XOM Gateway Control Protocol (XGCP) Specification

XOM Gateway Control Protocol (XGCP) Specification

Version 1.1 Edition 7.20141001
Updated October 25, 2014
Distributed with Package openss7-1.1.7.20141001

Abstract:

This document is a Specification containing technical details concerning the implementation of the XOM Gateway Control Protocol (XGCP) for OpenSS7. It contains recommendations on software architecture as well as platform and system applicability of the XOM Gateway Control Protocol (XGCP). It provides abstraction of the Gateway Control Protocol (GCP) interface to these components as well as providing a basis for Gateway Control Protocol control for other Gateway Control Protocol protocols.

Brian Bidulock <bidulock@openss7.org> **for**
The OpenSS7 Project <http://www.openss7.org/>

Published by:

OpenSS7 Corporation
1469 Jefferys Crescent
Edmonton, Alberta T6L 6T1
Canada

Notice:

Short Contents

Table of Contents

List of Figures

List of Tables

Preface

Notice

Abstract

This document is a Specification containing technical details concerning the implementation of the XOM Gateway Control Protocol (XGCP) for OpenSS7. It contains recommendations on software architecture as well as platform and system applicability of the XOM Gateway Control Protocol (XGCP).
This document specifies a XOM Gateway Control Protocol (XGCP) Specification in support of the OpenSS7 Gateway Control Protocol (GCP) protocol stacks. It provides abstraction of the Gateway Control Protocol interface to these components as well as providing a basis for Gateway Control Protocol control for other Gateway Control Protocol protocols.

Purpose

The purpose of this document is to provide technical documentation of the XOM Gateway Control Protocol (XGCP). This document is intended to be included with the OpenSS7 STREAMS software package released by *OpenSS7 Corporation*. It is intended to assist software developers, maintainers and users of the XOM Gateway Control Protocol (XGCP) with understanding the software architecture and technical interfaces that are made available in the software package.

Intent

It is the intent of this document that it act as the primary source of information concerning the XOM Gateway Control Protocol (XGCP). This document is intended to provide information for writers of OpenSS7 XOM Gateway Control Protocol (XGCP) applications as well as writers of OpenSS7 XOM Gateway Control Protocol (XGCP) Users.

Audience

The audience for this document is software developers, maintainers and users and integrators of the XOM Gateway Control Protocol (XGCP). The target audience is developers and users of the OpenSS7 SS7 stack.

Revision History

Take care that you are working with a current version of this documentation: you will not be notified of updates. To ensure that you are working with a current version, check the OpenSS7 Project website for a current version.
A current version of this specification is normally distributed with the *OpenSS7* package, `openss7-1.1.7.20141001`.[1]

[1] `http://www.openss7.org/repos/tarballs/openss7-1.1.7.20141001.tar.bz2`

Version Control

Although the author has attempted to ensure that the information in this document is complete and correct, neither the Author nor OpenSS7 Corporation will take any responsibility in it. *OpenSS7 Corporation* is making this documentation available as a reference point for the industry. While *OpenSS7 Corporation* believes that these interfaces are well defined in this release of the document, minor changes may be made prior to products conforming to the interfaces being made available. *OpenSS7 Corporation* reserves the right to revise this software and documentation for any reason, including but not limited to, conformity with standards promulgated by various agencies, utilization of advances in the state of the technical arts, or the reflection of changes in the design of any techniques, or procedures embodied, described, or referred to herein. *OpenSS7 Corporation* is under no obligation to provide any feature listed herein.

```
$Log: xgcp.texi,v $
Revision 1.1.2.2  2011-07-27 07:52:18  brian
- work to support Mageia/Mandriva compressed kernel modules and URPMI repo

Revision 1.1.2.1  2011-07-18 19:42:12  brian
- added documentation

Revision 1.1.2.2  2011-02-07 02:21:48  brian
- updated manuals

Revision 1.1.2.1  2009-06-21 10:58:46  brian
- added files to new distro
```

ISO 9000 Compliance

Only the TEX, texinfo, or roff source for this maual is controlled. An opaque (printed, postscript or portable document format) version of this manual is a **UNCONTROLLED VERSION**.

Disclaimer

OpenSS7 Corporation **disclaims all warranties with regard to this documentation including all implied warranties of merchantability, fitness for a particular purpose, non-infrincement, or title; that the contents of the manual are suitable for any purpose, or that the implementation of such contents will not infringe on any third party patents, copyrights, trademarks or other rights. In no event shall** *OpenSS7 Corporation* **be liable for any direct, indirect, special or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action or contract, negligence or other tortious action, arising out of or in connection with any use of this documentation or the performance or implementation of the contents thereof.**

U.S. Government Restricted Rights

If you are licensing this Software on behalf of the U.S. Government ("Government"), the following provisions apply to you. If the Software is supplied by the Department of Defense ("DoD"), it is classified as "Commercial Computer Software" under paragraph 252.227-7014 of the DoD Supplement to the Federal Aquisition Regulations ("DFARS") (or any successor regulations) and the Government is acquiring only the license rights granded herein (the license rights customarily provided to non-Government users). If the Software is supplied to any unit or agency of the Government other than DoD, it is classified as "Restricted Computer Software" and the Government's rights in the Software are defined in paragraph 52.227-19 of the Federal Acquisition Regulations ("FAR") (or any

successor regulations) or, in the cases of NASA, in paragraph 18.52.227-86 of the NASA Supplerment to the FAR (or any successor regulations).

Acknowledgements

The OpenSS7 Project was funded in part by:

- Monavacon Limited
- OpenSS7 Corporation

Thanks to the subscribers to and sponsors of The OpenSS7 Project. Without their support, open software like this would not be possible.

As with most open source projects, this project would not have been possible without the valiant efforts and productive software of the Free Software Foundation, the Linux Kernel Community, and the open source software movement at large.

1 Introduction

1.1 Overview

The XOM Gateway Control Protocol Programming Interface (abbreviated XGCP) defines an Application Program Interface (API) to gateway control services. It is referred to as *the interface* throughout this specification.

The interface is designed to offer services that are consistent with, but not limited to, the Telcordia, Level 3 and Cisco RFC 2705 MGCP Version 1.0, the IETF MEGACO Version 2.0, the ITU-T Recommendation H.248 GCP Version 2.0, and the ITU-T Recommendation H.248 GCP Version 3.0 standards. These standards have been published starting in 1999 and have been stable for many years. The ITU-T Recommendation H.248 GCP Version 3.0 was last updated in 2005.

The interface is also designed to offer services that are consistent with various packages provided by MGCP, MEGACO and H.248.

All of the above standards are referred to in this document as *the Standards*.

Access to other gateway control services through the API is not prohibited, but has not been explicitly considered.

The interface is designed to be used and implemented in conjunction with the use and implementation of the general-purpose XOM API (reference **XOM**).

A brief introduction to Gateway Control Services is given in Section 1.3 [Introductory Concepts], page 9. Following this is an overview of the OSI-Abstract-DATA Manipulation OM, which provides the Data Abstraction service as defined in the XOM specification (reference **XOM**). Then the optional features of this specification are described, and the chapter closes with a list of abbreviations. In all cases the reader should refer to the Standards (reference **MGCP**, reference **MEGACO**, reference **H.248**), or to the XOM Specifications (reference **XOM**) for further authoritative details.

The structure of the remaining chapters and appendices are described in the [Preface], page 5.

1.2 Format of the Specification

This specification described a programming language-independent interface to the Gateway Control Services together with a specific 'C' language binding of that interface. Several conventions are used to identify particular items. The general conventions are described in the [Preface], page 5, while the 'C' language binding conventions are described in Chapter 2 [C Language Binding], page 13.

1.3 Introductory Concepts

1.3.1 Relationship to GCP Protocols

The interaction between gateway control programs acting in a gateway control entity role are realized through the exchange of gateway control service information. The general communications service for gateway control is the *Gateway Control Protocol*. GCP defines the following operations:

Service	Type

This communication may be accomplished using the MGCP, MEGACO or H.248 protocol.

1.3.2 XGCP and the GCP Provider

The XGCP interface provides access to the GCP service provider, which offers all of the facilities defined in the Standards. It also provides facilities such as *Automatic Association Management*,

Automatic Message Handling and *Automatic Dialog Handling*. The interface is designed not to restrict the services offered to those of specific service packages of the H.248 protocol or any given profile of packages.

The interface defined in this specification is "symmetrical" in the sense that it can be used to implement gateway control programs acting in any of the GCP producer or consumer roles (e.g. MGC, MG, CA, S-SBG, D-SBG). The interface supports:

- a gateway control program acting as a consumer of gateway control services. This is done by submitting service operation requests and receiving service operation responses.
- a gateway control program acting as a producer of gateway control services. This is done be receiving service operation requests and sending back service operation responses.

The interface provides the ability to send *requests* on the consumer side and to receive *indications* on the producer side within a gateway control service interaction. Furthermore, if the service is confirmed, the producer will be able to send back *responses* that will be received as *confirmations* by the consumer.

1.4 Relationship to MGCP and MEGACO/H.248 GCP

The API is essentially based on the abstract services of MGCP and MEGACO/H.248, bit is independent of the underlying communications stack. The API allows the manipulation of ITU-T and pre-standard gateway control protocol service information. Thus this API does not preclude and does not force the use of either the MGCP pre-standard protocol or the MEGACO/H.248 standard protocol (of any version).

The XGCP API offers three abstract gateway control service views: that of MGCP Version 1.0, MEGACO Version 2.0, and H.248 GCP Version 3.0.

The contents of the MGCP messages are described in RFC 2705. These messages implicitly define MGCP GCP services. The mapping between MGCP GCP services and various service primitives and parameters of the XGCP API are described below.

The services offered by the XGCP API are a superset of those defined by MGCP, MEGACO and H.248 GCP. The general communications protocol for each is the User Datagram Protocol (UDP) or the Stream Control Transmission Protocol (SCTP) defined by the IETF.

The three abstract gateway control view of XGCP (MGCP, MEGACO and H.248) are independent of the underlying protocol.

1.5 Relationship to Data Abstraction Services

XGCP is dependent on standard data abstraction services to ensure portability of application software written to the XGCP specification. XGCP functions pass most arguments by reference. The data referenced by these arguments is modelled and manipulated in an object-oriented fashion. gateway control data abstraction services are provided by the XOM API (reference **XOM**).

The definitions below introduce various concepts that are used by Gateway Control data abstraction service.

Syntax — A *syntax* is the classification and representation of values in OSI-Abstract-Data Manipulation. Examples of syntaxes are *Boolean*, *Integer*, *Real*, *String(Octet)*, *String(Object-Identifier)* and *Object*.

Value — A *value* is a single datum, or piece of information. A value may be as simple as a Boolean value (for example, *True*), or as complicated as an entire OM object (for example, a *Message*).

OM Attribute
: An *OM attribute type* is an arbitrary category into which a specification places some values. An *OM attribute* is an OM Attribute Type, together with an ordered sequence of one or more values. The OM Attribute Type can be thought of as the name of the OM attribute.

OM Object
: An *OM object* is a collection of OM attributes.

OM Class
: An *OM class* is a category of OM objects set out in a specification. It determines the OM attributes that may be present in the OM object, and details the constraints on those OM attributes.

Package
: A *Package* is a set of OM classes that are grouped together by the specification, because tey are functionally related (for example, GSM service package).

Package Closure
: A *Package-Closure* is the set of classes that need to be supported to be able to create all possible instances of all classes defined in the package. Thus an OM Class may be defined to have an OM Attribute whose value is an OM Obejct of an OM Class that is defined in some other package, but within the same *Package-Closure*.

Workspace
: A *workspace* is allocated storage that contains one or more *Package-Closures*, together with an implementation of the Gateway Control data abstraction services, that supports all the OM classes of OM objects in the *Package-Closures*.

Descriptor
: A *descriptor* is a defined data structure that is used to represent an OM Attribute Type and a single value. The structure has three components: a type, a syntax and a value.

Public Object
: *Public Objects* are represented by data structures that are manipulated directly using programming language constructs. Use of Public Objects therefore simplifies programming by this direct access and by enabling objects to be statically defined, when appropriate. Programs can efficiently access public objects.

Private Objects
: *Private Objects* are held in data structures that are private to the service and can only be accessed from programs indirectly using interface functions. They are of particular use for structures that are infrequently manipulated by programs, being passed by reference to the service, which can then manipulate them efficiently. An example of such objects in XGCP is the *session* object.

1.6 Mandatory and Optional Features

The interface defines an Application Program Interface (API) that application programs can use to access the functionality of the underlying Gateway Control Services. The interface does not define or imply any profile of that service.

Note that nothing in this specification requires that the implementation of the interface or the Gateway Control Services itself actually makes use of UDP or SCTP or other parts of the model, just so long as it provides the defined service. Also, the *scope* of the Gateway Control Services to which an application has access is not determined; it is not restricted to H.248 GCP operations.

Some OM attributes are optional: these are marked *(Optional Functionality)* in the OM class definitions. They are:

- **File-Descriptor** in a **Session** object.

Some items of behaviour of the interface and a number of aspects of the Gateway Control Services provider are implementation-defined. These are:

- the maximum number of outstanding asynchronous operations
- whether an asynchronous function call returns before the operation is submitted to the Gateway Control Services provider
- the text and language of error messages
- the OM classes permitted as values of the **Address** and **Title** argument to interface functions.

The default values of some OM attributes on OM object **Session** are locally administered.

This API assumes the provision of *Automatic Association Management*, *Automatic Message Handling* and *Automatic Dialog Handling* by the GCP provider.

The interface enables negotiation of the use of the various defined features of the GCP provider and those of the interface.

1.6.1 Packages

The specification defines several Gateway Control packages (Common GCP package, MGCP package, MEGACO package, and H.248 package), Chapter 4 [Interface Packages], page 31. These packages define the OM classes required by the interface functions to perform GCP services. The common GCP package, which also includes the errors defined (see Chapter 7 [Errors], page 129), is mandatory. The MGCP, MEGACO and H.248 packages are optional, but at least one of them must be supported by the implementation. The different service views assume the support of the corresponding GCP package by the implementation.

The use of the optional packages is negotiated using the `Negotiate()` function.

1.6.2 Terminology

The terms *implementation-defined*, *may*, *should*, *undefined*, *unspecified*, and *will* are used in this document with the meanings ascribed to them in reference **XPG4**, see also [Glossary], page 135.

1.6.3 Abbreviations

API	Application Program Interface
ASN.1	Abstract Syntax Notation One
ANSI	American National Standards Institute
BER	Basic Encoding Rules
GSM	Global Services Mobile
ISO	International Organisation for Standardisation
ITU-T	International Telecommunications Union - Telecom Sector
MAP	Mobile Application Part
OM	OSI-Abstract-Data Manipulation
OSI	Open Systems Interconnect
ROSE	Remote Operations Service Element
TCAP	Transaction Capabilities Application Part
XMAP	XOM Mobile Application Part API
XOM	X/Open: OSI-Abstract-Data Manipulation API

2 C Language Binding

This chapter sets out certain characteristics of the C language binding to the interface. The binding specifies C identifiers for all the elements of the interface, so that application programs written in C can access the Gateway Control Services. These elements include function names, *typedef* names and constants. All of the C identifiers are mechanically derived from the language independent names as explained below. There is a complete list of all the identifiers in Appendix A [C Headers], page 131. For ease of use, some of these identifiers are defined in the specification alongside the language-independent name.

A *Function()* is indicated as shown.

A `CONSTANT` is in Roman font.

The names of `[ERRORS]` and other return codes are surrounded by square brackets.

The definitions of the C identifiers appear in four headers:

<xom.h> This header file contains definitions for the associated OM interface.

<xgcp.h> This header file contains common definitions for the access to the Gateway Control Protocol service (see Chapter 5 [Interface Functions], page 37, and Section 6.2 [Common GCP Package], page 88).

<xmap_mgcp.h>
This header file contains specific definitions that reflect the Abstract Services of the MGCP Gateway Control Services along with the ASN.1 productions of the related protocol (MGCP), See Section 6.3 [MGCP Package], page 110.

<xmap_megaco.h>
This header file contains specific definitions that reflect the Abstract Services of the MEGACO Gateway Control Services along with the ASN.1 productions of the related protocol (MEGACO), See Section 6.5 [MEGACO Package], page 111.

<xmap_h248.h>
This header file contains specific definitions that reflect the Abstract Services of the H.248 Gateway Control Services along with the ASN.1 productions of the related protocol (H.248), See Section 6.7 [H.248 Package], page 117.

<xmap_gsm_sm.h>
This header file contains specific definitions that reflect the Short Message GSM services along with ASN.1 productions of the related services (GSM MAP Short Message services), See ⟨undefined⟩ [⟨undefined⟩], page ⟨undefined⟩.

2.1 C Naming Conventions

The interfaces uses part of the 'C' public namespace for its facilities. All identifiers start with the letters gcp, GCP or OGCP, and more detail of the conventions used are given in the following table. Note that the interface reserves *all* identifiers starting with the letters *gcpP* for private (i.e. internal) use by implementations of the interface. It also reserves *all* identifiers starting with the letters *gcpX* or *GCPX* for vendor-specific extensions of the interface. Application programmers should not use any identifier starting with these letters.

The OSI-Abstract-Data Manipulation API uses similar, though not identical, naming conventions, that are described in XOM (reference **XOM**). All its identifiers are prefixes by the letters *OM* or *om*.

reserved for implementors *gcpP*

reserved for interface extensions	*gcpX*
reserved for interface extensions	*GCPX*
reserved for implementors	*OGCP*

functions	*gcp_*
error problem values	*GCP_E_*
enumeration tags (except errors)	*GCP_T_*
OM class names	*GCP_C_*
OM value length limits	*GCP_VL_*
OM value number limits	*GCP_VN_*
other constants	*GCP_*

A complete list of all identifiers used (except those beginning *gcpP*, *gcpX*, *GCPX* or *OGCP*) is given in Appendix A [C Headers], page 131. No implementation of the interface will use any other public identifiers. A *public identifier* is any name except those reserved in section 4.1.2.1 of the ISO C Standard, and the *public namespace* is the set of all possible public identifiers.

The C identifiers are derived from the language-independent names used throughput this specification by a purely mechanical process which depends on the kind of name:

- Interface function names are made entirely lower-case and prefixed by *gcp_*. Thus **Service-Req()** becomes `gcp_service_req()`.
- C function parameters are derived from the argument and result names by making them entirely lower-case. In addition, the names of results have *_return* added as a suffix. Thus the argument **Session** becomes *session*, while the result of **Result** becomes *result_return*.
- OM class names are made entirely upper-case and prefixed by *GCP_C_*. Thus **Service-Argument** becomes *GCP_C_SERVICE_ARGUMENT*. Note that the symbolic OM class names are strictly those used in the abstract syntax ASN.1 of the TCAP and GCP with the exception that names containing multiple words are separated by hyphens.
- Enumeration tags are derived from the name of the corresponding OM type and syntax by prefixing *GCP_*. The case of letters is left unchanged. Thus **Enum(User-reason)** befomes *GCP_User_reason*.
- Enumeration constants, except erros, are made entirely upper-case and prefixed by *GCP_T_*. Thus **resource-limitation** becomes *GCP_T_RESOURCE_LIMITATION*.
- The name of an OM attribute is local to its OM class, that measn the same name of an OM attribute may appear in different OM classes, for example, OM attribute **application-Context** is defined in both OM classes **Open-Arg** and **application-Context-List**. Independent-language atribute **application-Context** appears as *GCP_APPLICATION_CONTEXT* in C-language. Note that the symbolic OM attribute names are strictly those used in the abstract syntax ASN.1 of the TCAP and GCP with the exception that names containing multiple words are separated with hyphens.
- Errors are treated as a special case. Constants that are the possible values of the OM attribute **Error-Status** of a subclass of the OM class **Error** are made entirely upper-case and prefixed by *GCP_E_*. Thus **invalid-session** becomes *GCP_E_INVALID_SESSION*.
- The constants in the **Value Length** and **Value Number** columns of the OM class definition tables are also assigned identifiers. (They have no names in the language-independent specification.) Where the upper limit in on eof these columns is not “1” (one), it is given a name consisting of the OM attribute name, prefixed by *GCP_VL_* for value length, or *GCP_VN_* for value numbers.

- The sequence of octets for each object identifier is also assigned an identifier, for internal use by certain OM macros. These identifiers are all upper case and are prefixed by *OMP_O_*. See reference **XOM** for further details on the use of object identifiers.

  Note that hyphens are translated everywhere to underscores.

2.2 Use and Implementation of Interfaces

Each of the following statements applies unless explicitly state otherwise in the detailed descriptions that follow:

If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behaviour is *undefined*.

Any function declared in a header may be implemented as a macro defined in the header, so a library function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally be encoding the name of the function in parentheses, because the name is not then followed by the left parentheses that indicate expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a library function even if it is also defined as a macro. The use of `#undef` to remove any macro definition will also ensure that an actual function is referred to. Any invocation of a library function that is implemented as a macro will expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression anywhere a function with a compatible return type could be called.

2.3 Function Return Values

The return value of a C function is always bound to the result of the language-independent description. Functions return a value of `GCP_status`, which is an error indication. If and only if the function succeeds, its value will be **success**, expressed in C by the constant *GCP_SUCCESS*. If a function returns a status other than this, then it has not updated the return parameters. The value of the status, in this case, is an error as described in Chapter 7 [Errors], page 129. In most cases the integer returned in **Status** is sufficient for error processing. However, in a few cases additional information is available if desired.

Since C does not provide multiple return values, functions must return all other results by writing into storage passed by the application program. Any argument that is a pointer to such storage has a name ending with *_return*. For example, the C parameter declaration '`OM_sint *invoke_id_return`' in the *Service-Req()* function indicates that the function will return an signed integer **Invoke-Id** as a result, so the actual argument to the function must be the address of a suitable variable. This notation allows the reader to distinguish between an input parameter that happes to be a pointer, and an output parameter where the `*` is used to simulate the semantics of passing by reference.

2.4 Compilation and Linking

All applications programs that use this interface include the `<xom.h>` and `<xgcp.h>` headers in that order, and at least one of the `<xgcp_mgcp.h>`, `<xgcp_megaco.h>` and `<xgcp_h248.h>` headers.

3 Description

The interface comprises a number of functions together with many OM classes and OM objects that are used as the arguments and results of the functions. Both the functions and the OM objects are based closely on the *Abstract Service* that is specified in the Standards (references **MGCP**, **MEGACO** and **H.248**).

The interface models gateway control interactions as service requests made through a number of interface *functions*, that take a number of input *arguments*. Each valid request causes an *operation* within the producer that eventually returns a *status* and any *result* of the operation.

All interactions between a Consumer and a Producer belong to a *session*, that is represented by an OM object passed as the first argument to most interface functions.

The other arguments to the function include a *context* and various service-specific arguments. The *context* includes a number of parameters that are common to many functions and that seldom change from operation to operation.

Each of the components of this model is described below, along with other features of the interface such as asynchronous function calls and security.

3.1 Services

As mentioned above, the Standards define Abstract Services that Consumers and Producers use. Each of these Abstract Services maps to a single function call with the same name. The services are **Service-req** and **Service-rsp**.

There are three functions called *Receive()*, *Wait()*, and *Abandon()* that have no counterpart in the Abstract Service. *Receive()* is used to receive indications and results of asynchronous operations, and is explained in Chapter 5 [Interface Functions], page 37. *Wait()* is used to suspend execution until indications are available for specified sessions. *Abandon()* is used to abandon locally the result of a pending asynchronous operation. Two additional functions *Bind()*[1] and *Unbind()* are used to open and close a user-session.

There are also other interface specific functions called *Get-Assoc-Info()*, *Get-Last-Error()*, *Validate-object()*, *Error-Message()*, *Initialize()*, *Shutdown()* and *Negotiate()*.

The detailed specifications are given in Chapter 5 [Interface Functions], page 37.

3.1.1 Negotiation Sequence

The interface has an initialize and shutdown sequence that permits the negotiation of optional features. This involves the functions *Initialize()*, *Negotiate()*, and *Shutdown()*.

Every application program must first call *Initialize()*, that returns a workspace. This workspace supports only the standard common GCP package, See Chapter 6 [Interface Class Definitions], page 87.

The workspace can be extended to support one of the MGCP, MEGACO or H.248 packages, or any combination of them (see Chapter 6 [Interface Class Definitions], page 87, and any combination of the optional Gateway Control Services packages), or any vendor extensions. Vendor extensions may include additional packages, and may also include additional or modified functionality. All such packages or other extensions are identified by means of OSI Object Identifiers, and the Object Identifiers are supplied to the *Negotiate()* function to incorporate the extensions into the workspace. Features defined by this specification are described and assigned Object Identifiers in Chapter 5 [Interface Functions], page 37. A feature represents any package or additional or modified functionality

[1] See Section 5.8 [Bind()], page 49.

that is subject to negotiation. The *Negotiate()* function allows some particular features to be made available.

After a workspace with the required features has been negotiated in this way, the application can use the workspace as required. It can create and manipulate OM objects using the OM functions, and can start one or more gateway control sessions using *Bind()*.[2] All the sessions on a given workspace share the same features.

Eventually, when it has completed its tasks, terminated all its gateway control sessions using *Unbind()*, and released all its OM objects using *OM-Delete()*, the application should ensure that resources associated with the interface are freed by calling *Shutdown()*.

A miscellaneous error arise if an attempt is made to use an unavailable feature. If an instance of a class that is not in an available package is supplied as a function argument, the **bad-class** error arises.

3.1.2 Names, Addresses and Titles

To address a wide variety of gateway control transport protocols the interface is capable of accepting various forms of object names, system addresses and program or system titles.

- **Name** is an "abstract class" that contains various subclass types used to define specific systems responsible for producing gateway control services.
- **Address** is an "abstract class" that contains various subclass types used to define the specific location to contact a particular consumer or producer of gateway services. For example, the UDP-Address subclass is typically used to define the location of a producer or consumer.
- **Title** is an "abstract class" that contains various subclass types used to define a specific system name responsible for producing gateway control services.

All three abstract classes participate in an implementation-specific name resolution scheme. It is assumed that given a **Name**, an implementation can determine the **Title** responsible for that **Name**. It is also assumed that given a **Title**, an implementation can determine the **Address** of that **Title**.[3]

The producer of an invoked operation may be explicitly designated at the interface boundary using the following precedence rules:

1. A default Title or Address may be supplied as parameters to a bound "session". If both are provided, the implementation will verify that the Title resolves to the Address.
2. If *Automatic Association Management* is used, a provider Title or Address may be supplied as parameters within the "context" or a specific operation request. If both are provided, the implementation will verify that the Title resolves to the Address. The "context" Title or Address take precedence over the "session" Title or Address. The "context" Title or Address takes precedence over the "session" Title or Address for unassociated session objects.
3. A consumer address may be supplied as a parameter within the "argument" of a specific operation request. The "argument" Address take precedence over either the "session" Title or Address or the "context" Title or Address.
4. If the producer of an invoked operation is not explicitly designated at the interface boundary, the implementation will resolve the Name to the appropriate Title or Address.

[2] See Section 5.8 [Bind()], page 49.

[3] Note that the way in which these relationships are resolved is implementation-dependent, but use of DNS should play a significant role.

3.2 Session

A session identifies to which gateway control entity a particular operation will be sent. It contains some **Bind-Arguments**, such as the name of the consumer. The session is passed as the first argument to most interface functions.

A session is described by an OM object of OM class **Session**.[4] It is created, and appropriate parameter values may be set, using the OSI-Abstract-Data Manipulation functions. A gateway control session is then started with *Bind()* (see Section 5.8 [Bind()], page 49) and later is terminated with *Unbind()* (see Section 5.28 [Unbind()], page 82). A session with default parameters can be started by passing the constant **Default-Session** (‘`(OM_object)GCP_DEFAULT_SESSION`’) as the **Session** argument to *Bind()*. *Bind()* must be called before the **Session** can be used as an argument to any other function in the interface. After *Unbind()* has been called, *Bind()* must be called again if another session is to be started using the same session object.

The interface supports multiple concurrent sessions, so that an application implemented as a single process, such as a service in a client-service model, can interface with the Gateway Control Services using several identities; and so that a process can interact directly and concurrently with different gateway control services.

Detailed specifications of the OM class **Session** are given in Section 6.2.40 [Session], page 106.

A session can be used either acting as a consumer of gateway control services, or acting as a producer of gateway control services, or both.

A session can be restricted for use only with a designated program called the responder. When the responder is omitted and *Automatic Association Management* is used, the session can be used to exchange gateway control service information with all processes.

The responder (title and address) parameters of an opened session, if present, specifies the producer of the requested operation. The precedence rules on address and title of the responder are described in Section 3.1.2 [Names, Addresses and Titles], page 18.

Other OM attributes (vendors’ implementation extensions) may be included to specify characteristics of the underlying protocol used.

There are three aspects of session objects, as follows:

1. A session can have *Automatic Association Management* enabled or disabled.
2. A session can have *Automatic Dialog Handling* enabled or disabled.
3. A session can have *Automatic Message Handling* enabled or disabled.

These aspects are described in sub-sections below.

3.2.1 Association Management

3.2.1.1 AAM Enabled Session

The **Session** collects together all the information that described a particular gateway control interaction. The parameters that are to control such a session are set up in an instance of this OM class, which is then passed as an argument to *Bind()*.[5] This sets the OM attributes that describe the actual characteristics of the session, and starts the session. Such a started session can be passed as the first argument to interface functions.

No attribute of a bound or connected session may be changed. The result of modifying a started session is unspecified.

[4] See Section 6.2.40 [Session], page 106.

[5] See Section 5.8 [Bind()], page 49.

Finally, *Unbind()* is used to terminate the session, after which the parameters can be modified and a new session started using the same instance, if required. Multiple concurrent sessions can be run, by using multiple instances of this OM class.

A session allows a requesting program (the requester) to exchange gateway control information with another program designated (the responder) or by default to all programs.

An *AAM enabled* session thus allows a gateway control entity to access either a portion of the gateway control services (that is, that are accessible via the designated responder) or all gateway control services. In the later case, the producer gateway control entity resolution is performed by the Gateway Control Service provider, according to the gateway control services invoked.

This type of session object can not be used to receive or send ACSE related primitives or operations explicitly. To use ACSE explicitly, see Section 3.2.1.2 [AAM Disabled Session], page 20.

3.2.1.2 AAM Disabled Session

A session object can has *Automatic Association Management (AAM)* disabled when it belongs to a workspace that has had *AAM* disabled with *Negotiate()*. Disabling *AAM* allows the user to explicitly send and receive underlying transport operations to build and tear down associations. It gives explicit control over associations to the user. The Gateway Control Service provider does not automatically perform any underlying transport operations on behalf of the user.

When the user creates and binds a session object in a workspace with *AAM* disabled, only the following attributes within the session object can be specified:

- *requester-Address* — the address (IP address) used by the local gateway control entity to communicate with remote gateway control entities. When not specified for MGC entities, the *requester-Title* must resolve to an address (i.e. using DNS).
- *requester-Title* — the title (host name) used by the local gateway control entity to communicate with remote gateway control entities. When not specified for MGC entities, the *requester-Address* must be specified.
- *role* – the role of the local gateway control entity to be taken in communications with associated entities. This can either identify the MGC role or the MG role.

The session object is then passed as an argument to the *Bind()* function (see Section 5.8 [Bind()], page 49) that binds the session. This bound session can only used to send underlying transport related operations and to receive underlying transport related primitives. The following can be sent/received using this type of bound session.

- *Receive()* (`gcp_receive()`/'`GCP_TRANSPORT_IND`')
- *Receive()* (`gcp_receive()`/'`GCP_TRANSPORT_CNF`')
- *Assoc-req()* (`gcp_assoc_req()`)
- *Assoc-rsp()* (`gcp_assoc_rsp()`)

The other attributes that relate to the underlying transport are specified within an **Assoc-Argument** or **Assoc-Result** object that is passed to, or returned from, *Assoc-req()*, *Assoc-rsp()*, or *Receive()*.

3.2.1.3 Associated Session

Once a user has created a bound session that has AAM disabled, an association can be created. An association is represented by an *associated* or *partially associated* session object. An *associated* session is returned as the result of building a new association. The associated session is used, like a bound session, by sending an receiving gateway control transaction handling or service operations. The major difference is that an associated session object can only be used to send and receive

operations to, or from, a single remote gateway control entity. After a session is associated, the user can abort the association, which implicitly unbinds the associated, or partially associated, session.

The precedence rules for common parameters within the **Session** and the **Context** objects are different for associated session objects. Once a session is in the associated state; the *responder-Address* and *responder-Title* cannot be overridden by the context object.

To terminate this type of session, the user should either abort the session, which implicitly unbinds the session. If the user unbinds the associated session prior to aborting the association, the service provider will abort the association.

3.2.2 Message Handling

3.2.2.1 AMH Enabled Session

The AMH enabled session allows a gateway control entity to invoke and respond to gateway control services requests and indications without regard for message handling. The Gateway Control Services provider provides all message handling.

This type of session cannot be used to send messages explicitly. To dispatch messages explicitly, see Section 3.2.2.2 [AMH Disabled Session], page 21.

3.2.2.2 AMH Disabled Session

A session object can have *Automatic Message Handling* disabled when it belongs to a workspace that has *Automatic Message Handling* disabled using the *Negotiate()* function. This allows the user to explicitly send and receive message handling operations to establish, group and dispatch messages. it gives explicit control over messages to the user. The Gateway Control Service provider does no message handling operations on behalf of the user.

Oncee the session object is bound (AAM enabled) or associated (AAM disabled) and has AMH disabled, the session must explicitly issue message handling operations for each gateway control services transaction. This bound or associated session can only be used to send message handling primitives. The following can be sent/received using this type of bound or associated session:

- *Receive()* (`gcp_receive()`/'`GCP_MESSAGE_IND`')
- *Send()* (`gcp_send()`/'`GCP_MESSAGE_REQ`')

The other attributes that relate to message handling are specified within the **Send-Argument** or **Message-Result** objects that are passed to, or returned from, *Send()* or *Receive()*.

3.2.2.3 Message Session

Once a user has created a bound or associated session that has AMH disabled, a message can be created. A message is represented by a *fully formed*, or *partially formed message*, session obejct. A *message* session is returned as the result of building a new message. The message session is used, like a bound or associated session, by sending and receiving gateway control service operations. The major difference is that a message session object can only be used to send and receive operations within a single message with a single remote gateway control entity. After a session forms a message, the user can close or abort the message, which returns the session to the bound or associated state.

The precedence rules for common parameters within the **Session** and the **Context** objects are different for message session objects. Once a session has formed a message, the message related argument, *application-Context-Name*, cannot be overridden by the context object.

To terminate this type of session, the user should either abort or close the message, which implicitly unbinds the session. If the user unbinds the message session prior to either closing or aborting the

message, the service provider will first attempt to close the message, and if that is rejected, will abort the message.

3.2.3 Dialog Handling

MGCP only permits one service request or response per message. For MGCP dialog handling is trivial and can be handled automatically by the Gateway Control Service provider.

MEGACO and H.248, on the other hand, permit multiple commands within multiple communications contexts per transaction (where each transaction is essentially a dialogue). Actions or commands that are send within a transaction are executed in sequential order, where the first action or command must be completed before the next action or command is initiated. There is no guarantee of synchronization or timing between actions or commands that are placed in separate transactions. Therefore, for actions or commands that must be executed in order, there are two approaches from an API perspective:

1. Issue the commands or actions in separate transactions; however, do not issue a subsequent action or command until the response to the preceding action or command has been received. Actions or commands can be issued synchronously or asynchronously, without affecting the sequence of events, just allowing the calling process to proceed and do other things awaiting message receipt.
2. Collect the commands or actions into a single transaction.

Note that the first method is the only method applicable to MGCP because MGCP does not permit actions or commands to be grouped into transactions: there is imply one action per message. This does not cause any logical difficulties for MGCP with the minor exception that if a message is lost, a noticeable delay might occur between actions or commands.[6]

In general, the Gateway Control Services provider acting on behalf of a service invoker can select the mosts appropriate approach from those outlined above and made possible by the Standards; however, the XGCP interface requires a mechanism for communicating which actions or commands depend upon which others, and whether the remainder of the sequence be encountered should an error be encountered for one of the actions or commands in the sequence.

To distinguish groupings of actions or commands, actions or commands can be grouped into dialogues. Dialogues are a sequence of operations the form a unit. They are directly related to 'transactions' in ITU-T Recommendation H.248.1 (09/2005).

3.2.3.1 ADH Enabled Session

A session object has *Automatic Dialog Handling (ADH)* enabled by default, or when it belongs to a workspace that has had *ADH* disabled using the *Negotiate()* function. The *ADH* enabled session allows a gateway control entity to invoke and respond to gateway control services requests and indications without regards for dialog handling. The Gateway Control Services provider provides all dialog handling.

This type of session cannot be used to send dialog handling primitives or operations explicitly. To use dialog handling explicitly, see Section 3.2.3.2 [ADH Disabled Session], page 22.

3.2.3.2 ADH Disabled Session

A session object has *Automatic Dialog Handling (ADH)* disabled when it belongs to a workspace that has had *ADH* disabled using the *Negotiate()* function.

[6] In practise, MGC and MG equipment is often collocated or not very distant and is connected by clean, high bandwidth connections.

This allows the user to explicitly send and receive transaction handling operations to establish, group and dispatch transactions. It gives explicit control over transactions to the user. The Gateway Control Service provider does no transaction handling operations on behalf of the user.

Once the session object is bound (AAM enabled) or associated (AAM disabled) and has ADH disabled, the session must explicitly issue transaction handling operations for each gateway control services request or response. This bound or associated session can only be used to send transaction handling primitives. The following can be sent/received using this type of bound or associated session:

- *Receive()* (`gcp_receive()`/'`GCP_OPEN_IND`')
- *Receive()* (`gcp_receive()`/'`GCP_ACCEPT_CNF`')
- *Receive()* (`gcp_receive()`/'`GCP_REFUSE_CNF`')
- *Open()* (`gcp_open()`) See Section 5.16 [Open()], page 63.
- *Accept()* (`gcp_accept()`) See Section 5.5 [Accept()], page 43.
- *Refuse()* (`gcp_refuse()`) See Section 5.18 [Refuse()], page 68.

The other attributes that relate to transaction handling are specified within the **Open-Argument**, **Accept-Result** or **Refuse-Result** objects that are passed to, or returned from, *Open()*, *Accept()*, *Refuse()*, or *Receive()*.

3.2.3.3 Dialog Session

Once a user has created a bound or associated session that has ADH disabled, a transaction can be created. A transaction is represented by a *fully formed*, or *partially formed transaction*, session object. A *transaction* session is returned as the result of building a new transaction. The transaction session is used, like a bound or associated session, by sending and receiving gateway control service operations. The major difference is that a transaction session object can only be used to send and receive operations within a single transaction with a single remote gateway control entity. After a session forms a transaction, the user can close or abort the transaction, which returns the session to the bound or associated state.

The precedence rules for common parameters within the **Session** and the **Context** objects are different for transaction session objects. Once a session has formed a transaction, the transaction related argument, *application-Context-Name*, cannot be overridden by the context object.

To terminate this type of session, the user should either abort or close the transaction, which implicitly unbinds the session. If the user unbinds the transaction session prior to either closing or aborting the transaction, the service provider will first attempt to close the transaction, and if that is rejected, will abort the transaction.

3.3 Context

The context defines the characteristics of the gateway control interaction that are specific to a particular gateway control operation, but are often used unchanged for many operations. Since the parameters are presumed to be relatively static for a given user during a particular gateway control interaction, these arguments are collected into an OM object of OM class **Context**, which is supplied as the second argument of each gateway control operation. This serves to reduce the number of arguments passed to each function.

The context includes various administrative details, such as the *mode* defined in the Abstract Service, which affects the processing of each gateway control operation. These include a number of *Service Controls* and *Local Controls* that allow control over some aspects of the operation. The

Service Controls include **mode**, **responder-Address**, and **responder-Title**. The *Local Controls* include **asynchronous**, **reply-Limit**, **time-Limit**. Each of these is mapped onto an OM attribute in the **Context**, and they are detailed in Chapter 6 [Interface Class Definitions], page 87.

The effect is as if they were passed as a group of additional arguments on every function call. The value of each component of the context is determined when the interface function is called, and remains fixed throughout the operation.

The precedence rules on address and title of the responder are described in Section 3.1.2 [Names, Addresses and Titles], page 18.

Some of the OM attributes in the **Context** have default values, some of which are locally administered. The constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`') can be passed as the value of the **Context** argument to the interface functions, and has the same effect as a context OM object created with default values. The context must be a private object, unless it is **Default-Context**.

Detailed specifications of the OM class **Context** are given in Chapter 6 [Interface Class Definitions], page 87.

3.4 Function Arguments

The Abstract Service defines specific arguments for each operation. These are mapped onto corresponding arguments to each interface function (which are also called input parameters). Although each service has different arguments, some specific arguments recur in several operations; these are briefly introduced here. As far as the H.248 package is concerned, OM classes are defined with a one-to-one mapping to the ASN.1 Abstract Syntax of H.248. Full details of these and all the other arguments are given in the function definitions in Chapter 5 [Interface Functions], page 37, and the OM class definitions in Chapter 6 [Interface Class Definitions], page 87.

All arguments that are OM objects can generally be supplied to the interface functions as public objects (i.e, descriptor lists) or as private objects. Private objects must be created in the workspace that was returned by *Initialize()*. In some cases, constants can be supplied instead of OM objects.

Note that wherever a function is stated as accepting an instance of a particular OM class as the value of an argument, it will also accept an instance of any subclass of that OM class. For example, the **Service-Req** function has a parameter **argument**, which accepts values of OM class **Service-Argument**. Any of the subclasses of **Service-Argument** may be supplied as the value of **argument**.

Rules for interpretation of '`ANY`' syntax appearing in function arguments are defined in Section 3.4.1 [Encoding and Decoding], page 24.

3.4.1 Encoding and Decoding

XGCP specifies two alternatives for encoding and decoding of Gateway Control Packages OM-Attribute values of type '`ANY`', or any OM-Attribute values in a Gateway Control Services package.

1. The encoding and decoding functionality can be provided internally with the XGCP API, without requiring the application to invoke any encoding or decoding functions. This option allows the application to be free from any knowledge of encoding rules. In this case, the OM class and attribute type and corresponding representation are defined in a gateway control services package. The XGCP API uses the package definition to attempt encoding or decoding; if automatic decoding fails, an OM String(Encoding) is used.
2. The application can perform encoding and decoding itself. This option gives the application responsibility and control over the encoding and decoding of OM attributes. In this case, all OM attribute values appear as an OM String(Encoding).

The encoding and decoding alternative to be used is negotiated through the *Negotiate()* function; See Section 5.15 [Negotiate()], page 59.

The XGCP API does not specify the use of OM-Encode or OM-Decode for the OM classes defined in this specification, or in gateway control services packages used with this specification.

To ensure interoperability, the sender and receiver must follow the same encoding rules when converting between OM syntax and encoded syntax. If an algorithm is used to generate OM packages, then the algorithm must ensure that the generated OM syntax is consistent with the input abstract syntax (that is, the same encoded values must result from applying the encoding rules to either representation). The encoding rules used with MGCP, MEGACO and H.248 packages defined by this specification are ASN.1 BER. This does not imply that other encoding rules cannot be used with other packages defined in the future.

For the API to encode and decode the OM attribute values according to the ASN.1 standard scheme, ASN.1 tagging information must be stored for each OM object and each OM attribute. Thus, the package definitions in the workspace need to incorporate the ASN.1 tagging information for each OM object and each OM attribute definition for all Gateway Control Services packages.

As a minimum, the following requirements apply:

- All rules specified in ISO/IEC 8825 – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) shall be adhered to. Any exceptions or restrictions must be stated.
- ASN.1 tagging information must be retained for each OM object and each OM attribute in the Gateway Control Services packages.
- The specified encoding and decoding scheme (and any implementation thereof) should be extensible to accommodate the new encoding rules established subsequent to ISO/IEC 8825.

3.4.2 Argument and Response

Most operations and notifications take an argument to specify the argument of the operation and a response when issuing the response of the operation. These arguments and responses are specified to accept values of OM classes that are consistent with the abstract service view (MGCP, MEGACO or H.248) of the current operation.

The argument for a *Service-req()* function is represented by an instance of the OM Class **MGCP-Service-Req-Argument** for an MGCP operation, or an instance of the OM Class **MEGACO-Service-Req-Argument** for a MEGACO operation, or **H248-Service-Req-Argument** for an H.248 operation.

The reponse for a *Service-rsp()* function is represented by an instance of the OM Class **MGCP-Service-Result**, **MGCP-Linked-Reply-Argument Service-Error** or **MGCP-Service-Reject** to represent the possible responses of the MGCP service request operation, or an instance of the **MEGACO-Service-Result**, **MEGACO-Linked-Reply-Argument Service-Error** or **MEGACO-Service-Reject** to represent the possible responses of the MEGACO service request operation, or an instance of the **H248-Service-Result**, **H248-Linked-Reply-Argument Service-Error** or **H248-Service-Reject** to represent the possible responses of the H.248 service request operation.

3.5 Function Results

All functions return a **Status** (which is the C function result). Most return an **Dialog-ID** which identifies the particular invocation. The confirmed operations each return a **Result**. (The **Dialog-ID** and **Result** are returned using pointers that are supplied as arguments of the C function). These three kinds of function results are introduced below.

All OM objects returned by interface functions (results and errors) will be private objects in the workspace associated with the session private object.

3.5.1 Dialog-ID

All interface functions that invoke a gateway control service operation return an **Dialog-ID**; an integer that identifies the particular invocation of an operation. The **Dialog-ID** is only relevant for asynchronous confirmed operations and may be used later to receive the **Status** and **Result**, or to abandon them. The **Dialog-ID** is also used to respond to a previously requested confirmed operation. Asynchronous operations are fully described in Section 3.6 [Synchronous and Asynchronous Operation], page 27. The interface functions that can be used to start them are the *Service-req()* function.

The numerical value of the **Dialog-ID** returned from a call that successfully invoke an asynchronous confirmed operation is guaranteed to be unique amongst all outstanding operations in given session. Dialog IDs used by XGCP are not necessarily those that are actually sent via an underlying protocol such as H.248. Dialog IDs may be mapped or altered by the Gateway Control Service provider.

The value returned for a synchronous operation or an asynchronous non-confirmed operation is unspecified, as is that for a call that fails to invoke an operation.

3.5.2 Result

Functions invoking confirmed gateway control service operations return a result only if they succeed. All errors from these functions are reported in the **Status** described below, as are errors from all other functions.

The value returned by a function call that invokes an asynchronous operation is unspecified, as is that for a call that fails to invoke an operation. The result of an asynchronous operation is returned by a later call to *Receive()*.

The result of a function invoking a confirmed operation can be composed of a single reply, or of multiple linked replies. In the later case, the term *partial result* is used to designate one of these linked replies. Only a confirmed **Service-req** may produce multiple results. Multiple replies to a single gateway control service operation may only occur if the invoker specifies multiple-reply in the functional unit attribute of the Session object.

In asynchronous mode, the partial results can be retrieved one at a time by subsequent calls to *Receive()*, which each time returns an instance of OM class **Linked-Reply-Argument**. In synchronous mode, the function returns an instance of OM class **Multiple-Reply**, which contains a list of sub-objects of OM class **Linked-Reply-Argument**.

The result (or partial result) of an operation is returned in a private object whose OM class is appropriate to the particular operation. The format of gateway control service operation results is driven both by the Abstract Service and by the need to provide asynchronous execution of functions. To simplify processing of asynchronous results, the result (or partial result) of a single operation is returned in a single OM object (corresponding to the abstract result defined in the Standards). The components of the result (or partial result) of an operation are represented by OM attribute in the operation's result object. All information contained in the Abstract Service result is made available to the application program. The result (partial result) is inspected using the functions provided in the OSI-Abstract-Data Manipulation API.

Only confirmed gateway control service operations produce results, and each type of operation has a specific OM class of OM object for its result. These OM classes are defined in Chapter 6 [Interface Class Definitions], page 87.

The actual OM class of the result can always be a subclass of that named, to allow flexibility for extensions. Thus, the function *OM-Instance()* should always be used when testing the OM class.

3.5.3 Status

Every interface function returns a **Status** value, that is either the constant **success** ('`(GCP_status)0`' or '`GCP_SUCCESS`') or an error. Function call errors are represented as integer constants and grouped in categories of System, Library and Communications as described in Chapter 7 [Errors], page 129.

Additional error information is available for System and Communications errors via the *Get-Last-Error()* function call. Additional error information is available for the **bad-argument** Library error via the *Validate-object()* function call.

A synchronous call with multiple linked replies is considered successful unless the reply limit or time limit is exceeded. The function returns a **Status** value equal to success, and the argument *Result* is an OM object of OM class **Multiple-Reply**, which contains all the linked replies.

It should be noted that OM object **Linked-Reply-Argument** may contain an OM attribute that reflects an error.

If the reply limit or time limit is exceeded, the synchronous call fails and returns a status of the appropriate Library error. However, the *Result* is still considered valid and may contain an OM-Object **Multiple-Reply**, which contains all the received linked replies. A result of *GCP_ABSENT_OBJECT* means no replies were received.

In most cases other results of functions are initialized to Null (*GCP_ABSENT_OBJECT*) if the status does not have the value **success**. However, the *Result* is still considered valid and may contain an OM-Object of partial replies. A result of *GCP_ABSENT_OBJECT* means no replies were received.

3.6 Synchronous and Asynchronous Operation

The asynchronous or synchronous mode of a requested operation is specified at the interface, and determined for each operation by the value of the OM attribute *Asynchronous* in the **Context** passed to the interface function. The default value of this OM attribute is **false**, causing all operations to by synchronous. Support for both synchronous and asynchronous operation is mandatory. There is a limit to the number of pending asynchronous operations; this limit is given by the constant **max-outstanding-operations**, and has a minimum value of 10.

In synchronous mode, all functions wait until the operation is complete before returning. Thus the thread of control is blocked within the interface after calling a function, and the application can make use of the result immediately after the function returns.

In asynchronous mode, some functions return before the operation is complete. The application is then able to continue with other processing while the operation is being executed by the Gateway Control Service provider, and can then access the result by calling *Receive()*. An application may initiate several concurrent asynchronous operations on the same session before receiving any of the results, subject to the limit described below. The results are not guaranteed to be returned in any particular order. The functions that can execute asynchronously are the *Service-req()* function. This corresponds to the gateway control services of the Standards that operate in a confirmed mode. Moreover, only confirmed operations return service results.

An asynchronous function call of a confirmed service returns a **Dialog-ID** of the operation to the application. The same **Dialog-ID** will be returned by *Receive()* on the corresponding result.

A **Dialog-ID** is also returned by *Receive()* on an indication of an invoked gateway control service operation. The same **Dialog-ID** will be used to respond to this operation.

Implementations of the interface are free to return from asynchronous function calls as soon as possible or may wait until the operation has been submitted to the underlying Gateway Control Service provider. The actual policy used is implementation-defined.

Implementations will define a limit to the number of asynchronous operations that may be outstanding at any one time on any one session. An asynchronous operation is outstanding from the time that the function is called until the last reply of the result is returned by *Receive()*, or the operation is abandoned by *Abandon()*, or the session is closed by *Unbind()*. The limit is given by the constant **max-outstanding-operations** (‘`GCP_MAX_OUTSTANDING_OPERATIONS`’) and is at least 10 for conforming XGCP implementations. While this number of operations is outstanding, attempts to invoke further asynchronous operations will report a **Library-Error** (too many operations).

Asynchronous operation calls can be aborted by executing an *Abandon()* or *Unbind()* call. In this case, the operation is no longer outstanding and the result will never be returned by further *Receive()* function calls.

If an error is detected before an asynchronous request is submitted to the Gateway Control Service provider, the function will return immediately and there will be no outstanding operation generated. Other errors are notified later by *Receive()*, when the result of the outstanding asynchronous confirmed operation is returned. All errors occurring during a synchronous request are reported when the function returns. Full details of error handling are given in Chapter 7 [Errors], page 129.

Where vendors provide suitable system primitives (such as System V `poll(2s)`, or BSD `select(2)`), applications can obtain a file descriptor from the **Session** by inspecting the value of the OM attribute *File-Descriptor*. Applications may use the file descriptor to suspend the process until data is received on the particular file descriptor.

Applications should ensure that there are no outstanding asynchronous operations on a session when *Unbind()* is called on that session. Once *Unbind()* has been called there is no way to determine whether any outstanding operations succeed or even whether they were ever sent to the Gateway Control Service provider. Also no errors or results of any kind will be reported to the application. It is strongly recommended that *Receive()* is called repeatedly until **Completion-Flag** takes the value **nothing**.

3.7 Other Features

These features are not part of the interface itself, but are mandatory when specified by the Gateway Control Service provider.

The Gateway Control Protocols are not restricted to those defined by H.248.

All the features listed below are for the most part necessary for ease of use in a gateway control environment. These features are classified as given registered identifiers (Object Identifier). They can be negotiated using the *Negotiate()* function in the same manner as packages. Other types of information that are critical in servicing an environment that includes implementation from multiple vendors on various machines can also be classified and handled with the *Negotiate()* function. Features defined by this specification are described and assigned Object Identifiers in Chapter 5 [Interface Functions], page 37.

3.7.1 Automatic Association Management

When the Gateway Control Services provider makes use of association oriented communication services, such as TCP or SCTP, the Gateway Control Service provider implementations are assumed to provide automatic handling of the association between gateway control entities, establishing and releasing associations at its discretion. Such management is intended to bring benefits such as reduced communication charges. To allow this flexibility to the implementation, the interface does not specify when communication takes place. *Automatic Association Management (AAM)* may be enabled or disabled on a per-workspace basis using the *Negotiate()* function.

3.7.2 Automatic Message Handling

When the Gateway Control Services provider makes use of message oriented communication services, such as that provided by H.248, the Gateway Control Service provider implementations are assume to provide automatic handling of messages between gateway control entities, grouping transactions into messages at its discretion. Such handling is intended to bring benefits such as reduced communication overheads. To allow this flexibility to the implementation, the interface does not specify which transactions are packed into which messages. *Automatic Message Handling (AMH)* may be enabled or disabled on a per-workspace basis using the *Negotiate()* function.[7]

3.7.3 Automatic Dialog Handling

When the Gateway Control Services provider makes use of transaction oriented communication services, such as that provided by H.248, the Gateway Control Service provider implementations are assumed to provide automatic handling of transactions between gateway control entities, establishing and releasing transactions at its discretion. Such handling is intended to bring benefits such as reduced communication overheads. To allow this flexibility to the implementation, the interface does not specify when communication takes place. *Automatic Dialog Handling (ADH)* may be enabled or disabled on a per-workspace basis using the *Negotiate()* function.[8]

Under MEGACO and H.248, transactions are used to provide for sequencing of a set of commands. Commands that are contained within separate transactions provide no guarantees whatsoever of ordering. When ordering of commands is required, either the commands must be placed into the same transaction, or the result of each command must be awaited before issuing the subsequent command. Due to the possibility of message loss and the delays associated with retransmission of the lost messages, the later approach (awaiting the result of the previous command before issuing the current command) can introduce inter-command delays that may or may not be acceptable. On the other hand, preparing the full sequence of commands in advance may not be possible, as the next command chosen may depend on the results of a previous command. In general; however, grouping multiple commands into a transaction can serve to provide a limited set of compound commands from short sequences of simple commands.

3.7.4 Automatic Performer Resolution

The performer of an invoked operation may be explicitly designated by the responder name and responder address parameters of the bound session used.

However, in the case where the responder is specified as a wildcard, the Gateway Control Service provider may be assumed to provide automatic gateway control service and application context to consumer resolution: to find out the consumer that is in charge of the selected gateway control service specified in the gateway control service operation.

3.7.5 Responder Versatility

Responder versatility is the ability to change the consumer within a same bound-session at each function call. It is useful when the automatic consumer resolution is either not supported by the Gateway Control Service provider or not requested. This applies if the underlying Gateway Control Service provider is connection-less.

[7] Note that *Automatic Message Handling* is an independent concept from *Automatic Association Management*.
[8] Note that *Automatic Dialog Handling* is an independent concept from *Automatic Association Management*.

3.7.6 Automatic Name to Address Resolution

Gateway Control Service provider implementation may provide automatic resolution between program name and address to find the network address of a gateway control entity from its name using directory or translation services.

3.7.7 Automatic Dispatching to Appropriate Stack

The Gateway Control Services provider implementation may provide a loop back facility if the destination of the operation or notification is local. It also may provide routing of the gateway control services operation to the proper underlying communications stack according to the implied gateway control service and the destination (for example over UDP or SCTP).

3.8 Function Sequencing

A minimum set of sequencing rules applies when using the interface to exchange gateway control service information between gateway control programs acting as a gateway control entity. These rules need to be respected by gateway control programs to ensure that interface functions are called in the proper sequence and that the state of the interface is not violated, otherwise **Library-error** status will be returned.[9]

The general rules to follow are:

1. Initialize a workspace ('`gcp_initialize()`')
2. Negotiate features of the interface ('`gcp_negotiate()`')
3. Open one or several sessions ('`gcp_bind()`')
4. Perform gateway control service interactions (operations) using the offered interface functions. An interaction is identified by its **Dialog-Id**.
5. Close the opened sessions ('`gcp_unbind()`')
6. Discard the workspace ('`gcp_shutdown()`')

Seven states are defined in the interface to cover both interface service operations and gateway control service interactions:

UNINIT Workspace uninitialized.

INIT Workspace initialized.

UNBND Session closed.

BND Session opened.

IDLE

OUTOP Outstanding operation requested in a gateway control service interaction.

OPIND Operation indication received in a gateway control service interaction.

[9] Note the following is considered as tutorial information. The definitive information is contained in the Standards (see referenced documents).

4 Interface Packages

4.1 Feature Packages

4.1.1 Common GCP Package

The Object-Identifier associated with the Common GCP package is:

```
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591)
  xom-packages(1) xgcp(2) common(1) }
```

This Object-Identifier is represented by the constant **Common-Package** (‘GCP_COMMON_PKG’).

4.1.2 MGCP Package

The Object-Identifier associated with the MGCP package is:

```
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591)
  xom-packages(1) xgcp(2) mgcp(2) }
```

This Object-Identifier is represented by the constant **MGCP-Package** (‘GCP_MGCP_PKG’).

4.1.3 MEGACO Package

The Object-Identifier associated with the MEGACO package is:

```
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591)
  xom-packages(1) xgcp(2) megaco(3) }
```

This Object-Identifier is represented by the constant **MEGACO-Package** (‘GCP_MEGACO_PKG’).

The Object-Identifiers associated with the MEGACO package for various versions of the protocol are:

```
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591)
  xom-packages(1) xgcp(2) megaco(3) v1(1) }
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591)
  xom-packages(1) xgcp(2) megaco(3) v2(2) }
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591)
  xom-packages(1) xgcp(2) megaco(3) v3(3) }
```

This Object-Identifiers are represented by the constants **MEGACO-V1-Package** (‘GCP_MEGACO_V1_PKG’), **MEGACO-V1-Package** (‘GCP_MEGACO_V2_PKG’) and **MEGACO-V1-Package** (‘GCP_MEGACO_V3_PKG’), respectively.

4.1.4 H.248 Package

The Object-Identifier associated with the H.248 package is:

```
{ itu-t(0) recommendation(0) h(8) h248(248) part1(1) }
```

This Object-Identifier is represented by the constant **H248-Package** (‘GCP_H248_PKG’).

The Object-Identifiers associated with the H.248.1 package for various versions of the protocol are:

```
{ itu-t(0) recommendation(0) h(8) h248(248) part1(1) version1(1) }
{ itu-t(0) recommendation(0) h(8) h248(248) part1(1) version1(2) }
{ itu-t(0) recommendation(0) h(8) h248(248) part1(1) version1(3) }
```

This Object-Identifiers are represented by the constants **MEGACO-V1-Package** (‘GCP_H248_V1_PKG’), **MEGACO-V2-Package** (‘GCP_H248_V2_PKG’) and **MEGACO-V3-Package** (‘GCP_H248_V3_PKG’), respectively.

4.1.5 H.248 Extension Packages

4.1.5.1 H.248.x Sub-series Extension Packages

ITU-T Recommendation H.248.1 (09/2005) provides for its own extension package concept. Each extension package is in a part of the H.248 suite of specifications. There are over 50 of these extension packages defined and the list is too long to repeat here and the list would contain too much redundant information (see ITU-T Recommendations Series H Supplement 2 (07/2010) for a guide to the ITU-T H.248.x sub-series packages). Therefore, these recommendations and extension packages are identified by an Object-Identifier of the general form: `{ itu-t(0) recommendation(0) h(8) h248(248) partX(X) versionY(Y) }`, where, '`X`' is the part number in the H.248 suite (for example, for H.248.12, '`X`' is '`12`'); and, '`Y`' is the GCP protocol version (for example, for GCP Version 3, '`Y`' is '`3`'). The 'C' language binding of these Object-Identifiers are **H248-X-VY-Package** ('`GCP_H248_X_VY_PKG`'). To refer to the extension package without specifying the version, use `{ itu-t(0) recommendation(0) h(8) h248(248) partX(X) }` only. The 'C' language binding of these Object-Identifiers are **H248-X-Package** ('`GCP_H248_X_PKG`').

The ITU-T Recommendation H.248.x sub-series extension packages are itemized in Table 4.1, Table 4.2, Table 4.3 and Table 4.4.

ITU-T H.248.1 Annex E	Basic packages.
ITU-T H.248.2	Facsimile, text, conversation and call discrimination packages.
ITU-T H.248.3	User interface elements and actions package.
ITU-T H.248.6	Dynamic tone definition package.
ITU-T H.248.7	Generic announcement package.
ITU-T H.248.9	Advanced media server packages.
ITU-T H.248.10	Media gateway resource congestion handling package.
ITU-T H.248.11	Media gateway overload control package.
ITU-T H.248.12	H.248.1 packages for H.323 and H.324 interworking.
ITU-T H.248.12 Annex A	Extended H.324, H.245 command and H.245 indication packages.
ITU-T H.248.13	Quality alert ceasing package.
ITU-T H.248.14	Inactivity timer package.
ITU-T H.248.15	SDP H.248 package attribute.
ITU-T H.248.16	Enhanced digit collection packages and procedures.
ITU-T H.248.17	Line test packages.
ITU-T H.248.18	Package for support of multiple profiles.
ITU-T H.248.19	Decomposed MCU, audio, video and data conferencing packages.

Table 4.1: *ITU-T Rec. H.248.x Packages (1–19)*

ITU-T H.248.20	Local and remote descriptors with H.221/H.223 multiplexing.
ITU-T H.248.21	Semi-permanent connection handling package.
ITU-T H.248.22	Shared risk group package.
ITU-T H.248.23	Enhanced alerting packages.
ITU-T H.248.24	MF tone generation and detection packages.
ITU-T H.248.25	Basic CAS packages.
ITU-T H.248.26	Enhanced analogue lines package.
ITU-T H.248.27	Supplemental tones packages.
ITU-T H.248.28	International CAS packages.
ITU-T H.248.29	International CAS compelled register signalling packages.
ITU-T H.248.30	RTCP extended performance metrics package.
ITU-T H.248.31	Adaptive jitter buffer package.
ITU-T H.248.32	Detailed congestion reporting package.
ITU-T H.248.33	PCM frame spare bit package.
ITU-T H.248.34	Stimulus analogue line package.
ITU-T H.248.35	Coin-operated phone control package.
ITU-T H.248.36	Hanging termination detection package.
ITU-T H.248.37	IP NAPT traversal package.
ITU-T H.248.38	Base context package.
ITU-T H.248.39	SDP parameter identification and wildcarding.

Table 4.2: *ITU-T Rec. H.248.x Packages (20–39)*

ITU-T H.248.40	Application data inactivity detection package.
ITU-T H.248.41	IP domain connection package.
ITU-T H.248.42	DCME interworking package.
ITU-T H.248.43	Gate management packages.
ITU-T H.248.44	Multi-level precedence and pre-emption package.
ITU-T H.248.45	MGC information package.
ITU-T H.248.46	Connection capability control package.
ITU-T H.248.47	Statistic conditional reporting package.
ITU-T H.248.48	RTCP XR block reporting package.
ITU-T H.248.49	SDP RFC packages.
ITU-T H.248.50	NAT traversal toolkit packages.
ITU-T H.248.51	Termination connection model package.
ITU-T H.248.52	Quality of service packages.
ITU-T H.248.53	Traffic management packages.
ITU-T H.248.54	MPLS support package.
ITU-T H.248.55	Generic pull mode package.
ITU-T H.248.56	Virtual private network packages.
ITU-T H.248.57	RTP control protocol package.
ITU-T H.248.58	Package for application of level H.248 statistics.
ITU-T H.248.59	Event timestamp notification package.

Table 4.3: *ITU-T Rec. H.248.x Packages (40–59)*

ITU-T H.248.60	Identification of content of communication package.
ITU-T H.248.61	Packages for network level H.248 statistics.
ITU-T H.248.62	Re-answer package.
ITU-T H.248.63	Resource management package.
ITU-T H.248.64	IP router packages.
ITU-T H.248.65	Support of the resource reservation protocol.
ITU-T H.248.66	Packages for RTSP and H.248 interworking.
ITU-T H.248.67	GCP transport mode indication package.
ITU-T H.248.68	Package for removal of digits and tones.
ITU-T H.248.69	Packages for interworking between MSRP and H.248.
ITU-T H.248.70	Dialling method information packages.
ITU-T H.248.71	RTCP support packages.
ITU-T H.248.72	ITU-T H.248 support for MONA.
ITU-T H.248.73	MSCML and ITU-T H.248 interworking.
ITU-T H.248.74	Media resource control enhancement package.
ITU-T H.248.75	Package identifier publishing and application package.
ITU-T H.248.76	Filter group package and guidelines.
ITU-T H.248.77	SRTP package and procedures.
ITU-T H.248.78	Bearer-level application level gateway.
ITU-T H.248.80	Usage of the revised SDP offer/answer model with H.248.

Table 4.4: *ITU-T Rec. H.248.x Packages (60–80)*

4.1.5.2 Q.1950 Annex A Extension Packages

ITU-T Recommendation Q.1950 (12/2002) and ITU-T Recommendation Q.1950 Amendment 1 (01/2006) provide for additional H.248 extension packages that are numbered separately from H.248.x sub-series packages. These recommendations and extension packages are identified by an Object-Identifier of the general form: `{ itu-t(0) recommendation(0) q(17) q1950(1950) AnnexY(Y) PartX(X) }`, where, '`Y`' is the annex number (for example, for Q.1950 Annex A, '`Y`' is '`1`'); '`X`' is the part number in the Q.1950 suite (for example, for Q.1950 Annex A.6, '`X`' is '`6`'), but is only specified when there is a part of the Annex. The 'C' language binding of these Object-Identifiers is **CBC-AnnexY-X-Package** ('`GCP_CBC_ANNEXY_X_PKG`'). To refer to the complete set of CBC extensions without reference to a specific part, use `{ itu-t(0) recommendation(0) q(17) q1950(1950) }` only. The 'C' language binding of these Object-Identifiers are **CBC-Package** ('`GCP_CBC_PKG`').

The ITU-T Recommendation Q.1950 extension packages are itemized in Table 4.5.

ITU-T Q.1950/A.3	Bearer characteristics package.
ITU-T Q.1950/A.4	Bearer network connection cut through package.
ITU-T Q.1950/A.5	Reuse idle package.
ITU-T Q.1950/A.6	Generic bearer connection package.
ITU-T Q.1950/A.7	Bearer control tunnelling package.
ITU-T Q.1950/A.8	Basic call progress tones generator with directionality.
ITU-T Q.1950/A.9	Expanded call progress tones generator package.
ITU-T Q.1950/A.10	Basic services tones generation package.
ITU-T Q.1950/A.11	Expanded services tones generation package.
ITU-T Q.1950/A.12	Intrusion tones generation package.
ITU-T Q.1950/A.13	Business tones generation package.
ITU-T Q.1950/B	CBC continuity test. NOTE 1.
ITU-T Q.1950/C	CBC BIWF congestion handling. NOTE 2.
ITU-T Q.1950/D	CBC N x 64k package.
ITU-T Q.1950/E	CBC extensions for access networks that support BICC.
ITU-T Q.1950/F	CBC emergency call indication.
ITU-T Q.1950/G	CBC international emergency preference scheme.

NOTES:

1. This package requires the H.248.1 Basic continuity test control package.
2. This package requires the H.248.10 Media Gateway resource congestion handling package.

Table 4.5: *ITU-T Rec. Q.1950 (12/2002) Packages*

4.2 Feature Profiles

The Standards provide for a wide range of ITU-T Recommendation H.248 extension packages. Many standards bodies have defined profiles of extension packages that define the activation of sets of feature packages. These profiles are defined here. Activation of a given H.248 profile results in the activation deactivation of corresponding feature packages.

5 Interface Functions

5.1 Send()

NAME **Send** – send a gateway control message.

SYNOPSIS

```
#include <xom..h>
#include <xgcp.h>

GCP_status gcp_send(
        OM_private_object   session,
        OM_private_object   context,
        OM_object           argument,
        OM_private_object   *result_return,
        OM_sint32           *message_id-return
);
```

DESCRIPTION

This function is used to send a gateway control message when *Automatic Message Handling* is disabled on a session.

ARGUMENTS

Session (Object(Session))

The gateway control session on which to send the message. This must be a private object previously returned from *Bind()* or *Open()*.

Context (Object(Context))

The gateway control context to be used for this operation. This argument must be a private object or the constant **Default-Context** (‘`GCP_DEFAULT_CONTEXT`’).

The service may be requested in a confirmed mode or a non-confirmed mode. In confirmed mode, a reply is expected.

Argument (Object(Message))

The information supplied as the argument of a gateway control message is an instance of a subclass of the OM class **Message**. Normally, concrete subclasses of this class are defined in Gateway Control Services packages. For example, the **H248-Message** subclass of **Message** is defined in the H.248 Services package is defined in the H.248 Services package (see Section 6.7 [H.248 Package], page 117) and may be used as an argument to this function.

RESULTS

Status (Status)

When the function is called synchronously, the values **success** indicates that the action was completed. If called asynchronously, it indicates that the operation was initiated.

Result (Object(*))

Upon successful completion of a syncrhonous call, when the operation was requested in a confirmed mode, the result is one of the following:

- When the service is requested in a non-confirmed mode, no results are expected and the constant **Absent-Object** ('GCP_ABSENT_OBJECT') is returned as the result.
- When a confirmed mode service is requested, this is indicated by an instance of the OM class **Message-Result** or **Message-Error**, or when multiple replies are provided, an instance of OM class **Multiple-Reply**, which contains a set of instances of the OM class **Message-Linked-Reply**. Each **Message-Linked-Reply** contains exactly one of the following OM attributes:
  - message-Result
  - message-Error
  - processing-Failure

Message-ID (Integer)
: The **Message-Id** of the initiated gateway control message when invoked asynchronously. It is significant in the case of a confirmed mode request only.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-session, miscellaneous not-supported, session-terminated, reply-limit-exceeded, time-limit-exceeded.

This function can return a **Communications-Error**.

This function can also return the error constants, `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]`, `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Abandon()*,[1] *Response()*.

CORESPONDENCE

This function corresponds to the H248Message of ITU-T Recommendation H.248.1.

[1] See Section 5.2 [Abandon()], page 39.

5.2 Abandon()

NAME **Abandon** – abandon locally the result of a pending asynchronous operation.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_abandon(
        OM_private_object  session,
        OM_sint32          invoke_id
);
```

DESCRIPTION

This function abandons the result of an outstanding asynchronous function call. The function is no longer outstanding after this function returns, and the result (or the remaining results in case of multiple linked replies) will never be returned by *Receive()*.

Abandon() may, but need not, cause the Gateway Control Service provider to abandon the outstanding asynchronous operation itself (as opposed to simply discarding the result). Note that the specified behaviour is a local mater, and no statement is made about underlying gateway control service operations that may or may not be abandoned.

This function can only be called in synchronous mode.

ARGUMENTS

Session (Object(Session))
: The gateway control session in which the confirmed operation was requested. This must be a private object previously returned from *Bind()*.[1]

Dialog-ID (Integer)
: Selects the specific outstanding asynchronous operation submitted via the **Session** to be terminated. The outstanding operation may be a non-confirmed service. In that case the abandon is without effect. The value of **Dialog-ID** must be that which was returned by the function call that initiated the asynchronous gateway control operation that is now to be abandoned.

RESULTS

Status (Status)
: Indicates whether or not the abandon function succeeded.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-session, bad-procedural-use, miscellaneous, session-terminated.

This function can return a **Communications-Error**.

Note that the abandon function is successful even if the operation to be abandoned does not exist (any longer) or is not confirmed. The abandon is then without effect.

The function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCE]`.

[1] See Section 5.8 [Bind()], page 49.

5.3 Abort()

NAME **Abort** – Abort Association.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_abort(
        OM_private_object  session,
        OM_private_object  context,
        OM_object          argument
);
```

DESCRIPTION

This function is one of a group of dialog handling functions: *Open()*, *Accept()*, *Refuse()*, *Issue()*, *Close()*, *Abort()*, *Receive()*, used to manage the MAP dialog when AAM is disabled on a session. When AAM is enabled on a session, this dialog handling function is neither necessary nor permitted.

This function is used to abort a gateway control session that is either associated or partially associated. The service is defined as an unconfirmed service: a reply is not expected.

Once an abort is issued, the associated session is implicitly disassociated and unbound. All outstanding requests that pertain to this session are returned with the error session-terminated. This includes any *Wait()* request on that session.

ARGUMENTS

Session (Object(Session))
: The associated (or partially associated) session against which this operation is perfomed. This must be a private object previously returned as part of an **Accept-Result** or **Open-Argument**, or returned explicitly from an asynchronously called *Open()*. This session object must have AAM disabled.

Context (Object(Context))
: The gateway control context to be used for this operation. This argument must be a private object. Once a session is connected or partially connected, the precedence rules for common parameters within the **Session** and the **Context** objects are different. Once connected, the responder address and responder title cann not be overriden by the **Context** object. (See also Section 3.1.2 [Names, Addresses and Titles], page 18.)

Argument (Object(Abort-Argument))
: The information supplied as the argument of an **Abort** operation. (See Section 6.2.2 [Abort-Argument], page 90.)

RESULTS As this function is not confirmed, there are no results returned.

ERRORS This function can return a **Communications-Error**, or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-session, miscellaneous, missing-type, session-terminated, reply-limit-exceeded, time-limit-edceeded.

The function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCES]`.

CORRESPONDENCE

This function corresponds to the *MAP-U-ABORT* request primitive of 3GPP TS 29.002 Section 7.

5.4 Abort-req()

NAME **Abort-req** — abort a gateway control association.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_abort_req(
        OM_private_object   session,
        OM_private_object   context,
        OM_object           response
);
```

DESCRIPTION

This function is used to abort a gateway control session that is either associated or partially associated. The service is defined as an unconfirmed service. A reply is not expected.

Once an abort request has been issued, the associated session is implicitly disassociated and unbound. All outstanding requests that pertain to this session are returned with the error *session-terminated*. This includes any *Wait()* requests on that session.

ARGUMENTS

Session (Object(Session))

The associated (or partially associated) session against which this operation is performed. This must be a private object previously returned as part of an **Assoc-Result** or **Assoc-Argument**, or returned explicitly from an asynchronously called *Assoc-req()*. This session object must have AAM disabled.

Context (Object(Context))

The gateway control context to be used for this operation. This argument must be a private object.

Once a session is associated or partially associated, the precedence rules for common parameters within the **Session** and **Context** objects are different. Once associated, the *responder-Address* and *responder-Title* cannot be overridden by the **Context** object.

Response (Object(Abort-Result))

The information supplied as the response of an **Abort** operation.

RESULTS

Status (Status)

The value **success** indicates that the operation was completed.

ERRORS This function can return a **Communications-Error**, or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-session, miscellaneous, missing-type, session-terminated, reply-limit-exceeded, time-limit-exceeded.

This function can also return the error constants `[MP_NO_WORKSPACE]`, `[MP_INVALID_SESSION]` or `[MP_INSUFFICIENT_RESOURCES]`.

5.5 Accept()

NAME **Accept** — accept an indicated gateway control open operation.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_accept(
        OM_private_object       session,
        OM_private_object       context,
        OM_object               response,
        OM_sint32               dialog_id,
        OM_private_object      *connected_session_return
);
```

DESCRIPTION

This function is one of a group of dialog handling functions: *Open()*, *Accept()*, *Refuse()*, *Issue()*, *Close()*, *Abort()*, *Receive()*, used to manage the MAP dialog when AAM is disabled on a session. When AAM is enabled on a session, this dialog handling function is neither necessary nor permitted.

This function is used to accept a previously indicated **Open** operation. This function can only be called in syncrhonous mode.

ARGUMENTS

Session (Object(Session))
: The gateway control session against which this operation is to be performed. This must be a private object previously returned from *Bind()*.[1]

Context (Object(Context))
: The gateway control context to be used for this operation. This argument must be a private object or the constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`').

Response (Object(Accept-Result))
: The information supplied in acceptance of the previously indicated **Open** operation. This is an instance of OM class **Accept-Result**, indicating that the open indication is to be accepted. The user provides negotiated ACSE parameters in this object as input to the service provider. A newly formed dialog object is returned in the **Accept-Result** object, so this is an in/out parameter to this function. The new **Session** object represents a fully formed dialog session.

Dialog-ID (Integer)
: The **Dialog-ID** of the requested dialog to which the reply applies. This **Dialog-ID** must have been returned from a call to *Receive()* for the corresponding association that is being accepted.

RESULTS

Status (Status)
: Indicates whether or not the **Accept** response was completed.

[1] See Section 5.8 [Bind()], page 49.

Connected-Session (Object(Session))

When successful, this function returns a newly connected session object. The returned **Session** object is in a connected state, and contains the final negotiated ACSE parameters for the new association.

It is not specified whether the association has been formed by the underlying Gateway Control Service provider at the time that this function returns. The underlying Gateway Control Service might wait until a *Issue()* function call before fully forming the association. Nevertheless, the association is considered to be in the "connected" state from the standpoint of the interface.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-context, bad-result, bad-session, miscellaneous, no-such-operation, not-supported, session-terminated.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCES]`.

CORRESPONDENCE

This function corresponds to the *MAP-OPEN* response primitive (with a parameter indicating acceptance) of 3GPP TS 29.002 Section 7.

5.6 Assoc-req()

NAME **Assoc-req** — establish gateway control association.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_assoc_req(
        OM_private_object   session,
        OM_private_object   context,
        OM_object           argument,
        OM_private_object  *result_return
);
```

DESCRIPTION

This function is used to request the immediate establishment of a gateway control association when *Automatic Association Management* is not used. Note that association establishment performs end-to-end communication for all underlying transports, regardless of whether they are connection-oriented or connectionless. When the underlying transport is connectionless (such as UDP), the association establishment operation still includes the exchange of ServiceChange messages intended on initiating the association.

The service is defined as a confirmed service. A reply is expected.

This operation may be called in synchronous or asynchronous modes.

When *Automatic Association Management* is used for a gateway control session, this function is not used. See Section 5.16 [Open()], page 63.

ARGUMENTS

Session (Object(Session))
: The gateway control session against which this operation is performed. This must be a private object previously returned from *Bind()*.[1] This session must also have *Automatic Association Management (AAM)* disabled.

Context (Object(Context))
: The gateway control context to be used for this operation. This argument must be a private object.

Argument (Object(Assoc-Argument))
: The information supplied as an argument of an **Assoc** operation. This is an **Assoc-Argument** object with optional ACSE information contained within it. An associated **Session** object is returned in the *Result* of this function.

RESULTS

Status (Status)
: The value **success** indicates that the action was completed.

[1] See Section 5.8 [Bind()], page 49.

Result (Object(Assoc-Result))

Upon successful completion, when the **Assoc-req** has been accepted/rejected by the Gateway Control Service provider, one instance of the OM class **Assoc-Result** Object is returned. This **Assoc-Result** object either contains information as to why an association was rejected, or contains a **Session** object in an associated state, and contains the final negotiated ACSE parameters for the new association.

ERRORS This function can return one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-session, miscellaneous, missing-type, session-terminated, reply-limit-exceeded, time-limit-exceeded.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` or `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Assoc-rsp()*.[2]

[2] See Section 5.7 [Assoc-rsp()], page 47.

5.7 Assoc-rsp()

NAME **Assoc-rsp** — reply to a requested association operation.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_assoc_rsp(
        OM_private_object   session,
        OM_private_object   context,
        OM_object           response,
        OM_sint32           assoc_id
);
```

DESCRIPTION

This function is used to reply to a previously invoked *Assoc* operation.

This function can only be called in synchronous mode.

ARGUMENTS

Session (Object(Session))

The gateway control session against which this operation is performed. This must be a private object previously returned from *Bind()*.[1]

Context (Object(Context))

The gateway control context to be used for this operation. This must be a private object or the constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`').

Response (Object(Assoc-Result))

The information supplied as a response to an **Assoc** operation. This is one of the following:

- When an association is accepted, one instance of the OM class **Assoc-Result** is given as the response. The user provides negotiated ACSE parameters in this object as input to the service provider, and also indicates that the association is to be accepted by setting the *assoc-Result* attribute to **accept**. A newly associated session object is returned in the **Assoc-Result** object, so this is an in/out parameter to this function. The new **Session** object is in an associated state, and contains the final negotiated ACSE parameters for the new association.
- When an association is to be rejected, one instance of the OM class **Assoc-Result** is given as a response. The **Assoc-Result** should have the *assoc-Result* attribute set to either **reject-permanent** or **reject-transient**. The *assoc-Diagnostic* can also optionally be set to indicate why the reject has occurred.

Assoc-ID (Integer)

The **Assoc-ID** of the requested operation to which the reply applies.

RESULTS

[1] See Section 5.8 [Bind()], page 49.

Status (Status)
Indicates whether or note the **Assoc** response was completed.

ERRORS Ths function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-context, bad-result, bad-session, miscellaneous, no-such-operation, not-supported, session-terminated.

SEE ALSO *Assoc-req()*.[2]

[2] See Section 5.6 [Assoc-req()], page 45.

5.8 Bind()

NAME **Bind** — open a gateway control session.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_bind(
        OM_object           session,
        OM_workspace        workspace,
        OM_private_object  *bound_session_return
);
```

DESCRIPTION

This function opens a gateway control session. It creates a **Session** OM object describing the session suitable for supplying to other XGCP functions. A session must be opened before any gateway control service interactions can take place.

If the OM attribute **requester-Title** is specified, only one unconnected session can be opened with the same value of the OM attribute. There can be multiple connected or partially connected session objects with the same **requester-Title**.

To allow for the implementation of *Automatic Association Management*, it is undefined as to whether *Bind()* causes any communication with the remote gateway control entity.

ARGUMENTS

Session (Object(Session))

Specifies a program together with other details of the service required. This argument may be either a public object or a private object. The constant **Default-Session** (‘GCP_DEFAULT_SESSION’) may also be used as the value of this argument, causing a new session to be created with default values for all its OM attributes.

Workspace (Workspace)

Specifies the workspace (obtained from a call to *Initialize()*) which is to be associated with the session. All function results from gateway control operations using this session will be returned as private objects in this workspace. If the Session argument is a private object, it must be a private object in this workspace.

Whether the resulting session uses *Automatic Association Management (AAM)* or not depends upon the negotiation of this feature (using the *Netogiate()* interface function) for the workspace supplied as an argument.

RESULTS

Status (Status)

Indicates whether or not the function completed successfully.

Bound-Session (Object(Session))

Upon successful completion, contains an instance of a gateway control session that may be used as an argument to other functions (e.g. *Service-req()*). This will be a new private object if the value of **Session** was **Default-Session** or a public object, otherwise, it will be that supplied as

an argument. In the later case, the session provided should not be already in use. The function will supply default values for any of the OM attribute that were not present in the **Session** instance supplied as an argument. It will also set the value of the **File-Descriptor** OM attribute (the value will be **No-Valid-File-Descriptor** ('`GCP_NO_VALID_FILE_DESCRIPTOR`') if the functionality is not supported).

When *Automatic Association Management (AAM)* is disabled for the *Workspace* with *Negotiate()*, any session bound using *Bind()* is unconnected and may only be used to receive and send ACSE-related primitives, (i.e. it cannot be used for Gateway Control Service operations).

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-address, bad-session, bad-title, miscellaneous, not-supported, too-many-sessions.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Unbind()*,[1] *Negotiate()*.[2]

[1] See Section 5.28 [Unbind()], page 82.
[2] See Section 5.15 [Negotiate()], page 59.

5.9 Close()

NAME **Close** — terminate a gateway control association.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_close(
        OM_private_object       session,
        OM_private_object       context,
        OM_object               argument
);
```

DESCRIPTION

This function is used to request the termination of a gateway control association. The service is defined as an unconfirmed service. No reply is expected.

This operation may only be called in synchronous mode.

ARGUMENTS

Session (Object(Session))

The connected session against which this operation is performed. This must be a private object previously returned as part of an **Accept-Result** or **Open-Argument**. This session object must have *Automatic Association Management (AAM)* disabled, and it must be in a connected state.

Context (Object(Context))

The gateway control context to be used for this operation. This argument must be a private object.

Once a session is connected or partially connected, the precedence rules for common parameters within the **Session** and the **Context** objects are different. Once connected, the responder address and title cannot be overridden by the **Context** object.

Argument (Object(Close-Argument))

The information supplied as the argument of a **Close** operation.

RESULTS

Status (Status)

The value **success** indicates that the operation was completed.

ERRORS This function can return a **Communications-Error**, or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-session, miscellaneous, missing-type, session-terminated, reply-limit-exceeded, time-limit-exceeded.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Abort()*.[1]

CORRESPONDENCE

This function corresponds to the *MAP-CLOSE* request primitive of 3GPP TS 29.002 Section 7.

[1] See Section 5.3 [Abort()], page 40.

5.10 Error-Message()

NAME **Error-Message** — return an error message describing a particular error.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

OM_sint gcp_error_message(
        GCP_status          error,
        OM_sint             length,
        unsigned char      *error_text_return
);
```

DESCRIPTION

This function returns an error message string that describes the error. The caller provides a buffer-address and buffer-length argument. The error message is stored in the client's buffer.

ARGUMENTS

Error (Status)

Length The length of the buffer. The error text buffer is an unsigned character array. This is necessary if the intent is to support NLS (the X/Open Native Language System).

RESULTS

Error-text (String)

A message describing the error. The error message text is terminated by a NUL character.

The error message text will be truncated if the length of the error-text-buffer is less than the length of the error message text.

Length (Integer)

Indicates the length of the returned mesage. If the **length** parameter is 0 or the ***error_text_return** parameter is NULL, then the **length_return** value indicates the amount of buffer space required to host the error message.

ERRORS This function returns no errors. (A default error message reports faulty arguments or other problems).

5.11 Get-Assoc-Info()

NAME **Get-Assoc-Info** — retrieve negotiated association values.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_get_assoc_info(
        OM_private_object       receive_result_or_argument,
        OM_uint                 request_mask,
        OM_uint                 result_mask,
        OM_public_object        *pres_layer_args,
        OM_public_object        *acse_args,
        OM_public_object        *gcp_assoc_args
);
```

DESCRIPTION

This function returns the negotiated association values corresponding to an incoming **Result-Or-Argument** object previously supplied by *Receive()*. The caller provides a **request_mask** to identify which values are to be returned in result objects.

This function may be used with *Automatic Association Management* enabled or disabled or with *Automatic Message Handling* enabled or disabled. In any case, the values returned are those associated with the underlying association within which the incoming **Result-Or-Argument** arrived. In connectionless environments, the values returned are those associated with the incoming **Result-Or-Argument** object (for example, **Responder-Address**).

Certain requested values may not be available for the input object (that is, inappropriate for the underlying protocol) and may therefore be absent from the result. The values actually returned are indicated by the function result.

ARGUMENTS

Result-Or-Argument (Object(*))

This object contains an asynchronous response or indication, as previously returned to the user from the *Receive()* function.

Request-Mask (Integer)

The **request-mask** indicates which association values should be returned as result objects. The mask is composed of bit values that must be set *on* (‘1’) to request that the corresponding association value be returned. Association values that can be obtained by calling this function are:

- presentation-Context-List
- responder-Address
- responder-Title
- application-Context
- application-Context-List
- version-List

RESULTS

Result-Mask (Integer)
A mask indicating which association values have been returned as part of the result objects below. This mask has the same structure as the *Request-Mask* argument. All bits *off* ('0') indicates no values were available for the input object.

Pres-Layer-Args (Object(Presentation-Layer-Args))
Upon completion of this function, this object contains the negotiated values associated with the **Result-Or-Argument** object. This object is returned only when one of the following *Result-Mask* bits is set *on*:

- 'GCP_T_PRESENTATION_CONTEXT_LIST'

Otherwise, [GCP_ABSENT_OBJECT] is returned for this object.

Acse-Args (Object(Acse-Args))
Upon completion of this function, this object contains the negotiated values associated with the **Result-Or-Argument** object. This object is returned only when one of the following *Result-Mask* bits is set *on*:

- 'GCP_T_RESPONDER_ADDRESS'
- 'GCP_T_RESPONDER_TITLE'
- 'GCP_T_APPLICATION_CONTEXT'

Otherwise, [GCP_ABSENT_OBJECT] is returned for this object.

Gcp-Assoc-Args (Object(Gcp-Assoc-Args))
Upon completion of this function, this object contains the negotiated values associated with the **Result-Or-Argument** object. This object is returned only when the one of the following *Result-Mask* bits is set *on*:

- 'GCP_T_APPLICATION_CONTEXT_LIST'
- 'GCP_T_VERSION_LIST'

otherwise, [GCP_ABSENT_OBJECT] is returned for this object.

ERRORS This function can return error constants [GCP_NO_WORKSPACE], [GCP_INVALID_SESSION] or [GCP_INSUFFICIENT_RESOURCES].

5.12 Get-last-error()

NAME **Get-last-error** — retrieve secondary return code of the most recent function call Communications or System error.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_get_last_error(
        OM_workspace        workspace,
        OM_uint32          *additional_return_error
);
```

DESCRIPTION

This function is used to return additional error information related to the last function call that returned a status of:

- `[GCP_E_COMMUNICATIONS_PROBLEM]`
- `[GCP_E_BROKEN_SESSION]`
- `[GCP_E_INVALID_CONNECTION_ID]`
- `[GCP_E_SYSTEM]`

The returned integer value is implementation dependent.

In a multiple thread environment where there are multiple XGCP function calls, additional error information is stored in the workspace of the invoking call on a thread basis. The `gcp_get_last_error` call must be invoked from the same thread.

For most XGCP function calls, the Workspace anchor to store additional information is dervied from the '`bound_session`'. For *Bind()* and *Wait()*, the workspace parameter on the calls is used.

ARGUMENTS

Workspace (Workspace)
: The workspace (obtained from a prior call to *Initialize()* of the function call that had the status error.

RESULTS

Status (Status)
: Indicates whether or not the function call completed.

Additional-Error (Integer)
: The secondary integer related to the last function call that returned a Communications or System error.

ERRORS This function can return error constants `[GCP_NO_WORKSPACE]` or `[GCP_INVALID_SESSION]`.

5.13 Initialize()

NAME **Initialize** — initialize the interface.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

OM_workspace gcp_initialize(
        void
);
```

DESCRIPTION

This function performs any necessary initialization of the interface and allocates a workspace. It must be called before any other gateway control interface functions are called. It may be called multiple times, in which case each call returns a workspace that is distinct from other workspaces created by *Initialize()* but not yet deleted by *Shutdown()*.

ARGUMENTS

None.

RESULTS

Workspace (Workspace)

Upon successful completion, contains a handle to a workspace in which OM objects can be created and manipulated. Objects created in this workspace, and only such objects, may be used as arguments to the other gateway control interface functions. This function returns NULL if it fails.

ERRORS None.

SEE ALSO *Shutdown()*.[1]

EXAMPLE

```
OM_workspace workspace;

if ((workspace = gcp_initialize()) == NULL) {
         exit(1);
}
```

[1] See Section 5.24 [Shutdown()], page 77.

5.14 Issue()

NAME **Issue** — issue pending service requests and responses.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_issue(
        OM_private_object       session,
        OM_private_object       context,
        OM_sint32               dialog_id
);
```

DESCRIPTION

This function is one of a group of dialog handling functions: *Open()*, *Accept()*, *Refuse()*, *Issue()*, *Close()*, *Abort()*, *Receive()*, used to manage GCP dialog when *Automatic Association Management (AAM)* is disabled on a session. When *AAM* is enabled on a session, this dialog handling function is neither necessary nor permitted.

This function is used to issue pending gateway control service requests and responses. The function can only be called in synchronous mode.

When an *Open()*, *Accept()* or *Refuse()* function is called, the underlying Gateway Control Service provider might not issue the corresponding MGCP, MEGACO or H.248 transactions immediately, but may wait for the accumulation of gateway control service requests or responses to combine with the dialog. The *Issue()* function tells the underlying Gateway Control Service provider to release pending dialog handling primitives with the service requests and responses currently accumulated.

One typical order of dialog handling function calls would be as follows:

1. *Bind()* — bind the session.
2. *Open()* — open the dialog.
3. *Service-req()* — generate one or more service requests.
4. *Issue()* – issue pending dialog handing and accumulated service requests or responses.
5. *Service-req()* — generate one or more service requests.
6. *Close()* — close the dialog (also issuing any pending dialog handling and accumulated service requests or responses).

Note that MGCP only permits one transaction (and, therefore, one service request or response) per message. MEGACO and H.248 support multiple transactions per message, and multiple service requests or responses per transaction.

ARGUMENTS

Session (Object(Session))
: The gateway control session against which this operation is to be performed. This must be a private object previously returned from *Bind()*.[1]

[1] See Section 5.8 [Bind()], page 49.

Context (Object(Context))
: The gateway control context to be used for this operation. This argument must be a private object or the constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`').

Dialog-ID (Integer)
: The **Dialog-ID** of the requested dialog on which to issue accumulated service requests and responses. This **Dialog-ID** must have been returned from a call to *Open()*[2] or *Receive()*[3] for the corresponding association for which pending service requests and responses are being issued.

RESULTS

Status (Status)
: Indicates whether or not the **Issue** request was completed.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-context, bad-result, bad-session, miscellaneous, no-such-operation, not-supported, session-terminated.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCES]`.

CORRESPONDENCE

This function corresponds to the *GCP-DELIM* request primitive of 3GPP TS 29.002 Section 7.

[2] See Section 5.16 [Open()], page 63.
[3] See Section 5.17 [Receive()], page 65.

5.15 Negotiate()

NAME **Negotiate** — negotiate features of the interface and service.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_negotiate(
        GCP_feature         feature_list[],
        OM_workspace        workspace
);
```

DESCRIPTION

This function negotiates features of the interface; each feature is represented as an Object Identifier. Several features are defined and registered within this specification. Features may also include gateway control services packages, vendor extensions, and new features defined in future versions of this specification. Features can be negotiated after a workspace has been initialized, and can be renegotiated any time until the workspace is discarded. Note that all sessions on a given workspace share the same features.

ARGUMENTS

Feature-List (Feature-List)

An ordered sequence of features, each represented by an object identifier and a request value. The request value can contain one of the following values: Activate, Deactivate, Query State, and Query Supported.

The sequence is terminated by an object identifier having no components (a length of zero and a value of the data pointer in the C representation). The response value is returned upon completion of the Negotiate invocation.

In the C binding, the Feature-List argument is a single array of structures of type `GCP_feature`, which is defined as:

```
#define GCP_ACTIVATE            0
#define GCP_DEACTIVATE          1
#define GCP_QUERY_STATE         2
#define GCP_QUERY_SUPPORTED     3

typedef struct {
        OM_object_identifier    feature;
        OM_sint                 request;
        OM_boolean              response;
} GCP_feature;
```

The following Features are defined and registered by this specification:

- **Gateway Control Packages**
  The GSM Gateway Control package and the ANSI Gateway Control package are specified in Chapter 6 [Interface Class Definitions], page 87. Additional Gateway Control packages may be specified in the future. Note that multiple Gateway Control Packages can be activated within the same workspace.

- **Automatic Association Management**
  This feature provides for automatic establishment and release of the underlying protocol associations.[1] The Object-Identifier associated with this feature is '`{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) common(1) automatic-association-management(1) }`'. This Object-Identifier is represented by the constant '`GCP_AUTOMATIC_ASSOCIATION_MANAGEMENT`'.

  Automatic Association Management is enabled by default.
- **Automatic Message Handling**
  This feature provides for automatic establishment and release of the underlying protocol messages.[2] The Object-Identifier associated with this feature is '`{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) common(1) automatic-message-handling(2) }`'. This Object-Identifier is represented by the constant '`GCP_AUTOMATIC_DIALOG_HANDLING`'.
- **Automatic Dialog Handling**
  This feature provides for automatic establishment and release of the underlying protocol transaction.[3] The Object-Identifier associated with this feature is '`{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) common(1) automatic-transaction-handling(3) }`'. This Object-Identifier is represented by the constant '`GCP_AUTOMATIC_TRANSACTION_HANDLING`'.
- **Automatic ASN.1 BER Encoding and Decoding**
  This feature provides for automatic encoding and decoding of OM class and attribute types using ASN.1 BER.[4] The Object-Identifier associated with this feature is '`{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) common(1) automatic-decoding(4) }`'. This Object-Identifier is represented by the constant '`GCP_AUTOMATIC_DECODING`'.

  Automatic ASN.1 BER Encoding and Decoding is enabled by default.

Gateway Control Services packages are also negotiated as part of the Feature-List. Gateway Control Services packages may be defined by the OpenSS7 Project, by standards organizations or consortia, by vendors, or by users.

Registered Object Identifiers representing future features and vendor extensions may also be included in the Feature-List for negotiation.

Workspace The handle to the workspace for which features are negotiated.

[1] See Section 3.7.1 [Automatic Association Management], page 28.
[2] See Section 3.7.2 [Automatic Message Handling], page 29.
[3] See Section 3.7.3 [Automatic Dialog Handling], page 29.
[4] See Section 3.4.1 [Encoding and Decoding], page 24.

RESULTS

Status (Status)

Whether or not the function completed successfully.

Response (Boolean-List)

If the function completed successfully, this result contains an ordered list of Boolean values, with the same number of elements as the Feature-List. The significance of the values is shown as follows:

Request	Response	Meaning
Activate	True	Activated
	False	Cannot activate feature (or the feature is not supported).
Deactivate	True	Deactivated
	False	Cannot deactivte feature (or the feature is not supported).
Query-state	True	Activated
	False	Deactivated (or the feature is not supported).
Query-supported	True	Supported
	False	Not supported
Invalid	True	Cannot be returned
	False	Invalid argument

In the C binding, this result is combined with the Feature-List argument as a single array of structures of type `GCP_feature` as defined above.

ERRORS This function can return a **System-Error** or **Library-Error** "miscellaneous".

This function does not return a **Communications-Error**, nor any gateway control errors.

The function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCE]`.

EXAMPLE

```
GCP_status status;
OM_workspace workspace;
int i;

workspace = gcp_initialize();

GCP_feature feature_list[] = {
          {GCP_COMMON_PKG, GCP_ACTIVATE, 0}
          , {GCP_GSM_PKG, GCP_ACTIVATE, 0}
          , {GCP_GSM_SM_PKG, GCP_ACTIVATE, 0}
          , {GCP_GSM_LS_PKG, GCP_ACTIVATE, 0}
          , {GCP_GSM_IN_PKG, GCP_ACTIVATE, 0}
          , {GCP_GSM_CH_PKG, GCP_ACTIVATE, 0}
          , {NULL, 0, 0}
};

if ((status = gcp_negotiate(feature_list, workspace)) != GCP_SUCCESS)
          exit(1);

for (i = 0; i < 6; i++)
```

```
if (!feature_list[i].response)
        exit(1);
```

5.16 Open()

NAME **Open** – Request establishment of a Gateway Control Services dialog.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_open(
        OM_private_object       session,
        OM_private_obect        context,
        OM_object               argument,
        OM_private_object       *result_return,
        OM_sint32               *dialog_id_return
);
```

DESCRIPTION

This function is used to request the creation of a gateway control entity association dialog. The service is defined as a confirmed service: a reply is expected.

This operation may be called in asynchronous mode. Note that when operating in this mode, results may not only be locally discarded (when **Abandon()** is used), as may be done with other asynchronous calls.

ARGUMENTS

Session (Object(Session))
: The gateway control session against which this operation is performed. This must be a private object previously returned from *Bind()*.[1] This session must also have ADH[2] disabled.

Context (Object(Context))
: The gateway control context to be used for this operation. This argument must be a private object.

Argument (Object(Assoc-Argument))
: The information supplied as the argument of an **Assoc** operation. This is an **Assoc-Argument** object with optional ACSE information contained within it. When called asynchronously, a partially connected **Session** object is returned in the Result of this function.

RESULTS

Status (Status)
: If the function is called synchronously, the value **success** indicated that the action was completed. If called asynchronously, it indicates that the operation was initiated.

Result (Object(*))
: Upon successful completion of a synchronous call, the results is one of the following:

[1] See Section 5.8 [Bind()], page 49.
[2] See Section 3.7.3 [Automatic Dialog Handling], page 29.

- When the *Open* request has been accepted by the remote peer, one instance of the OM class **Accept-Result** Object is returned. This **Accept-Result** object contains a *session* attribute that returns a **Session** object for which a transaction has been fully formed. The **Accept-Result** object also contains the final negotiated transaction parameters for the new transaction.
- When the *Open* request has been refused by the remote peer, one instance of the OM class **Refuse-Result** Object is returned. This **Refuse-Result** object contains information as to why the transaction was refused. No *session* attribute is present in this result and no transaction session is formed.
- When the *Open* request has been aborted by the remote peer or by the provider, one instance of the OM class **Abort-Argument** is returned. This **Abort-Argument** object contains information pertaining to the abort. No *session* attribute is present in this result and no transaction session is formed.

Upon successful completion of an asynchronous call, a partially formed transaction **Session** object is returned in the *Result*.

Note: The original **Session** object passed to this function is unaffected and still remains in the bound or associated state. This session object can still be used in additional concurrent *Open()* or *Receive()* function calls.

Dialog-ID (Integer)
The returned **Dialog-ID** of the gateway control operation when used asynchronously.

ERRORS This function can return one of the following **Library-Errors**: bad argument, bad-class, bad-context, bad-session, miscellaneous, missing-type, session-terminated, reply-limit-exceeded, time-limit-exceeded.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Abort()*,[3] *Assoc-rsp()*, *Accept()*,[4] *Refuse()*.[5]

CORRESPONDENCE
This function corresponds to the *GCP-OPEN* request primitive of 3GPP TS 29.002 Section 7.

[3] See Section 5.3 [Abort()], page 40.
[4] See Section 5.5 [Accept()], page 43.
[5] See Section 5.18 [Refuse()], page 68.

5.17 Receive()

NAME **Receive** — get the argument of an operation or retrieve the (partial) result of an asynchronously executed operation.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_receive(
        OM_private_object   session,
        OM_sint            *mode_return,
        OM_sint            *primitive_return,
        OM_sint            *completion_flag_return,
        GCP_status         *operation_status_return,
        OM_private_object  *result_or_argument_return,
        OM_sint32          *invoke_or_dialog_id_return
);
```

DESCRIPTION

This function is used to retrieve the argument of an invoked operation and to retrieve a partial result (linked reply) or the completed result of a previous asynchronous operation.

The function results include two status indications. One, called **Status**, indicates that the function call itself was successful; it is always returned. The other, called **Operation-Status**, is used to return the status of the completed asynchronous operation, and is only returned if there is one.

ARGUMENTS

Session (Object(Session))

The moble application session against which this gateway control operation is performed. This must be a private object previously returned from *Bind()*,[1] or a connected or partially connected session object returned from *Open()*,[2] *Accept()*,[3] or *Receive()*.

RESULTS

Status (Status)

Takes an error value if one of the library errors or system errors listed below occurred during execution of this function. Takes the value **success** (‘GCP_SUCCESS’) if this function returned successfuly.

Primitive (Integer)

The gateway control service primitives (‘GCP_SERVICE_IND’, ‘GCP_SERVICE_CNF’).

The GCP association service primitives (‘GCP_OPEN_IND’, ‘GCP_ACCEPT_CNF’, ‘GCP_REFUSE_CNF’, ‘GCP_DELIM_IND’, ‘GCP_CLOSE_IND’, ‘GCP_ABORT_IND’, ‘GCP_P_ABORT_IND’).

[1] See Section 5.8 [Bind()], page 49.
[2] See Section 5.16 [Open()], page 63.
[3] See Section 5.5 [Accept()], page 43.

Determines the operation of this result or argument.

This result is only valid if **Completion-Flag** has the value **completed**, **incoming** or **partial**.

Mode (Integer)

This indicates the mode of an indication. When **confirmed** (‘GCP_T_CONFIRMED’) the invoked operation has to be confirmed, a reply is expected. When **non-confirmed** (‘GCP_T_NON_CONFIRMED’), the requested service is not to be confirmed.

This result is only valid if **Completion-Flag** has the value **incoming**.

Completion-Flag (Integer)

This flag indicates the statue of the received data, if any.

completed (‘GCP_COMPLETED’)
This flag indicates that a *final* response has been received. For gateway control primitives this may be the confirmation for a service request or the last confirmation of a linked reply. In the latter case, the **Result-Or-Argument** parameter will be the **Absent-Object**.

incoming (‘GCP_INCOMING’)
An indication has been received.

nothing (‘GCP_NOTHING’)
There are no indications or confirmations to receive. Further, there are no outstanding asynchronous requests.

outstanding
(‘GCP_OUTSTANDING’)
There are no indications or confirmations to receive. There are still outstanding requests, but no confirmations have yet arrived.

parial (‘GCP_PARTIAL’) A confirmation has been received which is part of a linked reply. This is used for all but the last in a series of linked replies. (See completed, above.)

This result is only valid if **Status** has the value **success**. In that case, the validity of the other results is given as follows:

Completed	yes(1)	no	yes	yes(1)	yes
Incoming	yes	yes	no	yes	yes
Nothing	no	no	no	no	no
Outstanding	no	no	no	no	no
Partial(3)	yes(1)	no	yes	yes(1)	yes

Operation-Status (Status)

Takes an error value if a communications error occurred during the execution of the asynchronous operation, and **success** (‘GCP_SUCCESS’) otherwise. The possible error values are listed for each individual operation in the corresponding function description.

This result is only valid if **Completion-Flag** has the value **completed** or **partial**.

Result-or-Argument (Object(*))

This object contians the results of an asynchronous request, or information about an indication. The class of object received is dependent upon the values of the **Primitive** and **Completion-Flag** parameters. The following table for the three applicable **Completion-Flag** values. The actual class returned is dependent on the value of **Primitive**.

Completion-Flag set to **completed**:

Service-Result	'GCP_SERVICE_CNF'
Service-Error	'GCP_SERVICE_CNF'
Service-Reject	'GCP_SERVICE_CNF'
Accept-Result	'GCP_ACCEPT_CNF'
Refuse-Result	'GCP_REFUSE_CNF'
Absent-Object	All confirmations.

Note that **Absent-Object** may be returned in two cases:

1. The confirmation contains no data.
2. As the terminator of a linked reply list. In this case, the **Invoke-or-Dialog-ID** parameter can be used to determine which linked reply has been terminated.

Completion-Flag set to **incoming**:

Service-Argument	'GCP_SERVICE_IND'
Open-Argument	'GCP_OPEN_IND'
Close-Argument	'GCP_CLOSE_IND'
Abort-Argument	'GCP_ABORT_IND'

Completion-Flag set to **partial**:

Linked-Reply-Argument

For **Completion-Flag** values of **completed** or **partial**, the **Result-or-Argument** parameter is valid only if the **Operation-Status** contains the value **success**. The parameter is not valid for **Completion-Flag** values of **nothing** or **outstanding**.

Invoke-or-Dialog-ID (Integer)

The Dialog-ID or Dialog-ID of the operation whose error, result or argument is being returned.

This result is only valid of the **Status** has the value **success** and **Completion-Flag** has the value **completed**, **partial** or **incoming**.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-context, bad-session, miscellaneous, session-terminated, time-limit-exceeded.

This function does not report any **Communication-Errors**, in its **Status** result. (Any such errors related to the completed asynchronous operation is reported in **Operation-Status**, as described above.)

This function can also return the error constants [GCP_NO_WORKSPACE], [GCP_INVALID_SESSION] and [GCP_INSUFFICIENT_RESOURCES].

5.18 Refuse()

NAME **Refuse** – refuse an indicated association operation.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_refuse(
        OM_private_object       session,
        OM_private_object       context,
        OM_object               response,
        OM_sint32               dialog_id
);
```

DESCRIPTION

This function is one of a group of dialog handling functions: *Open()*, *Accept()*, *Refuse()*, *Issue()*, *Close()*, *Abort()*, *Receive()*, used to manage the GCP dialog when AAM is disabled on a session. When AAM is enabled on a session, this dialog handling function is neither necessary nor permitted.

This function is used to refuse a previously indicated **Open** operation. This function can only be called in syncrhonous mode.

ARGUMENTS

Session (Object(Session))
: The gateway control session against which this operation is to be performed. This must be a private object previously returned from *Bind()*.[1]

Context (Object(Context))
: The gateway control context to be used for this operation. This argument must be a private object or the constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`').

Response (Object(Refuse-Result))
: The information supplied in refusal of the previously indicated **Open** operation. This is an instance of OM class **Refuse-Result**, indicating that the open indication is to be refused. The user provides alternate ACSE parameters in this object as input to the service provider. The **refuse-Reason** and **diagnostic-Information** can be set to indicate why the reject occured.

Dialog-ID (Integer)
: The **Dialog-ID** of the requested dialog to which the reply applies. This **Dialog-ID** must have been returned from a call to *Receive()* for the corresponding association that is being accepted.

RESULTS

Status (Status)
: Indicates whether or not the **Refuse** response was completed.

[1] See Section 5.8 [Bind()], page 49.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-context, bad-result, bad-session, miscellaneous, no-such-operation, not-supported, session-terminated.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCES]`.

CORRESPONDENCE

This function corresponds to the *GCP-OPEN* response primitive (with a parameter indicating refusal) of 3GPP TS 29.002 Section 7.

5.19 Release-req()

NAME **Release-req** —

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_release_req(
);
```

DESCRIPTION
ARGUMENTS

RESULTS

ERRORS

5.20 Release-rsp()

NAME **Release-rsp** —

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_release_rsp(
);
```

DESCRIPTION
ARGUMENTS

RESULTS

ERRORS

5.21 Service-req()

NAME **Service-req** — request gateway control service.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_service_req(
        OM_private_object   session,
        OM_private_object   context,
        OM_object           argument,
        OM_private_object  *result_return,
        OM_sint32          *invoke_id_return
);
```

DESCRIPTION

This function is used to request gateway control service.

ARGUMENTS

Session (Object(Session))

The gateway control session against which this operation is performed. This must be a private object previously returned from *Bind()*.

Context (Object(Context))

The gateway control context to be used for this operation. This argument must be a private object or the constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`').

The service may be requested in a confirmed mode or a non-confirmed mode. In confirmed mode, a reply is expected.

Argument (Object(Service-Argument))

The information supplied as the argument of a gateway control service request is an instance of a subclass of the OM class **Service-Argument**. Normally, concrete subclasses of this class are defined in Gateway Control Services packages.

RESULTS

Status (Status)

If the function is called synchronously, the values **success** indicated that the action was completed. If called asynchronously, it indicates that the operation was initiated.

Result (Object(*))

Upon successful completion of a synchronous call, when the operation was requested in a confirmed mode, the result is one of the following:

- When the service is requested in a non-confirmed mode, no results are expected and the constant **Absent-Object** ('`GCP_ABSENT_OBJECT`') is returned as the result.
- When a confirmed mode services is requested, this is indicated by an instance of the OM class **Service-Result**, or **Service-Error**, or when multiple replies are provided, an instance of OM class

Multiple-Reply, which contains a set of instances of the OM class **Service-Linked-Reply-Argument**. Each **Service-Linked-Reply-Argument** contains exactly one of the following OM attributes:

- service-Result
- service-Error
- processing-Failure

Dialog-ID (Integer)

The **Dialog-ID** of the initiated gateway control service operation when invoked asynchronously. It is significant in the case of a confirmed mode request only.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-session, miscellaneous not-supported, session-terminated, reply-limit-exceeded, time-limit-exceeded.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` or `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Abandon()*,[1] *Service-rsp()*.[2]

CORRESPONDENCE

This function corresponds to the *GCP-XXX* request primitive of 3GPP TS 29.002 Section 7.

[1] See Section 5.2 [Abandon()], page 39.
[2] See Section 5.22 [Service-rsp()], page 74.

5.22 Service-rsp()

NAME **Service-rsp** — reply to a requested gateway control service operation.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_service_rsp(
        OM_private_object   session,
        OM_private_object   context,
        OM_object           response,
        OM_sint32           invoke_id
);
```

DESCRIPTION

This function is used to reply to a previously invoked confirmed gateway control service operation. This function can only be called in synchronous mode.

ARGUMENTS

Session (Object(Session))

The gateway control session against which the operation is performed. This must be a private object previously returned from *Bind()*.

Context (Object(Context))

The gateway control context to be used for this operation. This argument must be a private object or the constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`').

Response (Object(*))

The information supplied as response to an **service** operation. The response is one of the following:

- An instance of a concrete subclass of the OM class **Service-Result** as the response.
- An instance of a concrete subclass of the OM class **Service-Error** as the response, including the problem cause and its associated parameter that may be returned. See Section 6.2.37 [Service-Error], page 105.
- An instance of a concrete subclass of the OM class **Service-Reject** including the problem cause and its associated parameter may be returned: duplicate-invocation, mistyped-argument, resource-limitation, unrecognized-operation.
- When a service request requires multiple responses, this is indicated by one or more *Service-rsp()* calls, once for each response, followed by a final "empty" *Service-rsp()*. Each *Service-rsp()* call includes a response that contains an instance of a concrete subclass of OM class **Service-Linked-Reply-Argument**, containing exacly one of the following OM attributes:
  - *service-Result*
  - *service-Error*

— *processing-Failure*

The final "empty" *Service-rsp()* call includes a response that contains only the constant **Absent-Object** ('`GCP_ABSENT_OBJECT`').

For more details about the OM classes and OM attributes mentioned above, refer to Chapter 6 [Interface Class Definitions], page 87.

Dialog-ID (Integer)
The **Dialog-ID** of the requested operation to which the reply applies. This is the **Dialog-ID** that was returned from a call to *Receive()* that indicated the service request to which this service response corresponds.

RESULTS

Status (Status)
Indicates whether or not the action response was completed.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-context, bad-error, bad-linked-reply, bad-result, bad-session, miscellaneous, no-such-operation, not-supported, session-terminated.

This function can return a **Communications-Error**.

This function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` or `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Receive()*,[1] *Service-req()*.[2]

CORRESPONDENCE
This function corresponds to the *GCP-XXX* response primitive of 3GPP TS 29.002 Section 7.

[1] See Section 5.17 [Receive()], page 65.
[2] See Section 5.21 [Service-req()], page 72.

5.23 Service-parameter()

NAME **Service-parameter** –

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_xxx(
);
```

DESCRIPTION
ARGUMENTS
RESULTS

ERRORS

CORRESPONDENCE

This function corresponds to the *GCP-PARAMETER* request primitive of 3GPP TS 29.002 Section 7.

5.24 Shutdown()

NAME **Shutdown** — delete a workspace and the associated resources.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_shutdown(
        OM_workspace    workspace
);
```

DESCRIPTION

This function deletes a workspace established by *Initialize()*[1] and all the associated resources. It may enable the service to release resources.

All the remaining opened sessions are closed, all the remaining OM objects are deleted, and the workspace is deleted.

No other function may reference the specified workspace after it has been deleted.

ARGUMENTS

Workspace (Workspace)
Specifies the workspace (obtained from a call to *Initialize()*) which is to be deleted.

RESULTS

Status (Status)
Inidcates whether or not the shutdown function succeeded.

ERRORS This function can return the error constants `[GCP_NO_WORKSPACE]` or `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Initialize()*.[2]

EXAMPLE

```
OM_workspace workspace;
GCP_status status;

if ((workspace = gcp_initialize()) == NULL)
         exit(1);

/* perform functions within the workspace */

if ((status = gcp_shutdown(workspace)) != GCP_SUCCESS)
         exit(1);
exit(0);
```

[1] See Section 5.13 [Initialize()], page 56.
[2] See Section 5.13 [Initialize()], page 56.

5.25 Transaction-pnd()

NAME **Transaction-pnd** — notify of a pending requested gateway control transaction.

SYNOPSIS

DESCRIPTION
ARGUMENTS
RESULTS

ERRORS

5.26 Transaction-req()

NAME **Transaction-req** — request a gateway control transaction.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_transaction_req(
        OM_private_object   session,
        OM_private_object   context,
        OM_object           argument,
        OM_private_object   *result_return,
        OM_sint32           *transaction_id-return
);
```

DESCRIPTION

This function is sued to request a gateway control transaction.

ARGUMENTS

Session (Object(Session))

The gateway control session against which this transaction is issued. This must be a private object previously returned from *Bind()*.

Context (Object(Context))

The gateway control context to be used for this transaction (not to be confused with a "Context" in the MEGACO/H.248 protocol). This argument must be a private object or the constant **Default-Context** ('`GCP_DEFAULT_CONTEXT`').

The transaction may be requested in a confirmed mode or a non-confirmed mode. In confirmed mode, a reply is expected.

Argument (Object(Transaction-Request))

The information supplied as the argument of a gateway control transaction request is an instance of a subclass of the OM class **Transaction-Request**. Normally, concrete subclasses of this class are defined in Gateway Control packages. For example, the **H248-Transaction-Request** subclass of **Transaction-Request** is defined in the H.248 package (see Section 6.7.32 [H248-Transaction-Request], page 127) and may be used as an argument to this function.

RESULTS

Status (Status)

If the function is called synchronously, the values **success** indicated that the action was completed. If called asynchronously, it indicates that the operation was initiated.

Result (Object(*))

Upon successful completion of a synchronous call, when the transaction was requested in a confirmed mode, the result is one of the following:

- When the transaction is requested in a non-confirmed mode, no results are expected and the constant **Absent-Object** ('`GCP_ABSENT_OBJECT`') is returned as the result.

- When a confirmed mode transaction is requested, this is indicated by an instance of the OM class **Transaction-Response** or **Transaction-Error**, or when multiple replies are provided, an instance of OM class **Multiple-Reply**, that contains a set of instances of the OM class **Service-Linked-Reply**, which contains a set of instances of the OM class **Transaction-Segmented-Reply**. Each **Transaction-Linked-Reply-Argument** contains exactly one of the following OM attributes:
  - transaction-Result
  - transaction-Error
  - processing-Failure

ERRORS

5.27 Transaction-rsp()

NAME **Transaction-rsp** — reply to a requested gateway control transaction.

SYNOPSIS

DESCRIPTION
ARGUMENTS
RESULTS

ERRORS

5.28 Unbind()

NAME **Unbind** — unbind from a gateway control session.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_unbind(
        OM_private_object   session
);
```

DESCRIPTION

This function terminates the given gateway control session, and makes the argument unavailable for use with other interface functions (except *Bind()*).

Note that this means the results of any outstanding asynchronous operations that were initiated using the given **Session** can no longer be received. Any such operations may be terminated prematurely. For this reason it is recommended that all outstanding asynchronous operations are processed using *Receive()* before *Unbind()* is called.

The unbound session may be used again as an argument to *Bind()* possibly after modification by the XOM functions (reference **XOM**). When it is no longer required, it must be deleted using the XOM functions.

The **Library-Error** "session-terminated" will be returned as the error value to a synchronous function call using a terminated session.

ARGUMENTS

Session (Object(Session))

The gateway control session that is to be unbound. This must be a private object previously returned by *Bind()*. The value of the **File-Descriptor** OM attribute will be **No-Valid-File-Descriptor** ('`GCP_NO_VALID_FILE_DESCRIPTOR`') if the function succeeds. The other OM attributes will be unchanged.

RESULTS

Status (Status)

Takes the value **success** if **Session** was unbound, and take an error value if not.

ERRORS This function can return a **System-Error** or one of the following **Library-Errors**: bad-class, bad-session, miscellaneous, session-terminated.

This function does not return a **Communications-Error** or any gateway control errors.

This function can return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` or `[GCP_INSUFFICIENT_RESOURCES]`.

SEE ALSO *Bind()*.[1]

[1] See Section 5.8 [Bind()], page 49.

5.29 Validate-object()

NAME **Validate-object** — analyze OM-Object and return Bad-Argument details if necessary.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_validate_obejct(
        OM_workspace        workspace,
        OM_object           test_object,
        OM_private_object  *bad_argument_return
);
```

DESCRIPTION

This function is used to analyze any OM-Object to validate its structure. It may be used as a debug tool prior to issuing other XGCP function calls. It may also be used after XGCP function calls that return `[GCP_E_BAD_ARGUMENT]`.

It is not the intention of this function to be able to validate all OM objects in all packages. Its purpose is to validate only those primary OM objects that may be validly passed as arguments to function defined in this specification.

This function is not intended to validate private objects, only client or service generated public objects. When passed a private object, provided that the private object is of a known OM class, success will be returned. That is, the function does not analyze the contents of a private object.

ARGUMENTS

Test-Object (Object(*))
: The OM-Object to analyze and validate.

Workspace (Workspace)
: Specifies the workspace (obtained from a call to *Initialize()*, in which **Bad-Argument** OM object will be created if the return status is `[GCP_E_BAD_ARGUMENT]`. Test-Objects does not need to be from this workspace.

RESULTS

Status (Status)
: Indicates whether or not the validation was successful. A value of `[GCP_E_BAD_ARGUMENT]` indicates a validation failure and problem details are in the **Bad-Argument** parameter.

Bad-Argument (Object(Bad-Argument))
: When Status is `[GCP_E_BAD_ARGUMENT]`, the result is one instance of the OM class **Bad-Argument**.

ERRORS This function can return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INSUFFICIENT_RESOURCES]`, `[GCP_E_SYSTEM]` or `[GCP_E_BAD_ARGUMENT]`.

EXAMPLE Following is a 'C' binding example of validating a user-generated public object from the GSM MAP Short Message services package:

```
OM_object argument = {
```

```
    OM_OID_DESC(OM_CLASS, GCP_C_MO_FORWARD_SM_ARG)
    , {GCP_SM_RP_DA, OM_S_OBJECT, {
        OM_OID_DESC(OM_CLASS, GCP_C_SM_RP_DA)
        , {GCP_IMSI, GCP_S_TBCD_STRING, OM_STRING("\x21\x43\x65\x87")}
        , OM_NULL_DESCRIPTOR
      } }
    , {GCP_SM_RP_OA, OM_S_OBJECT, {
        OM_OID_DESC(OM_CLASS, GCP_C_SM_RP_OA)
        , {GCP_NO_SM_RP_OA, OM_S_NULL, NULL}
        , OM_NULL_DESCRIPTOR
      } }
    , {GCP_SM_RP_UI, OM_S_OCTET_STRING, (OM_string) {data_length, data_pointer} }
    , {GCP_IMSI, GCP_S_TBCD_STRING, OM_STRING("\x21\x43\x65\x87")}
    , OM_NULL_DESCRIPTOR
};
GCP_status status;
OM_private_object bad_argument;

status = gcp_validate_object(workspace, argument, &bad_argument);

if (status == GCP_E_BAD_ARGUMENT)
    exit(1);
```

5.30 Wait()

NAME **Wait** — wait for the availability of manaement message(s) from one or more bound Sessions.

SYNOPSIS

```
#include <xom.h>
#include <xgcp.h>

GCP_status gcp_wait(
        GCP_waiting_sessions    bound_session_list[],
        OM_workspace            workspace,
        OM_uint32               timeout
);
```

DESCRIPTION

This function is used to suspend the caller until a gateway control operation is available for a bound Session. A timeout value specifies the maximum number of milliseconds to suspend before returning when no messages are available. It should be noted that, in a multithreaded environment, *Wait()* may report the presence of a message that will have been processed by another thread by the time the first thread call *Receive()* to process it.

ARGUMENTS

Bound_session_list (Bound-Session-List)

An ordered sequence of gateway control sessions to wait upon. The last value must evaluate to NULL.

Workspace (Workspace)

Specifies the workspace (obtained from a call to *Initialize()*, in which a `GCP_status` object will be created if the return status is other than `[GCP_SUCCESS]`. Session(s) specified in the **bound-session-list** do no need to be from this workspace.

Timeout (Integer)

The maximum number of milliseconds to suspend before returning when there are no messages from the list of Session(s). A value of zero specifies an indefinite timeout.

RESULTS

Status (Status)

Indicates whether or not the function completed successfully. A successful completion means that either a message is available from a Session or that the timeout limit has been reached. The *Receive()* function must be called to determine whether a message is available. (See note in description above.)

Activated (Boolean-List)

If the function was completed successfully, this result is an ordered list of Boolean values, with the same number of elements as the **bound-session-list**. If true, each value indicates that the corresponding Session has data waiting in queue. If false, each valud indicates that the corresponding Session does *not* has data waiting in queue.

In the C binding, this result is combined with the **bound-session-list** argument as a single array of structures of type `GCP_waiting_sessions`, which is defined as:

```
typedef struct {
        OM_private_object   bound_session;
        OM_boolean          activated;
} GCP_waiting_sessions;
```

ERRORS This function can return one of the following **Library-Errors**: bad-address, bad-session, bad-workspace, miscellaneous, session-terminated.

The function can also return the error constants `[GCP_NO_WORKSPACE]`, `[GCP_INVALID_SESSION]` and `[GCP_INSUFFICIENT_RESOURCE]`.

SEE ALSO *Initialize()*,[1] *Receive()*,[2] *Bind()*.[3]

EXAMPLE Following is a '`C`' language binding example of using the *Wait()* function in conjunction with the *Receive()* function to process indications or confirmations:

```
GCP_waiting_sessions bound_session_list[] =
    { {session, OM_FALSE} , {NULL,} };

for (;;) {
    if ((status = gcp_wait(bound_session_list, workspace, 0)) != GCP_SUCCESS)
        exit(1);
    if (bound_session_list[0].activated) {
        OM_sint mode;
        OM_sint primitive;
        OM_sint completion_flag;
        GCP_status operation_status;
        OM_private_object result;
        OM_sint32 received_invoke_id;

        if ((status = gcp_receive(session, &mode, &primitive, &operation_status,
                          &result, &received_invoke_id)) != GCP_SUCCESS)
                exit(1);
        /* process received result */
        om_delete(result);
    }
}
```

[1] See Section 5.13 [Initialize()], page 56.
[2] See Section 5.17 [Receive()], page 65.
[3] See Section 5.8 [Bind()], page 49.

6 Interface Class Definitions

This chapter defines, in alphabetical order, the OM classes that constitute the Common GCP package (COMMON), the MGCP package (MGCP), the MEGACO package (MEGACO), and the H.248 package (H248). The common errors are defined in the Common GCP package, while the variant specific errors are defined in the MGCP, MEGACO and H.248 packages.

The Object-Identifier associated with the Common GCP package is:

> { iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) common(1) }

This Object-Identifier is represented by the constant **Common-Package** (‘`GCP_COMMON_PKG`’).

The Object-Identifier associated with the MGCP package is:

> { iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) mgcp(2) }

This Object-Identifier is represented by the constant **MGCP-Package** (‘`GCP_MGCP_PKG`’).

The Object-Identifier associated with the MEGACO package is:

> { iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) megaco(3) }

This Object-Identifier is represented by the constant **MEGACO-Package** (‘`GCP_MEGACO_PKG`’).

The Object-Identifier associated with the H.248 package is:

> { iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591) xom-packages(1) xgcp(2) h248(4) }

This Object-Identifier is represented by the constant **H248-Package** (‘`GCP_H248_PKG`’).

The XGCP API may also make use of Gateway Control Services packages. These optional packages define OM classes that are additional to those in the Gateway Control packages, to extend the capabilities of the interface.

The concepts of OSI-Abstract-Data Manipulation are briefly described in Section 1.5 [Relationship to Data Abstraction Services], page 10. The notation is introduced below. Both are fully explained in the XOM Specification (see reference XOM).

Each OM class is described in a separate section, which identifies the OM attributes specific to that OM class. The OM classes are listed in alphabetic order; the OM attributes for each OM class are listed in the order in which they occur in corresponding ASN.1 definitions. The OM attributes that may be found in an instance of an OM class are those OM attributes specific to that OM class and those inherited from each of its super-classes. The OM class-specific OM attributes are defined in a table. The table gives the name of each OM attribute, the syntax of each of its values, any restrictions upon the length (in bits, octets (bytes), or characters) of each value, any restrictions upon the number of values, and the value, if any, the *OM-Create()* function supplies.

Vendor Extensions

Vendors may provide additional OM attributes in their implementation of particular OM classes and their individual documentation will give details of the specification and usage of these. Extensions must be negotiated through use of the *Negotiate()* function.

All such OM attributes have default values which lead to the behaviour described in this specification.

6.1 Global Call Hierarchy

This section depicts the hierarchical organization of the OM classes defined in this chapter, and thus shows which OM classes inherit additional OM attributes from their super-classes. Sub-classification

is indicated by indentation, and the names of abstract OM classes are rendered in italics. Thus, for example, the concrete class **SCTP-Address** is an immediate subclass of the abstract class *Address* which in turn is an immediate subclass of the abstract class *Object*. The *Create()* function applies to all concrete OM classes.

The application is not permitted to create or modify instances of some OM classes, because these OM classes are only returned by the interface and never supplied to it (for example, some subclasses of *Error*).

- *Object* (defined in the XOM Specification: see reference **XOM**)

6.1.1 Interface Common Objects

- *Object* (defined in the XOM Specification: see reference **XOM**)

6.1.2 Interface Common Error Definitions

- *Object* (defined in the XOM Specification: see reference **XOM**)

6.1.3 MGCP Package Objects

- *Object* (defined in the XOM Specification: see reference **XOM**)

6.1.4 MEGACO Package Objects

- *Object* (defined in the XOM Specification: see reference **XOM**)

6.1.5 H248 Package Objects

- *Object* (defined in the XOM Specification: see reference **XOM**)

6.2 Common GCP Package

The Common GCP package introduces some additional OM syntaxes that are derivations of the *String(Octet)* syntax. These additional OM syntaxes are used to represent digit strings for telephony numbers and SCCP addresses.

6.2.1 Basic GCP Types

6.2.1.1 Auth-Data

This OM class contains the following class-specific OM attributes:

auth-Data A *String(Octet)* of 12 to 32 octets in length.

6.2.1.2 Authentication-Header

This OM class contains the following class-specific OM attributes:

sec-Parm-Index
A **Security-Parm-Index** object.

seg-Num A **Sequence-Num** object.

ad A **Auth-Data** object.

6.2.1.3 Context-ID

This OM class contains the following class-specific OM attributes:

context-ID An integer value between 0 and 4294967295 (inclusive). The following values are significant:

0 The *NULL* context value.

0xFFFFFFFE
The *CHOOSE* context value.

0xFFFFFFFF
The *ALL* context value.

6.2.1.4 Domain-Name

This OM class contains the following class-specific OM attributes:

name An IA5 string containing a properly formated domain name.

port-Number
A optional **Port-Number** object.

6.2.1.5 Error-Code

This OM class contains the following class-specific OM attributes:

error-Code An integer value between 0 and 65535 inclusive.

6.2.1.6 Error-Text

This OM class contains the following class-specific OM attributes:

error-Text An IA5 string containing the human readable error text.

6.2.1.7 IP4-Address

This OM class contains the following class-specific OM attributes:

address A 4-octet octet string containing the IP version 4 address.

port-Number
An optional **Port-Number** object.

6.2.1.8 IP6-Address

This OM class contains the following class-specific OM attributes:

address A 16-octet octet string containing the IP version 6 address.

port-Number
An optional **Port-Number** object.

6.2.1.9 Message-ID

This OM class contains the following class-specific OM attributes:

6.2.1.10 MTP-Address

This OM class contains the following class-specific OM attributes:

6.2.1.11 Path-Name

This OM class contains the following class-specific OM attributes:

path-Name An IA5 string between 1 and 64 octets long (inclusive).

6.2.1.12 Port-Number

This OM class contains the following class-specific OM attributes:

port-Number
An integer between 0 and 65535, inclusive. The value zero (0) has special meaning.

6.2.1.13 Security-Parm-Index

This OM class contains the following class-specific OM attributes:

security-Parm-Index
A 4-octet octet string.

6.2.1.14 Sequence-Num

This OM class contains the following class-specific OM attributes:

sequence-Num
A 4-octet octet string.

6.2.2 Abort-Argument

An instance of abstract OM class *Abort-Argument* represents the base information passed as the *Argument* argument to the *Abort()* function (see Section 5.3 [Abort()], page 40) or returned from the *Receive()* function (see Section 5.17 [Receive()], page 65) in the *Result-Or-Argument* result for a user abort indication.
This OM class is an abstract class with several defined concrete subclasses: **MGCP-Abort-Argument**, **MEGACO-Abort-Argument** and **H248-Abort-Argument**. An instance of this OM class has the OM attributes of its super-classes, *Object* and *Error*. It does not define any attributes of its own. Because abort causes are Gateway Control package specific, this abstract OM class does not define any of the values for the *problem* or *parameter* attributes of the *Error* super-class.

6.2.3 Abort-Result

An instance of abstract OM class *Abort-Result* represents the base information passed as the *Argument* argument to the *Abort-req()* function (see Section 5.4 [Abort-req()], page 42) or returned from the *Receive()* function (see Section 5.17 [Receive()], page 65) in the *Result-Or-Argument* result for an association abort indication.
This OM class is an abstract class with several defined concrete subclasses: **MGCP-Abort-Result**, **MEGACO-Abort-Result** and **H248-Abort-Result**. An instance of this OM class has the OM attributes of its super-classes: *Object* and *Error*. It does not define any attributes of its own. Because abort result error codes are Gateway Control package specific, this abstract class does not define any of the *problem* or *parameter* syntaxes of the *Error* super-class.

6.2.4 Accept-Result

An instance of abstract OM class *Accept-Result* represents the base information passed as the *Response* argument to the *Accept()* function (see Section 5.5 [Accept()], page 43) or returned from the *Receive()* function (see Section 5.17 [Receive()], page 65) in the *Result-Or-Argument* result for

an accept indication. This OM class is an abstract class with several defined concrete subclasses: **MGCP-Accept-Result**, **MEGAGO-Accept-Result** and **H248-Accept-Result**.

An instance of this OM class has the OM attributes of it super-classes, *Object*, and additionally the OM attributes listed in Table 6.1.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
session	Object(Session)	—	1	—

Table 6.1: *OM Attributes of OM class Accept-Result*

This OM class contains the following class-specific OM attributes:

session Used to return to the user a fully formed dialog session object. This object may be used for all future interface calls pertaining to this new dialog.

When this object is received as a *Result-Or-Argument* result in a call to *Receive()* (see Section 5.17 [Receive()], page 65), the *Get-Assoc-Info()* interface function can be used to retrieve additional details concerning the association. See Section 5.11 [Get-Assoc-Info()], page 53.

6.2.5 Acse-Args

An instance of OM class **Acse-Args** identifies ACSE specific arguments that will be used during association establishment to a remote peer entity. These can be specified in the **Session** object if *Automatic Association Management (AAM)* is enabled, or in the **Assoc-Argument** or **Assoc-Result** object if *AAM* is disabled. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.2.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
responder-Address	Object(Address)	—	0-1	—
responder-Title	Object(Title)	—	0-1	—
application-Context	Integer or String(Object-Identifier)	—	0-1	—
security-Context	Integer or String(Object-Identifier)	—	0-1	—
authentication-Information	Object(Authentication-Information)	—	0-1	—

Table 6.2: *OM Attributes of OM class* **Acse-Args**

This OM class contains the following class-specific OM attributes:

responder-Address
Indicates the address of the responding program named by *responder-Title*.

responder-Title
Indicates the name of the program which will be used to service gateway control requests. It may be a distinguished name (instance of OM class Form1) or a registered name (instance of OM class Form2) which is used in name/network address resolution phased to get the application process title, the application entity qualifier and the presentation address. It may also be the Entity-Name of the responding program.

application-Context
: The application context name proposed by the user to be used on the gateway control association. When automatic dialog handling is enabled on the session, the application context will be selected by the Gateway Control Service provider consistent with the Gateway Control Service request or indication. Gateway Control Service packages define a set of security contexts that are applicable to the service package.

security-Context
: The security context name proposed by the user to be used on the gateway control association. By default the security context is not present. Gateway Control Service packages define a set of security contexts that are applicable to the service package, defined as integers (Form 1 local values) or Object-Identifiers (Form 2 global values), for use as the value of this object. For example, for H.248 Version 3 there are four possibilities:

- ESP header in accordance with Section 5 of RFC 2406, for IP version 6.
- AH header in accordance with Section 5 of RFC 2402, for IP version 6.
- ESP or AH header as above, but tunnelling IP version 6 over IP version 4, or vise verse.
- The interim AH scheme described in ITU-T Recommendation H.248.1 (09/2005) Clause 10.2.

authentication-Information
: Provides authentication information for use with the security context (if any).

Note that, although this OM attribute is named after ACSE (Association Control Service Execution), which is an OSI layer, none of MGCP, MEGACO nor H.248 provide an ACSE layer. Nevertheless, MGCP, MEGACO and ACSE support the concept of an association and requires equivalent arguments as would the OSI or SS7 ACSE layer.

6.2.6 Action

An instance of OM class *Action* represents an action, that is a sequence of commands, pending notifications or responses that are applied to a gateway control context. This OM class is an abstract class containing several defined subclasses: **Action-Request** and **Action-Reply**. An instance of this OM class has the OM attributes of its super-classes, **Object**, and additionally the OM attributes listed in Table 6.3.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
context	Object(Gcp-Context)	—	0-1	—

Table 6.3: *OM Attributes of OM class* **Action**

This OM class contains the following class-specific OM attributes:

context
: Provides the Gateway Control Service context, which is the context as defined by the Standards, rather than the Context supplied as an argument to most XGCP interface functions.

6.2.7 Action-Request

An instance of OM class *Action-Request* represents an action that is a sequence of commands that are applied to a gateway control context. This OM class is an abstract class containing several defined classes: **MGCP-Action-Request**, **MEGACO-Action-Request** and **H248-Action-Request**. An instance of this OM class has the OM attributes of its super-classes, **Object** and **Action**, and additionally the OM attributes listed in Table 6.4.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
context-Id	Object(Context-ID)	—	1	—
context-Request	Object(Context-Request)	—	0-1	—
context-Attr-Audit-Req	Object(Context-Attr-Audit-Req)	—	0-1	—
command-Requests	Object(Command-Request)	—	1-more	—

Table 6.4: *OM Attributes of OM class* **Action-Request**

This OM class contains the following class-specific OM attributes:

context-Id A **Context-ID** object providing the MEGACO context identifier to which the commands in this action apply.

context-Request
An optional **Context-Request** object specifying some of the attributes requested to be assigned to a context or set of contexts.

context-Attr-Audit-Req
An optional **Context-Attr-Audit-Req** object that, when present, requests the audit of some of the characteristics associated with the context or a selected set of contexts.

command-Requests
One or more commands of OM class **Command-Request** or derived classes, that are applicable to the context of the action.

6.2.8 Context-Request

This OM class contains the following class-specific OM attributes:

priority An optional call priority.

emergency An optional boolean value indicating an emergency call.

topology-Req
Zero or more **Topology-Request** objects describing the topology of the context.

ieps-Call-Ind
An optional boolean value indicating an IEPS call.

context-Prop
Zero or more **Property-Parm** object providing property parameters for the context.

context-List
Zero or more **Context-ID** objects representing a list of contexts.

6.2.9 Context-Attr-Audit-Request

An instance of OM class **Context-Attr-Audit-Request** represents a request to audit attributes of a context or set of contexts. This OM class contains the attributes of its super-classes, *Object*, plus the additional OM attributes listed in Table 6.5.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
topology	Boolean	—	1	—
emergency	Boolean	—	1	—
priority	Boolean	—	1	—
ieps-Call-Ind	Boolean	—	1	—
context-Prop-Aud	Object(Ind-Aud-Property-Parm)	—	0-more	—
select-Priority	Object(Priority)	—	0-1	—
select-Emegency	Object(Emergency)	—	0-1	—
select-Ieps-Call-Ind	Object(Ieps-Call-Ind)	—	0-1	—
select-Logic	Enum(Select-Logic)	—	0-1	—

Table 6.5: *OM Attributes of OM class* **Context-Attr-Audit-Request**

This OM class contains the following class-specific OM attributes:

topology An boolean value that, when TRUE, specifies that the topology of the context is requested.

emergency An boolean value that, when TRUE, specifies that the emergency status of the context is requested.

priority An boolean value that, when TRUE, specifies that the priority of the context is requested.

ieps-Call-Ind
An boolean value that, when TRUE, specifies that the IEPS call indicator of the context is requested.

context-Prop-Aud
Zero or more **Ind-Aud-Property-Parm** objects, each specifying that the context property is requested.

select-Priority
An optional **Priority** object that specifies which value of priority to audit.

select-Emergency
An optional **Emergency** object that specifies which emergency settings to audit.

select-Ieps-Call-Ind
An optional **Ieps-Call-Ind** object that specifies which IEPS call indicators to audit.

select-Logic
An enumerated value specifying what logical operation to perform across selections. The value can be one of the following:

and-audit-select
The audit selection items must all be met for the context to be selected for audit.

or-audit-select
: Only one audit selection item must be met for the context to be selected for audit.

When not specified, the default is **and-audit-select**.

6.2.10 Action-Reply

An instance of OM class *Action-Reply* represents the response to an action that is a sequence of responses that result from commands applied to a gateway control context. This OM class is an abstract class containing several defined classes: **MGCP-Action-Reply**, **MEGACO-Action-Reply** and **H248-Action-Reply**. An instance of this OM class has the OM attributes of its super-classes, **Object** and **Action**, and additionally the OM attributes listed in Table 6.6.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
response-List	Object(Response)	—	1-more	—

Table 6.6: *OM Attributes of OM class* **Action-Reply**

This OM class contains the following class-specific OM attributes:

context-Id An object of OM class **Context-ID** that specifies the primary context to which this action reply corresponds.

error-Descriptor
: An optional object of OM class **Error-Descriptor** that is used to describe an error situation.

context-Reply
: An optional object of OM class **Context-Request** that is used to return requested audit properties of the context.

command-Reply
: One or more command replies of OM class **Command-Reply** or derived classes, that are applicable to the context of the action.

6.2.11 Address

An instance of OM class Address represents the address of a particular gateway control entity. It is used to define the specific location to contact a particular gateway control entity. An instance of this OM class has all of the attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.7.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
internet-Address	String(Internet-Address)	4-16	1	—

Table 6.7: *OM Attributed of OM class* **Address**

This OM class contains the following class-specific OM attributes:

internet-Address

The address is represented using an octet string that contains the encoding of an IP version 4 or IP version 6 address with the following considerations:

- The encoded address *shall* have both an IP address and port number.
- The encoded address *shall* not have a host name in the address.
- The encoded address will distinguish between IP version 4 format and IP version 6 format using the address family indicator.

6.2.12 Application-Context-List

An instance of OM class **Application-Context-List** represents a list of application context names. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.8.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
application-Context-Name	Integer or String(Object-Identifier)	—	1-more	—

Table 6.8: *OM Attributes of OM class* **Application-Context-List**

This OM class contains the following class-specific OM attributes:

application-Context-Name

This attribute is one or more instances of an application context name consisting of an integer syntax or an object-identifier syntax. The integer syntax is only applicable to some Gateway Control packages (e.g. the H.248 service package).

6.2.13 Assoc-Argument

An instance of OM class **Assoc-Argument** is the information supplied as an argument of an ACSE Association operation to be performed. This object is passed to *Assoc-req()* or returned from *Receive()*. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.9.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
presentation-Layer-Args	Object(Presentation-Layer-Args)	—	0-1	—
acse-Args	Object(Acse-args)	—	0-1	—
gcp-Assoc-Args	Object(GCP-Assoc-Args)	—	0-1	—
session	Object(Session)	—	0-1	—

Table 6.9: *OM Attributes of OM class* **Assoc-Argument**

This OM class contains the following class-specific OM attributes:

presentation-Layer-Args

Indicates any presentation layer arguments needed during association establishment. If none are supplied, the Gateway Control Service provider will supply defaults if required. Presentation layer arguments are not required for MGCP. For MEGACO and H.248; however, the presentation layer arguments specify whether the ASN.1 BER

binary encoding of ITU-T Recommendation H.248.1 (09/2005) Annex A or the text encoding of Annex B is used on the association.

acse-Args Indicates ACSE related arguments that will be used during association establishment. If none are supplied, the Gateway Control Service provider will supply defaults if required.

gcp-Assoc-Args
Indicates GCP related arguments that will be used during association establishment. If none are supplied, the Gateway Control Service provider will supply defaults if required.

session Used to return an associated **Session** object. This attribute must be empty when *Assoc-req()* is called, and is supplied as a **Result** parameter by the Gateway Control Service provider upon return from that call. When the object is returned from *Receive()*, this attribute is a partially associated **Session** object. Partially associated session objects can be aborted by calling *Abort-req()* using the new partially associated session object. Partially associated session objects can become fully associated by calling *Assoc-rsp()*. To accept or refuse the new association, call *Assoc-rsp()* using the new partially associated session object that was returned by *Receive()* for the association indication.

6.2.14 Assoc-Result

An instance of OM class **Assoc-Result** is the information supplied as a response of an ACSE Associate operation to be performed. This object is passed to *Assoc-rsp()* or returned from *Receive()*. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.10.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
presentation-Layer-Args	Object(Presentation-Layer-Args)	—	0-1	—
acse-Args	Object(Acse-args)	—	0-1	—
gcp-Assoc-Args	Object(GCP-Assoc-Args)	—	0-1	—
session	Object(Session)	—	0-1	—

Table 6.10: *OM Attributes of OM class* **Assoc-Result**

This OM class contains the following class-specific OM attributes:

presentation-Layer-Args
Indicates any presentation layer arguments needed during association establishment. If none are supplied, the Gateway Control Service provider will supply defaults when required. Presentation layer arguments are not required for MGCP. For MEGACO and H.248; however, the presentation layer arguments specify whether the ASN.1 BER binary encoding of Annex B or the text encoding of Annex C is used on the association.

acse-Args Indicates ACSE related arguments that will be used during association establishment. If none are supplied, the Gateway Control Service provider will supply defaults when required.

gcp-Assoc-Args
Indicates GCP related arguments that will be used during association establishment. If none are supplied, the Gateway Control Service provider will supply defaults when required.

session Used to return to the user an associated session object. This object may be used for all future interface calls pertaining to this new association.

6.2.15 Close-Argument

An instance of abstract OM class *Close-Argument* represents the base information that is supplied as the *Argument* argument to the *Close()* interface function (see Section 5.9 [Close()], page 51) or is returned by the *Receive()* function (see Section 5.17 [Receive()], page 65) in the *Result-Or-Argument* result for a close indication.

This OM class is an abstract class with several defined concrete subclasses: **MGCP-Close-Argument**, **MEGACO-Close-Argument** and **H248-Close-Argument**. An instance of this abstract OM class has the OM attributes of its super-classes, *Object*. It does not define any OM attributes of its own.

6.2.16 Command-Request

An instance of OM class *Command-Request* represents a command issued for gateway control. This OM class is an abstract class containing a number of defined classes: **MGCP-Command-Request**, **MEGACO-Command-Request** and **H248-Command-Request**. An instance of this OM class has the OM attributes of its super-classes, **Object**. This OM class does not have any defined class-specific attributes.

This OM class contains the following class-specific OM attributes:

command

optional

wildcard-Return

6.2.17 Context

An instance of OM class **Context** comprises per operation arguments that are accepted by most of the interface functions. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.11.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Common Arguments				
extensions	Object(Extension)	—	0-more	—
Service Controls				
mode	Enum(Mode)	—	0-1	—
application-Context	Integer or String(Object-Identifier)	—	0-1	—
responder-Address	Object(Address)	—	0-1	—
responder-Title	Object(Title)	—	0-1	—
Local Controls				
asynchronous	Boolean	—	1	false
reply-Limit	Integer	—	0-1	—
time-Limit	Integer	—	0-1	—

Table 6.11: *OM Attributes of OM class* **Context**

This OM class contains the following class-specific OM attributes:

Common Arguments

extensions

Service Controls

mode

application-Context
responder-Address
responder-Title

Local Controls

asynchronous
reply-Limit

time-Limit

6.2.18 Extension

An instance of OM class **Extension** represents an extension to ASN.1 syntax. This object is coded using the *any* syntax. The particular syntax of the extension is dependent upon the object within which an instance of this extension object is contained. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.12.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
extension	any	—	1	—

Table 6.12: *OM Attributes of OM class* **Extension**

This OM class contains the following class-specific attributes:

extension This attribute is a syntax that represents the ASN.1 encoding of the extension complete with tags.

6.2.19 Generic-Service-Argument

An instance of OM class **Generic-Service-Argument** represents a generic service request independent of a defined Gateway Control Services package. This object is used to support service requests that have not been defined in a Gateway Control Services package, or for a Gateway Control Services package that is not supported by the implementation workspace.
An instance of this OM class has the attributes of its super-classes, *Service-Request*, and additionally the OM attributes listed in Table 6.13.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
confirmation	Boolean	—	1	—
operation	Integer or String(Object-Identifier)	—	1	—
arguments	Object(Encoding)	—	1-more	—

Table 6.13: *OM Attributes of OM class* **Generic-Service-Argument**

This OM class contains the following class-specific OM attributes:

confirmation
: This attribute is a boolean value that specifies whether the operation is performed in the confirmed or unconfirmed mode. When performed in a confirmed mode, a response is expected. When performed in an unconfirmed mode, no response is expected. This attribute determines which operations class will be used for the underlying invoke operation.

operation
: This attribute specifies the ROSE/TCAP operation code associated with the service request as either an integer or object identifier. Whether an integer or object identifier is used depends upon the underlying ROSE/TCAP provider and may differ depending on the Gateway Control package in use.

arguments
: This attribute specifies the parameters of the operation invocation as one or more instances of OM class *Encoding*. This encoding class will be encapsulated in a Parameter Sequence by the underlying ROSE/TCAP provider.

6.2.20 Generic-Service-Result

An instance of OM class **Generic-Service-Result** represents a generic service response independent of a defined Gateway Control Services package. This object is used to support service responses that have not been defined in a Gateway Control Services package, or for a Gateway Control Services package that is not supported by the implementation workspace.

An instance of this OM class has the attributes of its super-classes, *Service-Result*, and additionally the OM attributes listed in Table 6.14.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
operation	Integer or String(Object-Identifier)	—	1	—
response	Object(Encoding)	—	1-more	—

Table 6.14: *OM Attributes of OM class* **Generic-Service-Result**

This OM class contains the following class-specific OM attributes:

operation
: This attribute specifies the ROSE/TCAP operation code associated with the service request as either an integer or an object identifier. Whether an integer or object identifier is used depends upon the underlying ROSE/TCAP provider and may differ depending on the Gateway Control package in use.

response
: This attribute specifies the parameters of the operation invocation as one or more instances of OM class *Encoding*. This encoding class will be encapsulated in a Parameter Sequence by the underlying ROSE/TCAP provider.

6.2.21 Gcp-Assoc-Args

An instance of OM class **Gcp-Assoc-Args** represents the GCP association arguments that are optionally supplied by the **Session**, **Assoc-Argument** or **Assoc-Result** objects and can be interrogated with the *Get-Assoc-Info()* function. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.15.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
application-Context-List	Object(Application-Context-List)	—	0-1	—
version-List	Object(Version-List)	—	0-1	—

Table 6.15: *OM Attributes of OM class* **Gcp-Assoc-Args**

This OM class contains the following class-specific OM attributes:

application-Context-List
version-List

6.2.22 Notice-Result

An instance of abstract OM class *Notice-Result* represents the base information that is returned in the *Result-Or-Argument* result by the *Receive()* function (see Section 5.17 [Receive()], page 65) for a notice indication.
This OM class is an abstract class with several defined concrete subclasses: **MGCP-Notice-Result**, **MEGACO-Notice-Result** and **H248-Notice-Result**. An instance of this OM class has the OM attributes of its super-classes, *Object*. It does not define any OM attributes of its own.

6.2.23 Message

An instance of OM class *Message* represents the base information that is supplied as an argument to the *Send()* function (see Section 5.1 [Send()], page 37) or is returned by the *Receive* function (see Section 5.17 [Receive()], page 65) when Automatic Message Handling is disabled on a **Session**.
This OM class is an abstract class containing several defined subclasses: **MGCP-Message**, **MEGACO-Message** and **H248-Message**.
An instance of this OM class has the OM attributes of its super-classes, **Object**, and additionally the OM attributes listed in Table 6.16.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially

Table 6.16: *OM Attributes of OM class* **Message**

This OM class contains the following class-specific OM attributes:

message-ID

transactions
One ore more transactions of OM class **Transaction** or derived classes.

6.2.24 Open-Argument

An instance of abstract OM class *Open-Argument* represents the base information that is supplied as an argument to the *Open()* function (see Section 5.16 [Open()], page 63) or is returned by the *Receive()* function (see Section 5.17 [Receive()], page 65) for an open indication.
This OM class is an abstract class with several defined concrete subclasses: **MGCP-Open-Argument**, **MEGACO-Open-Argument** and **H248-Open-Argument**. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.17.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
responder-Address	Object(Address)	—	0-1	—
responder-Title	Object(Title)	—	0-1	—
requesting-Address	Object(Address)	—	0-1	—
requesting-Title	Object(Title)	—	0-1	—
session	Object(Session)	—	0-1	—

Table 6.17: *OM Attributes of OM abstract class Open-Argument*

This OM class contains the following class-specific OM attributes:

responder-Address
: Specifies the responder address (the address of the remote gateway control entity) for a bound session object with *Automatic Association Management (AAM)* enabled. This address takes precedence over any *responder-Address* contained in the **Context** or **Session** objects. For an associated session object, the precedence rules are different and the *Open-Argument* object cannot be used to override the *responder-Address* contained in the **Session** object.

responder-Title
: Specifies the responder title. This title takes precedence over any *responder-Title* contained in the **Context** or **Session** objects. When not specified by any object, the Gateway Control Service provider will not include a responder title in the open request.

requesting-Address
: The *requesting-Address* contained in the *Open-Argument* cannot be used to override any requesting address contained in the **Session** object.

requesting-Title
: Specifies the requesting title. This title takes precedence over any *requesting-Title* contained in the **Context** or **Session** objects. When not specified by any object, the Gateway Control Service provider will not include a requesting title.

session
: Used to return a partially formed dialog **Session** object. This attribute must be empty when *Open()* is called, and is supplied as a *Result* parameter by the Gateway Control Service provider upon return from that call. The partially formed dialog session can either be aborted with a call to *Abort()* (see Section 5.3 [Abort()], page 40), or can become a fully formed dialog session with a call to *Accept()* (see Section 5.5 [Accept()], page 43). To abort the partially formed dialog session, call *Abort()* using the new partially formed dialog session object. To accept or refuse the new dialog, call *Accept()* or *Refuse()* using the session object that received the association indication.

6.2.25 Operation-Argument

6.2.26 Operation-Error

6.2.27 Operation-Reject

6.2.28 Operation-Result

6.2.29 P-Abort-Result

An instance of OM abstract class *P-Abort-Result* represents the base information returned in the *Result-Or-Argument* result from the *Receive()* function (see Section 5.17 [Receive()], page 65) for a provider abort indication.

This OM class is an abstract class with several defined concrete subclasses: **MGCP-P-Abort-Result**, **MEGACO-P-Abort-Result** and **H248-Abort-Result**. An instance of this OM class has the OM attributes of its super-classes, *Object* and *Error*. It does not defined any OM attributes of its own.

6.2.30 Presentation-Context

An instance of OM class **Presentation-Context** lists the presentation contexts for association establishment. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.18.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
presentation-Id	Integer	—	1	—
presentation-Abstract	String(Object-Identifier)	—	1	—

Table 6.18: *OM Attributes of OM class* **Presentation-Context**

This OM class contains the following class-specific OM attributes:

presentation-Id
: The integer user identification of the presentation context.

presentation-Abstract
: Identifies the abstract syntax. If no value is supplied, by default the abstract syntax name is provided by the Gateway Control package.

MGCP only supports a single presentation context: text encoding, and in the case of MGCP, no *presentation-Abstract* need be specified. The MGCP services package does, however, define an Object-Identifier used to specify the default text encoding. MEGACO/H.248 (all versions), on the other hand, supports both ITU-T Recommendation H.248.1 (09/2005) Annex A binary encoding (ASN.1 BER), as well as text encoding. The MEGACO/H.248 services packages specify the Object-Identifiers to use with the *presentation-Abstract* OM attribute.

6.2.31 Presentation-Layer-Args

An instance of OM class **Presentation-Layer-Args** identifies the presentation layer arguments that will be used during association establishment. These can be specified in the session object if AAM is enabled, or in the **Assoc-Argument** or **Assoc-Result** object if AAM is disabled. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.19.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
presentation-Context-List	Object(Presentation-Context)	—	0-more	—

Table 6.19: *OM Attributes of OM class* **Presentation-Layer-Args**

This OM class contains the following class-specific OM attributes:

presentation-Context-List
: The list of presentation contexts.

MGCP only supports a single presentation context: text encoding, and in the case of MGCP, no presentation-Layer-Args need be specified. On the other hand, MEGACO and H.248 (all versions) may support ITU-T Recommendation H.248.1 (09/2005) Annex A binary encoding (ASN.1 BER), or Annex B text encoding, or both.

When used on an "invoker-role" session (MGC) and the transport address port number is not specified in the *requester-Address* or *requester-Title*, the *presentation-Context-List* specifies which well-known ports on which the invoker will listen for remote Gateway Control Service indications. There are two well-known ports defined: one for binary encoding and another for text encoding.

6.2.32 Refuse-Result

An instance of abstract OM class *Refuse-Result* represents the base information that is supplied as the *Response* argument to the *Refuse()* function (see Section 5.18 [Refuse()], page 68) or is returned by the *Receive()* function (see Section 5.17 [Receive()], page 65) in the *Result-Or-Argument* result for a refuse indication.

This OM class is an abstract class with several defined concrete subclasses: **MGCP-Refuse-Result**, **MEGACO-Refuse-Result** and **H248-Refuse-Result**. An instance of this OM class has the OM attributes of its super-classes, *Object*. It does not define any OM attributes of its own.

6.2.33 Release-Argument

6.2.34 Release-Result

6.2.35 Command-Reply

An instance of OM class *Command-Reply* represents the response to a gateway control command. This OM class is an abstract class containing a number of defined classes: **MGCP-Command-Reply**, **MEGACO-Command-Reply** and **H248-Command-Reply**. An instance of this OM class has the OM attribuges of its super-classes, **Object**. This OM class does not have any defined class-specific attributes.

6.2.36 Service-Argument

An instance of abstract OM class *Service-Argument* represents the base information that is supplied as an argument to the *Service-req()* function (see Section 5.21 [Service-req()], page 72) or is returned by the *Receive()* function (see Section 5.17 [Receive()], page 65) for a service indication.

This OM class is an abstract class containing several defined subclasses: **MGCP-Service-Argument**, **MEGACO-Service-Argument** and **H248-Service-Argument**. An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.20.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
responder-Address	Object(Address)	—	0-1	—
responder-Title	Object(Title)	—	0-1	—
requesting-Address	Object(Address)	—	0-1	—
requesting-Title	Object(Title)	—	0-1	—

Table 6.20: *OM Attributes of OM Class Service-Argument*

This OM class contains the following class-specific OM attributes:

responder-Address
responder-Title
requesting-Address
requesting-Title

6.2.37 Service-Error

An instance of abstract OM class *Service-Error* represents the base information that is supplied in response to a service operation, supplied as the *Argument* argument to the *Service-rsp()* function (see Section 5.22 [Service-rsp()], page 74) or is returned as a *Result-Or-Argument* result by the *Receive()* function (see Section 5.17 [Receive()], page 65) for a service confirmation.

This OM class is an abstract class containing several defined subclasses: **MGCP-Service-Error**, **MEGACO-Service-Error** and **H248-Service-Error**. An instance of this OM class has the OM attributes of its super-classes, *Object* and *Error*. As the *problem* values and *parameter* syntaxes are Gateway Control package specific, this OM class defines no attributes or error values of its own.

6.2.38 Service-Reject

An instance of abstract OM class *Service-Reject* represents the base information that is supplied in response to a service operation, supplied as the *Argument* argument to the *Service-rsp()* function (see Section 5.22 [Service-rsp()], page 74) or is returned as *Result-Or-Argument* result by the *Receive()* function (see Section 5.17 [Receive()], page 65) for a service confirmation.

This OM class is an abstract class containing several defined subclasses: **MGCP-Service-Reject**, **MEGACO-Service-Reject** and **H248-Service-Reject**. An instance of this OM class has the OM attributes of its super-classes, *Object* and *Error*. As the *problem* values and *parameter* syntaxes are Gateway Control package specific, this OM class defines no attributes or error values of its own.

Note that when *Automatic Association Management (AAM)* or *Automatic Dialog Handling (ADH)* are enabled on a session for which a concrete subclass of the *Service-Reject* is used as a response, the *Service-Reject* object also represents the errors that would otherwise be reported using a concrete subclass of *Abort-Argument*, *P-Abort-Result* (were *ADH* to be disabled) or *Refuse-Result* or *Abort-Result* (were *AAM* to be disabled). Therefore, when *ADH* is enabled on a session, the *problem* and *parameter* attributes of the *Error* super-class may also contain any of the error values defined for the corresponding *Abort-Argument* or *P-Abort-Result* subclasses. When *AAM* is enabled on a session, the *problem* and *parameter* attributes of the *Error* super-class may also contain any of the error values defined for the corresponding *Refuse-Result* or *Abort-Result* subclasses.

6.2.39 Service-Result

An instance of abstract OM class *Service-Result* represents the base information that is supplied as an argument to the *Service-rsp()* function (see Section 5.22 [Service-rsp()], page 74) or is returned as a *Result-Or-Argument* result by the *Receive()* function (see Section 5.17 [Receive()], page 65) for a service confirmation.

This OM class is an abstract class containing several defined subclasses: **MGCP-Service-Result**, **MEGACO-Service-Result** and **H248-Service-Result**. An instance of this OM class has the OM attributes of its super-classes, *Object*. As the error codes are Gateway Control package specific, this OM class defines not attributes of its own.

6.2.40 Session

An instance of OM class **Session** identifies a particular communications link from the application program to the Gateway Control Service provider or to a remote gateway control entity depending upon the state of the session object. Session objects can be created with *Automatic Association Management (AAM)* enabled or disabled. If *AAM* is enabled, then all of the ACSE connection management, needed to deliver a gateway control service operation, is done by the Gateway Control Service provider. If *AAM* is disabled, the ACSE connection management is done by the user before issuing or receiving Gateway Control Service operations. The use and rules for the session object are different depending upon whether *AAM* is enabled or disabled. *AAM* is specified as part of *Negotiate()*.[1] An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.21.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
requester-Address	Object(Address)	—	0-1	—
requester-Title	Object(Title)	—	0-1	—
role	Integer	—	0-1	see below
file-Descriptor	Integer	—	1	see below
presentation-Layer-Args	Object(Presentation-Layer-Args)	—	0-1	—
asce-Args	Object(Acse-Args)	—	0-1	—
gcp-Assoc-Args	Object(Gcp-Assoc-Args)	—	0-1	—

Table 6.21: *OM Attributes of OM class* **Session**

For a description of the three types of **Session** object: *AAM* Enabled Session, *AAM* Disabled Session, and Connected Session, see Section 6.2.40 [Session], page 106. The OM attributes of a **Session** are:

requester-Address
: Indicates the address of the requesting program named by *requester-Title*. If both the *requester-Address* and *requester-Title* are not specified, the service provider will supply a default *requester-Address*.

requester-Title
: Indicates the name of the requesting program, user of this session. It may be a distinguished name (instance of OM class **Form1**) or a registered name (instance of OM class **Form2**) which is used in the name/network address resolution phase to get the application process title, the application entity qualifier and the presentation address. It may also be the entity-name of the requesting program.

[1] See Section 5.15 [Negotiate()], page 59.

role Indicates the roles acted by the requester. The role value is specified by OR-ing none, one, or more of the following values:

performer-role
('`GCP_T_PERFORMER_ROLE`') Responder and performer of gateway control service operations. Unless this role is set or implied, no indications will be returned by calls to *Receive()* (only confirmations and notifications) and the Gateway Control Service provider does not need to provide indications to the bound session user. In the terms of the Standards, the performer role is that of the *Media Gateway (MG)*.

invoker-role
('`GCP_T_INVOKER_ROLE`') Requester and invoker of gateway control service operations. If **performer-role** is set and **invoker-role** is clear, the Gateway Control Service provider does not need to honor operation or service requests (other than **Notify-Request**) from the bound session user. In terms of the Standards, the invoker role is that of the *Media Gateway Controller (MGC)*.

When both role flags are clear, or this attribute is not present in the session, the Gateway Control Service provider will still honor operation and service requests. Implementations of this interface are not required to permit the performer and invoker roles simultaneously.

file-Descriptor
Indicates the file descriptor associated with the session. The file descriptor may be used by implementations of the *Wait()* function. The file descriptor may be used in subsequent calls to vendor-specific system facilities to suspend the process (for example, System V `poll(2s)` or BSD `select(2)`. Its use for any other purpose is unspecified. If the implementation does not define any suitable suspension facilities, or if the session is not started, the value is **No-Valid-File-Descriptor** ('`-1`' or '`GCP_NO_VALID_FILE_DESCRIPTOR`').

presentation-Layer-Args
Indicates any presentation layer arguments needed during association establishment. If none are supplied, the Gateway Control Service providers will supply defaults when required.

When the session supports the **invoker-role**, the *presentation-Layer-Args* specifies which encodings the invoker supports, and determines which well-known port numbers upon which the invoker will listen for association indications.

acse-Args Indicates the ACSE related arguments that will be used during association establishment. If none are supplied, the Gateway Control Service provider will supply defaults when required.

gcp-Assoc-Args
Indicates the GCP association related arguments that will be used during association establishment. If none are supplied, the Gateway Control Services provider will supply defaults when required.

6.2.41 Title

An instance of OM class **Title** represents the title of a particular gateway control service entity. It contains various subclass types used to define the specific gateway control process or system name

responsible for a gateway control service entity. An instances of this OM class has all of the attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.22.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
internet-Address	String(Internet-Address)	4-16	1	—

Table 6.22: *OM Attributes of OM class* **Title**

This OM class contains the following class-specific OM attributes:

internet-Address

The title is represented using an octet string that contains the encoding of an internet address, either IP version 4 or IP version 6 address with the following considerations:

- The encoded address *may* have either an IP address or a port number, or both, in the address.
- The encoded address *shall* have a host name in the address (the host name indicated *shall not* be null).
- The encoded address will distinguish between IP version 4 format and IP version 6 format using the address family indicator.

6.2.42 Transaction

An instance of OM class *Transaction* represents the base information that is supplied as an argument to the *Request()*, *Pending()* or *Response()* functions, or is returned by the *Receive()* function (see ⟨undefined⟩ [⟨undefined⟩], page ⟨undefined⟩) when Automatic Message Handling is enabled on a *Session*.

This OM class is an abstract class containing several defined subclasses: **Transaction-Request**, **Transaction-Pending** and **Transaction-Response**.

An instance of this OM class has the OM attributes of its super-classes, *Object*, and additionally the OM attributes listed in Table 6.23.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially

Table 6.23: *OM Attributes of OM class* **Transaction**

This OM class contains the following class-specific OM attributes:

transaction-Id

context-ID

6.2.43 Transaction-Request

An instance of OM class *Transaction-Request* represents the base information that is supplied as an argument to the *Request()* function (see ⟨undefined⟩ [⟨undefined⟩], page ⟨undefined⟩) or is returned by the *Receive()* function (see Section 5.17 [Receive()], page 65) for a transaction indication. This OM class is an abstract class containing several defined subclasses: **MGCP-Transaction-Request**, **MEGACO-Transaction-Request** and **H248-Transaction-Request**. An instance of this OM class has

the OM attributes of its super-classes, *Object* and *Transaction*, and additionally the OM attributes listed in Table 6.24.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
responder-Address	Object(Address)	—	0-1	—
responder-Title	Object(Title)	—	0-1	—
requesting-Address	Object(Address)	—	0-1	—
requesting-Title	Object(Title)	—	0-1	—

Table 6.24: *OM Attributes of OM class* **Transaction-Request**

This OM class contains the following class-specific OM attributes:

requests One or more requests of OM class **Action-Request** or derived classes.

6.2.44 Transaction-Response

An instance of OM class *Transaction-Response* represents the base information that is supplied as an argument to the *Response()* function (see ⟨undefined⟩ [⟨undefined⟩], page ⟨undefined⟩) or is returned by the *Receive()* function (see Section 5.17 [Receive()], page 65) for a transaction confirmation. This OM class is an abstract class containing several defined subclasses: **MGCP-Transaction-Response**, **MEGACO-Transaction-Response** and **H248-Transaction-Response**. An instance of this OM class has the OM attributes of its super-classes, *Object* and *Transaction*, and additionally the OM attributes listed in Table 6.25.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially

Table 6.25: *OM Attributes of OM class* **Transaction-Response**

This OM class contains the following class-specific OM attributes:

responses One or more responses of OM class **Action-Reply** or derived classes.

6.2.45 Transaction-Pending

An instance of OM class *Transaction-Pending* represents the base information that is supplied as an argument to the *Pending()* function (see ⟨undefined⟩ [⟨undefined⟩], page ⟨undefined⟩) or is returned by the *Receive()* function (see ⟨undefined⟩ [⟨undefined⟩], page ⟨undefined⟩) for a transaction indication. This OM class is an abstract class containing several defined subclasses: **MGCP-Transaction-Pending**, **MEGACO-Transaction-Pending** and **H248-Transaction-Pending**. An instance of this OM class has the OM attributes of its super-classes, *Object* and *Transaction*. This OM class does not have any defined class-specific OM attributes.

6.2.46 Version-List

6.2.47 Common GCP Data Objects

6.2.47.1 Audit-Descriptor

6.2.47.2 Digit-Map-Descriptor

6.2.47.3 Error-Descriptor

6.2.47.4 Event-Buffer-Descriptor

6.2.47.5 Events-Descriptor

6.2.47.6 Media-Descriptor

6.2.47.7 Modem-Descriptor

6.2.47.8 Mux-Descriptor

6.2.47.9 Observed-Events-Descriptor

6.2.47.10 Packages-Descriptor

6.2.47.11 Service-Change-Descriptor

6.2.47.12 Signals-Descriptor

6.2.47.13 Statistics-Descriptor

6.2.47.14 Transaction-ID-Or-List

6.3 MGCP Package

6.3.1 MGCP-Action-Request

6.3.2 MGCP-Action-Reply

6.3.3 MGCP-Audit-Connection-Request

6.3.4 MGCP-Audit-Connection-Response

6.3.5 MGCP-Audit-Endpoint-Request

6.3.6 MGCP-Audit-Endpoint-Response

6.3.7 MGCP-Command-Request

mgcp-command-Verb
mgcp-transaction-Id
mgcp-endpoint-Name
mgcp-version

6.3.8 MGCP-Create-Connection-Request

6.3.9 MGCP-Create-Connection-Response

6.3.10 MGCP-Delete-Connection-Request

6.3.11 MGCP-Delete-Connection-Response

6.3.12 MGCP-Endpoint-Configuration-Request

6.3.13 MGCP-Endpoint-Configuration-Response

6.3.14 MGCP-Message

6.3.15 MGCP-Modify-Connection-Request

6.3.16 MGCP-Modify-Connection-Response

6.3.17 MGCP-Notification-Request

6.3.18 MGCP-Notification-Response

6.3.19 MGCP-Notify-Request

6.3.20 MGCP-Notify-Reply

6.3.21 MGCP-Command-Reply

mgcp-response-Code
mgcp-transaction-Id
mgcp-response-String

6.3.22 MGCP-Restart-In-Progress-Request

6.3.23 MGCP-Restart-In-Progress-Response

6.3.24 MGCP-Transaction-Request

6.3.25 MGCP-Transaction-Response

6.4 MGCP Extension Packages

6.5 MEGACO Package

6.5.1 MEGACO-Action-Request

6.5.2 MEGACO-Action-Reply

6.5.3 MEGACO-Add-Request

An instance of OM class **MEGACO-Add-Request** represents the Add Request command of MEGACO (RFC 3705). This OM class contains the attributes of its super-classes: *MEGACO-Amm-Request*. It does not define any OM attributes of its own.

6.5.4 MEGACO-Add-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.26.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.26: *OM Attributes of OM class MEGACO-Add-Reply*

This OM class contains the following class-specific OM attributes:

6.5.5 MEGACO-Audit-Request

An instance of a subclass of this abstract OM class represents either an Audit Capability command request or an Audit Value command request. This is an abstract class that defines several derived subclasses: **MEGACO-Audit-Capability-Request** and **MEGACO-Audit-Value-Request**. This OM class contains the OM attributes of its super-classes: *MEGACO-Command-Request*; plus the additional OM attributes listed in Table 6.27.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
audit-Descriptor	Object(Audit-Descriptor)	—	1	—

Table 6.27: *OM Attributes of OM class MEGACO-Audit-Request*

This OM class contains the following class-specific OM attributes:

audit-Descriptor
: A single instance of OM class **Audit-Descriptor**.

6.5.6 MEGACO-Audit-Reply

6.5.7 MEGACO-Audit-Capability-Request

This OM class has the attributes of its super-classes: *MEGACO-Service-Request*; plus the additional OM attributes listed in Table 6.28.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
audit-Descriptor	Object(Audit-Descriptor)	—	0-1	—

Table 6.28: *OM Attributes of OM class MEGACO-Audit-Capabilities-Command*

This OM class contains the following class-specific OM attributes:

6.5.8 MEGACO-Audit-Capability-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.29.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—

Table 6.29: *OM Attributes of OM class MEGACO-Audit-Capabilities-Response*

This OM class contains the following class-specific OM attributes:

6.5.9 MEGACO-Audit-Value-Request

This OM class has the attributes of its super-classes: *MEGACO-Service-Request*; plus the additional OM attributes listed in Table 6.30.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
audit-Descriptor	Object(Audit-Descriptor)	—	0-1	—

Table 6.30: *OM Attributes of OM class MEGACO-Audit-Value-Request*

This OM class contains the following class-specific OM attributes:

6.5.10 MEGACO-Audit-Value-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.31.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.31: *OM Attributes of OM class MEGACO-Audit-Value-Reply*

This OM class contains the following class-specific OM attributes:

6.5.11 MEGACO-Command-Request

6.5.12 MEGACO-Message

6.5.13 MEGACO-Modify-Request

An instance of OM class **MEGACO-Modify-Request** represents the Modify Request command of MEGACO (RFC 3705). This OM class contains the attributes of its super-classes: *MEGACO-Amm-Request*. It does not define any OM attributes of its own.

6.5.14 MEGACO-Modify-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.32.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.32: *OM Attributes of OM class MEGACO-Modify-Reply*

This OM class contains the following class-specific OM attributes:

6.5.15 MEGACO-Move-Request

An instance of OM class **MEGACO-Move-Request** represents the Move Request command of MEGACO (RFC 3705). This OM class contains the attributes of its super-classes: *MEGACO-Amm-Request*. It does not define any OM attributes of its own.

6.5.16 MEGACO-Move-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.33.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.33: *OM Attributes of OM class MEGACO-Move-Reply*

This OM class contains the following class-specific OM attributes:

6.5.17 MEGACO-Notify-Request

This OM class has the attributes of its super-classes: *MEGACO-Command-Request*; plus the additional OM attributes listed in Table 6.34.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	1	—
error-Descriptor	Object(Error-Descriptor)	—	0-1	—

Table 6.34: *OM Attributes of OM class MEGACO-Notify-Request*

This OM class contains the following class-specific OM attributes:

observed-Events-Descriptor
: One instance of OM class **Observed-Events-Descriptor**.

error-Descriptor
: An optional instance of OM class **Error-Descriptor**.

6.5.18 MEGACO-Notify-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.35.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially

Table 6.35: *OM Attributes of OM class MEGACO-Notify-Reply*

This OM class contains the following class-specific OM attributes:

6.5.19 MEGACO-Command-Reply

6.5.20 MEGACO-Service-Change-Request

This OM class has the attributes of its super-classes: *MEGACO-Service-Request*; plus the additional OM attributes listed in Table 6.36.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
service-Change-Descriptor	Object(Service-Change-Descriptor)	—	1	—

Table 6.36: *OM Attributes of OM class MEGACO-Service-Change-Request*

This OM class contains the following class-specific OM attributes:

6.5.21 MEGACO-Service-Change-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.37.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
service-Change-Descriptor	Object(Service-Change-Descriptor)	—	0-1	—

Table 6.37: *OM Attributes of OM class MEGACO-Service-Change-Reply*

This OM class contains the following class-specific OM attributes:

6.5.22 MEGACO-Subtract-Request

This OM class has the attributes of its super-classes: *MEGACO-Command-Request*; plus the additional OM attributes listed in Table 6.38.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
audit-Descriptor	Object(Audit-Descriptor)	—	0-1	—

Table 6.38: *OM Attributes of OM class MEGACO-Subtract-Request*

This OM class contains the following class-specific OM attributes:

6.5.23 MEGACO-Subtract-Reply

This OM class has the attributes of its super-classes: *MEGACO-Service-Result*; plus the additional OM attributes listed in Table 6.39.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.39: *OM Attributes of OM class MEGACO-Subtract-Reply*

This OM class contains the following class-specific OM attributes:

6.5.24 MEGACO-Transaction-Pending

6.5.25 MEGACO-Transaction-Request

6.5.26 MEGACO-Transaction-Response

6.6 MEGACO Extension Packages

6.7 H.248 Package

The H.248 package provides specialization of the argument and result objects of the Common GCP package to provide for the services under the H.248 GCP as described in ITU-T Recommendation H.248.1. This package is organized as a separate OM package that can be negotiated using the *Negotiate()* function.[2]

The Object-Identifier associated with the H.248 package is:

```
{ iso(1) org(3) dod(6) internet(1) private(4) enterprises(1) openss7(29591)
  xom-packages(1) xgcp(2) h248(4) }
```

This Object-Identifier is represented by the constant **H248-Package** (‘`GCP_H248_PKG`’). This constant can be used to negotiate support for the package using the *Negotiate()* function.

The H.248 GCP package introduces some additional OM syntaxes that are derivations of the *String(Octet)* syntax. These additional OM syntaxes are used to represent digit strings and telephony numbers.

The following outlines the OM Class Hierarchy for the H.248 package:

— *Object* (defined in the XOM Specification: see reference **XOM**)

[2] See Section 5.15 [Negotiate()], page 59.

6.7.1 H248-Amm-Request

This OM class contains the OM attributes of its super-classes, *H248-Command-Request*, plus the additional OM attributes listed in Table 6.40.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
descriptors	Object(Amm-Descriptors)	—	1	—

Table 6.40: *OM Attributes of OM class H248-Amm-Request*

This OM class contains the following class-specific OM attributes:

descriptors An object of OM class **Amm-Descriptors**.

6.7.2 H248-Amm-Descriptors

An instance of the abstract *Amm-Descriptors* OM class represents the descriptors that can be included with an Add, Move or Modify command. This OM class contains the attributes of its super-classes, *Object*, plus the additional attributes listed in Table 6.41.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
audit-Descriptor	Object(Audit-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—

Table 6.41: *OM Attributes of OM class Amm-Descriptors*

This OM class contains the following class-specific OM attributes:

media-Descriptor
An optional instance of OM class **Media-Descriptor**.

modem-Descriptor
An optional instance of OM class **Modem-Descriptor**.

mux-Descriptor
An optional instance of OM class **Mux-Descriptor**.

events-Descriptor
An optional instance of OM class **Events-Descriptor**.

signals-Descriptor
An optional instance of OM class **Signals-Descriptor**.

digit-Map-Descriptor
An optional instance of OM class **Digits-Descriptor**.

audit-Descriptor
: An optional instance of OM class **Audit-Descriptor**.

statistics-Descriptor
: An optional instance of OM class **Statistics-Descriptor**.

6.7.3 H248-Amm-Reply

This OM class contains the following class-specific OM attributes:

6.7.4 H248-Action-Request

This OM class contains the following class-specific OM attributes:

6.7.5 H248-Action-Reply

This OM class contains the following class-specific OM attributes:

6.7.6 H248-Add-Request

An instance of the OM clss **H248-Add-Request** represents the Add Request command of ITU-T Recommendation H.248.1 (09/2005). This OM class has the attributes of its super-classes: *H248-Amm-Request*. It does not define any OM attributes of its own. See Section 6.7.1 [H248-Amm-Request], page 118.

6.7.7 H248-Add-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.42.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.42: *OM Attributes of OM class H248-Add-Reply*

This OM class contains the following class-specific OM attributes:

media-Descriptor
modem-Descriptor
mux-Descriptor
events-Descriptor
signals-Descriptor
digit-Map-Descriptor
observed-Events-Descriptor
event-Buffer-Descriptor
statistics-Descriptor
packages-Descriptor

6.7.8 H248-Audit-Request

An instance of a subclass of this abstract OM class represents either an Audit Capability Request command or an Audit Value Request command. This is an abstract class that defines several derived subclasses: **H248-Audit-Capability-Request** and **H248-Audit-Value-Request**. This OM class contains the OM attributes of its super-classes: *H248-Command-Request*; plus the additional OM attributes listed in Table 6.43.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
audit-Descriptor	Object(Audit-Descriptor)	—	1	—

Table 6.43: *OM Attributes of OM class H248-Audit-Request*

This OM class contains the following class-specific OM attributes:

audit-Descriptor
A single instance of OM class **Audit-Descriptor**.

6.7.9 H248-Audit-Reply

This OM class contains the following class-specific OM attributes:

6.7.10 H248-Audit-Capability-Request

An instance of OM class **H248-Audit-Capability-Request** represents an Audit Capability command request of ITU-T Recommendation H.248.1 (09/2005). This OM class contains the attributes of its super-classes: *H248-Audit-Request*; it does not define any OM attributes of its own.

6.7.11 H248-Audit-Capability-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.44.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—

Table 6.44: *OM Attributes of OM class H248-Audit-Capabilities-Response*

This OM class contains the following class-specific OM attributes:

media-Descriptor
modem-Descriptor
mux-Descriptor
events-Descriptor
signals-Descriptor
observed-Events-Descriptor
event-Buffer-Descriptor
statistics-Descriptor

6.7.12 H248-Audit-Value-Request

An instance of OM class **H248-Audit-Value-Request** represents an Audit Value command request of ITU-T Recommendation H.248.1 (09/2005). This OM class contains the attributes of its super-classes: *H248-Audit-Request*; it does not define any OM attributes of its own.

6.7.13 H248-Audit-Value-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.45.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.45: *OM Attributes of OM class H248-Audit-Value-Reply*

This OM class contains the following class-specific OM attributes:

media-Descriptor
modem-Descriptor
mux-Descriptor
events-Descriptor
signals-Descriptor
digit-Map-Descriptor
observed-Events-Descriptor
event-Buffer-Descriptor
statistics-Descriptor
packages-Descriptor

6.7.14 H248-Command-Request

An instance of a concrete subclass of this abstract OM class represents an ITU-T Recommendation H.248.1 (09/2005) CommandRequest. This abstract OM class defines several abstract and concrete subclasses: *H248-Amm-Request*, **H248-Subtract-Request**, *H248-Audit-Request*, **H248-Notify-Request** and **H248-Service-Change-Request**.
This abstract OM class has the attributes of its super-classes, *Command-Request*, plus the additional OM attributes listed in Table 6.46.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
termination-Id	Object(Termination-ID-List)	—	1	—

Table 6.46: *OM Attributes of OM class H248-Command-Request*

This OM class contains the following class-specific OM attributes:

termination-Id
: An instance of OM class **Termination-ID** that specifies the termination to which the command request applies.

6.7.15 H248-Command-Reply

An instance of a concrete subclass of this abstract OM class represents an ITU-T Recommendation H.248.1 (09/2005) CommandReply. This abstract OM class defines several abstract and concrete subclasses: *H248-Amm-Reply*, *H248-Audit-Reply*, **H248-Notify-Reply** and **H248-Service-Change-Reply**.
This abstract OM class has the attributes of its super-classes, *Command-Reply*, plus the additional OM attributes listed in Table 6.47.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
termination-Id	Object(Termination-ID-List) or	—	1	—

Table 6.47: *OM Attributes of OM class H248-Command-Reply*

This OM class contains the following class-specific OM attributes:

termination-Id
: An instance of OM class **Termination-ID** that specifies the termination to which the command reply applies.

6.7.16 H248-Message

6.7.17 H248-Modify-Request

An instance of OM class **H248-Modify-Request** represents the Modify Request command of ITU-T Recommendation H.248.1 (09/2005). This OM class contains the attributes of its super-classes: *H248-Amm-Request*. It does not define any attributes of its own. See Section 6.7.1 [H248-Amm-Request], page 118.

6.7.18 H248-Modify-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.48.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.48: *OM Attributes of OM class H248-Modify-Reply*

This OM class contains the following class-specific OM attributes:

media-Descriptor
modem-Descriptor
mux-Descriptor
events-Descriptor
signals-Descriptor
digit-Map-Descriptor
observed-Events-Descriptor
event-Buffer-Descriptor
statistics-Descriptor
packages-Descriptor

6.7.19 H248-Move-Request

An instance of OM class **H248-Move-Request** represents the Move Request command of ITU-T Recommendation H.248.1 (09/2005). This OM class contains the attributes of its super-classes: *H248-Amm-Request*. It does not define any attributes of its own. See Section 6.7.1 [H248-Amm-Request], page 118.

6.7.20 H248-Move-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.49.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.49: *OM Attributes of OM class H248-Move-Reply*

This OM class contains the following class-specific OM attributes:

media-Descriptor
modem-Descriptor
mux-Descriptor
events-Descriptor
signals-Descriptor
digit-Map-Descriptor
observed-Events-Descriptor
event-Buffer-Descriptor
statistics-Descriptor
packages-Descriptor

6.7.21 H248-Notify-Request

This OM class has the attributes of its super-classes: *H248-Service-Request*; plus the additional OM attributes listed in Table 6.50.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	1	—
error-Descriptor	Object(Error-Descriptor)	—	0-1	—

Table 6.50: *OM Attributes of OM class H248-Notify-Request*

This OM class contains the following class-specific OM attributes:

observed-Events-Descriptor
error-Descriptor

6.7.22 H248-Notify-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.51.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially

Table 6.51: *OM Attributes of OM class H248-Notify-Reply*

This OM class contains the following class-specific OM attributes:

6.7.23 H248-Service-Change-Request

This OM class has the attributes of its super-classes: *H248-Service-Request*; plus the additional OM attributes listed in Table 6.52.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
service-Change-Descriptor	Object(Service-Change-Descriptor)	—	1	—

Table 6.52: *OM Attributes of OM class H248-Service-Change-Request*

This OM class contains the following class-specific OM attributes:

service-Change-Descriptor

6.7.24 H248-Service-Change-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.53.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
service-Change-Descriptor	Object(Service-Change-Descriptor)	—	0-1	—

Table 6.53: *OM Attributes of OM class H248-Service-Change-Reply*

This OM class contains the following class-specific OM attributes:

service-Change-Descriptor

6.7.25 H248-Service-Request

This OM class contains the following class-specific OM attributes:

termination-Id

6.7.26 H248-Service-Result

This OM class contains the following class-specific OM attributes:

termination-Id

6.7.27 H248-Service-Error

This OM class contains the following class-specific OM attributes:

6.7.28 H248-Service-Reject

This OM class contains the following class-specific OM attributes:

6.7.29 H248-Subtract-Request

This OM class has the attributes of its super-classes: *H248-Command-Request*; plus the additional OM attributes listed in Table 6.54.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
audit-Descriptor	Object(Audit-Descriptor)	—	0-1	—

Table 6.54: *OM Attributes of OM class H248-Subtract-Request*

This OM class contains the following class-specific OM attributes:

audit-Descriptor

6.7.30 H248-Subtract-Reply

This OM class has the attributes of its super-classes: *H248-Service-Result*; plus the additional OM attributes listed in Table 6.55.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
media-Descriptor	Object(Media-Descriptor)	—	0-1	—
modem-Descriptor	Object(Modem-Descriptor)	—	0-1	—
mux-Descriptor	Object(Mux-Descriptor)	—	0-1	—
events-Descriptor	Object(Events-Descriptor)	—	0-1	—
signals-Descriptor	Object(Signals-Descriptor)	—	0-1	—
digit-Map-Descriptor	Object(Digit-Map-Descriptor)	—	0-1	—
observed-Events-Descriptor	Object(Observed-Events-Descriptor)	—	0-1	—
event-Buffer-Descriptor	Object(Event-Buffer-Descriptor)	—	0-1	—
statistics-Descriptor	Object(Statistics-Descriptor)	—	0-1	—
packages-Descriptor	Object(Packages-Descriptor)	—	0-1	—

Table 6.55: *OM Attributes of OM class H248-Subtract-Reply*

This OM class contains the following class-specific OM attributes:

media-Descriptor
modem-Descriptor
mux-Descriptor
events-Descriptor
signals-Descriptor
digit-Map-Descriptor
observed-Events-Descriptor
event-Buffer-Descriptor
statistics-Descriptor
packages-Descriptor

6.7.31 H248-Transaction-Pending

6.7.32 H248-Transaction-Request

6.7.33 H248-Transaction-Response

6.8 H.248 Extension Packages

Commands:

- *Add()*: The add command adds a termination to a context. The *Add()* command on the first termination in a context is used to create a context.
- *Modify()*: The modify command modifies the properties, events and signals of a termination.
- *Subtract()*: The subtract command disconnects a termination from its context and returns statistics on the termination's participation in the context. The *Subtract()* command on the last termination in a context deletes the context.
- *Move()*: The move command atomically moves a termination to another context.
- *AuditValue()*: The audit value command returns the current state of properties, events, signals and statistics of terminations.
- *AuditCapabilities()*: The audit capabilities command all the possible values for termination properties, events and signals allowed by the Media Gateway.
- *Notify()*: The notify command allows the Media Gateway to inform the Media Gateway Controller of the occurrence of events in the Media Gateway.
- *ServiceChange()*: The service change command allows the Media Gateway to notify the Media Gateway Controller that a termination or group of terminations is about to be taken out of service or has just been returned to service. The service change command is also used by the MG to announce its availability to an MGC (registration), and to notify the MGC of impending or completed restart of the MG. The MGC may announce a handover to the MG by sending it a service change command. The MGC may also use the service change command to instruct the MG to take a termination or group of terminations in or out of service.

7 Errors

Appendix A C Headers

Appendix B Examples

Glossary

Licenses

All code presented in this manual is licensed under the [GNU Affero General Public License], page 137. The text of this manual is licensed under the [GNU Free Documentation License], page 147, with no invariant sections, no front-cover texts and no back-cover texts. Please note, however, that it is just plain wrong to modify statements of, or attribute statements to, the Author or *OpenSS7 Corporation*.

GNU Affero General Public License

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU Affero General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the

source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers

where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to

downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

    Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a "Source" link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see http://www.gnu.org/licenses/.

GNU Free Documentation License

GNU FREE DOCUMENTATION LICENSE
Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

   B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

   D. Preserve all the copyright notices of the Document.

   E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

   F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

H

I

L

M

N

O

P

R

S

T

V

W