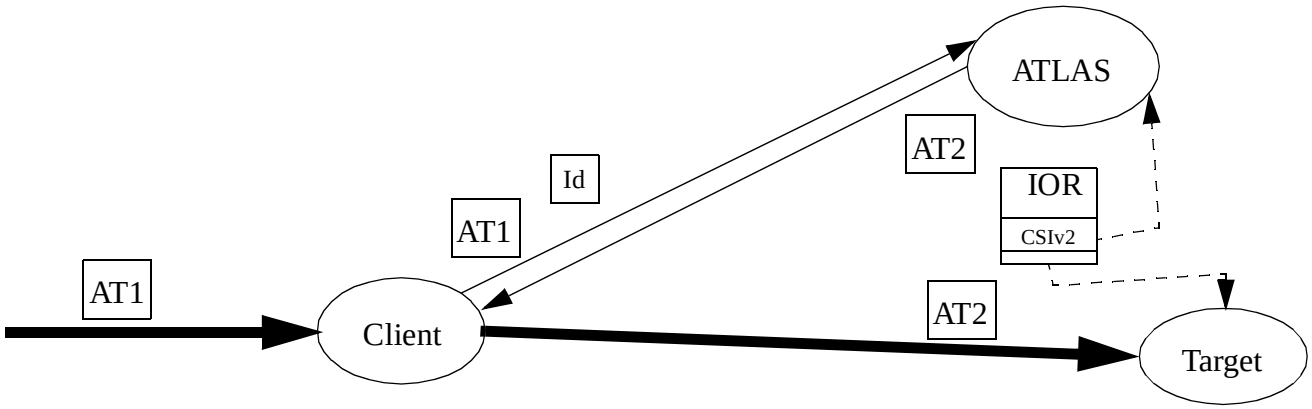

Figure 27-2

[25] In the above model, the Intermediary Client has no prior agreements with the Target. The Client has no knowledge as to the authorization domain, and the format and encoding of an Authorization Token that the Target will understand. However, the Client has an Authorization Token, AT1, from another Authorization Domain. The following scenario ensues:

1. The Client acquires the IOR of the Target.
2. The Client looks at the CSIv2 component in the Target's IOR and locates the ATLAS based on the information in the **CSIIOP::SAS_ContextSec:privilege_authorities** field.
3. The Client intends to make a request on the Target in some other principal's behalf. The Client requests AT2 from the ATLAS based on its own authentication to the ATLAS, the other principal's authorization token (AT1), and the identity token representing the other principal.
4. The Client makes its intended CSIv2 protected invocation on the Target pushing AT2 that was retrieved from the ATLAS in the authorization layer of the CSIv2 protocol and asserting the identity of the other principal.

[26] Since the Target specified the ATLAS, the Target will understand the format and encoding of the AT2 that is produced by the ATLAS. The Client need not understand the format or encoding of the AT2.

27.8 The ATLAS Module

[27] The ATLAS module contains data types, exceptions, and interfaces used by an ATLAS. Some of the important types are described here. The ATLAS module contains the following IDL:

```
// File: ATLAS.idl

#ifndef _ATLAS_IDL_
#define _ATLAS_IDL_

#include <TimeBase.idl>
#include <CosNaming.idl>
#include <CSI.idl>
#include <CSIIOP.idl>

#pragma prefix "omg.org"

module ATLAS {

...

};
#endif // _ATLAS_IDL_
```

[28] The ATLAS module depends on some data types in the **TimeBase**, **CosNaming**, **CSI**, and **CSIIOP** modules. Some important types are described in the following sections. The full IDL description of the module is defined in Section C.1, “Module ATLAS,” on page 27-17.

27.8.1 The ExpiryTime Type

[29] The **ExpiryTime** structure is a component of an AuthTokenData structure that stipulates the time that authorization token will expire, if known. The **ExpiryTime** is a sequence of at most one element of coordinated universal time. A zero element sequence indicates that the expiry time of the authorization token is not known. The **ExpiryTime** type has the following definition:

```
typedef sequence<TimeBase::UtcT,1> ExpiryTime;
```

27.8.2 The IdTokenOption Type

[30] The CSIv2 protocol requires the use of a **CSI::IdentityToken**. The intended identity shall be asserted along with the **CSI::AuthorizationToken** to give the TSS an indication of the identity to which the **CSI::AuthorizationToken** pertains. The return of a **CSI::IdentityToken** also facilitates translation of the identity token as different privilege scopes may map identities to different encodings, or even different identities.

[31] The **IdTokenOption** structure is a component of an **AuthTokenData** structure that stipulates the **CSI::IdentityToken** that the CSS shall use in conjunction with the **CSI::AuthorizationToken**. The **IdTokenOption** is a sequence of at most one element containing a **CSI::IdentityToken**. A zero element sequence indicates that the **CSI::IdentityToken** used by the CSS in its call to the ATLAS is acceptable in conjunction with the accompanying **CSI::AuthorizationToken**. This approach removes the need to return the same token back to the CSS, as the identity token can be lengthy.

```
typedef sequence<CSI::IdentityToken,1> IdTokenOption;
```

27.8.3 The AuthTokenData Type

[32] The **AuthTokenData** structure is used for a return value in some ATLAS operations. It returns the **CSI::IdentityToken**, **CSI::AuthorizationToken**, and an expiry time. The expiry time shall indicate the time the token will expire, if known. It has the following definition:

```
struct AuthTokenData {
    IdTokenOption               ident_token;
    CSI::AuthorizationToken     auth_token;
    ExpiryTime                  expiry_time;
};
```

[33] The **CSI::AuthorizationToken** type is actually a sequence of **CSI::AuthorizationElement**. Therefore, the **auth_token** field may be a sequence that contains zero elements, which means that the authorization token is empty. In this case, the CSS shall send an empty authorization token to the intended TSS.

27.8.4 The AuthTokenDispenser Interface

[34] The **AuthTokenDispenser** interface delivers **AuthTokenData** elements to the client. It has the following definition:

```
interface AuthTokenDispenser {
        // ... attributes and operations
};
```

[35] The operations of the **AuthTokenDispenser** interface are defined in the following subsections.

27.8.4.1 get_my_authorization_token

[36] A client shall use this operation to retrieve an authorization token based on the client's own identity. It has the following definition:

```
AuthTokenData get_my_authorization_token()
 raises (
    IllegalTokenRequest
```

```
);
```

Return Value

[37] The value returned by this operation shall be the data structure containing the **CSI::AuthorizationToken**, and **CSI::IdentityToken** for the client. An IllegalTokenRequest exception shall be raised in the event that the client is not granted an authorization token.

27.8.4.2 translate_authorization_token

[38] A client shall use this operation to translate an authorization token from one privilege scope to that of the scope supported by this ATLAS for the intended subject. It has the following definition:

```
AuthTokenData translate_authorization_token(
    in  CSI::IdentityToken      the_subject,
    in  AuthorizationToken      the_token
) raises (
    IllegalTokenRequest,
    TokenOkay
);
```

[39] The client may use this operation to "request" privileges within the target scope by creating an authorization token and having it translated.

Parameters

the_subject	This parameter specifies the identity for which the token is being translated. The **CSI::IdentityToken** type is a discriminated union that accommodates different name forms. A client shall not use a **CSI::IdentityToken** with a discriminator of **CSI::ITTAbsent**.
the_token	This parameter contains the token to be translated.

Return Value

[40] The value returned shall be the structure containing the **CSI::AuthorizationToken** and the **CSI::IdentityToken** that has been translated to the target's privilege scope.

[41] An IllegalTokenRequest exception shall be raised in the event that the client is not granted an authorization token, or if the given authorization token is not translatable by this ATLAS.

[42] The TokenOkay exception shall be raised in the event that the token is understood and is deliverable to the target. In other words, the token did not need to be translated by this ATLAS.

27.9 The Target ATLAS Interoperability Profile

[43] The target shall indicate the specific ATLAS from which the CSS gets authorization tokens to deliver to the TSS. Once a CSS gets this profile, it shall locate the target's ATLAS. Locating an ATLAS is defined as retrieving an object reference to an **AuthTokenDispenser** interface.

The **ATLASProfile** has the following definition:

```
struct ATLASProfile {
    ATLASCacheId        the_cache_id;
    ATLASLocator        the_locator;
);
```

[44] The field, **the_cache_id**, is a byte sequence on which the client may cache authorization tokens associated with the located ATLAS. The field, **the_locator**, shall contain a locator that leads to the object reference of the **AuthTokenDispenser** interface.

[45] The **ATLASCacheId** is defined as a byte sequence. The caching identifier is said to be present if it is a non-empty byte sequence. The caching identifier is said not to be present if it is an empty byte sequence.

[46] If the caching identifier is present, and the CSS caches authorization tokens, the CSS shall use the caching identifier and shall ignore the locator for the caching of authorization tokens. The locator shall not enter into the CSS caching scheme because the locator is insufficient to determine whether two ATLAS's are the same. For, example, there may be many different servers for one ATLAS, which results in many different object references and locator specifications. The caching identifier matching algorithm is byte sequence equality.

[47] If the caching identifier is not present, the target considers the locator sufficient for caching purposes. In this case, the default matching algorithm used by the CSS is byte sequence equality on the locator. The CSS can use better matching algorithms based on the its understanding of the locator and its resolution.

[48] The target shall make the caching identity unique enough to facilitate correct client caching of authorization tokens amongst its clients. One approach would be to create a Universal Unique Identifier (UUID) [4]. For multiple targets that use the sameATLAS, it is advisable to allocate a common identifier for that ATLAS.

[49] Targets using different privilege scopes shall have different locators, and if caching identities are supplied, they shall be different as well.

[50] Specification of caching identifiers and procedures for allocation of caching identifiers is outside the scope of this specification.

27.10 Locating the Target's ATLAS

[51] The **ATLASLocator** shall be, or shall lead to, the object reference of an ATLAS **AuthTokenDispenser** interface. Using an object reference that directly points to the ATLAS may not be desirable in all cases. In some cases, a level of indirection, such as using the CORBA NameService, may be useful. The **ATLASLocator** combines all these methods by using a discriminated union.

```
struct CosNamingLocator {
    CosNaming::NamingContext   name_service;
    CosNaming::Name            the_name;
};

typedef CosNaming::NamingContextExt::URLString URLocator;

typedef unsigned long ATLASLocatorType;

const  ATLASLocatorType   ATLASCosNaming   =  1;
const  ATLASLocatorType   ATLASURL         =  2;
const  ATLASLocatorType   ATLASObject      =  3;

union ATLASLocator switch (ATLASLocatorType) {
    case ATLASCosNaming:   CosNamingLocator    naming_locator;
    case ATLASURL:         URLocator           the_url;
    case ATLASObject:      AuthTokenDispenser  the_dispenser;
};
```

[52] The **ATLASCosNaming** branch of the union is a CORBA NameService specification in which the object reference of the naming context is supplied along with the name of the **AuthTokenDispenser**. The object reference that is resolved at the end of this name path shall resolve to a target of the **AuthTokenDispenser** type.

[53] The **ATLASURL** branch indicates a Universal Resource Locator (URL), which is specified by the Extended Interoperable Naming Service Specification [2].

[54] The **ATLASObject** branch indicates that an object reference points to a target of the AuthTokenDispenser type directly.

[55] Given an **ATLASLocator** that is a URL, the client shall locate the ATLAS by resolving successive locates until it gets an object reference, because the content of the URL may be another URL. This procedure shall be followed until the URL resolution results in an invalid URL or an object reference. The object reference that results shall be that of an **AuthTokenDispenser**.

Warning – To alleviate a denial of service attack on the client directly, a client would place a limit on the number of URLs it will resolve in a chain of resolutions as well as check for loops.

27.11 The Target ATLAS Interoperability Specification

[56] The ATLAS Interoperability Specification is contained in the **privilege_authorities** field of the **CSIIOP::SAS_ContextSec** structure. Its type is that of **CSIIOP::ServiceConfiguration**, and its definition is listed below:

```
typedef short ServiceConfigurationSyntax;

typedef sequence<octet>      ServiceSpecificName;

struct ServiceConfiguration {
    ServiceConfigurationSyntax      syntax;
    ServiceSpecificName             name;
};
```

[57] The **syntax** field of the structure stipulates the encoding of the **name** field.

[58] The **ServiceConfiguration** for an ATLAS is as follows:

[59] The “syntax” field of the **ServiceConfiguration** structure shall contain the value of the following constant.

```
const CSIIOP::ServiceConfigurationSyntax SCS_ATLAS = 3;
```

[60] The **name** field of the **ServiceConfiguration** structure shall contain the CDR encapsulation of the **ATLAS::ATLASProfile** structure.

27.12 Security Concerns

[61] This section describes the security concerns one should have in both implementing, deploying, configuring, and using the ATLAS.

[62] The ATLAS is intended to implemented with CORBA Security. Its implementations shall be security aware to support some of the operations. One such operation is the **AuthTokenDispenser::get_my_authorization_token** operation. This operation shall determine its client’s identity to return the correct authorization token for the client.

[63] The ATLAS is a potentially sensitive service. All of its operations shall be protected by CORBA Security services and have an access control policy based on its clients’ identities. The reason the ATLAS is a sensitive service is discussed in the following subsections.

27.12.1 Confidentiality and Privacy

[64] Authorization information may be a privacy concern to some individuals and organizations. For example, a nemesis discovers an individual to have privileges that allow that individual access to sensitive data. That discovery may warrant an attack on that individual to gain access to the sensitive data. Therefore, the ATLAS shall take

care and perform access control when dispensing authorization tokens. Also, where privilege information is a privacy concern in suspicious networks, the ATLAS should mandate the use of confidential security services.

[65] The use of the ATLAS must also be considered for privacy concerns. A rogue target may know the ATLAS locator or its caching identifier. It can then spoof the CSS into giving up a cached authorization token. The CSS shall trust the target before sending authorization tokens. A rogue target may also collect authorization tokens by specifying an ATLAS for the purpose of getting the CSS to perform token translation. Therefore, the CSS shall trust the target and its located ATLAS before sending authorization tokens to the ATLAS for translation.

27.12.2 Integrity

[66] To stop spoofing of targets by clients, the TSS shall verify that the authorization tokens delivered by the client are from the target's ATLAS. Also, the TSS shall make some trust determination that the tokens are valid for the particular client that delivered them.

27.12.3 Availability

[67] Beyond the normal problems of dealing with denial of service attacks, there is one important concern that a CSS must take into account when trying to locate a target's ATLAS. A rogue target may put a URL loop or an exceedingly long URL resolution chain in its ATLAS locator. A CSS that is not careful may spin indefinitely trying to locate the ATLAS; and therefore, it may not do anything else. A CSS should impose a limit on chasing chains of URLs, look for loops, and possibly make a trust determination on URLs.

[68] Another great concern for the CSS is a recursive need for authorization tokens to access an ATLAS. In the reference for interoperability the ATLAS is effectively just another target, which is protected by CORBA security services. An ATLAS's TSS may require privileges from another privilege scope. ATLAS's that require authorization tokens shall not specify themselves, or one from some mutually recursive set, as the ATLAS.

27.13 IllegalTokenRequest Error Codes

[69] This section describes the error codes returned in the **IllegalTokenRequest** code that are returned from implementations. The exception has the following format:

```
exception IllegalTokenRequest {
    unsigned long     the_errnum;
    string            the_reason;
};
```

[70] The field, **the_errnum**, contains an error code, of which the values are defined in Table 27-1. An implementation shall use the standard error codes below where possible, and indicate minor errors with the reason field. For error codes that are not standard, the code should be placed in the least significant 16 bits of the unsigned long with the following restrictions.

[71] An error code between 300 and 400 indicates a server error, of which 300 as the default for server errors. The range 200 to 300 hundred is for errors pertaining to the client's invocation on the ATLAS. Codes between 100 and 200 are for other errors. Specific vendors may use their VMCIDs to indicate their own errors.

Table 27-1 IllegalTokenRequest Error Codes

Error Code	Meaning
0100	Generic error
0200	Generic client error
0201	Authorization token is malformed
0202	Identity token is malformed
0300	Generic server error
0301	Authorization token is not granted
0302	Authorization token type is not supported for translation
0303	Authorization token translation failed.

Appendix A References

[1] The Object Management Group, *Common Secure Interoperability Version 2 Joint Revised Submission*, http://cgi.omg.org/cgi-bin/doc?orbos/00-08-04, 2000

[2] The Object Management Group, *CORBA Name Service v1.1*, http://cgi.omg.org/cgi-bin/doc?orbos/99-10-11.pdf, 1999

[3] The Object Management Group, *CORBA Time Service*, http://cgi.omg.org/cgi-bin/doc?formal/97-02-22, 1997

[4] The Open Group, *Universal Unique Identifier*, http://wwww.opengroup.org/onlinepubs/9629399/adxa.htm, 1997

[5] Yergeau F., *UTF-8, a transformational format of Unicode and ISO 10646*, RFC 2044, Alis Technologies, October 1996

Appendix B Conformance Points

B.1 Conformance of ATLAS Implementations

[72] This Appendix describes the terms of conformance for implementations of ATLAS. All implementations shall implement all operations of the **AuthTokenDispenser** interface.

[73] Implementations may not support the notion of translating authorization tokens. However, they shall still raise an IllegalTokenRequest exception with the appropriate error code defined in Section 27.13, “IllegalTokenRequest Error Codes,” on page 27-13 for the “**translate_authorization_token**” operation of the **AuthTokenDispenser** interface.

B.2 Conformance of Standard CORBA Security Implementations

[74] CORBA Security implementations that support pushing authorization tokens shall support CSS functionality defined in Section 27.6, “Reference Model for CSIv2 Authorization Interoperability,” on page 27-5” when the ATLAS profile is contained as a privilege authority within the CSS selected security mechanism component in the target IOR. CORBA Security implementations that support acceptance of authorization tokens and indicate their privilege authorities in security mechanism components of their IOR shall have implementation support to indicate an ATLAS profile as a privilege authority.

Appendix C - OMG IDL

C.1 Module ATLAS

```
// File: ATLAS.idl

#ifndef _ATLAS_IDL_
#define _ATLAS_IDL_

#include <TimeBase.idl>
#include <CosNaming.idl>
#include <CSI.idl>
#include <CSIIOP.idl>

#pragma prefix "omg.org"

module ATLAS {

typedef sequence<TimeBase::UtcT,1> ExpiryTime;

typedef sequence<CSI::IdentityToken,1> IdTokenOption;

struct AuthTokenData {
    IdTokenOption               ident_token;
    CSI::AuthorizationToken     auth_token;
    ExpiryTime                  expiry_time;
};

exception IllegalTokenRequest {
    unsigned long       the_errnum;
    string              the_reason;
};

exception TokenOkay {};

interface AuthTokenDispenser {

    AuthTokenData     get_my_authorization_token()
      raises (
        IllegalTokenRequest
    );

    AuthTokenData     translate_authorization_token(
        in   CSI::IdentityToken          the_subject,
        in   CSI::AuthorizationToken     the_token
    ) raises (
        IllegalTokenRequest,
        TokenOkay
    );
```

```
};

struct CosNamingLocator {
    CosNaming::NamingContext   name_service;
    CosNaming::Name            the_name;
};

//
// This type specifies a string encoded in UTF-8 form [IETF RFC 2044].
//
typedef sequence<octet>  UTF8String;

typedef CosNaming::NamingContextExt::URLString URLocator;

typedef unsigned long          ATLASLocatorType;

const   ATLASLocatorType    ATLASCosNaming    =   1;
const   ATLASLocatorType    ATLASURL          =   2;
const   ATLASLocatorType    ATLASObject       =   3;

union ATLASLocator switch (ATLASLocatorType) {
    case ATLASCosNaming:   CosNamingLocator    naming_locator;
    case ATLASURL:         URLocator           the_url;
    case ATLASObject:      AuthTokenDispenser  the_depenser;
};

typedef sequence<octet>        ATLASCacheId;

struct ATLASProfile {
    ATLASLocator          the_locator;
    ATLASCacheId          the_cache_id;
};

const CSIIOP::ServiceConfigurationSyntax SCS_ATLAS = 3;

};

#endif // _ATLAS_IDL_
```