INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

T.124

(02/98)

SERIES T: TERMINALS FOR TELEMATIC SERVICES

Generic Conference Control

ITU-T Recommendation T.124

(Previously CCITT Recommendation)

ITU-T T-SERIES RECOMMENDATIONS

TERMINALS FOR TELEMATIC SERVICES

For further details, please refer to ITU-T List of Recommendations.

ITU-T RECOMMENDATION T.124

GENERIC CONFERENCE CONTROL

Summary

This Recommendation provides a high-level framework for conference management and control of multimedia terminals and Multipoint Control Units (MCUs). It encompasses Generic Conference Control (GCC) functions such as conference establishment and termination, managing the roster of terminals participating in a conference, managing the roster of applications and application capabilities within a conference, registry services for use by applications, coordination of conference conductorship, as well as other miscellaneous functions. It depends on companion Recommendations T.122 and T.125 (MCS) and T.123 as part of the T.120 infrastructure.

Source

ITU-T Recommendation T.124 was revised by ITU-T Study Group 16 (1997-2000) and was approved under the WTSC Resolution No. 1 procedure on the 6th of February 1998.

FOREWORD

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the ITU. The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

The ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. The ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, the ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

		Page
1	Scope	1
2	Normative References	2
3	Definitions	3
4	Abbreviations	7
5	Conventions	7
6	Overview	8
6.1	System model for a conference node	9
6.2	Conference establishment and termination	10
6.3	The conference roster	11
6.4	The application roster	11
6.5	The Application Registry	11
6.6	Conference conductorship	12
6.7	Miscellaneous functions	12
6.8	Scalable conferences	12
6.9	Summary of GCC abstract services	14
7	GCC service definition	18
7.1	Conference establishment and termination	18
7.1.1	The conference profile	18
7.1.2	Description of abstract services	18
7.1.3	Conference establishment requirements	48
7.1.4	Examples of conference establishment procedures	52
7.2	The conference roster	55
7.2.1	Description of abstract services	56
7.3	The application roster	61
7.3.1	Contents of the application roster	61
7.3.2	Description of the application roster exchange process	64
7.3.3	Description of abstract services	66
7.4	The Application Registry	76
7.4.1	Registry keys	76
7.4.2	Ownership and persistence	76
7.4.3	Dynamic allocation	77
7.4.4	Description of abstract services	77
7.5	Conference conductorship	87
7.5.1	Description of abstract services	88

Page

7.6 Miscellaneous functions 96
7.6.1 Description of abstract services 96
8 GCC Protocol Specification 101
8.1 General operation 101
8.2 Conference establishment and termination 102
8.2.1 Conference creation 102
8.2.2 Querying conferences 108
8.2.3 Joining a conference 110
8.2.4 Inviting a node to a conference 116
8.2.5 Requesting to add a node to a conference 121
8.2.6 Locking a conference 122
8.2.7 Unlocking a conference 124
8.2.8 Disconnecting from a conference 125
8.2.9 Terminating a conference 126
8.2.10 Ejecting a node from a conference 127
8.2.11 Transferring nodes between conferences 130
8.3 The conference and application rosters 131
8.3.1 Original Roster Protocol vs. Scalable Roster Protocol 131
8.3.2 Original Roster Protocol 131
8.3.3 Scalable Roster Protocol 141
8.3.4 Collapsing Application Capabilities Lists 152
8.3.5 Application and conference roster inquiry 153
8.3.6 Remotely invoking an Application Protocol Entity 153
8.4 The Application Registry 154
8.4.1 Registering a channel 155
8.4.2 Assigning a Token 156
8.4.3 Setting a Parameter 157
8.4.4 Retrieving an Entry 159
8.4.5 Deleting an Entry 159
8.4.6 Monitoring an Entry 160
8.4.7 Allocation of Unique Handles 162
8.4.8 Changes in ownership and Registry Clean-up 163
8.5 Conference conductorship 163
8.5.1 Grabbing conductorship 163
8.5.2 Releasing conductorship 164
8.5.3 Conductor Assignment and Release Indications 165
8.5.4 Asking to be given conductorship 166
8.5.5 Giving conductorship 166

Page

8.5.6 Getting conductorship status 167
8.5.7 Conductorship announcement when new nodes enter a conference 167
8.5.8 Unexpected disconnection of the conductor 168
8.5.9 Asking to be given conducted-mode permission 168
8.5.10 Granting conducted-mode permission 169
8.6 Miscellaneous functions 170
8.6.1 Timed conferences 170
8.6.2 Requesting conference assistance 171
8.6.3 Broadcasting a text message 171
8.7 GCCPDU definitions 172
9 Use of the Multipoint Communication Service 194
9.1 MCS services 194
9.2 Channel allocation 195
9.3 Token allocation 196
9.4 Use of MCS data transmission services 196
9.5 Encoding of PDUs in MCS primitives 199
9.6 Format of User Data parameter of MCS-Connect-Provider 199
9.7 Interpretation of the MCS Domain Selector 199
Annex A – Static channel and token ID assignments 200
A.1 Static channel ID assignments 200
A.2 Static token ID assignments 201
Annex B – Object Identifier assignments 201
Annex C – Network Address Parameter – Description and use 201
Appendix I – Relationship of T.120 to H.243 in H.320 conferences 202
I.1 Introduction 202
I.2 Conference selection and Password protection 202
I.2.1 T.124 conference establishment 203
I.2.2 H.243 conference establishment 203
I.3 Alternative Node ID 203

Recommendation T.124

GENERIC CONFERENCE CONTROL

(Geneva, 1995; revised in 1998)

1 Scope

This Recommendation provides a high-level framework for conference management and control of audiographic and audiovisual terminals and Multipoint Control Units (MCUs). It encompasses Generic Conference Control (GCC) functions such as conference establishment and termination, managing the roster of nodes participating in a conference, managing the roster of Application Protocol Entities and Application Capabilities within a conference, registry services for use by Application Protocol Entities, coordination of conference conductorship, as well as other miscellaneous functions.

This Recommendation is defined within the framework of Recommendation T.120. Included within this framework are companion Recommendations T.122 and T.125, which define the multipoint delivery mechanism used in this Recommendation, and Recommendation T.123, which specifies the Audiovisual Protocol Stacks for each of the communication networks supported.

Figure 1-1 presents an overview of the scope of this Recommendation and its relationship to the other elements of the T.120 framework within a single node.

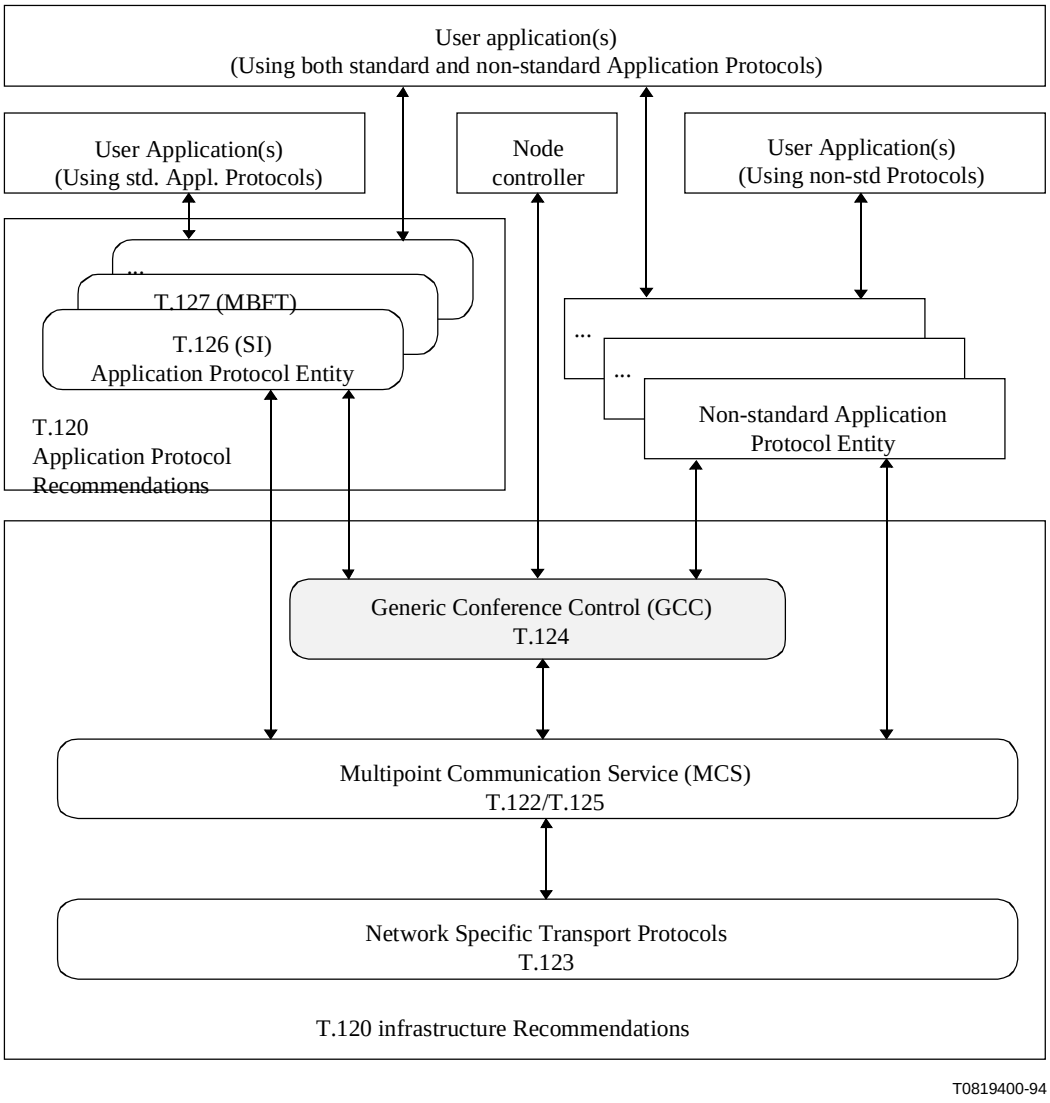

Figure 1-1/T.124 – Scope of T.124

2 Normative References

The following ITU-T Recommendations, and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

– ITU-T Recommendation F.702 (1996), *Multimedia conference services.*

– ITU-T Recommendation H.221 (1997), *Frame structure for a 64 to 1920 kbits/s channel in audiovisual teleservices.*

– CCITT Recommendation T.35 (1991), *Procedure for the allocation of CCITT defined codes for non-standard facilities.*

– ITU-T Recommendation T.120 (1996), *Data protocols for multimedia conferencing.*

– ITU-T Recommendation T.122 (1998), *Multipoint Communication Service – Service definition.*

– ITU-T Recommendation T.123 (1996), *Network specific data protocol stacks for multimedia conferencing.*

– ITU-T Recommendation T.125 (1998), *Multipoint Communication Service Protocol Specification.*

– ITU-T Recommendation T.126 (1997), *Multipoint still image and annotation protocol.*

– ITU-T Recommendation T.127 (1995), *Multipoint binary file transfer protocol.*

– ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

– ITU-T Recommendation X.691 (1997) | ISO/IEC 8825-2:1998, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).*

– ISO/IEC 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane.*

3 Definitions

This Recommendation defines the following terms.

3.1 anonymous node: This node category describes a T.120 node that can participate in a conference while not being visible to any other nodes in the conference. The presence of Anonymous nodes is needed to facilitate large-scale T.120 conferences.

3.2 application protocol: Any standard or non-standard protocol specification which is designed to make use of T.120 services.

3.3 application protocol entity: The instantiation of an Application Protocol in a terminal or MCU. Application Protocol Entities are employed by User Applications, but are not themselves User Applications. An Application Protocol Entity communicates with GCC through the GCC Provider present at its local terminal or MCU via a GCCSAP. Only Application Protocol Entities communicate with GCC Providers; User Applications do not. Multiple Application Protocol Entities based on the same Application Protocol may enroll at a single node. These may be either in the same or separate Application Protocol Sessions. A single Application Protocol Entity is assumed to communicate with the local GCC Provider via a single GCCSAP, and is also assumed to have a single MCS User ID if it has enrolled in the active state. An Application Protocol Entity which has enrolled in the active state is part of a single Application Protocol Session as indicated by its Session ID (or lack of a Session ID which indicates that it is part of the Default Session). An Application Protocol Entity which has enrolled in the inactive state is not considered part of any Application Protocol Session; however, an inactive Application Protocol Entity may make use of GCC services and information associated with such an Application Protocol Entity is included in the Application Roster.

3.4 application protocol key: The Application Protocol Key identifies the type of Application Protocol for an Application Protocol Session. Multiple Application Protocol Sessions of the same type would be identified using the same Application Protocol Key (but different Session IDs). An Application Protocol Key is either an ASN.1 OBJECT IDENTIFIER belonging to a Recommendation, Standard, or non-standard protocol, or, alternatively, it is a non-standard identifier using the encoding conventions of Recommendation H.221.

3.5 application protocol session: A set of Peer Application Protocol Entities.

3.6 application record: A set of information for a specific Application Protocol Entity at a specific node. This set includes the Application User ID, the Active/Inactive flag, as well as other parameters.

3.7 application registry: A central repository located at the Top GCC Provider where an Application Protocol Entity can register its use of tokens, channels, and other parameters. Peer Application Protocol Entities can then access the registry to discover this registered information.

3.8 application roster: The set of all Application Records from all enrolled Application Protocol Entities at all nodes in a conference, including the Application Capabilities List for each Application Protocol Session.

3.9 application user ID: MCS User ID assigned by MCS to an Application Protocol Entity.

3.10 conducted mode: Conducted Mode allows a Conductor node the ability to control Application Protocol Entities at all nodes in a conference, and to restrict operation of Application Protocol Entities by other nodes. Conducted mode is established when a Conductor has been assigned to a conference. This is achieved when a node has successfully grabbed the conference Conductor Token.

3.11 conductor: The Conductor, if present, is a node in a conference which controls certain aspects of the conference (e.g. control of communication between Application Protocol Entities, control over conference participants, and conference termination). There shall be either zero or one Conductors in a conference. A node becomes conductor by grabbing the conductor token, or by requesting or accepting conductorship from the current conductor.

3.12 conference: A number of nodes that are joined together and that are capable of exchanging audiographic and audiovisual information across various communication networks.

3.13 conference application roster: A database maintained by each GCC Provider consisting of a set of Application Records, one for every Application Protocol Entity at every node in the conference, as well as other information such as the Application Capabilities List for each Application Protocol Session.

3.14 conference mode: When a conference is created, a Conference Mode is specified, by the Convener, that defines which Categories of nodes will be allowed to participate in the conference. These Conference Modes include the following: *conventional-only*, which allows only Conventional nodes to join the conference; *counted-controlled*, which allows only Counted and Conventional nodes to join the conference; *anonymous-controlled*, which allows only Anonymous and Conventional nodes to join the conference; and *unrestricted-mode*, which allows all three node categories to participate in the conference. The Node Controller must use the T.124 password to determine which nodes may join as Conventional when the Conference Mode is either *counted-controlled or anonymous-controlled*.

3.15 conference profile: A database maintained by each GCC Provider consisting of information pertinent to a conference as a whole such as Conference Name, Password (if any), etc.

3.16 conference roster: A database maintained by each GCC Provider consisting of a list of nodes in a conference. For each node, this list includes the Node ID of the node, type of the node, the name of the node, and may include a list of participants at the node, as well as other optional information.

3.17 control GCC service access point: The communication interface between a GCC Provider and the Node Controller within a single node.

3.18 convener: Node that created a conference by issuing the GCC-Conference-Create request primitive.

3.19 convener password: An identifying numeric string, as well as optional text string, which may be used when a conference is created to allow the convener of the conference to leave the conference and re-enter at a later time using the password to regain convener privileges. A Convener Password must be included during conference creation (in one of its two possible forms) for this to be possible. Use of the correct Convener Password allows joining of a locked conference, but it does not avoid the need to specify the correct Password when joining a Password protected conference.

3.20 conventional node: This is the most basic category of a T.120 node. The Conventional node category defines a fully capable node that is included in all roster exchanges.

3.21 counted node: This node category describes a node with the following properties: the node appears in rosters of conventional nodes, the node does not affect conference capabilities, and the node can participate in acknowledge-mode APE sessions. A counted node provides a way for (which provide the content) and a large number of counted nodes (which can acknowledge the reception of the content being sourced).

3.22 default session: Active Application Protocol Entities with no Session ID included in their Session Key are considered part of a separate unique session referred to as the Default Session.

3.23 entity ID: A 16-bit numeric identifier used to identify each Application Protocol Entity enrolled at a node. The value of the Entity ID is unique among all Application Protocol Entities of any type within a single node. It need not be unique between nodes. A particular Application Protocol Entity in a conference may be uniquely identified by the combination of the Entity ID and the Node ID corresponding to the node at which the Application Protocol Entity is enrolled.

3.24 GCC provider: Agent providing GCC services to local Node Controller and Application Protocol Entities at a terminal or MCU.

3.25 GCC service access point: The communication interface between a GCC Provider and an Application Protocol Entity within a single node.

3.26 handle: A 32-bit integer number allocated by the Top GCC Provider using the GCC-Registry-Allocate-Handle request primitive. This number is guaranteed to be unique within a conference.

3.27 local application roster: A database maintained by each GCC Provider consisting of one Application Record as well as other information, such as the Application Capabilities List, for each Application Protocol Entity which has locally enrolled with the GCC Provider. This is used to form the information exchanged with other nodes to determine the Conference Application Roster.

3.28 MCS domain: A hierarchy of MCS connections between nodes. Nodes may use MCS services to communicate within a single domain, but not between separate domains. A GCC Conference corresponds one-to-one with a single MCS Domain.

3.29 MCS domain selector: A locally unique identifier of an MCS Domain.

3.30 MCS user ID: Unique identification number assigned by MCS to an MCS User. GCC Providers as well as Application Protocol Entities are MCS Users. An MCS User ID assigned to a GCC Provider is referred to as a Node ID. An MCS User ID assigned to an Application Protocol Entity is referred to as an Application User ID. An MCS User ID is valid only within a single MCS Domain.

3.31 multipoint: The ability to exchange data among multiple nodes simultaneously as compared with point-to-point, where data are exchanged between two directly connected nodes.

3.32 multipoint control unit: Commonly referred to as an MCU or bridge, a multiport device that serves to connect terminals and other MCUs in a multipoint fashion. A GCC-capable MCU runs GCC and MCS. An MCU is not primarily intended as an end-point for user communication.

3.33 multiport terminal: End-point audiographic or audiovisual equipment that also includes the ability to bridge T.120 information. Like a terminal, the behavior of a multiport terminal is typified by automatic establishment of a single conference. But, like an MCU, for a given conference in a multiport terminal, there may be more than one MCS connection.

3.34 node: A terminal, multiport terminal, or MCU. A single node comprises a single GCC provider. A single node may consist of one or more physical devices. Similarly, one physical device may host several logical nodes.

3.35 node category: Every T.120 node falls into one of three categories: Conventional, Counted, or Anonymous. These node categories allow for different degrees of scalabilitz within a T.120 conference.

3.36 node controller: A functional entity for which there is one for each terminal or MCU, which serves as the controller of that node.

3.37 node ID: MCS User ID assigned by MCS to the GCC Provider at a node.

3.38 non-conducted mode: The mode in which a conference has no Conductor.

3.39 participant: A person participating in a conference at a node.

3.40 password: A numeric string, as well as an optional text string, which may be specified when a conference is created. If so, when attempting to join a conference, a node must include this Password (in one of the two possible forms) in the GCC-Conference-Join primitive in order for that node to be accepted into the conference.

3.41 peer application protocol entity: Application Protocol Entities which have enrolled in the active state using identical Session Keys, including the Session ID portion of the Session Key. Peer Application Protocols are those which may communicate with each other during a conference.

3.42 resource: Something that can be used and shared by nodes in a conference. A resource comprises both channels and tokens.

3.43 session ID: An optional parameter included in a Session Key used to distinguish between multiple sets of Peer Application Protocol Entities which are based on the same base Application Protocol. Each set of Peer Application Protocol Entities, defined by the use of a common Application Protocol and identical Session IDs, separately communicate among themselves. Active Application Protocol Entities with no Session ID included in their Session Key are considered part of a separate unique session referred to as the Default Session. A set of Peer Application Protocol Entities is referred to as an Application Protocol Session. Session IDs are in the form of MCS Channel IDs.

3.44 session Key: An identifier which is common to Peer Application Protocol Entities. A Session Key consists of two components. One component, the Application Protocol Key identifies the type of Application Protocol. The second component, which is an optional part of a Session Key, is the Session ID which identifies the specific session for this Application Protocol (the lack of a Session ID indicates the Default Session). Application Protocol Entities whose entire Session Keys are in common, including the Application Protocol Key as well as the Session ID, if any, are considered Peer Application Protocol Entities.

3.45 terminal: End-point audiographic or audiovisual equipment. A GCC-capable terminal runs GCC and MCS. A terminal is limited, within a conference, to a single MCS connection.

3.46 top GCC Provider: The GCC Provider which is co-resident with the Top MCS Provider in a conference. The Top GCC Provider has responsibilities not required of other GCC Providers in a conference. The location of the Top GCC Provider remains unchanged for the duration of a conference.

3.47 Unicode: Multilingual text string format as defined by the Basic Multilingual Plane of ISO/IEC 10646-1.

3.48 Unicode Row 00: A subset of Unicode consisting of 256 code positions containing the Basic Latin and Latin-1 Supplement character sets plus control characters and reserved codes.

3.49 user application: An entity which makes use of one or more Application Protocol Entities. A User Application is limited in its scope to those tasks which have no effect on the interpretation of information between Peer Application Protocol Entities such as presentation to the end-user. User Applications therefore do not require specification, either by standard Recommendation or otherwise, to allow interoperability between the Application Protocol Entities which they make use of and their Peer Application Protocol Entities.

4 Abbreviations

This Recommendation uses the following abbreviations.

CSDN Circuit Switched Data Network

GCC Generic Conference Control

GCCSAP Generic Conference Control Service Access Point

ISDN Integrated Services Digital Network

MCS Multipoint Communication Service

MCU Multipoint Control Unit

PDU Protocol Data Unit

PSDN Packet Switched Data Network

PSTN Public Switched Telephone Network

The following abbreviations are used in this Recommendation and defined in Recommendation T.122:

MCS Multipoint Communication Service

MCSAP Multipoint Communication Service Access Point

5 Conventions

The primitive parameters of the abstract services defined in this Recommendation use the following key:

M Parameter is mandatory

C Parameter is conditional

O Parameter is optional

Blank Parameter is absent

(=) Value of the parameter is identical to the value of the corresponding parameter of the preceding primitive, where preceding is defined relative to the order: request, indication, response, confirm.

(=RQ) Value of the parameter is identical to the value of the corresponding parameter in a preceding primitive, where RQ = request, IN = indication, RS = response, and CF = confirm.

Primitives are categorized in up to four types: Request, Indication, Response, and Confirm. Some primitives support all of these types, while others do not. These four types are defined as follows:

– Request primitive: Those that are sourced from a Node Controller or Application Protocol Entity to initiate a certain action.

– Indication primitive: Those that are sourced from a GCC Provider either as a result of a Request primitive, or as a result of a GCC initiated action.

– Response primitive: Those that are sourced from a Node Controller or Application Protocol Entity in response to an Indication primitive which is defined to require a response.

– Confirm primitive: Those that are sourced from a GCC Provider as a result of a Response primitive.

PDUs are categorized into three types. PDU names all include the words Request, Indication, or Response to indicate the intended use of the PDU. These are defined as follows:

– Request PDUs: Those that require a Response PDU in return. If the request is for a function which is not supported by the receiving node (e.g. an optional or non-standard PDU), a generic response PDU, FunctionNotSupportedResponse, shall be used to provide a response to the requesting node (sent to the Node ID Channel of the requesting node).

– Indication PDUs: Those that do not require a response (e.g. those that are for informational purposes).

– Response PDUs: Those which are in response to a particular Request PDU to be sent on the Node ID Channel of the requesting node.

6 Overview

Within the context of the ITU-T Audiovisual Conferencing Service, a conference refers to a group of geographically dispersed nodes that are joined together and that are capable of exchanging audiographic and audiovisual information across various communication networks. Participants taking part in a conference may have access to various types of media handling capabilities such as audio only (telephony), audio and data (audiographics), audio and video (audiovisual), and audio, video, and data (multimedia).

The F-, G-, H-, and T-series Recommendations provide a framework for the interworking of audio, video, and graphics terminals on a point-to-point basis through existing telecommunication networks. They also provide the capability for three or more terminals in the same conference to be interconnected by means of an MCU.

This Recommendation provides a high-level framework for conference management and control of audiographics and audiovisual terminals, and MCUs. It coexists with companion Recommendations T.122 and T.125 (MCS) and T.123 (AVPS) to provide a mechanism for conference establishment and control. Recommendations T.122, T.123, T.124, and T.125 form the minimum set of Recommendations to develop a fully functional terminal or MCU.

This Recommendation includes the following Generic Conference Control (GCC) functional components: conference establishment and termination, managing the Conference Roster, managing the Application Roster, Application Registry services, and conference conductorship. The service definitions for the primitives associated with these functional components are contained in clause 7. The corresponding protocol definitions are contained in clause 8.

Figure 6-1 shows an example of how GCC components are distributed throughout an MCS domain. The GCC components are shown in white. Each terminal or MCU contains a GCC Provider that provides GCC services to the local Node Controller and Application Protocol Entities.

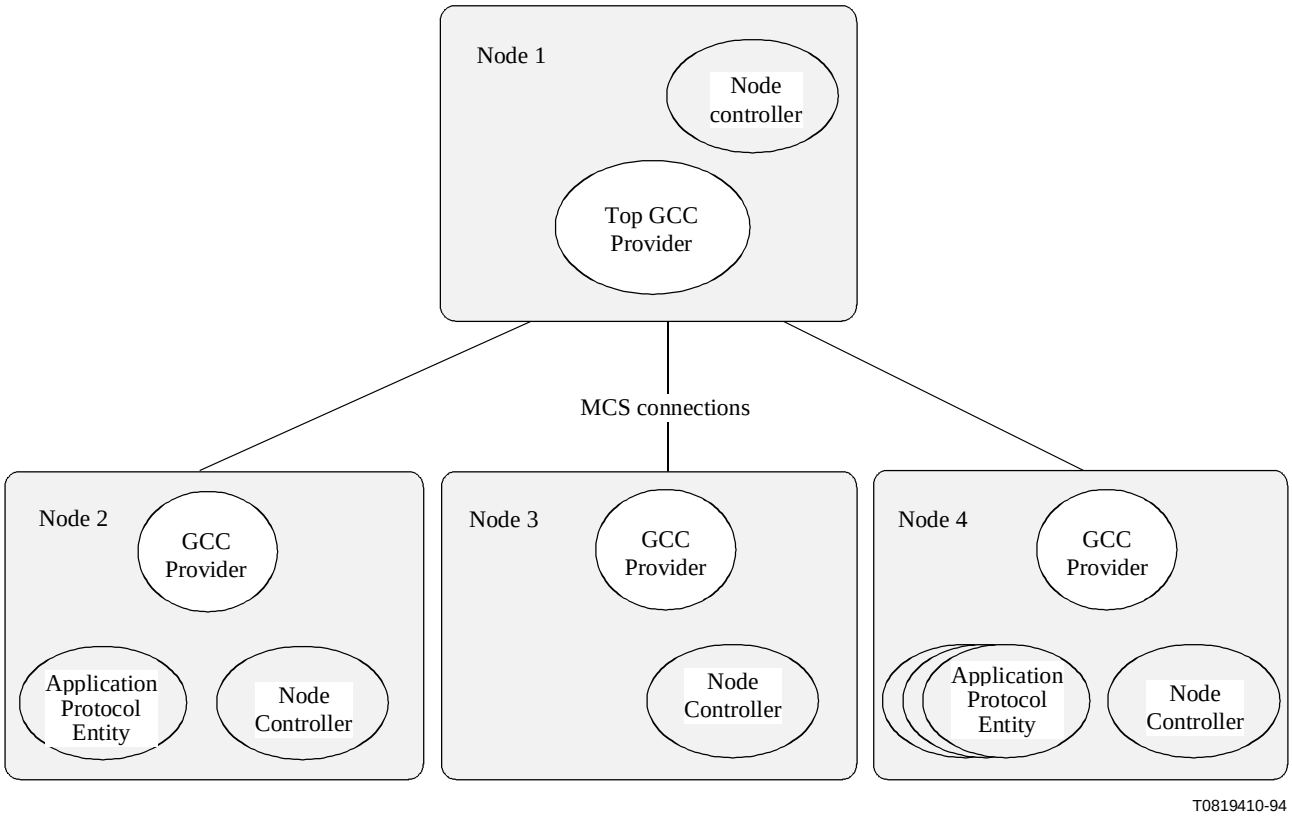

Figure 6-1/T.124 – Example of GCC components distributed throughout an MCS domain

6.1 System model for a conference node

Each Node participating in a GCC conference consists of an MCS layer, a GCC layer, a Node Controller and may also include one or more Application Protocol Entities. The relationship between these components within a single node is illustrated in Figure 6-2. The Node Controller is the controlling entity at a node, dealing with the aspects of a conference that apply to the entire node. The Node Controller interacts with GCC, but does not interact directly with MCS. Application Protocol Entities also interact with GCC, as well as directly with MCS. The services provided by GCC to Application Protocol Entities are primarily to enable Peer Application Protocol Entities to communicate directly, via MCS. Local communication between individual Application Protocol Entities or between Application Protocol Entities and the Node Controller may take place, but is a local implementation matter not covered by this Recommendation. Within a node, more than one Application Protocol Entity may be based on the same Application Protocol. In this case, they may either be part of the same Application Protocol Session, allowing them to communicate within the node as well as to other Peer Application Protocol Entities at other nodes, or they may be part of separate Application Protocol Sessions, allowing them to communicate separately among their peers, but using the same protocol.

The service primitives, as described in clause 7, apply to the GCC Service Access Point and the Control GCC Service Access Point as indicated in Figure 6-2. The PDUs described in clause 8 are communicated using MCS service primitives available at the Control MCS Access Point (MCSAP).

NOTE 1 – The normative intent of this Recommendation is to specify the procedures and contents of external communication – sequences of primitive operations and data exchanges acting through the Control MCSAP for purposes of conference control. The internal decomposition of a node suggested in Figure 6-2 serves to motivate features of the GCC protocol but is not normative. GCC service primitives whose effect is purely local need not exist at all nodes in the form that they are described here. Statements about what a node controller or an application protocol entity shall do in certain circumstances should be interpreted loosely if the same results in external communication can be achieved through different internal mechanisms.

NOTE 2 – The system model assumes that service requests may be issued at any time. A Node Controller or Application Protocol Entity need not wait until it has received a confirm from the GCC Provider corresponding to a previous request before issuing another request. A specific implementation may, however, impose stricter requirements.

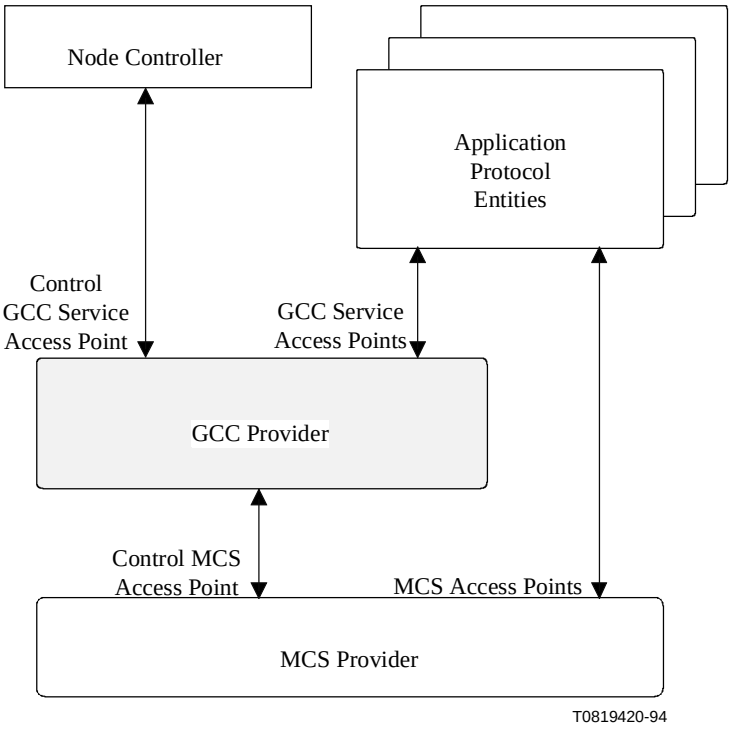

Figure 6-2/T.124 – System model showing GCC Service Access Point and relationship with MCS

6.2 Conference establishment and termination

GCC provides a set of services for establishment and termination of conferences. A conference can be viewed as a meeting room in which any number of participants may meet in order to exchange audiographic or audiovisual information. As with physical meeting rooms, services such as finding out what conferences are in progress, joining a conference, leaving a conference, restricting access to a conference, etc., are meaningful in audiographic and audiovisual conferences as well.

Prior to joining a conference, participants at a node may not know all of the information needed to join. GCC provides a means for participants to view a list of Conference Names and select the one they wish to join. This service is analogous to the conference schedule typically posted in a lobby, allowing someone to find the meeting room in which a particular meeting is taking place.

GCC provides a means to create new conferences. This may be done either by a conference participant, or by a conference administrator. When a new conference is created, its characteristics, referred to as its Conference Profile, are specified by its creator. The Conference Profile includes

such things as the Conference Name, whether it has restricted access by means of a Password, whether it is open to be freely joined (unlocked) or restricted to be joined by invitation only (locked).

Expanding an existing conference may be initiated by the joining node, or by a conference Convener (or convener designated node). If a conference is Password protected, a joining node is required to supply the correct Password to be allowed into the conference. If a conference is locked, joining is only allowed at the request of the Convener, or convener-designated node. Nodes may join or be added at any time during a conference. It is possible to be joined with more than one conference simultaneously. GCC also provides a means of transferring participants from one conference to another. This function may be used to achieve the effect of merging two conferences, or splitting a conference into more than one conference.

At any time a node may wish to disconnect from the conference, leaving the other nodes to continue the conference. Depending on the choice of termination method, a conference is either automatically terminated if all nodes disconnect from it, or manually terminated by an explicit termination request. The convener, or a convener-designated node may also forcibly terminate the entire conference at any time, or eject a particular node from it.

6.3 The conference roster

Once a node has joined a conference, it announces its presence to all other nodes in the conference. A GCC primitive is provided to allow each node in a conference to announce its presence in a conference, and another to provide each node with either updates to the conference roster or a full conference roster. The conference roster is a list of Conventional and possible Counted nodes in the conference. For each node the conference roster includes information such as the name of the node, a list of participants at that node, as well as other information needed for proper communication between nodes. A Conventional or Counted node is not considered part of a conference until it has been included in the conference roster.

6.4 The application roster

GCC provides a means of identifying which Application Protocol Entities are available at each node and also provides necessary information for Peer Application Protocol Entities to communicate with each other. Upon joining a conference, each Conventional or Counted node sends to all other nodes its local list of Application Protocol Entities – its Local Application Roster – which it may update at any time thereafter. From this information, the Conference Application Roster is formed and broadcast to all interested nodes. Relevant portions of this roster are then communicated locally to each Application Protocol Entity as well as to the Node Controller. In addition to a simple roster, GCC also provides a service for Application Protocol Entities at Conventional Nodes to include a list of Application Protocol-specific capabilities in the information exchanged. GCC applies a fixed set of rules to this information from all nodes in the conference to determine a common set of Application Capabilities. This information is also communicated locally to each Application Protocol Entity.

6.5 The Application Registry

The Application Registry is an active database residing at the Top GCC Provider that may be used to manage channels, tokens, and other shared resources used in a conference. The Application Registry can aid in establishing communication among peer Application Protocol Entities.

6.6 Conference conductorship

GCC provides a method for allowing a node to become a conductor for a conference. A token is used by GCC to determine whether a conference is conducted or non-conducted. The node which grabs the conductor token becomes the conductor of the conference. A node may also request conductorship or accept conductorship from the current conductor. Upon request, GCC provides the identity of the current conference conductor. On creation of a conference, it may be specified that conducted mode is not permitted for the duration of the conference.

Conducted mode is available as a means to provide order to the course of a conference. The actual means by which this order is provided is determined by the Application Protocols. Specifically, Recommendations specifying Application Protocols may define alternative procedures depending on whether the conference is conducted or non-conducted. GCC does provide a mechanism for basic conducted operation which may be made use of by Application Protocols. A mechanism is provided by which a node may request permission from the conducting node, and if permission is granted, all Application Protocol Entities at that node may act accordingly as specified by the Application Protocol specification. Application Protocols may, for example, specify strict limitations in the allowed operations for nodes that do not have permission from the conductor, while removing some or all of these limitations for nodes which have this permission. Application protocols may also, for example, specify that once overall permission has been granted by the conductor, further permission must be granted by the Peer Application Protocol Entity at the conducted node, if one exists, before allowing certain operations to be performed.

6.7 Miscellaneous functions

A method is provided for coordinating timed conferences. A mechanism is provided for a node to find out how much time is remaining in a timed conference, as well as a mechanism for announcing to all nodes how much time is remaining (which would typically be used to announce that the time is almost up), and a mechanism for nodes to request more time to be added, if available.

A method is also provided to request assistance from an unspecified operator. Another function is provided to allow transmission of simple text messages.

6.8 Scalable conferences

The T.124 protocol is designed to allow for a number of different degrees of scalability, including small "tightly coupled" conferences, medium-sized "acknowledged" conferences, and very large "loosely coupled" conferences. This is accomplished through the use of three Node Categories, each of which allows for a varying degree of scalability. These categories include Conventional, Counted and Anonymous. An overview of each of these categories follows.

Conventional Nodes provide a large amount of node information to every node in a conference and are the most heavy-weight in terms of how they affect the other members of the conference. Using this category is appropriate for principal members of a loosely-coupled conference or for all nodes involved in a small tightly-coupled conference. Conventional Nodes are responsible for creating all APE sessions within a conference and also dictate the capabilities negotiated in those sessions. Each T.124 conference must include at least one Conventional Node. Also, each APE session must include at least one conventional node as long as the session exists. Once all Conventional nodes have left an APE session, that session ceases to exist.

Counted Nodes are somewhat less heavy-weight than Conventional Nodes. Nodes of this type only appear in rosters received by Conventional Nodes. Counted nodes cannot create APE sessions and they do not affect session capabilities. The capabilities for each APE session are established by the Conventional Nodes participating in the conference. Counted Nodes are typically used in situations

where there is a need to acknowledge receiving data that is sourced by a Conventional Node. Conferences with Counted Node participants scale better than conferences with only Conventional Nodes, but are still limited in size due to the exchange of roster information. Counted nodes may source content for the conference, but that content can never be acknowledged.

Anonymous Nodes are the lightest-weight of the three Node Categories. Nodes in this category do not affect conference or application rosters, cannot create APE sessions, and do not affect conference capabilities. Their purpose is to allow for very large-scale conferences in environments where there are a small number of presenters and a large number of observers. A typical conference scenario could include a small number of Conventional nodes, which would be the principal contributors of the conference and a large number of Anonymous node observers.

Prior to the introduction of Node Categories, the number of nodes that could participate in a GCC conference was severely limited. Node Categories make it possible to support a number of different conference scaling models. Because legacy nodes that are not aware of Node Categories must continue to be supported within the T.120 framework, a few restrictions are necessary to ensure backward compatibility. Also, an additional feature of GCC that facilitates scalable conferences is the use of Roster Delta Updates by the Top Provider when notifying conference participants that a node has joined, left, or changed a Roster record. Again, this feature was not supported in the original version of GCC. The following rules must be abided by to ensure backward compatibility with legacy nodes:

- A node can only be considered Anonymous or Counted if it joins the conference through a node that is Node Category-aware. Also, all nodes from the joining node to the Top Provider (including the Top Provider) must also be Node Category aware.
- Nodes that are not Node Category-aware, or do not indicate an understanding of Node Categories when joining a conference, are always treated as Conventional.
- The Top Provider, Convener, and all Management nodes are always categorized as Conventional nodes.
- All T.124 nodes, except those that will only participate in a T.120 conference as Anonymous or Counted terminal nodes must continue to support the legacy GCC-Broadcast-Channel as well as the GCC-Conventional-Broadcast-Channel and the GCC-Counted-Broadcast-Channel if necessary (see explanation of all broadcast channels in the Protocol definition of this Recommendation).
- Only Conventional nodes are allowed to add, delete, or change the parameters within the registry. To support Counted and Anonymous nodes, the registry can be read by any node that makes a request to do so, including nodes that are not listed in the Conference or Application rosters.
- There are no topology restrictions on which categories of nodes can be connected to which categories of nodes as long as all the nodes above a node that is expecting to use the Node Category related protocol is also Node Category-aware.

In summary, when a conference is created, a Conference Mode is specified that defines which Categories of nodes will be allowed to participate in the conference. It is the responsibility of both the GCC Provider and the Node Controller at the Top Provider to determine which Categories a joining node must fall into. These Conference Modes include the following: *conventional-only*, which allows only Conventional nodes to join the conference; *counted-controlled*, which allows only Counted and Conventional nodes to join the conference; *anonymous-controlled*, which allows only Anonymous and Conventional nodes to join the conference; and *unrestricted-mode*, which allows all three node categories to participate in the conference. The Node Controller must use the T.124 password to determine which nodes may join as Conventional when the Conference Mode is either *counted-controlled* or *anonymous-controlled*.

6.9 Summary of GCC abstract services

Table 6-1 is a list of all GCC primitives and their associated PDUs. The table also shows whether or not each primitive is mandatory (M), conditionally required (C), or optional (O) for a terminal or an MCU. For a multiport terminal, for each primitive, the requirement shall be taken to be the most restrictive of either a terminal or MCU for that primitive. A conditionally required primitive is one which is required if the Application Protocol specification for one or more Application Protocol Entities located at that node mandates its use. The table also shows, for the corresponding PDUs, whether these are mandatory (M) or conditionally required (C) for both the transmit (T) and receive (R) directions. Again, for a multiport terminal, the requirement shall be taken to be the most restrictive of either a terminal or MCU for that PDU. A conditionally required PDU is one which is required only if the corresponding primitive is to be supported at that node. In the case that a primitive is not mandatory, but its corresponding PDU is mandatory, this implies that there is some portion of the protocol, not related to the primitive, that relies on the use of that PDU which is required to be supported.

Table 6-1/T.124 – GCC Primitives and PDUs

Functional unit	Primitives	Term	MCU	Associated PDUs	Dir.	Term	MCU
Conference establishment and termination	GCC-Conference-Create request	M	O	ConferenceCreateRequest	T	M	C
	GCC-Conference-Create indication	M	M	ConferenceCreateRequest	R	M	M
	GCC-Conference-Create response	M	M	ConferenceCreateResponse	T	M	M
	GCC-Conference-Create confirm	M	O	ConferenceCreateResponse	R	M	C
				UserIDIndication	T, R	M	M
	GCC-Conference-Query request	M	M	ConferenceQueryRequest	T	M	M
	GCC-Conference-Query indication	M	M	ConferenceQueryRequest	R	M	M
	GCC-Conference-Query response	M	M	ConferenceQueryResponse	T	M	M
	GCC-Conference-Query confirm	M	M	ConferenceQueryResponse	R	M	M
	GCC-Conference-Join request	M	O	ConferenceJoinRequest	T	M	C
	GCC-Conference-Join indication	O	M	ConferenceJoinRequest	R	C	M
	GCC-Conference-Join response	O	M	ConferenceJoinResponse	T	C	M
	GCC-Conference-Join confirm	M	O	ConferenceJoinResponse	R	M	C
				UserIDIndication	T, R	M	M
	GCC-Conference-Invite request	O	M	ConferenceInviteRequest	T	C	M
	GCC-Conference-Invite indication	M	M	ConferenceInviteRequest	R	M	M
	GCC-Conference-Invite response	M	M	ConferenceInviteResponse	T	M	M
	GCC-Conference-Invite confirm	O	M	ConferenceInviteResponse	R	C	M
				UserIDIndication	T, R	M	M
	GCC-Conference-Add request	O	O	ConferenceAddRequest	T	C	C
	GCC-Conference-Add indication	O	O	ConferenceAddRequest	R	C	C
	GCC-Conference-Add response	O	O	ConferenceAddResponse	T	C	C
	GCC-Conference-Add confirm	O	O	ConferenceAddResponse	R	C	C
	GCC-Conference-Lock request	O	O	ConferenceLockRequest	T	C	C
	GCC-Conference-Lock indication	O	O	ConferenceLockRequest	R	C	C
	GCC-Conference-Lock response	O	O	ConferenceLockResponse	T	C	C
	GCC-Conference-Lock confirm	O	O	ConferenceLockResponse	R	C	C
	GCC-Conference-Unlock request	O	O	ConferenceUnlockRequest	T	C	C
	GCC-Conference-Unlock indication	O	O	ConferenceUnlockRequest	R	C	C
	GCC-Conference-Unlock response	O	O	ConferenceUnlockResponse	T	C	C
	GCC-Conference-Unlock confirm	O	O	ConferenceUnlockResponse	R	C	C

Table 6-1/T.124 – GCC Primitives and PDUs (*continued*)

Functional unit	Primitives	Term	MCU	Associated PDUs	Dir.	Term	MCU
	GCC-Conference-Lock-Report indication	O	O	ConferenceLockIndication	T, R	C	C
				ConferenceUnlockIndication	T, R	C	C
	GCC-Conference-Disconnect request	M	M	–	–	–	–
	GCC-Conference-Disconnect indication	M	M	–	–	–	–
	GCC-Conference-Disconnect confirm	M	M	–	–	–	–
	GCC-Conference-Terminate request	O	O	ConferenceTerminateRequest	T	C	C
				ConferenceTerminateRequest	R	M	M
	GCC-Conference-Terminate indication	M	M	ConferenceTerminateIndication	T	M	M
				ConferenceTerminateIndication	R	M	M
	GCC-Conference-Terminate confirm	O	O	ConferenceTerminateResponse	T	M	M
				ConferenceTerminateResponse	R	C	C
	GCC-Conference-Eject-User request	O	O	ConferenceEjectUserRequest	T	C	C
				ConferenceEjectUserRequest	R	M	M
	GCC-Conference-Eject-User indication	M	M	ConferenceEjectUserIndication	T	M	M
				ConferenceEjectUserIndication	R	M	M
	GCC-Conference-Eject-User confirm	O	O	ConferenceEjectUserResponse	T	M	M
				ConferenceEjectUserResponse	R	C	C
	GCC-Conference-Transfer request	O	M	ConferenceTransferRequest	T	C	M
				ConferenceTransferRequest	R	M	M
	GCC-Conference-Transfer indication	M	M	ConferenceTransferIndication	T	M	M
				ConferenceTransferIndication	R	M	M
	GCC-Conference-Transfer confirm	O	M	ConferenceTransferResponse	T	M	M
				ConferenceTransferResponse	R	C	M
Conference roster	GCC-Conference-Announce-Presence request	M	M	RosterUpdateIndication	T, R	M	M
	GCC-Conference-Announce-Presence confirm	M	M	–	–		
	GCC-Conference-Roster-Report indication	M	M	RosterUpdateIndication	T, R	M	M
	GCC-Conference-Roster-Inquire request	O	O	–	–	–	–
	GCC-Conference-Roster-Inquire confirm	O	O	–	–	–	–
Application roster	GCC-Application-Permission-To-Enroll indication	M	C	–	–	–	–
	GCC-Application-Enroll request	M	C	RosterUpdateIndication	T, R	M	M
				RosterRefreshRequest	T, R	C	C
	GCC-Application-Enroll confirm	M	C	–	–		
	GCC-Application-Roster-Report indication	M	C	RosterUpdateIndication	T, R	M	M
	GCC-Application-Roster-Inquire request	O	O	–	–	–	–
	GCC-Application-Roster-Inquire confirm	O	O	–	–	–	–
	GCC-Application-Invoke request	O	O	ApplicationInvokeIndication	T	C	C
	GCC-Application-Invoke indication	O	O	ApplicationInvokeIndication	R	C	C
	GCC-Application-Invoke confirm	O	O	–	–	–	–

Table 6-1/T.124 – GCC Primitives and PDUs (*continued*)

Functional unit	Primitives	Term	MCU	Associated PDUs	Dir.	Term	MCU
Application registry	GCC-Registry-Register-Channel request	C	C	RegistryRegisterChannelRequest	T	C	C
				RegistryRegisterChannelRequest	R	M	M
	GCC-Registry-Register-Channel confirm	C	C	RegistryResponse	T	M	M
				RegistryResponse	R	C	C
	GCC-Registry-Assign-Token request	C	C	RegistryAssignTokenRequest	T	C	C
				RegistryAssignTokenRequest	R	M	M
	GCC-Registry-Assign-Token confirm	C	C	RegistryResponse	T	M	M
				RegistryResponse	R	C	C
	GCC-Registry-Set-Parameter request	C	C	RegistrySetParameterRequest	T	C	C
				RegistrySetParameterRequest	R	M	M
	GCC-Registry-Set-Parameter confirm	C	C	RegistryResponse	T	M	M
				RegistryResponse	R	C	C
	GCC-Registry-Retrieve-Entry request	C	C	RegistryRetrieveEntryRequest	T	C	C
				RegistryRetrieveEntryRequest	R	M	M
	GCC-Registry-Retrieve-Entry confirm	C	C	RegistryResponse	T	M	M
				RegistryResponse	R	C	C
	GCC-Registry-Delete-Entry request	C	C	RegistryDeleteEntryRequest	T	C	C
				RegistryDeleteEntryRequest	R	M	M
	GCC-Registry-Delete-Entry confirm	C	C	RegistryResponse	T	M	M
				RegistryResponse	R	C	C
	GCC-Registry-Monitor request	C	C	RegistryMonitorEntryRequest	T	C	C
				RegistryMonitorEntryRequest	R	M	M
	GCC-Registry-Monitor indication	C	C	RegistryMonitorEntryIndication	T	M	M
				RegistryMonitorEntryIndication	R	C	C
	GCC-Registry-Monitor confirm	C	C	RegistryResponse	T	M	M
				RegistryResponse	R	C	C
	GCC-Registry-Allocate-Handle request	C	C	RegistryAllocateHandleRequest	T	C	C
				RegistryAllocateHandleRequest	R	M	M
	GCC-Registry-Allocate-Handle confirm	C	C	RegistryAllocateHandleResponse	T	M	M
				RegistryAllocateHandleResponse	R	C	C
Conference conductorship	GCC-Conductor-Assign request	O	O	–	–	–	–
	GCC-Conductor-Assign indication	C	C	ConductorAssignIndication	T	M (Note)	M (Note)
				ConductorAssignIndication	R	C	C
	GCC-Conductor-Assign confirm	O	O	–	–	–	–
	GCC-Conductor-Release request	O	O	ConductorReleaseIndication	T	C	C
				ConductorReleaseIndication	R	M (Note)	M (Note)
	GCC-Conductor-Release indication	C	C	ConductorReleaseIndication	T	M (Note)	M (Note)
				ConductorReleaseIndication	R	C	C
	GCC-Conductor-Release confirm	O	O	–	–	–	–
	GCC-Conductor-Please request	O	O	–	–	–	–
	GCC-Conductor-Please indication	O	O	–	–	–	–
	GCC-Conductor-Please confirm	O	O	–	–	–	–

Table 6-1/T.124 – GCC Primitives and PDUs (*concluded*)

Functional unit	Primitives	Term	MCU	Associated PDUs	Dir.	Term	MCU
	GCC-Conductor-Give request	O	O	–	–	–	–
	GCC-Conductor-Give indication	O	O	–	–	–	–
	GCC-Conductor-Give response	O	O	ConductorAssignIndication	T	C	C
	GCC-Conductor-Give confirm	O	O	–	–	–	–
	GCC-Conductor-Inquire request	C	C	–	–	–	–
	GCC-Conductor-Inquire confirm	C	C	–	–	–	–
	GCC-Conductor-Permission-Ask request	O	O	ConductorPermissionAskIndication	T	C	C
	GCC-Conductor-Permission-Ask indication	O	O	ConductorPermissionAskIndication	R	C	C
	GCC-Conductor-Permission-Ask confirm	O	O	–	–	–	–
	GCC-Conductor-Permission-Grant request	O	O	ConductorPermissionGrantIndication	T	C	C
	GCC-Conductor-Permission-Grant indication	O	O	ConductorPermissionGrantIndication	R	C	C
	GCC-Conductor-Permission-Grant confirm	O	O	–	–	–	–
Miscellaneous functions	GCC-Conference-Time-Remaining request	O	O	ConferenceTimeRemainingIndication	T	C	C
	GCC-Conference-Time-Remaining indication	O	O	ConferenceTimeRemainingIndication	R	C	C
	GCC-Conference-Time-Remaining confirm	O	O	–	–		
	GCC-Conference-Time-Inquire request	O	O	ConferenceTimeInquireIndication	T	C	C
	GCC-Conference-Time-Inquire indication	O	O	ConferenceTimeInquireIndication	R	C	C
	GCC-Conference-Time-Inquire confirm	O	O	–	–		
	GCC-Conference-Extend request	O	O	ConferenceTimeExtendIndication	T	C	C
	GCC-Conference-Extend indication	O	O	ConferenceTimeExtendIndication	R	C	C
	GCC-Conference-Extend confirm	O	O	–	–	–	–
	GCC-Conference-Assistance request	O	O	ConferenceAssistanceIndication	T	C	C
	GCC-Conference-Assistance indication	O	O	ConferenceAssistanceIndication	R	C	C
	GCC-Conference-Assistance confirm	O	O	–	–	–	–
	GCC-Text-Message request	O	O	TextMessageIndication	T	C	C
	GCC-Text-Message indication	O	O	TextMessageIndication	R	C	C
	GCC-Text-Message confirm	O	O	–	–	–	–
	–	–	–	FunctionNotSupported	T, R	M	M

NOTE – Reception of ConductorReleaseIndication PDUs is mandatory to allow the Top GCC Provider to properly handle requests which require privileges. Privileges depend on whether the conference is in conducted or non-conducted mode. Transmission of ConductorAssignIndication and ConductorReleaseIndication PDUs by the Top GCC Provider is mandatory for handling new nodes joining a conference.

7 GCC service definition

7.1 Conference establishment and termination

In this clause, primitives needed for conference establishment and conference termination are described. All of the primitives in this clause are intended for use only by the Node Controller at a terminal or MCU.

7.1.1 The conference profile

All conferences have the following characteristics which are defined when the conference is created and communicated to each node as it enters the conference. These characteristics remain unchanged for the duration of the conference. This information is collectively referred to as the Conference Profile:

- *Conference name* – A numerical string and an optional Unicode Row 00 text string identifying the conference. If both forms of Conference Name are used when a conference is created, when that conference is joined, either form may be specified to indicate the conference to be joined.
- *Conference description* – An optional text string to describe the conference. For a listed conference, this string is reproduced in the Conference Descriptor List in response to a GCC-Conference-Query request.
- *Password protected vs. not password protected* – Choice of whether the conference is Password protected or not.
- *Listed vs. unlisted* – Choice of whether the conference is listed or not listed on the conference list provided when querying the list of available conferences.
- *Conductible vs. non-conductible* – Choice of whether the conference is able to be placed in conducted mode or whether the conference is always non-conducted.
- *Termination method* – Choice of whether conference should last until explicitly terminated (manually terminating), or if it should last until all participants disconnect (automatically terminating).
- *Privilege lists* – A set of lists indicating which privileges, normally only available to the Convener, are also allowed to the Conductor, to any node in a conducted conference, or to any node in a non-conducted conference.
- *Conference Mode* – An optional choice that specifies which categories of nodes will be allowed to join the conference after it is created. If not specified, the conference should default to conventional-only mode.

7.1.2 Description of abstract services

The following is a list of the primitives defined in this subclause and a brief summary of the function of each:

- GCC-Conference-Create – Used by the Node Controller to create a new conference, specifying the characteristics of that conference.
- GCC-Conference-Query – Used by the Node Controller to query what conferences are currently in progress as well as the information needed to attempt to join them.
- GCC-Conference-Join – Used by the Node Controller to join an existing conference.
- GCC-Conference-Invite – Used by the Node Controller to invite a node into an existing conference.

- GCC-Conference-Add – Allowed only by the conference convener or convener-designated node; this is used by the Node Controller to request that a node be added to the conference by dialling out from an MCU.
- GCC-Conference-Lock – Allowed only by the conference convener or convener-designated node; this is used by the Node Controller to prevent new participants from joining a conference without being explicitly added.
- GCC-Conference-Unlock – Allowed only by the conference convener or convener-designated node; this is used by the Node Controller to allow new participants to join a conference.
- GCC-Conference-Lock-Report – Provides an indication to the Node Controller that a conference has changed from being locked to being unlocked or vice versa.
- GCC-Conference-Disconnect – Used by a Node Controller to disconnect the local node from an ongoing conference.
- GCC-Conference-Terminate – Allowed only by the conference convener or convener-designated node; this is used by the Node Controller to terminate an entire conference, disconnecting all nodes.
- GCC-Conference-Eject-User – Allowed only by the conference convener or convener-designated node (or by the node directly above the ejected node in the connection hierarchy); this is used by the Node Controller to disconnect a specific node from an ongoing conference.
- GCC-Conference-Transfer – Allowed only by the conference convener or convener-designated node; this is used by the Node Controller to transfer nodes joined with one conference to another conference. This may be used as part of the process of merging or splitting conferences.

7.1.2.1 GCC-Conference-Create

The GCC-Conference-Create request primitive is used by a Node Controller to create a new conference at a remote node to which the local node is automatically joined. This primitive may be issued at any time. When a conference is created, the node to which the creation request is directed (the node which receives the GCC-Conference-Create indication) is also automatically joined to the conference and becomes the Top GCC Provider for that conference. This node remains the Top GCC Provider for the conference as long as the conference continues to exist. In some implementations, it may be possible to create a conference locally without the use of GCC primitives. In this case, the node at which the conference is created becomes the Top GCC Provider. Table 7-1 shows the parameters and types of this primitive. Figure 7-1 shows the sequence of events when using this primitive.

Table 7-1/T.124 – GCC-Conference-Create – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Conference Name	M	M(=)	M(=)	M(=)
Conference Name Modifier	C		C	
Conference ID		M	M(=)	M
Convener Password	O	O(=)		
Password	O	O(=)		
Conference Locked	M	M(=)		
Conference Listed	M	M(=)		
Conference Conductible	M	M(=)		
Termination Method	M	M(=)		
Conductor Privilege List	O	O(=)		
Conducted-mode Conference Privilege List	O	O(=)		
Non-conducted-mode Conference Privilege List	O	O(=)		
Conference Description	O	O(=)		
Caller Identifier	O	O(=)		
Calling Address	O	O(=)		
Called Address	O	O		
Domain Parameters	M	M	M	M(=)
Quality of Service	M	M	M	M(=)
Local Network Address	O		O	
Conference Priority	O	O(=)		
Conference Mode	O	O(=)		
User Data	O	O(=)	O	O(=)
Result			M	M(=)

Conference Name: Name by which the conference to be created is identified. This consists of a numerical string along with an optional Unicode Row 00 text string, from zero to 255 characters each. If both forms of a Conference Name are used, if a node wishes to join this conference, it may specify either form of the name in the join request. In the join request, a numeric value will necessarily be included in numeric variant of the Conference Name. As a result, use of a text Conference Name including only numeric characters will never be compared against and therefore should not be used – that is, the text variant of the Conference Name should include at least one non-numeric character.

Conference Name Modifier: If the requesting or responding node is already joined to a conference with the same Conference Name (either numerical or text portion) as that included in the request, this parameter shall also be included in the corresponding request or response primitive. The value of this parameter shall be unique among all conferences at the corresponding node which have this Conference Name. This modifier, if included, shall be used as the Called Node Conference Name Modifier parameter in a GCC-Conference-Join request by another node attempting to join the conference through a direct connection with the corresponding node. This modifier is also included in the response to a GCC-Conference-Query directed at this node. This parameter is a numerical string up to 255 digits in length.

Conference ID: Locally-allocated identifier of the newly-created conference. All subsequent references to the conference are made using the Conference ID as a unique identifier. The Conference ID shall be identical with the MCS Domain Selector used locally to identify the MCS Domain associated with the conference.

Convener Password: This optional parameter contains a numeric string, as well as an optional Unicode Row 00 text string, used for the convener to identify itself in later operations, allowing the convener to disconnect and later rejoin the conference, maintaining convener privileges (only when rejoined with a direct connection to the Top GCC Provider). This is the private password which will allow the convener to perform convener-only operations (maximum 255 digits and 255 characters). If this parameter is NULL, then it is not possible for the convener to disconnect and later rejoin maintaining convener privileges. In the join request, a numeric value will necessarily be included in the numeric variant of the Convener Password. As a result, use of a text Convener Password including only numeric characters will never be compared against and therefore should not be used – that is, the text variant of the Convener Password should include at least one non-numeric character.

Password: This is a numeric string, as well as an optional Unicode Row 00 text string, to serve as a Password to enter the conference (maximum 255 digits and 255 characters). If no Password is specified, the conference is not Password protected. In the join request, a numeric value will necessarily be included in the numeric variant of the Password. As a result, use of a text Password including only numeric characters will never be compared against and therefore should not be used – that is, the text variant of the Password should include at least one non-numeric character.

NOTE – If the conference is Password protected, the Node controller must specify a numeric Password, and may also specify a text Password. The numeric Password is required to allow for nodes which have no suitable text entry mechanism. In the case that a text password is used, there is no assumption that the numeric Password that must also be included is generated by the user. It may be more convenient and secure to use a machine generated numeric Password.

Conference Locked: Setting this flag immediately locks a conference, preventing anyone from joining this conference unless they are explicitly added using the GCC-Conference-Invite primitive (or indirectly inviting via the GCC-Conference-Add primitive). To lock a conference at any time after issuing this primitive, the primitive GCC-Conference-Lock may be used. To unlock a conference, GCC-Conference-Unlock may be used.

Conference Listed: The TRUE setting of this flag indicates that this conference may be listed when using the conference-query facility. The FALSE setting of this flag indicates that this conference shall not be listed.

Conference Conductible: The TRUE setting of this flag indicates that this conference may be placed in conducted mode using the GCC-Conductor-Assign primitive. The FALSE setting of this flag indicates that this conference shall be non-conducted only, and attempts to assign a conductor shall be rejected.

Termination Method: This parameter indicates whether the conference shall remain in existence until explicitly terminated by the Convener or convener-designated node using the GCC-Conference-Terminate primitive (manually terminating), or if the conference will terminate when there are no nodes joined to it or if explicitly terminated (automatically terminating).

Conductor Privilege List: This is a list of flags indicating which functions the convener is designating as allowable to be used by the conference conductor, if any. The flags in this list correspond to the operations: GCC-Conference-Terminate, GCC-Conference-Eject-User, GCC-Conference-Add, GCC-Conference-Lock, GCC-Conference-Unlock and GCC-Conference-Transfer.

Conducted-mode Conference Privilege List: This is a list of flags indicating which functions the convener is designating as allowable to be used by any node in a conducted-mode conference. The flags in this list correspond to the same operations as for the above parameter.

Non-conducted-mode Conference Privilege List: This is a list of flags indicating which functions the convener is designating as allowable to be used by any node in a non-conducted-mode conference. The flags in this list correspond to the same operations as for the above parameter.

Conference Description: An optional Unicode text string, up to 255 characters in length, which may be used to describe the conference. This string is maintained by the GCC Providers in the conference to use as part of the response to GCC-Conference-Query requests.

Caller Identifier: Optional Unicode text string (maximum 255 characters) which may be used to identify the calling node to the node at which the conference is to be created. The use of this information at this node is beyond the scope of this Recommendation. It may be used, for example, to allow a user at that node to select among a limited set of participants which are allowed to create conferences. Because this string can be set to any value, it does not necessarily add to the security of a conference to do this, however.

Calling Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Called Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Domain Parameters: Domain parameters to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Quality of Service: Quality of Service parameters to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Local Network Address: If included in either the request or response, the local GCC Provider at the corresponding node shall use this information to include as the Network Address parameter in the Conference Descriptor List sent as part of the response to a GCC-Conference-Query request from another node. In the GCC protocol, this parameter is reflected by ASN.1 structures NetworkAddress and NetworkAddressV2. See Annex B for the description and use of these.

Conference Priority: An optional parameter to specify the priority of a conference. This may be used in some situations to determine whether or not the indication should be accepted. The parameter includes two subparameters: the priority and the scheme. The priority is an integer value ranging from zero to 65535. The scheme indicates the procedures by which the priority value is to be interpreted. At this time, only non-standard schemes are supported. Standardized procedures for interpretation of this parameter are for further study.

Conference Mode: An optional parameter used to specify the mode of a conference. This parameter is used by GCC and the Node Controller at the Top GCC Provider to determine which Node Categories a joining node must fall into before being allowed to join the conference. Conference Modes include the following: *conventional-only*, which allows only Conventional nodes to join the conference; *counted-controlled*, which allows only Counted and Conventional nodes to join the conference; *anonymous-controlled*, which allows only Anonymous and Conventional nodes to join the conference; and *unrestricted-mode*, which allows all three node categories to participate in the

conference. Note that Conventional nodes wishing to join either a counted-controlled or an anonymous-controlled conference must specify a password upon joining.

User Data: Optional user data which may be used for functions outside the scope of this Recommendation such as authentication, billing, etc.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, user rejected, resources not available, rejected for symmetry-breaking, locked conference not supported, Conference Name and Conference Name Modifier already exist, domain parameters unacceptable, domain not hierarchical, lower-layer initiated disconnect, unspecified failure to connect. A negative result in the GCC-Conference-Create confirm does not imply that the physical connection to the node to which the connection was being attempted is disconnected.

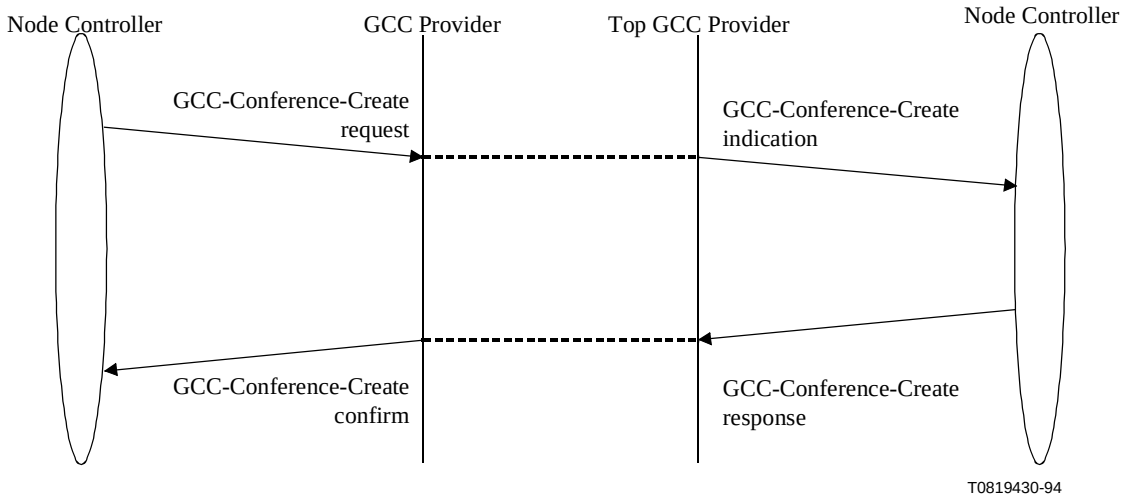

Figure 7-1/T.124 – GCC-Conference-Create – Sequence of primitives

7.1.2.2 GCC-Conference-Query

The GCC-Conference-Query request primitive may be used by a Node Controller to determine what conferences are currently in existence at a particular MCU. Table 7-2 shows the parameters and types of this primitive. Figure 7-2 shows the sequence of events when using this primitive.

Table 7-2/T.124 – GCC-Conference-Query – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Node Type	M	M(=)	M	M(=)
Asymmetry Indicator	C	C(=)	C	C(=)
Conference Descriptor List				C
Wait for Invitation Flag			O	O
No Unlisted Conference Flag			O	O
Calling Address	O	O(=)		
Called Address	O	O		
User Data	O	O(=)	O	O(=)
Result			M	M(=)

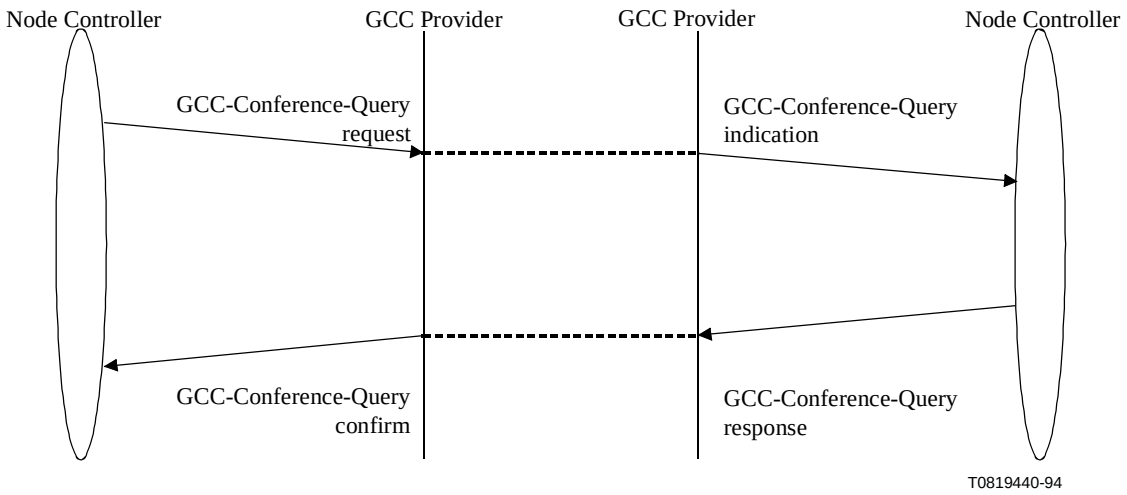

Figure 7-2/T.124 – GCC-Conference-Query – Sequence of primitives

Node Type: The node type is either terminal, MCU, or multiport terminal.

Asymmetry Indicator: This is a field which is needed for certain conference establishment procedures which require knowledge of which node across a single connection was the calling node (the initiator of the physical connection), and which is the called node. This parameter is required in the case of a physical connection between the two nodes (that is, an underlying connection between the two nodes using the PSTN, ISDN, or CSDN cases of Recommendation T.123). Otherwise, this parameter is optional. If a node is aware of its status as the calling or called node, it shall set this indicator to the proper value. In some cases a node may not be certain whether it is the calling or called node. In this case, this parameter shall include a 32-bit random number. If both nodes indicate that they are not certain of their status, the random numbers are used to determine which node should be considered the calling node and which the called node for the purposes of the conference establishment procedure (note that this may not correctly reflect the true calling or called node). The node which transmitted the largest of the two random numbers shall be considered the calling node for the purpose of the conference establishment procedure. If both random numbers are identical, this decision shall be considered unsatisfied and the requester shall re-issue the GCC-Conference-Query request using a different random number (a different random number shall also be used in the resulting response). If both nodes respond with actual values for this parameter (indicating knowledge of whether they are the called or calling node), but the resulting exchange indicates a disagreement (i.e. both think they are the calling node or both think they are the called node), this decision shall also be considered unsatisfied and the requester shall re-issue the GCC-Conference-Query request using a random number (the response shall also use a random number). During the period where a query request remains unconfirmed on one side of a given connection, the value for this parameter in all query requests and query responses issued by this node over the same physical connection shall remain unchanged. The random numbers should be generated to be uniformly distributed over the entire numerical range.

Conference Descriptor List: A variable length list of conference descriptors each indicating an active conference available to be joined. This list does not include conferences which had been designated unlisted at their time of creation. If there are no available conferences, the list contains zero entries. Each conference descriptor includes the parameters shown in Table 7-3.

Calling Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection.

Called Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection.

User Data: Optional user data which may be used for functions outside the scope of this Recommendation such as authentication, billing, etc.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, user-rejected, domain parameters unacceptable, domain not hierarchical, lower-layer initiated disconnect, unspecified failure to connect.

Table 7-3/T.124 – Contents of a Conference Descriptor

Parameter	Description
Conference Name	Conference Name of the conference. If the requesting node wishes to join this conference, this parameter is the value that shall be used in the Conference Name parameter of the GCC-Conference-Join request. This parameter is a numerical string along with an optional Unicode Row 00 text string, maximum 255 characters each. If both forms of the name are given, either form may be specified in the join request.
Conference Name Modifier (conditional)	If at the node returning the response, the conference is known by a name which includes a Conference Name Modifier, this parameter is included. If the requesting node wishes to join this conference, this is the Conference Name Modifier that shall be used in the Called Node Conference Name Modifier parameter of the GCC-Conference-Join request. This parameter is a numerical string up to 255 digits in length.
Conference Description (conditional)	An optional Unicode text string, up to 255 characters in length, used to describe the conference. This parameter may be particularly useful in cases where more than one conference in the Conference Descriptor List has the same Conference Name as a means of distinguishing between these conferences.
Locked/Unlocked	Flag indicating whether the conference is currently locked or unlocked.
Password In The Clear Required	Indicates that the conference is password protected with a password that may be used without encryption in a GCC-Conference-Join request without first being challenged for the password.
Network Address (conditional)	Address information provided to the requesting node. This is provided only if the optional Local Network Address parameter had been included in the connection establishment primitive at that node (either GCC-Conference-Create, GCC-Conference-Join, or GCC-Conference-Invite). In the GCC protocol, this parameter is reflected by ASN.1 structures NetworkAddress and NetworkAddressV2. See Annex B for the description and use of these.
Default Conference Flag	An optional flag indicating whether a particular conference should be considered the default conference to join. In a meet-me conference where the user is expected to manually choose a conference to join from a list, this parameter should be FALSE for all conferences. In a meet-me conference where *a priori* information allows an MCU to determine which conference should be joined (and security reasons, for example, preclude the use of GCC-Conference-Invite), this flag may be set to TRUE for one and only one conference. If this flag is TRUE for more than one conference, or is TRUE for a Locked conference, it should be ignored by the receiver.
Conference Mode	An optional parameter used to specify the mode of a conference. This parameter can be used to determine the appropriate Node Category to specify when joining the conference.

Wait for Invitation Flag: Optional flag that may be set by an MCU. When TRUE this flag indicates that the receiving Node should wait to receive an invitation to a conference and need not attempt to join or create a conference. The absense of this flag, or a FALSE value, does not imply whether or not attempts to join or create a conference should be made. This flag shall not be set to TRUE by a terminal or multiport terminal. No time-limit is implied by a TRUE value for this flag. A node wishing to determine whether the MCU continues to indend it to wait for the invitation may re-issue a GCC-Conference-Query request in order to receive the current setting of this flag in the confirm.

No Unlisted Conference Flag: Optional flag that when TRUE indicates that there are no unlisted conferences available to be joined. Absense of this flag, or a FALSE value, does not imply any information as to whether or not unlisted conferences are available.

7.1.2.3 GCC-Conference-Join

The GCC-Conference-Join request primitive may be used by the Node Controller to cause the local node to join an existing conference. This primitive may be issued at any time. A node may be joined to more than one conference simultaneously. If the conference is Password protected, the Password parameter must contain the correct information for the conference join to be successful. Table 7-4 shows the parameters and types of this primitive. Figure 7-3 shows the sequence of events when using this primitive.

NOTE – It is up to the Node Controller, not the Top GCC Provider, to determine if the Password is correct. It is possible that its definition of correct may be less stringent than a strict character-by-character match. For example, in the case of the text form of a Password, the Node Controller may choose to use a case-insensitive matching criterion.

Table 7-4/T.124 – GCC-Conference-Join – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Conference Name	M			M
Called Node Conference Name Modifier	C			C(=RQ)
Calling Node Conference Name Modifier	C			C(=RQ)
Conference ID		M	M(=)	M
Convener Password	O	O(=)		
Password	C	C(=)	C	C(=)
Caller Identifier	O	O(=)		
Calling Address	O	C(=)		
Called Address	O	C		
Domain Parameters	M			M
Quality of Service	M			M
Password In The Clear Required				M
Conference Locked				M
Conference Listed				M
Conference Conductible				M
Termination Method				M
Conductor Privilege List				C
Conducted-mode Conference Privilege List				C

Table 7-4/T.124 – GCC-Conference-Join – Types of primitives and their parameters (*concluded*)

Parameter	Request	Indication	Response	Confirm
Non-conducted-mode Conference Privilege List				C
Conference Description				C
Local Network Address	O			
Node Category	O	O(=)	O	O(=)
Conference Mode			O	O(=)
User Data	O	O(=)	O	O(=)
Result			M	M(=)

Conference Name: Name of the conference being joined. In the request, this parameter is either a numeric string or a Unicode Row 00 text string, maximum 255 characters. If both the numeric and text parts of the Conference Name were used at the time the conference was created, the Node Controller shall determine whether to send the Conference Name as a text string or numeric string based on its value. A value consisting only of numeric digits shall be specified as a numeric string, while a value including at least one non-numeric character shall be specified as a text string. In the confirm, this parameter includes the full Conference Name including both numerical and text forms, if both were used at the time the conference was created.

Called Node Conference Name Modifier: If the node directly connected to the joining node (the node to which a connection is attempting to be established) has included a Conference Name Modifier as part of the name by which this conference is known, this parameter shall be included in the request primitive and shall indicate the Conference Name Modifier as it is known to the node directly connected to the joining node. This parameter is a numeric string up to 255 digits in length.

Calling Node Conference Name Modifier: If a conference already exists at the node issuing the join request with a name identical with the Conference Name of the conference to be joined, this parameter shall be included in the GCC-Conference-Join request and shall indicate the Conference Name Modifier by which the conference shall be known at the local node. This parameter, if included, shall be different from any Conference Name Modifier already in use for any other conference the local node is currently joined to with the same Conference Name. If used, this parameter becomes the Called Node Conference Name Modifier by which another node attempting to join this conference through a connection to the local node refers to this conference. This modifier is also included as the Conference Name Modifier parameter in any GCC-Conference-Query response from this node (if the conference is listed). This parameter is a numeric string up to 255 digits in length.

Conference ID: At the Top GCC Provider (the indication/response primitives), this parameter is the Conference ID of the conference to which the requesting node wishes to join. In the confirm primitive, this parameter is returned by GCC indicating the locally allocated ID by which all subsequent references to the conference are indicated. The Conference ID shall be identical with the MCS Domain Selector used locally to identify the MCS Domain associated with the conference.

Convener Password: This is an optional parameter which is either a numeric string or a Unicode Row 00 text string which may be used by a conference convener rejoining a conference after disconnecting (maximum 255 digits or characters). If this identifier matches the corresponding identifier used when the conference was created, the joining node is given the privileges of the convener, but only if joining with a direct connection to the Top GCC Provider (rather than via an

intermediate MCU). The convener, with the correct Convener Password, is allowed to join even conferences which are locked. If the conference is Password protected, the correct Password must be given in addition to the Convener Password to successfully join the conference. If the Convener Password is present but does not match, the request to join shall be rejected. The criterion used to determine if the Convener Password matches the originally specified value is determined by the Node Controller. The Node Controller shall determine whether to send the Convener Password as a text string or numeric string based on its value. A value consisting only of numeric digits shall be specified as a numeric string, while a value including at least one non-numeric character shall be specified as a text string.

Password: The password parameter is used to gain access to a password protected conference. In the request form of this primitive, this parameter shall only contain a password if a result of Challenge Response Required has been received in a previous GCC-Conference-Join confirm for this conference or if the Password In The Clear Required parameter is set in the Conference Descriptor for this conference in a previous GCC-Conference-Query confirm. In the case of a Password In The Clear, this is either a numeric string or a Unicode Row 00 text string (maximum 255 digits or characters). A text string may only be used if a text password was defined at the time of conference creation in addition to the numeric password. The Node Controller shall determine whether to send the Password as a text string or numeric string based on its value. A value consisting only of numeric digits shall be specified as a numeric string, while a value including at least one non-numeric character shall be specified as a text string. In the case of an encrypted password, this parameter contains the password encoded using one of the algorithms specified in the previously received challenge. In the case of a password sent in response to a challenge (either in the clear or encrypted), this parameter shall also include a tag which shall be identical to the tag received in the challenge. In the case of a password initiated in response to the Password In The Clear Required flag in the GCC-Conference-Query indication, no tag is required. In the request form of this primitive, this parameter may also include a challenge to the receiving node. There are no restrictions on when a challenge may be included in this parameter.

In the response form of this primitive, this parameter may contain a challenge to the requester indicating that a password is required for joining this conference. For this case, this parameter includes information specifying which forms of the password will be accepted (either in the clear, and/or encrypted via a list of non-standard encryption algorithms), an integer tag used to identify this challenge, and any additional information required for encryption. Should this parameter contain challenge, the result parameter of this primitive shall be set to Challenge Response Required. In this case, no connection is established by this exchange. This parameter in the response form of this primitive may also include a Password (either in the clear or encrypted) in response to a challenge by the requesting node.

If this parameter in the indication is not in a format satisfactory to the receiving node, that node should issue a response with Invalid Challenge Response as the result. If this parameter in the indication is of the correct format, but does not contain the correct password, the response should include Invalid Password as the result.

Caller Identifier: Optional Unicode text string (maximum 255 characters) which may be used to identify the calling node to the node at which the Top GCC Provider resides. The use of this information at this node is beyond the scope of this Recommendation. It may be used, for example, to allow a user at that node to select among a limited set of participants to allow into the conference. Because this string can be set to any value, it does not necessarily add to the security of a conference to do this, however.

Calling Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter. The presence of this parameter in the indication is conditional. It is present only when the Node initiating the GCC-Conference-Join request is attempting to establish a direct connection to the Node containing the Top GCC Provider for the conference being joined.

Called Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter. The presence of this parameter in the indication is conditional. It is present only when the Node initiating the GCC-Conference-Join request is attempting to establish a direct connection to the Node containing the Top GCC Provider for the conference being joined.

Domain Parameters: Domain parameters to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Quality of Service: Quality of Service parameters to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Password In The Clear Required: This is a Boolean parameter that indicates that the conference is password protected with a password that may be used without encryption in a GCC-Conference-Join request without first being challenged for the password. If the joining node is an MCU, this information is used by the GCC Provider in generating the response to a GCC-Conference-Query indication. This information shall be used by the Node Controller in determining whether or not the Password parameter may be included in a GCC-Conference-Transfer request.

Conference Locked: This flag indicates whether or not the joined conference is locked or unlocked. If the joining node is an MCU, this information shall be used in generating the response to a GCC-Conference-Query indication.

Conference Listed: This flag indicates whether or not the joined conference is listed or unlisted. If the joining node is an MCU, this information shall be used in generating the response to a GCC-Conference-Query indication. In particular, conferences which are indicated as being unlisted shall not be listed in the conference list provided with the GCC-Conference-Query response.

Conference Conductible: The TRUE setting of this flag indicates that this conference may be placed in conducted mode using the GCC-Conductor-Assign primitive. The FALSE setting of this flag indicates that this conference shall be non-conducted only, and attempts to assign a conductor shall be rejected.

Termination Method: This flag indicates the termination rule for the joined conference. The conference may be either manually or automatically terminating.

Conductor Privilege List: This is a list of flags indicating which functions the convener has designated as allowable to be used by the conference conductor, if any. The flags in this list correspond to the operations: GCC-Conference-Terminate, GCC-Conference-Eject-User, GCC-Conference-Add, GCC-Conference-Lock, GCC-Conference-Unlock and GCC-Conference-Transfer.

Conducted-mode Conference Privilege List: This is a list of flags indicating which functions the convener has designated as allowable to be used by any node in a conducted-mode conference. The flags in this list correspond to the same operations as for the above parameter.

Non-conducted-mode Conference Privilege List: This is a list of flags indicating which functions the convener has designated as allowable to be used by any node in a non-conducted-mode conference. The flags in this list correspond to the same operations as for the above parameter.

Conference Description: This parameter is a Unicode text string, up to 255 characters in length, describing the conference being joined. It is present in the confirm primitive only if included at the time of conference creation.

Local Network Address: If included in the request, the local GCC Provider at the corresponding node shall use this information to include as the Network Address parameter in the Conference Descriptor List sent as part of the response to a GCC-Conference-Query request from another node. In the GCC protocol, this parameter is reflected by ASN.1 structures NetworkAddress and NetworkAddressV2. See Annex B for the description and use of these.

Node Category: If included in the request, the local GCC Provider at the node being joined can use this to determine the joining node's preferred node category. This may be overridden in the join response if the suggested category is not acceptable by the node being joined. If not specified, the default requested category is Conventional.

Conference Mode: An optional parameter used to specify the mode of a conference. This parameter informs the joining node which Node Categories are allowed to participate in the conference. Conference Modes include: *conventional-only*, which allows only Conventional nodes to join the conference; *counted-controlled*, which allows only Counted and Conventional nodes to join the conference; *anonymous-controlled*, which allows only Anonymous and Conventional nodes to join the conference; and *unrestricted-mode*, which allows all three node categories to participate in the conference. Note that Conventional nodes wishing to join either a counted-controlled or an anonymous-controlled conference must specify a password upon joining.

User Data: Optional user data which may be used for functions outside the scope of this Recommendation such as authentication, billing, etc.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, user-rejected, invalid conference, invalid Password, Challenge Response required, invalid Challenge Response, invalid Convener Password, domain parameters unacceptable, domain not hierarchical, lower-layer initiated disconnect, unspecified failure to connect. A negative result in the GCC-Conference-Join confirm does not imply that the physical connection to the node to which the connection was being attempted is disconnected.

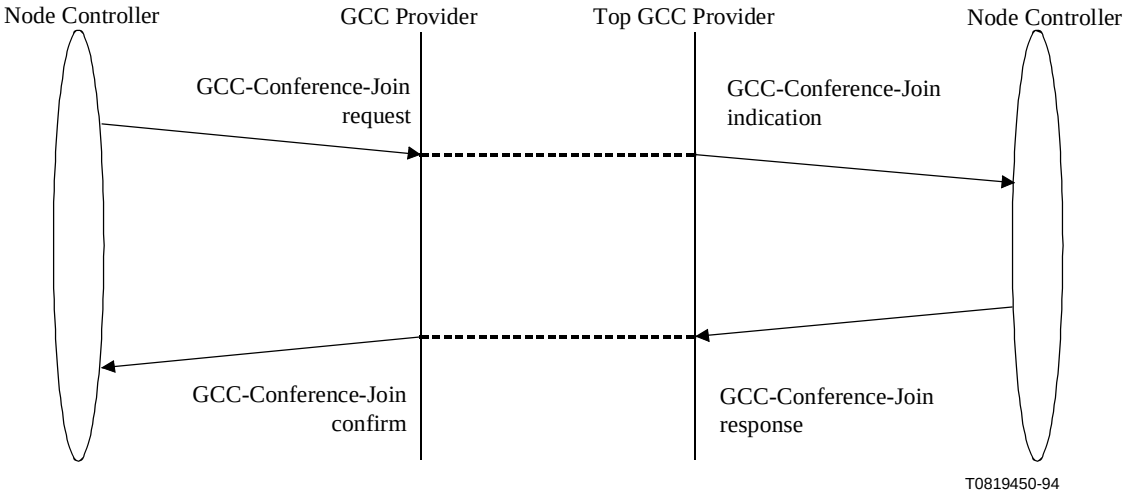

Figure 7-3/T.124 – GCC-Conference-Join – Sequence of primitives

7.1.2.4 GCC-Conference-Invite

The GCC-Conference-Invite request primitive may be used by a Node Controller to invite a node to join a conference. This primitive may be used as a result of GCC-Conference-Add indication, or may be issued directly by the inviting node. If the add is successful, the adding MCU invites the added node to join the conference by issuing the GCC-Conference-Invite request primitive. Note that even if the conference is Password protected, no Password is needed by the invited node in order to accept the invitation to join the conference. Table 7-5 shows the parameters and types of this primitive. Figure 7-4 shows the sequence of events when using this primitive.

Table 7-5/T.124 – CC-Conference-Invite – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Conference ID	M	M	M(=IN)	M(=RQ)
Conference Name		M		
Conference Name Modifier			C	
Caller Identifier	O	O(=)		
Calling Address	O	O(=)		
Called Address	O	O		
Domain Parameters		M	M	
Quality of Service		M	M	
Password In The Clear Required		M		
Conference Locked		M		
Conference Listed		M		
Conference Conductible		M		
Termination Method		M		
Conductor Privilege List		C		
Conducted-mode Conference Privilege List		C		
Non-conducted-mode Conference Privilege List		C		
Conference Description		C		
Local Network Address			O	
Conference Priority	O	O(=)		
Node Category	O	O(=)		
Conference Mode	O	O(=)		
User Data	O	O(=)	O	O(=)
Result			M	M(=)

Conference ID: Identifier of the conference to which the primitive refers.

Conference Name: The name of the conference as specified in the Conference Profile. If the Conference Name includes both numerical and text forms, both forms shall be included in this parameter.

Conference Name Modifier: If a conference already exists at the node issuing the invite response with a name identical with the Conference Name of the conference to be joined, this parameter shall be included in the GCC-Conference-Invite response and shall indicate the Conference Name Modifier by which the conference shall be known at the local node. This parameter, if included, shall be different from any Conference Name Modifier already in use for any other conference the local node is currently joined to with the same Conference Name. If used, this parameter becomes the Calling Node Conference Name Modifier by which another node attempting to join this conference through a connection to the local node refers to this conference. This name is also included as the Conference Name Modifier parameter in any GCC-Conference-Query response from this node (if the conference is listed). This is a numerical string up to 255 digits in length.

Caller Identifier: Optional Unicode text string (maximum 255 characters) which may be used to identify the calling node to the node at which the conference is to be created. The use of this information at this node is beyond the scope of this Recommendation. It may be used, for example, to allow a user at that node to select among a limited set of participants which are allowed to invite this node into a conference. Because this string can be set to any value, it does not necessarily add to the security of a conference to do this, however.

Calling Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Called Address: Optional address to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Domain Parameters: Domain parameters to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Quality of Service: Quality of Service parameters to be included in the MCS-Connect-Provider primitive on establishing an MCS connection. See Recommendation T.122 for the interpretation of this parameter.

Password In The Clear Required: This is a Boolean parameter that indicates that the conference is password protected with a password that may be used without encryption in a GCC-Conference-Join request without first being challenged for the password. If the joining node is an MCU, this information is used by the GCC Provider in generating the response to a GCC-Conference-Query indication. This information shall be used by the Node Controller in determining whether or not the Password parameter may be included in a GCC-Conference-Transfer request.

Conference Locked: This flag indicates whether or not the joined conference is locked or unlocked. If the joining node is an MCU, this information shall be used in generating the response to a GCC-Conference-Query indication.

Conference Listed: This flag indicates whether or not the joined conference is listed or unlisted. If the joining node is an MCU, this information shall be used in generating the response to a GCC-Conference-Query indication. In particular, conferences which are indicated as being unlisted shall not be listed in the conference list provided with the GCC-Conference-Query response.

Conference Conductible: The TRUE setting of this flag indicates that this conference may be placed in conducted mode using the GCC-Conductor-Assign primitive. The FALSE setting of this flag indicates that this conference shall be non-conducted only, and attempts to assign a conductor shall be rejected.

Termination Method: This flag indicates the termination rule for the joined conference. The conference may be either manually or automatically terminating.

Conductor Privilege List: This is a list of flags indicating which functions the convener has designated as allowable to be used by the conference conductor, if any. The flags in this list correspond to the operations: GCC-Conference-Terminate, GCC-Conference-Eject-User, GCC-Conference-Add, GCC-Conference-Lock, GCC-Conference-Unlock and GCC-Conference-Transfer.

Conducted-mode Conference Privilege List: This is a list of flags indicating which functions the convener has designated as allowable to be used by any node in a conducted-mode conference. The flags in this list correspond to the same operations as for the above parameter.

Non-conducted-mode Conference Privilege List: This is a list of flags indicating which functions the convener has designated as allowable to be used by any node in a non-conducted-mode conference. The flags in this list correspond to the same operations as for the above parameter.

Conference Description: This parameter is a Unicode text string, up to 255 characters in length, describing the conference. It is present in the confirm primitive only if included at the time of conference creation.

Local Network Address: If included in the response, the local GCC Provider at the corresponding node shall use this information to include as the Network Address parameter in the Conference Descriptor List sent as part of the response to a GCC-Conference-Query request from another node. In the GCC protocol, this parameter is reflected by ASN.1 structures NetworkAddress and NetworkAddressV2. See Annex B for the description and use of these.

Conference Priority: An optional parameter to specify the priority of a conference. This may be used in some situations to determine whether or not the indication should be accepted. The parameter includes two subparameters: the priority and the scheme. The priority is an integer value ranging from zero to 65535. The scheme indicates the procedures by which the priority value is to be interpreted. At this time, only non-standard schemes are supported. Standardized procedures for interpretation of this parameter are for further study.

Node Category: This informs the node that is being invited of the node category into which it falls. The invited node has no say in this selection. The invite should fail when an older protocol node that does not understand Node Categories is invited. When this occurs, the result should state that the node category parameter was unacceptable.

Conference Mode: An optional parameter used to specify the mode of a conference. This parameter informs the invited node which Node Categories are allowed to participate in the conference. Conference Modes include: *conventional-only*, which allows only Conventional nodes to join the conference; *counted-controlled*, which allows only Counted and Conventional nodes to join the conference; *anonymous-controlled*, which allows only Anonymous and Conventional nodes to join the conference; and *unrestricted-mode*, which allows all three node categories to participate in the conference. Note that Conventional nodes wishing to join either a counted-controlled or an anonymous-controlled conference must specify a password upon joining.

User Data: Optional user data which may be used for functions outside the scope of this Recommendation such as authentication, billing, etc.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, user rejected, invalid conference, domain parameters unacceptable, domain not hierarchical, lower-layer initiated disconnect, unspecified failure to connect and, node category unacceptable. A negative result in the GCC-Conference-Invite confirm does not imply that the physical connection to the node to which the connection was being attempted is disconnected.

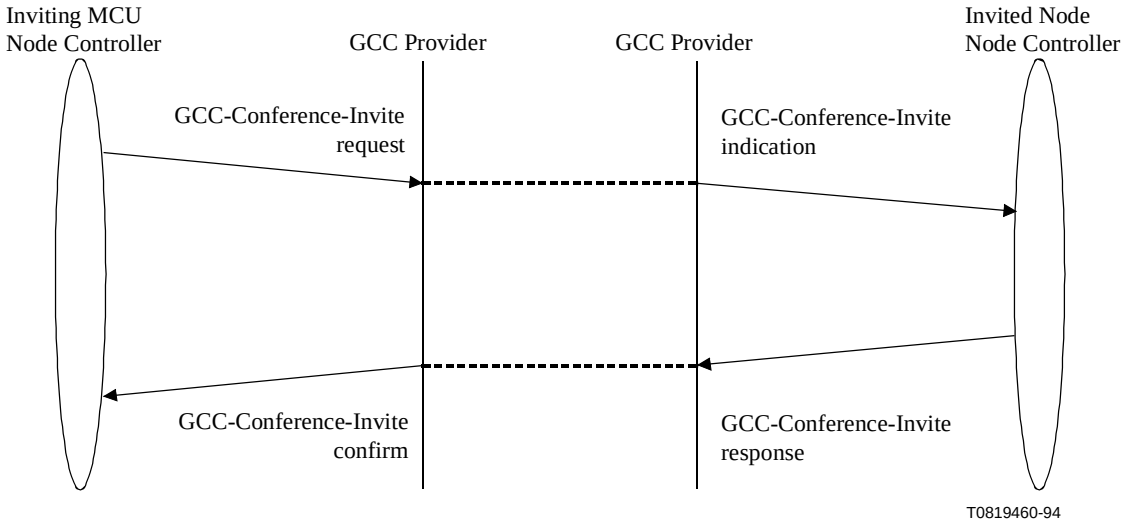

Figure 7-4/T.124 – GCC-Conference-Invite – Sequence of primitives

7.1.2.5 GCC-Conference-Add

The GCC-Conference-Add request primitive may be used by a Node Controller to add a single additional node to an existing conference by requesting that an MCU dial out the specified node. This primitive is only valid if issued by the Convener or a convener-designated node. If the Adding MCU is specified, the sequence of primitives is direct between the requester and the adding MCU. If the Adding MCU is not specified, the indication is issued from the Top GCC Provider. In this case, if a port capable of performing the add is available at this node, the exchange is also direct. If the Node Controller at the Top GCC Provider maintains or has access to a central database of port information for all MCUs in the conference, it may perform the add indirectly by issuing another GCC-Conference-Add request, specifying an Adding MCU (even if the top MCU is not a convener-designated privileged node for this operation).

Once provision has been made to set up the physical connection to the Added Node (if needed), the GCC-Conference-Invite request primitive shall be issued by the Adding MCU to invite the Added Node into the conference. If the Adding MCU already has a physical connection to the Added Node, the Adding MCU may issue the GCC-Conference-Invite request directly to the Added Node without establishing a new physical connection. If this connection had been established without knowledge of the Network Address of the Added Node (e.g. the Added Node had dialed into the Adding MCU to connect to a separate conference), the Network Address parameters in the Conference Roster may be used to provide the information needed to match the Network Address of an already connected node with that of a node to be added. Table 7-6 shows the parameters and types of this primitive. Figures 7-5 and 7-6 show the sequence of events when using this primitive in the direct and indirect cases, respectively.

Table 7-6/T.124 – GCC-Conference-Add – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Conference ID	M	M	M(=IN)	M(=RQ)
Network Address	M	M(=)	M(=)	M(=)
Adding MCU Node ID	O			
Requesting Node ID		M	M(=)	
Node Category	O	O(=)		
User Data	O	O(=)	O	O(=)
Result			M	M(=)

Conference ID: Identifier of the conference to which the primitive refers.

Network Address: This parameter consists of a list of one or more connection descriptions, each representing a portion of the total logical connection toward the added node. A particular connection description provides information on the network connection to be established toward the added node (type of network, type of circuit, network address), and optionally the possible ways to aggregate digital circuits when relevant, the multimedia profiles that the adding MCU may operate over the connection, and the media (i.e. audio, video, data) concerned with the connection. Parameters composing each element of the network address are presented in 7.1.2.5.1. In the GCC protocol, this parameter is reflected by ASN.1 structures NetworkAddress and NetworkAddressV2. See Annex B for the description and use of these.

The NSAP part of a transport address may encapsulate within itself certain patterns that by convention suggest the choice of a switched connection or a connectionless network protocol. In such cases, the listed element stands alone as an independent alternative to any aggregated channel or non-standard addresses that may also be present. On the other hand, a transport address may instead be incomplete and may depend implicitly on the shared use of a data channel opened within a multimedia multiplex of aggregated channels, as specified in Recommendation T.123 or in some non-standard way. In these cases, a locally-specified NSAP address and/or transport selector may be needed to steer new connections in the data channel to the correct termination point. NSAP addresses are conveyed, according to Recommendation T.123, as Calling or Called Party Subaddress information elements in SETUP, and transport selectors as TSAP identifiers in an X.224 connection request.

Adding MCU Node ID: This is an optional parameter which may be used to specify the particular MCU node from which the called node is to be added. If this parameter is not specified, the called node may be added from any available MCU in the conference. In this case, the indication occurs from the Top GCC Provider. The node controller at that MCU may either service it locally, or issue another request to a specific MCU.

Requesting Node ID: Node ID of the requesting node.

Node Category: This informs the node that is being invited of the node category into which it falls. The invited node has no say in this selection. The invite should fail when an older protocol node that does not understand Node Categories is invited. When this occurs, the result should state that the node category parameter was unacceptable.

User Data: Optional user data which may be used for functions outside the scope of this Recommendation such as authentication, billing, etc.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, invalid adding MCU, not convener or convener-designated node, invalid network type, invalid network address, added node busy, network busy, connection unsuccessful, no ports available.

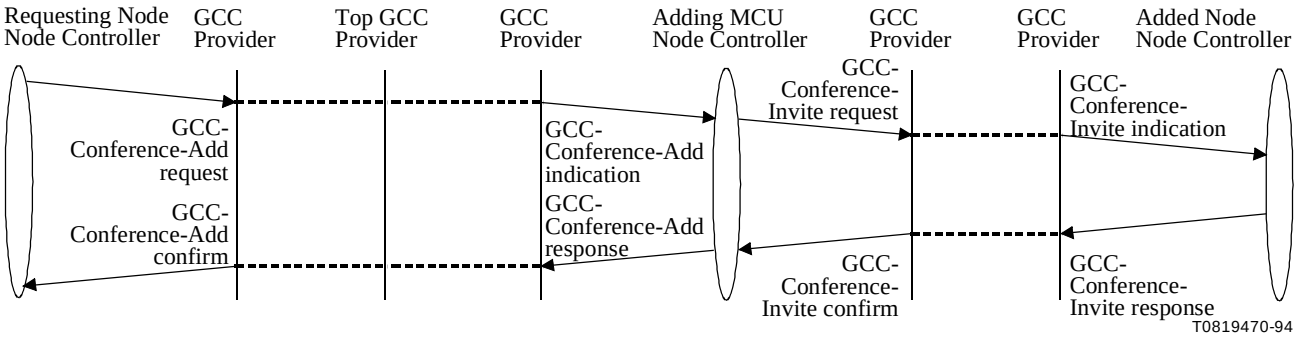

Figure 7-5/T.124 – GCC-Conference-Add (direct case) – Sequence of primitives

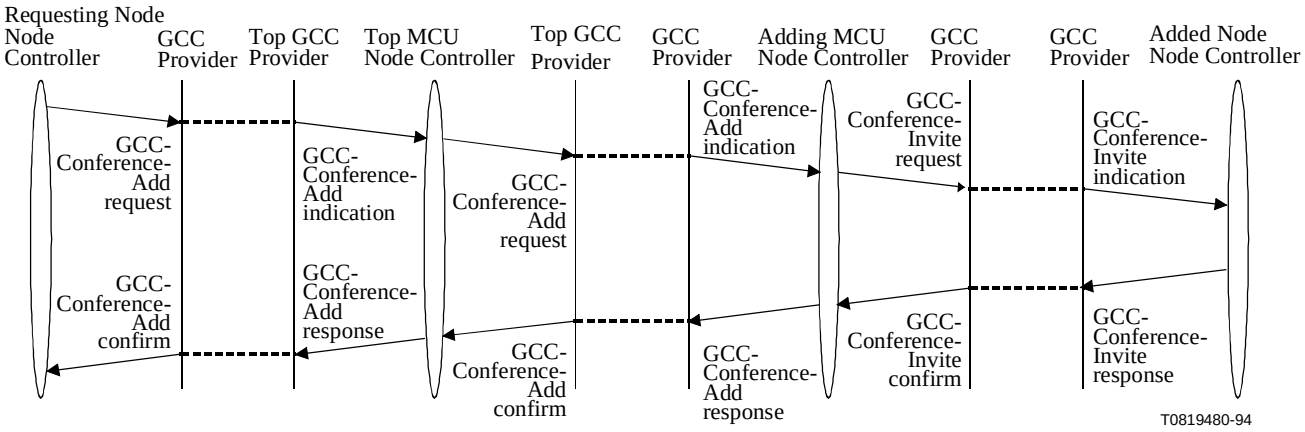

Figure 7-6/T.124 – GCC-Conference-Add (indirect case) – Sequence of primitives

7.1.2.5.1 Network Address Element

The structure of a Network Address element is presented by Table 7-7.

Table 7-7/T.128 – Structure of a Network Address element

Parameter	Description
Network Connection	This parameter consists in either of a single connection description or the description of a connection formed by aggregation of ISDN or CSDN digital circuits. This parameter is described in details through points 1) and 2) hereafter.
Profiles	This parameter is a list of one or more profiles that may be operated over the connection. Each profile may describe a basic transfer mode such as simple telephony or a more complex suites of protocols such as H.320/T.120. Possible values are "*Speech*", "*3 kHz telephony*", "*7 kHz telephony*", "*Voice-band*", "*Frame Relay*", *"T.123 basic profile for PSTN/GSTN*", "*T.123 basic profile for PSDN*", "*T.123 basic profile for B-ISDN*", "*H.310*", "*H.320*", "*H.321*", "*H.322*", "*H.323*", "*H.324*", "*H.324m*", "*ASVD*", "*DSVD*", "*DSM-CC Download Profile*" and "*Non Standard*". For multimedia profiles (H.32x, ASVD and DSVD), a boolean flag is associated to indicate whether the T.120 suites of protocols shall be operated for the data portion of the multiplex. The exact structures of non-standard profiles are user-defined.
Media Concerned	This is a list of three boolean flags that gives the list of media (i.e. audio, video and data) that are concerned by the connection.

1) *Single connections*

When *Network Connection* consists in a single connection, it may take different forms, depending on whether or not the connection is accurately described, and, when it is the case, on the type of network to which the added node is connected. Networks considered by this version of T.124 are GSTN, ISDN, CSDN, PSDN and ATM networks.

1a) GSTN connections:

For GSTN networks, *Network Connection* simply consists in an *Extended E.164 Network Address*, where optional parameter *Sub-Address* is not applicable. The structure of an extended E.164 network address is presented by Table 7-7-2 below:

Table 7-7-2/T.124 – Structure of an extended E.164 network address

Parameter	Description
International Number	This is a string of digits, up to 16 digits in length, which represents the full international number of the node to be added.
Sub-Address (optional)	This is an optional parameter, valid only in the case of ISDN transfer modes, which represents the ISDN sub-address of the node to be added. This is a string of digits, up to 40 digits in length.
Extra Dialling String (optional)	This is an optional parameter which indicates that additional information is needed to reach the node to be added once the physical connection has been established. In the case of a speech or voice-band data connection, for example, this may represent DTMF tones to be transmitted over the voice channel once it has been established. Alternatively, the extra dialling may represent a virtual private network number. This is a string up to 255 characters which may be either the digits 1 through 9, the "#" character, the "*" character, or the "," (comma) character. The comma character is meant to represent a one-second delay the Adding MCU is to insert prior to the characters which follow.

1b) ISDN connections:

For ISDN networks, *Network Connection* consists in a structure of two elements, the first being a list of circuit types that the added node and its access network are capable of operating, the second being an extended E.164 network address that shall be dialed to reach the added node. Table 7-7-3 shows the structure of a single ISDN connection descriptor:

Table 7-7-3/T.124 – Structure of a single connection – ISDN networks case

Parameter	Description
Circuit Types	This is a list of one or more types of circuits that may be switched to reach the added node. Possible types correspond to the codepoints of octet 4 of IE Bearer Capability of Rec. Q.931. These are "*digital 64k channel*" (B-Channel), "*2 x digital 64k channel*", "*digital 384k channel*" (H0 channel), "*digital 1536k channel*" (H11 channel), "*digital 1920k channel*" (H12 channel) and "*multirate based 64k channels*". For the latest option, an integer value is indicated, the *multiplier*, that gives the effective rate. When more than one types is passed, the choice is left to the discretion of the adding MCU.
Extended E.164 Network Address	This parameter has the format described by Table 7-7-2.
High Layer Compatibility Information (optional)	This is an optional parameter which indicates the mode of operation this portion of the connection is to use. This information is required for connections made in some countries. The modes of operation are one or more of telephony at 3 kHz bandwidth, telephony at 7 kHz bandwidth, videotelephony, videoconferencing, audiographics, audiovisual, or multimedia. If more than one of these is selected, this indicates that the Adding MCU may use one of the indicated modes at its discretion. The definitions of these choices may depend on the country of operation of the Adding MCU. The codepoints defined for this parameter match those defined for Information Element Higher Layer Compatibility of the ISDN signalling protocol (Rec. Q.931).

1c) CSDN connections:

For CSDN networks, *Network Connection* consists in a structure of two elements, the first being a list of circuit types that the added node and its access network are capable of operating, the second being an extended E.164 network address that shall be dialed to reach the added node. Table 7-7-4 shows the structure of a single CSDN connection descriptor:

Table 7-7-4/T.124 – Structure of a single connection – CSDN networks case

Parameter	Description
Circuit Types	This is a list of one or two types of circuits that may be switched to reach the added node. Possible types are "*digital 56k channel*" and "*digital 64k channel*". When both types are passed, the choice is left to the discretion of the adding MCU.
Extended E.164 Network Address	This parameter has the format described by Table 7-7-2.

1d) ATM connections:

For ATM networks, *Network Connection* consists in a structure of two elements, the first being either of an extended E.164 network address, a NSAP address, or a non-standard address format assumed to be understood by the adding MCU, and the second being an optional integer value giving the maximum possible transfer rate toward the added node, expressed in ATM cells per seconds.

NOTE – Certain private ATM networks use the E.164 standard address format coupled with an NSAP to reach enpoints. In that case, optional parameter Extra Dialling String may be used to convey the NSAP.

1e) PSDN connections:

For PSDN networks, *Network Connection* simply consists in either of the extended E.164 network address described by Table 7-7-2, a *Transport Address Descriptor*, or a non standard address format assumed to be understood by the adding MCU. Table 7-7-5 gives the structure of a transport address descriptor:

Table 7-7-5/T.124 – Structure of transport address descriptor

Parameter	Description
NSAP Address	This is an Octet String of up to 20 octets in length which is the preferred binary encoding (per A.8.3.1/X.213) of the Network Service Access Point address of the node to be added.
Transport Selector (optional)	This is an optional parameter which may be used to select the Transport Service Access Point at the node to be added.

1f) Undescribed connections:

In this case, *Network Connection* consists in either of the extended E.164 network address described by Table 7-7-2, a *Transport Address Descriptor*, or a non-standard address format assumed to be understood by the adding MCU.

2) *ISDN or CSDN digital circuits aggregation*

When parameter *Network Connection* of Table 7-7 is used for aggregated CSDN or ISDN channels, it has a structure of two elements, the first being a list of digital circuits that shall be aggregated together, the second being an optional list of channel aggregation algorithms usable for aggregating the circuits. Table 7-7-6 details this structure:

Table 7-7-6/T.124 – Structure of an aggregated channels connection

Parameter	Description
Aggregated Connections	This is a list of one ore more elements, each having either of the structures described in Tables 7-7-3 and 7-7-4. When one or more elements of the list gives multiple circuit types, the choice is left to the adding MCU for the final combination.
Aggregation Methods (optional)	This is a list of one or more standard or non-standard channel aggregation algorithms that may be used to aggregate digital circuits together. Possible standard algorithms are "H.221", "H.244" and "ISO/IEC 13871". Non-standard algorithms are supposed to be understood by the adding MCU. This parameter is optional. If not supplied, the choice is left to the adding MCU. NOTE – The actual algorithm is implicit for certain profiles that may be listed by parameter *Profiles* of Table 7-7 (ex. Rec.H.320).

7.1.2.6 GCC-Conference-Lock

The GCC-Conference-Lock request primitive may be used by a Node Controller to lock a conference, preventing other nodes from dialling into the conference at all. This primitive is valid only if issued by the Convener or a convener-designated node. While locked, participants may be added to a conference only by using the GCC-Conference-Invite primitive (or indirectly inviting via the GCC-Conference-Add primitive). The order of GCC-Conference-Lock and GCC-Conference-Unlock primitives exchanged between a node and the Top GCC Provider is preserved. Table 7-9 shows the parameters and types of this primitive. Figure 7-7 shows the sequence of events when using this primitive.

Table 7-9/T.124 – GCC-Conference-Lock – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Conference ID	M	M	M(=IN)	M(=RQ)
Source Node ID		M	M(=)	
Result			M	M(=)

Conference ID: Identifier of the conference to which the primitive refers.

Source Node ID: Node ID of the requesting node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, function not available, already locked, not convener or convener-designated node.

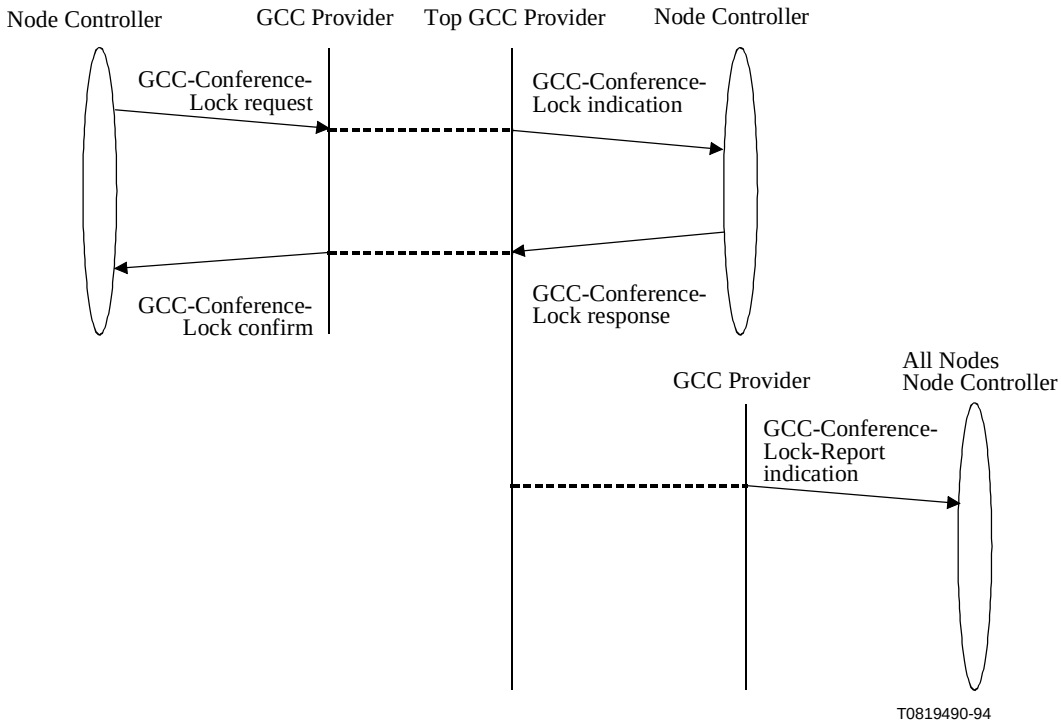

Figure 7-7/T.124 – GCC-Conference-Lock – Sequence of primitives

7.1.2.7 GCC-Conference-Unlock

The GCC-Conference-Unlock request primitive may be used by a Node Controller to unlock a previously locked conference. Unlocking a conference allows other nodes to join the conference by dialling into it in a meet-me style. If the conference was created specifying the use of Password protection, the Password is still required for any participant attempting to join the unlocked conference. This primitive is valid only if issued by the Convener or a convener-designated node. The order of GCC-Conference-Lock and GCC-Conference-Unlock primitives exchanged between a node and the Top GCC Provider is preserved. Table 7-10 shows the parameters and types of this primitive. Figure 7-8 shows the sequence of events when using this primitive.

Table 7-10/T.124 – GCC-Conference-Unlock – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Conference ID	M	M	M(=IN)	M(=RQ)
Source Node ID		M	M(=)	
Result			M	M(=)

Conference ID: Identifier of the conference to which the primitive refers.

Source Node ID: Node ID of the requesting node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, function not available, already unlocked, not convener or convener-designated node.

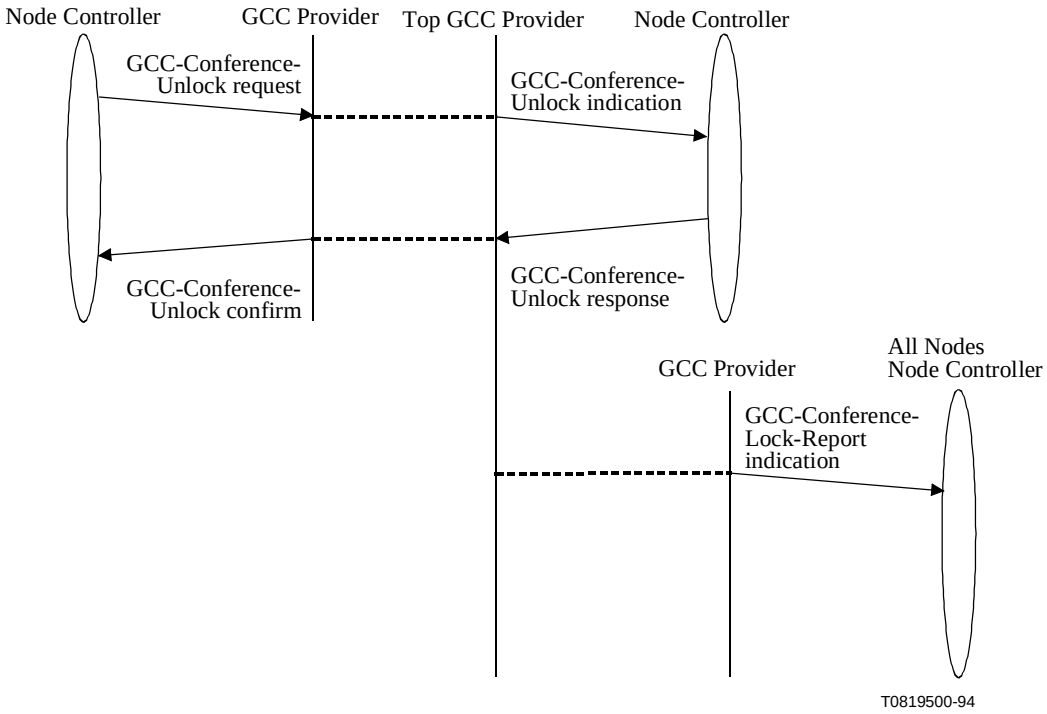

Figure 7-8/T.124 – GCC-Conference-Unlock – Sequence of primitives

7.1.2.8 GCC-Conference-Lock-Report

The GCC-Conference-Lock-Report indication primitive is issued to the Node Controller at all nodes in a conference as a result of a successful GCC-Conference-Lock request or a successful GCC-Conference-Unlock request. Figures 7-7 and 7-8 show the sequence of events leading to the use of this primitive. See also Table 7-11.

Table 7-11/T.124 – GCC-Conference-Lock-Report – Types of primitives and their parameters

Parameter	Indication
Conference ID	M
Locked/Unlocked	M

Conference ID: Identifier of the MCS Domain corresponding to the locked conference.

Locked/Unlocked: Flag indicating if the conference has switched into locked or unlocked mode.

7.1.2.9 GCC-Conference-Disconnect

The GCC-Conference-Disconnect request primitive is used by a Node Controller to disconnect itself from a conference. Disconnecting from a conference does not imply disconnecting the corresponding physical connection. Once disconnected from the conference, a terminal may then join another conference. If GCC detects that a node has abnormally disconnected from a conference (e.g. the physical call has been disconnected), it shall send a GCC-Conference-Disconnect indication to all remaining nodes in the conference. Table 7-12 shows the parameters and types of this primitive. Figures 7-9 and 7-13 show the sequence of events when using this primitive for client-initiated disconnects. Figure 7-11 shows the case of a GCC-initiated abnormal disconnect.

Table 7-12/T.124 – GCC-Conference-Disconnect – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Reason for disconnect		M	
Disconnecting Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Reason for disconnect: Indication of the reason for disconnecting from the conference. Either user-initiated, ejected node, or unknown.

Disconnecting Node ID: Node ID corresponding to the disconnected node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

NOTE – If it is necessary to allow a Node Controller to disconnect from a conference prior to completion of the connection establishment (e.g. prior to reception of a GCC-Conference-Join confirm), it would be necessary to provide a local means of association of the conference to be disconnected with the conference being established. This is because the Conference ID is not known by the Node Controller until after creating or joining a conference. The mechanism for doing so is a local matter and is beyond the scope of this Recommendation.

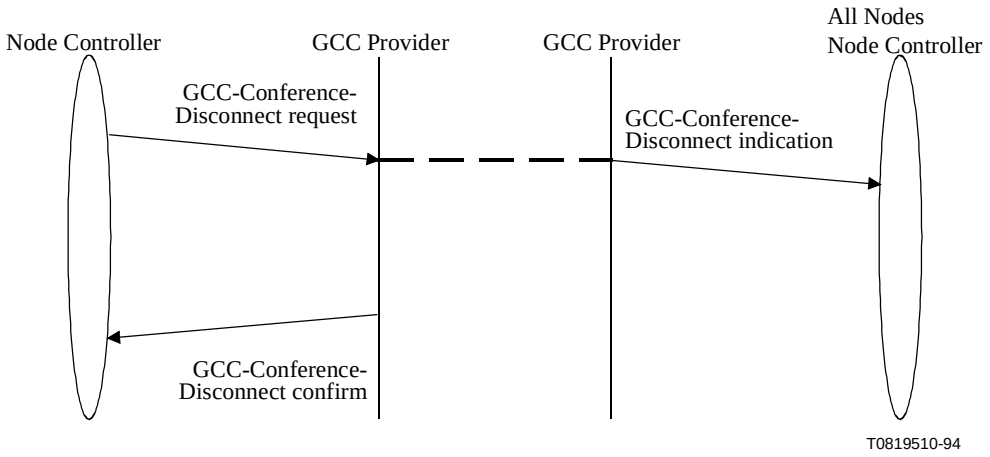

Figure 7-9/T.124 – GCC-Conference-Disconnect (client-initiated) – Sequence of primitives

7.1.2.10 GCC-Conference-Terminate

The GCC-Conference-Terminate request primitive is used by a Node Controller to terminate an entire conference. This primitive shall only be issued by the conference Convener or a convener-designated node. Terminating a conference does not imply termination of the corresponding physical connection. Table 7-13 shows the parameters and types of this primitive. Figures 7-10, 7-11 and 7-12 show the sequence of events when using this primitive for client-initiated and GCC-initiated cases. The GCC-initiated case can result either from abnormal termination at the lower layers of the protocol, or at the Top GCC Provider, if the conference had been created as an automatically terminating conference and all other nodes have disconnected from the conference.

NOTE – In most cases, reception of the GCC-Conference-Terminate indication implies that the entire conference has been terminated – i.e. all members have been disconnected. In the case that error termination is given as the reason code, this primitive implies that the local node has been unexpectedly removed from the conference, but does not necessarily imply that the entire conference has been terminated.

Table 7-13/T.124 – GCC-Conference-Terminate –
Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Reason for termination	O	O(=)	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Reason for termination: Indication of the reason for termination of the conference. This contains one of a list of possible reasons: requested normal termination, requested timed conference termination, no more participants in automatically terminating conference, error termination.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, not convener or convener-designated node.

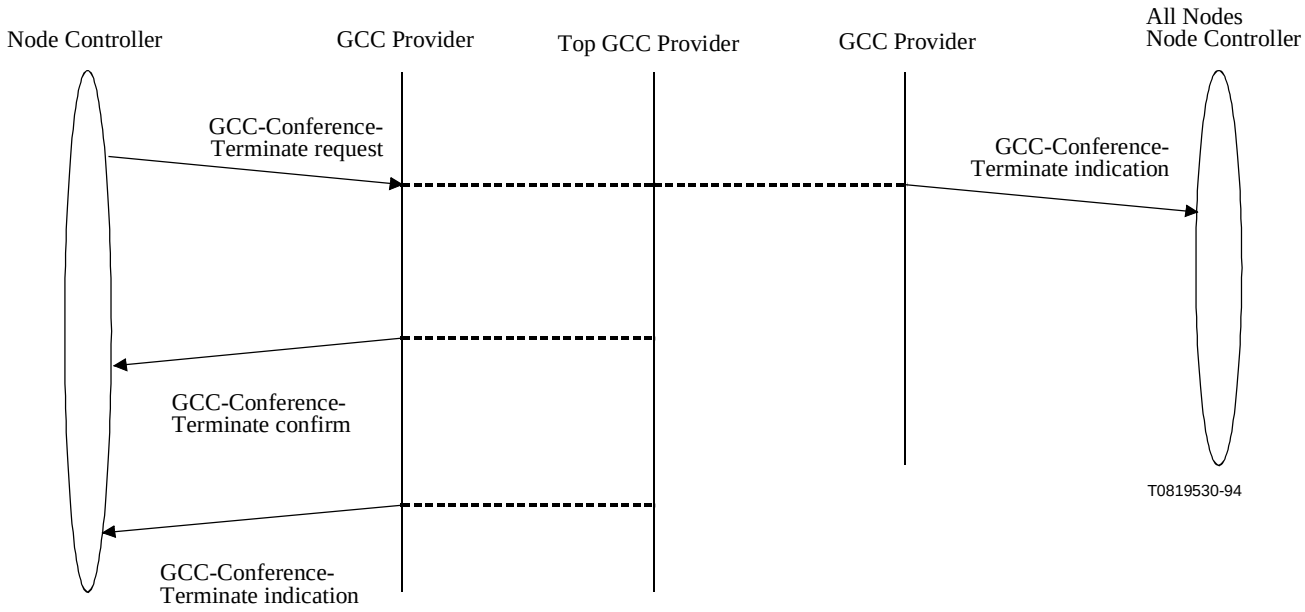

Figure 7-10/T.124 – GCC-Conference-Terminate (client-initiated) – Sequence of primitives

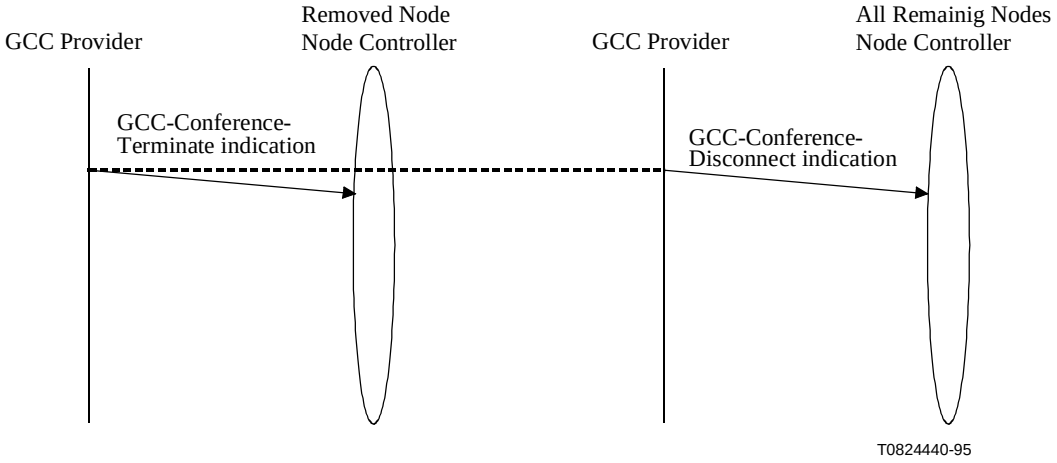

Figure 7-11/T.124 – GCC-Conference-Terminate (error termination) – Sequence of primitives

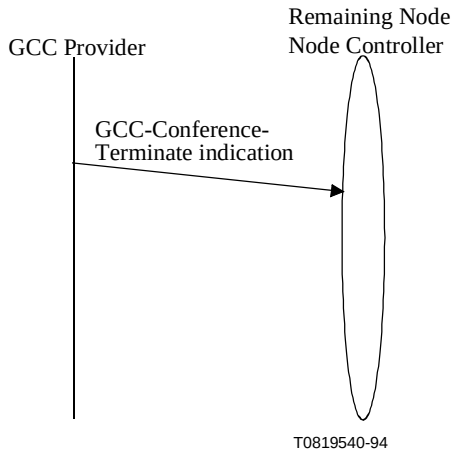

Figure 7-12/T.124 – GCC-Conference-Terminate (automatic termination) – Sequence of primitives

7.1.2.11 GCC-Conference-Eject-User

The GCC-Conference-Eject-User request primitive is used by a Node Controller to force a particular node to be disconnected from a conference. This primitive shall only be issued by the conference Convener or a convener-designated node or by a node directly above the ejected node in the connection hierarchy. Being ejected from a conference does not imply termination of the corresponding physical connection. When a node is ejected, a GCC-Conference-Disconnect indication is issued to the Node Controller at all nodes remaining in the conference indicating that the ejected node has disconnected due to being ejected. Table 7-14 shows the parameters and types of this primitive. Figure 7-13 shows the sequence of events when using this primitive.

Table 7-14/T.124 – GCC-Conference-Eject-User – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Ejected Node ID	M	M(=)	M(=)
Reason for ejection	O	O(=)	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Ejected Node ID: The node to be ejected.

Reason for ejection: Indication of the reason for ejection: user-initiated, higher node disconnected, or higher node ejected.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, invalid user to eject, not convener or convener-designated node.

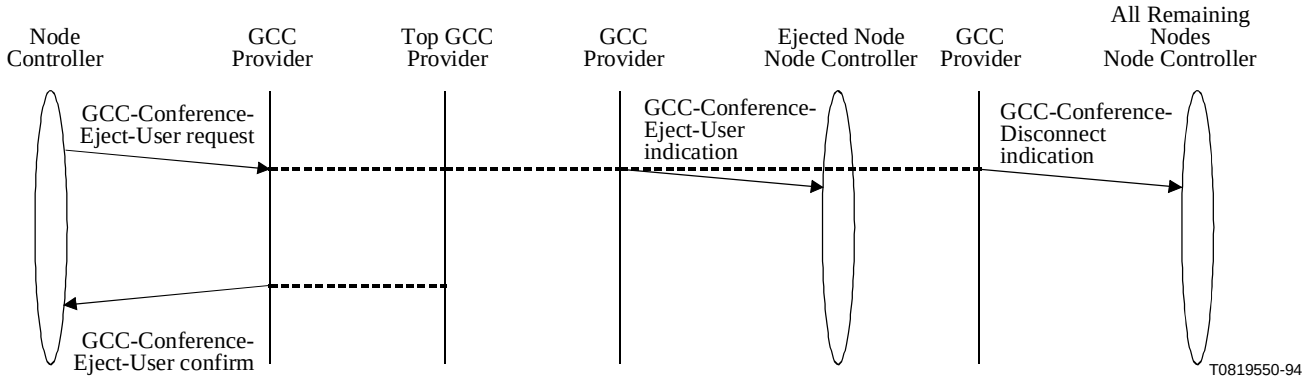

Figure 7-13/T.124 – GCC-Conference-Eject-User (client-initiated) – Sequence of primitives

7.1.2.12 GCC-Conference-Transfer

The GCC-Conference-Transfer request primitive is used by a Node Controller to cause selected nodes in a conference to disconnect from that conference and join another conference. This primitive shall only be issued by the conference Convener or a convener-designated node. Some MCUs in a conference may be already joined to both the originating and destination conferences prior to the transfer taking place. If so, these MCUs shall not be included in the list of Transferring Nodes in this request. If an MCU is not joined with both conferences but will be connected to nodes which, after the transfer, will be joined to both conferences (if not all nodes are transferred), then that MCU shall be joined to the destination conference prior to issuing the GCC-Conference-Transfer request. Any MCU which is included in the list of Transferring Nodes (those which are intended to transfer) shall complete the transfer operation (disconnecting from the originating conference and joining the destination conference) prior to processing any new GCC-Conference-Join indications. This allows the join request from the nodes below that MCU in the connection hierarchy to be successfully completed. Nodes which are in the process of transferring and receiving a GCC-Conference-Terminate indication for the originating conference may proceed directly to joining the destination conference without disconnecting if they have not already done so. This situation could arise if the MCU to which the node is connected was also instructed to transfer. Table 7-15 shows the parameters and types of this primitive. Figure 7-14 shows the sequence of events when using this primitive.

Table 7-15/T.124 – GCC-Conference-Transfer – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Destination Conference Name	M	M(=)	M(=)
Destination Conference Name Modifier	O	O(=)	O(=)
Destination Network Address	O	O(=)	
Transferring Nodes (List of Node IDs or null)	O		O(=)
Password	C	C(=)	
Result			M

Conference ID: Identifier of the MCS Domain corresponding to the conference to which the designated nodes are joined prior to the transfer operation.

Destination Conference Name: Name of the conference to which the designated nodes are instructed to join. This is the name by which the conference is known at the MCU or MCUs to which the transferring nodes are connected. If the conference had been created using both the numerical and text forms of the Conference Name, either form may be used in this primitive.

Destination Conference Name Modifier: If an MCU in the destination conference had chosen to use the optional Conference Name Modifier (as a result of local naming uniqueness problem), nodes to be transferred which are connected to this MCU must be transferred separately, with a separate exchange of the GCC-Conference-Transfer primitive from those nodes which are connected to MCUs using the unmodified (or differently modified) base Conference Name. The manner by which the requesting node becomes aware of Conference Name Modifiers at various MCUs is considered a matter outside the scope of this Recommendation.

Destination Network Address: This parameter optionally describes a nominal logical connection that each transferred node shall use to join the destination conference, if it is not directly connected to an MCU already joined (or hosting) the destination conference. Each portion of the logical connection is described in terms of network type, network address, transfer rate when relevant, and optionally a multimedia profile that shall be operated, a digital circuits aggregation algorithm when applicable, and the list of media (e.g. audio, video, data) that are concerned by this portion. The relevant parts (i.e. network address elements) are to be passed as parameter Called Address in the GCC-Conference-Join-request to be issued by a transferring node. In the GCC protocol, this parameter is reflected by ASN.1 structures NetworkAddress and NetworkAddressV2. See Annex B for the description and use of these.

NOTE – The model assumes that a nominal logical connection is available for all transferring nodes, which may not be valid assumption in all cases. For instance, in the case of an ISDN 2 B-Channels/H.320 based conference, this assumes that the same E.164 number (or pair of E.164 numbers) may be used by all nodes transferring to the destination conference. In the case where the ISDN network does not supply the Calling Party Number at the called side, the connected MCU has no mean to associate individual B-Channels into pairs. Providing a destination network address on a per transferring node basis is left for further study.

Transferring nodes (List of Node IDs or null): List of Node IDs identifying GCC Providers to which the indication should go, or omitted, to indicate it should go to all nodes in the conference designated by the Conference ID.

Password: This parameter indicates the password that the transferring nodes shall use in the GCC-Conference-Join request to join the new conference. This is a numeric string or a Unicode Row 00 text string (maximum 255 digits or characters). The Node Controller shall determine whether to send the Password as a text string or numeric string based on its value. A value consisting only of numeric digits shall be specified as a numeric string, while a value including at least one non-numeric character shall be specified as a text string. This parameter shall only be used if the Password In The Clear Required flag is set for this conference.

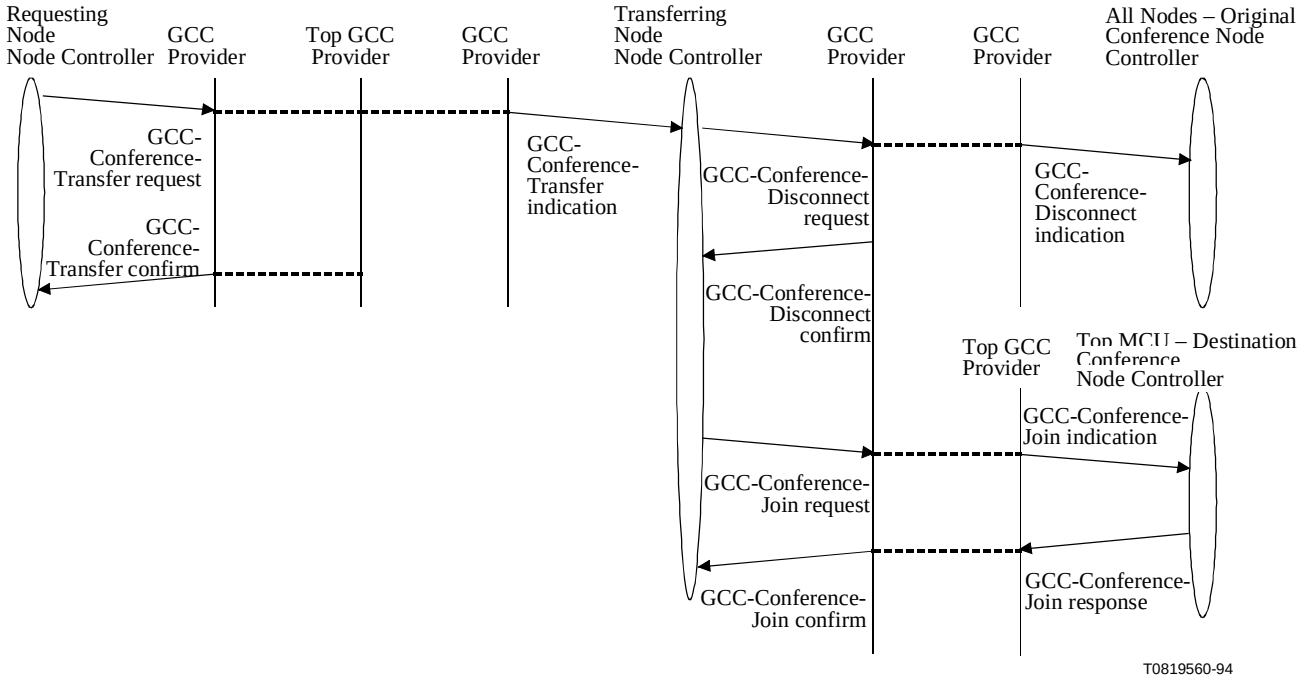

Figure 7-14/T.124 – GCC-Conference-Transfer – Sequence of primitives

7.1.3 Conference establishment requirements

In order to avoid the possibility of a deadlock situation in which both sides of a physical connection fail to initiate the GCC conference establishment procedure, waiting for the other to do so, the following requirements are defined for the conference establishment procedure when a physical connection is established (that is, an underlying connection between the two nodes using the PSTN, ISDN, or CSDN cases of T.123). These requirements may be superseded by bilateral agreement between the nodes involved, either via *a priori* arrangement, or via an exchange of information defined outside of the scope of this Recommendation.

- It must first be known by both ends of the physical connection which end is the calling node and which end is the called node. It must also be known what type of node each of the two nodes are (terminal, multiport terminal, or MCU). This information may be obtained either using the GCC-Conference-Query primitive, or it may be known *a priori*. For determining the calling vs. called node, if neither node is aware of which it is, the symmetry breaking procedure defined for the GCC-Conference-Query primitive is used to arbitrarily choose one node to be the caller for purposes of the conference establishment procedure.
- The calling node shall be responsible for initiating the initial conference establishment procedure (either requesting creation of a new conference, joining a conference at the called node, or inviting the called node into a conference). Note that this, in general, does not preclude the called node from taking actions to establish a conference over the same physical connection as well, although care must be taken in this case to ensure that this action does not interfere with the action of the calling node. A called MCU may set the Wait for Invite Flag in the GCC-Conference-Query response and in doing so relieve the calling node of the responsibility of initiating a connection.

- When establishing calls among the various node types (terminals, multiport terminals, and MCUs), constraints are placed on the actions of the calling node in initiating the initial connection. Table 7-16 shows these constraints for each permutation of calling and called node type. The definition of the actions shown in the table, numbered 1, 2, 3, and 4 are as follows:

  1. Calling node requests creation of a new conference at the Called node.
  2. Calling node attempts to join existing conference at the Called node.
  3. Calling node creates a conference locally and invites the Called node.
  4. Calling node invites the Called node into an existing conference.

Table 7-16/T.124 – Actions of the Calling Node for Conference Establishment

Calling Node	Called Node		
	Terminal	**Multiport Terminal**	**MCU**
Terminal	Either 1 or 3	Either 1 or 2	Either 1 or 2
Multiport Terminal	Either 3 or 4	Either 1, 2, 3, or 4	Either 1 or 2
MCU	Either 3 or 4	Either 3 or 4	Either 1, 2, 3, or 4

In some cases, the calling node may need to make use of information from the called node contained in the GCC-Conference-Query confirm to help choose among the allowed procedures defined in Table 7-16. For example, when a Terminal or Multiport Terminal calls a Multiport Terminal, it may make use of the presence or absence of unlocked conferences in the Conference Descriptor List to determine whether it is most appropriate to create a conference automatically (actions 1, 3, or 4), or to allow the user to attempt to join an existing conference at the called node (action 2). Therefore, when creating any conference, making it listed and unlocked indicates that it is to be available to be joined by a calling node. If the intent is not to allow a conference to be joined by a caller, the conference shall be created as locked, unlisted, or both.

Table 7-17 defines the set of rules for determining the default action of the calling node (as well as the called node) as a function of the node types and the parameter settings contained in the GCC-Conference-Query confirm.

Table 7-17 – Rules for determining the default action of the called and calling nodes

Calling Node Type	Called Node Type	Unlocked Conferences in List	Default Conference Flag	Wait for Invite Flag	No Unlisted Conference Flag	Default Action of Calling Node	Default Action of Called Node
Terminal	Terminal	*	*	*	*	Caller invites	Wait for caller
Multiport Terminal	Terminal	*	*	*	*	Caller invites	Wait for caller
MCU	Terminal	*	*	*	*	Caller invites	Wait for caller
Terminal	Multiport Terminal	FALSE	*	*	*	Caller creates remotely	Wait for caller
Terminal	Multiport Terminal	TRUE	NOT PRESENT or FALSE for all conferences	*	NOT PRESENT or FALSE	Caller chooses conference to join	Wait for caller
Terminal	Multiport Terminal	TRUE	NOT PRESENT or FALSE for all conferences	*	TRUE	Caller chooses listed conference to join	Wait for caller
Terminal	Multiport Terminal	TRUE	TRUE for one conference	*	*	Caller joins default conference	Wait for caller
Multiport Terminal	Multiport Terminal	FALSE	*	*	*	Caller invites	Wait for caller
Multiport Terminal	Multiport Terminal	TRUE	NOT PRESENT or FALSE for all conferences	*	NOT PRESENT or FALSE	Caller chooses conference to join	Wait for caller
Multiport Terminal	Multiport Terminal	TRUE	NOT PRESENT or FALSE for all conferences	*	TRUE	Caller chooses listed conference to join	Wait for caller
Multiport Terminal	Multiport Terminal	TRUE	TRUE for one conference	*	*	Caller joins default conference	Wait for caller
MCU	Multiport Terminal	*	*	*	*	Caller invites	Wait for caller
Terminal	MCU	FALSE	*	NOT PRESENT or FALSE	NOT PRESENT or FALSE	Caller chooses unlisted conference to join OR Caller repeats query	Wait for caller

Table 7-17 – Rules for determining the default action of the called and calling nodes (*concluded*)

Calling Node Type	Called Node Type	Unlocked Conferences in List	Default Conference Flag	Wait for Invite Flag	No Unlisted Conference Flag	Default Action of Calling Node	Default Action of Called Node
Terminal	MCU	FALSE	*	NOT PRESENT or FALSE	TRUE	Caller repeats query	Wait for caller
Terminal	MCU	TRUE	NOT PRESENT or FALSE for all conferences	NOT PRESENT or FALSE	NOT PRESENT or FALSE	Caller chooses conference to join	Wait for caller
Terminal	MCU	TRUE	NOT PRESENT or FALSE for all conferences	NOT PRESENT or FALSE	TRUE	Caller chooses listed conference to join	Wait for caller
Terminal	MCU	TRUE	TRUE for one conference	NOT PRESENT or FALSE	*	Caller joins default conference	Wait for caller
Terminal	MCU	*	*	TRUE	*	Caller waits for invite from called node	Called node invites
Multiport Terminal	MCU	FALSE	*	NOT PRESENT or FALSE	NOT PRESENT or FALSE	Caller chooses unlisted conference to join OR Caller repeats query	Wait for caller
Multiport Terminal	MCU	FALSE	*	NOT PRESENT or FALSE	TRUE	Caller repeats query	Wait for caller
Multiport Terminal	MCU	TRUE	NOT PRESENT or FALSE for all conferences	NOT PRESENT or FALSE	NOT PRESENT or FALSE	Caller chooses conference to join	Wait for caller
Multiport Terminal	MCU	TRUE	NOT PRESENT or FALSE for all conferences	NOT PRESENT or FALSE	TRUE	Caller chooses listed conference to join	Wait for caller
Multiport Terminal	MCU	TRUE	TRUE for one conference	NOT PRESENT or FALSE	*	Caller joins default conference	Wait for caller
Multiport Terminal	MCU	*	*	TRUE	*	Caller waits for invite from called node	Called node invites
MCU	MCU	*	*	*	*	FOR FURTHER STUDY	FOR FURTHER STUDY

7.1.4 Examples of conference establishment procedures

Conference establishment may be done in a variety of ways and under a variety of conditions. The simplest conference is of the point-to-point variety where there is no MCU involved in the call. In the cases where the conference is established through one or more MCUs, a call would typically be done in either the meet-me style (all participants call into an MCU), in the call-out style (the MCUs set up the call by calling out to all participants), or in the call-through style (one participant calls into an MCU, then adds other participants which are called by the MCU).

7.1.4.1 Meet-me conference establishment

In a meet-me conference, a conference is established at an MCU and terminal nodes (as well as other MCUs if necessary) call into the MCU and join the conference. If other MCUs have joined the conference, terminal nodes may call into any of these MCUs to join the conference.

Initial creation of a meet-me conference may be done out-of-band (e.g. locally initiated at the MCU), or the conference may be created by the first node to call into the MCU. In the former case, GCC is not involved. In the latter, the conference is created at the MCU by issuing a GCC-Conference-Create request over the connection from the convening node to the MCU. In either case, the node at which the conference was created becomes the Top-GCC-Provider. The convening node (the MCU itself in the former case, and the requesting node in the latter) is granted special status as the Conference Convener.

When a conference is created remotely (using a GCC-Conference-Create request), a Conference Name is specified in the request primitive. If that name is already in use at the MCU, a Conference Name Modifier is assigned by the Node Controller at the MCU to make the name locally unique at that MCU. When other nodes attempt to join the conference at that MCU, this name modifier must be specified as part of the join request.

When a conference is created remotely it may also include an optional Convener Password. This password is needed only if the Convener intends to disconnect from the conference and rejoin at a later time expecting Convener privileges to continue.

A meet-me conference would typically be specified as non-locked so that other nodes may join. A meet-me conference may optionally be created to require Password protection to prevent unwanted nodes from joining the conference.

A node joining a meet-me conference would issue a GCC-Conference-Join request over the connection from the node to any MCU joined to the conference. Typically, a node does not know *a priori* the name of the conference to be joined. In this case, prior to joining the conference, a node may query the MCU for a list of conferences that are available to be joined. This is done by issuing a GCC-Conference-Query request. The response is a GCC-Conference-Query confirm which indicates the type of node to which the terminal is connected (i.e. an MCU in this case), and a list of all listed conferences to which that MCU is currently joined. This list includes the Conference Name of each conference, a Conference Name Modifier if one is needed, as well as other characteristics of the conference such as whether or not the conference is Password protected, which may be used to request a Password from the user prior to attempting to join the conference. Once the name of the conference is selected, the conference may be joined by issuing a GCC-Conference-Join request specifying the Conference Name of the desired conference, and if needed, the Conference Name Modifier. If the conference requires a Password, it is included in the join request.

Note that it is typical that the case of a meet-me conference with no *a priori* knowledge of the conference to join is identical to the case of a point-to-point conference. That is, the sequence of events used in starting up the conference cannot be different from that of a point-to-point conference since the a priori knowledge in each case is the same. It is only the identification in the

GCC-Conference-Query confirm that the directly connected node is an MCU and that there are ongoing conferences to be joined that allow a distinction to be made.

In the case that the Conference Name is distributed to joining nodes out-of-band, a node may directly join the conference without first querying the available conferences by specifying the Conference Name in the GCC-Conference-Join request. If there are multiple independent MCUs to which nodes may connect to join a conference, it is possible that, due to naming conflicts, a Conference Name Modifier is needed on some MCUs. In this case, a node joining a conference when connected to one of these MCUs must specify the Conference Name Modifier for that MCU. As it may be difficult to determine this modifier without the use of the GCC-Conference-Query primitive, it is recommended that if there is any possibility of a naming conflict, GCC-Conference-Query should be used prior to attempting to join a meet-me conference. It is also recommended that a Conference Description be used when a meet-me conference is first created. In this case, it is more likely that multiple conferences with the same Conference Name may be distinguished by having different Conference Descriptions. In general, however, when creating a meet-me conference, it is better to choose a Conference Name which will be unique at all MCUs without the need of a Conference Name Modifier.

7.1.4.2 Call-out conference establishment

In call-out conference establishment, a conference is created locally at the MCU and the conference participants are called and invited to the conference by that MCU. The conference would typically be created specifying that it is a locked conference and may also be specified as unlisted.

The convening MCU would then make physical connections to each of the terminals to participate in the conference followed by inviting each node in turn to the conference. This is done by issuing a GCC-Conference-Invite request to the node to be invited. Since the conference was created at the MCU, that MCU is the Top-GCC-Provider of a call-out conference.

7.1.4.3 Call-through conference establishment

A call-through conference is very similar to call-out case with the exception that the conference is initially created remotely by an initiating terminal. In this case, the terminal connects to the MCU and creates a conference using a GCC-Conference-Create request. As in the call-out case, it would typically be locked and unlisted. A call-through conference would typically be created as an automatically terminating conference indicating that the conference will be terminated when all nodes disconnect. Typically, a NULL Conference Name would be specified for a call-through conference since there is no need for nodes to explicitly join the conference. If there is already a conference at the convening node with a NULL conference name, the Node Controller would simply choose an arbitrary unique name to use as the Conference Name. In either case, the name need not be human readable since it will never be used for joining.

7.1.4.4 Point-to-point conference establishment

A point-to-point conference is distinct from the other varieties in that it involves only two terminal nodes with no MCU present. In the case where it is known which terminal is the calling terminal (the initiator of the physical connection) and which is the called terminal, a point-to-point conference may be established by the calling terminal first querying the called terminal by issuing a GCC-Conference-Query request. This allows the terminal to determine if the other node is a terminal, MCU, or multiport terminal without requiring *a priori* knowledge to that effect. The GCC-Conference-Query confirm generated in response to this request indicates, in the case of a point-to-point call, that the directly connected node is a user-terminal. Once it is known that the directly connected node is a terminal, the conference is established by the calling terminal by issuing a GCC-Conference-Create request to create a new conference or by creating a conference locally,

and inviting the other terminal by issuing a GCC-Conference-Invite request. Typically, the conference would be specified with an arbitrary Conference Name such as "0", and would be locked, unlisted, and automatically terminating.

In the case that it is not known by a node which terminal is the calling or called terminal, that terminal should issue GCC-Conference-Query request to determine whether the other node is the called or calling node (unless it has already received a GCC-Conference-Query indication which has this information). In the request, the Asymmetry Indicator parameter indicates that it is unknown whether the local node is the called or calling node. If the other node does have this knowledge about itself, it will so indicate in the resulting GCC-Conference-Query confirm. If so, it is now known which is the calling node and the appropriate actions are taken as described above. If neither node knows its called/calling status, the confirm (or the contents of a received GCC-Conference-Query indication) will specify the unknown setting. The unknown settings of this parameter include a 32-bit random number. In this case, the random number is used to break the resulting symmetry. The node which had generated the smaller of the two random numbers should be considered the called node and should not attempt to establish the conference. The node which generated the larger number should be considered the calling node and should attempt to establish the conference.

If a terminal does have *a priori* knowledge that the call is point-to-point between two terminals (and in the case that it is known whether that terminal is the called or calling terminal), that terminal need not issue a GCC-Conference-Query request. Instead, if the terminal is the calling terminal, it may issue the GCC-Conference-Create request or the GCC-Conference-Invite request immediately. If it is the called terminal with this *a priori* knowledge, it may also skip the GCC-Conference-Query request and simply wait to receive a GCC-Conference-Create indication or GCC-Conference-Invite indication. If the far-end terminal does not have *a priori* knowledge of the connection type, it is possible that the local terminal will receive a GCC-Conference-Query indication from the far-end terminal to which it is required to respond.

7.1.4.5 Conference establishment among multiport terminals

A multiport terminal is a device which is, in general, to be treated as a terminal, but has the ability to establish connections to multiple nodes simultaneously as an MCU does. When a terminal or multiport terminal calls a node which it finds to be a multiport terminal (either through *a priori* knowledge or via the GCC-Conference-Query exchange), the action taken depends on whether or not there are conferences available on that multiport terminal and on the characteristics of those conferences.

In general, it is typically desirable that a connection of this kind be made automatically, like a point-to-point call rather than like a meet-me conference. Specifically, if neither the called nor calling nodes are already part of conferences connecting them to other nodes, the call should be treated exactly as a point-to-point call. If the calling node already has an ongoing conference, it is typical that this node would simply invite the new node into the existing conference.

If instead the called node has an ongoing conference, the action depends on whether that conference is locked or unlocked, listed or unlisted. If the conference is unlisted, without *a priori* information, the calling node would not be aware of its presence and would treat the call as if there were no conference present at that node (if it did have *a priori* information that an unlisted conference was present, it could join that conference). If the conference is locked, the calling node has no way to join that conference and again would treat the call as if that conference were not present. In either of these cases, once a new conference is established, the called node may choose to transfer the nodes connected to the previous conference into the new conference. If there are one or more listed conferences at the called node which are not locked, the calling node should assume that the called node is hosting a meet-me conference and would typically require a user to decide which one to join – in this case, the procedure would not be entirely automatic.

If there are conferences present at both nodes, the calling node would normally either invite the called node into its conference, or would attempt to join the called node's conference. In either case, the multiport terminal with the existing conference that was not enlarged to include the other multiport terminal should transfer all subordinate nodes to the conference that it has newly become joined (or invited) to.

Transferring nodes to the new conference in these cases may be done in one of two ways. The GCC-Conference-Transfer primitive may be used to command each of the nodes to disconnect from the current conference and join the new conference. This, of course, can only be done if the new conference is not locked, and if the requesting node had Transfer privilege for the previous conference (or was the convener of the previous conference). Alternatively, the GCC-Conference-Invite primitive could be used to invite the directly connected nodes to the new conference. Typically this would be done after disconnecting these nodes from the previous conference. If there were more nodes originally present than the directly connected nodes, it is the responsibility of the directly connected nodes, recursively, to invite the nodes directly connected to them into the new conference. That is, the invitations would propagate through the hierarchy of physically connected nodes. Nodes would typically do this; however, there is no assurance that this action would be taken. For this reason, the use of GCC-Conference-Transfer is a safer mechanism in the cases where it can be performed.

7.2 The conference roster

The Conference Roster allows a node participating in a conference to learn what other nodes are participating in the same conference, and provides information about each node. Immediately after joining a conference (by means of either creating, joining, or being invited to the conference), the Node Controller at that node shall announce its presence to the conference by issuing a GCC-Conference-Announce-Presence request. The results of this request depend on the joining node's Node Category.

For Conventional nodes (where all nodes that existed prior to the introduction of Node Categories are considered Conventional), the GCC Providers of the nodes in the conference exchange information needed to update the Conference Roster to include the newly-joined node. The updated Conference Roster or a delta update of the Conference roster is distributed to all nodes in the conference, generating a GCC-Conference-Roster-Report indication. A Conventional node joining a conference is not considered an active member of the conference until it has received a Conference Roster in which it is included.

For Counted Nodes, the GCC Providers of the affected nodes in the conference exchange information needed to update the Conference Roster to include the newly-joined node. The updated Conference Roster or a delta update of the Conference roster is distributed to all Conventional nodes in the conference and to the Counted node joining the conference, generating a GCC-Conference-Roster-Report indication. A Counted node joining a conference is not considered an active member of the conference until it has received a Conference Roster in which it is included.

Anonymous Nodes do not affect the Conference Roster. Regardless, their local Node Controllers are still required to announce their presence to the local GCC Provider.

The information in the Conference Roster may be changed at any time during a conference by the Node Controller at any Conventional and Counted node. This is done by re-issuing a GCC-Conference-Announce-Presence request. This results in an updated roster being distributed to all nodes, generating another GCC-Conference-Roster-Report indication.

As Conventional and Counted nodes leave a conference for any reason, a new Conference Roster is distributed by GCC, again generating a GCC-Conference-Roster-Report indication.

7.2.1 Description of abstract services

The following is a list of the primitives defined in this subclause and a brief summary of the function of each:

- GCC-Conference-Announce-Presence – Used by the Node Controller to announce the presence of a node into a conference. Use of this primitive is required immediately after joining or being joined to any conference.
- GCC-Conference-Roster-Report – Generated by GCC in response to any change to the Conference Roster due to nodes either entering or leaving a conference.
- GCC-Conference-Roster-Inquire – Used by either the Node Controller or by Application Protocol Entities to retrieve the current Conference Roster at any time during a conference.

7.2.1.1 GCC-Conference-Announce-Presence

Immediately after a node has joined any conference, it shall announce its presence to the conference using the primitive GCC-Conference-Announce-Presence request. The Node Controller is responsible for issuing this primitive. This primitive may be re-issued at any time during a conference if the included information has changed. Table 7-18 shows the parameters and types of this primitive. Figure 7-15 shows the sequence of events when using this primitive.

NOTE – The Conference Roster includes the list of all terminals, and MCUs currently joined to a conference. A user display of the roster, as well as user indications of nodes entering and leaving the conference are likely to only include terminals and multiport terminals. The Node Type parameter of this primitive may be used to distinguish among these node types, excluding uninteresting node types from user displays. In addition, the Node Properties parameter also adds information as to the use of the device, specifying whether a node is a management device, and whether a node is a peripheral, subordinate to another node. Both of these characteristics may also be used by a system to determine whether or not to include a particular node in its display of conference participants. Typically, neither management devices, nor peripherals would be included in such a list.

Table 7-18/T.124 – GCC-Conference-Announce-Presence – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=RQ)
Node Type	M	
Node Properties	M	
Node Name	O	
Participant Name(s)	O	
Site Information	O	
Network Address	O	
Alternative Node ID	O	
User Data	O	
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Node Type: The node type is either a terminal, MCU, or multiport terminal.

Node Properties: Made up of two independent flags. One indicating whether or not the node is a management device (e.g. a reservation system), and the other indicating whether or not the node is a peripheral, subordinate to another node.

Node Name: Unicode text string containing the name of this node (e.g. "London"). Maximum 255 characters.

Participant Name (s): A list of Unicode text strings each containing the name of a meeting participant. Maximum 255 characters for each string.

Site Information: A Unicode text string containing other information about the node. It may be used, for example, to indicate such things as the voice-phone or FAX numbers at the site.

Network Address: Optional parameter to indicate the network address of this node. This parameter includes sub-fields which specify network type information followed by the actual network address or addresses. This parameter should be used if it is possible that the announcing node may later be attempted to be added to another conference via the GCC-Conference-Add primitive. This gives an MCU knowledge of the Network Address which may be compared to the Network Address of a node to be added to determine if a physical connection already exists. If this parameter is not included, it may not be possible to later add this node to another conference over the same physical connection. This parameter should also be used if it is possible that an initial point-to-point conference may be automatically re-routed through an MCU to add additional nodes. This parameter allows one of the original two nodes to re-connect with the other by use of a GCC-Conference-Add primitive through the MCU.

Alternative Node ID: This field may be used to associate the announcing node (and its corresponding Node ID) with an alternative node ID which has been defined for some other purpose. This alternative node ID is not intended to represent IDs in the same numbering space as Node IDs, but rather a separate numbering scheme not specified by this Recommendation. For example, in the case of ISDN, for nodes supporting Recommendation H.243, the alternative node ID could be the Recommendation H.243 site ID which is two octets in length.

User Data: Optional user data which may be used for functions outside the scope of this Recommendation. Note that this data is stored within the Top GCC Provider as part of Conference Roster. Therefore, this parameter is not intended to contain large amounts of information. Doing so could risk being involuntarily ejected from the conference.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

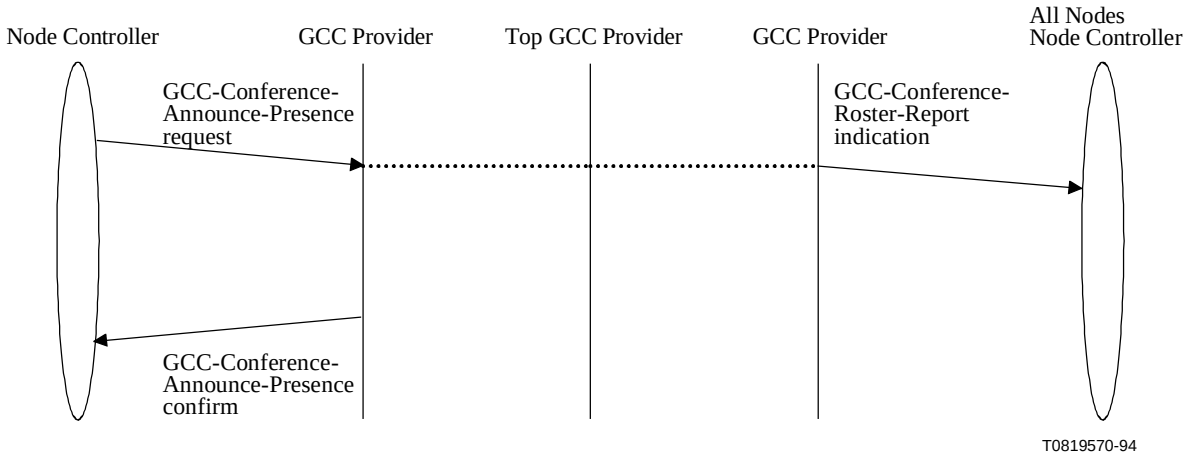

Figure 7-15/T.124 – GCC-Conference-Announce-Presence – Sequence of primitives

7.2.1.2 GCC-Conference-Roster-Report

Whenever the Conference Roster has changed for any reason (a new node entering the conference, a node leaving the conference, or updated information in a roster entry), the roster is distributed by GCC to all nodes by issuing a GCC-Conference-Roster-Report primitive to the Node Controller at each node. Table 7-19 shows the parameters and types of this primitive. Figure 7-16 shows the sequence of events when using this primitive.

Table 7-19/T.124 – GCC-Conference-Roster-Report – Types of primitives and their parameters

Parameter	Indication
Conference ID	M
Conference Roster	M

Conference ID: Identifier of the conference to which the primitive refers.

Conference Roster: A list of each node joined to the specified conference. The format of the Conference Roster is shown in Table 7-20.

Table 7-20/T.124 – Contents of the Conference Roster parameter

Parameter	Description
List of Conference Nodes	A list of the Nodes joined to the conference along with information about each node. The contents of each entry in this list is shown in Table 7-22.
Instance Number	The instance number for the Conference Roster. This is a 16-bit number which is incremented modulo 2^{16} each time the contents of the Conference Roster changes.
Nodes Added Flag	A flag indicating whether one or more Nodes have been added to the Conference Roster since the last instance. This flag is not mutually exclusive of the Nodes Removed Flag.
Nodes Removed Flag	A flag indicating whether one ore more Nodes have been removed from the Conference Roster since the last instance. This flag is not mutually exclusive of the Nodes Added Flag.

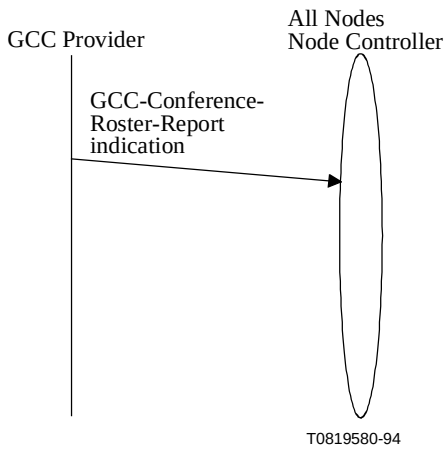

Figure 7-16/T.124 – GCC-Conference-Roster-Report – Sequence of primitives

7.2.1.3 GCC-Conference-Roster-Inquire

The GCC-Conference-Roster-Inquire request primitive returns the complete Conference Roster for the specified conference. This primitive is available to Application Protocol Entities as well as the Node Controller, allowing them to independently obtain a Conference Roster from their local GCC Provider. Table 7-21 shows the parameters and types of this primitive. Figure 7-17 shows the sequence of events when using this primitive.

Table 7-21/T.124 – GCC-Conference-Roster-Inquire – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Conference Name		M
Conference Description		C
Conference Roster		M
Result		M

Conference Name: The Conference Name as contained within the Conference Profile.

Conference Description: The Conference Description as contained within the Conference Profile. This parameter is present, only if a Conference Description had been defined at the time of conference creation.

Conference ID: Identifier of the conference to which the primitive refers.

Conference Roster: A list of each node joined to the specified conference. The format of the Conference Roster is shown in Table 7-23.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

Table 7-22/T.124 – Contents of each entry in the List of Conference Nodes

Parameter	Description
Node ID	MCS User ID of the GCC Provider at this node.
Node ID of Superior Node (conditional)	MCS User ID of the GCC Provider at the node directly above this node in the connection hierarchy, if any (not present for the Top GCC Provider).
Node Type	The node type is either a terminal, MCU, or multiport terminal.
Node Properties	Made up of two independent flags. One indicating whether or not the node is a management device (e.g. a reservation system), and the other indicating whether or not the node is a peripheral, subordinate to another node.
Node Name (conditional)	Unicode text string containing the name of this node (e.g. "London"). Maximum 255 characters.
Participants Names (conditional)	A list of Unicode text strings each containing the name of a meeting participant. Maximum 255 characters for each string.
Site Information (conditional)	A Unicode text string containing other information about the node. It may be used, for example, to indicate such things as the voice-phone or FAX numbers at the site.

Table 7-22/T.124 – Contents of each entry in the List of Conference Nodes (*concluded*)

Parameter	Description
Network Address (conditional)	This parameter describes the logical connection used by the node to join the conference. Each portion of the logical connection is described in terms of network type, network address, transfer rate when relevant, and optionally a multimedia profile that shall be operated, a digital circuits aggregation algorithm when applicable, and the list of media (e.g. audio, video, data) that are concerned by this portion. In the GCC protocol, this parameter is reflected by ASN.1 structures NetworkAddress and NetworkAddressV2. See Annex B for the description and use of these.
Alternative Node ID (conditional)	This field is used to associate the announcing node (and its corresponding Node ID) with an alternative node ID which has been defined for some other purpose. This alternative node ID is not intended to represent IDs in the same numbering space as Node IDs, but rather a separate numbering scheme not specified by this Recommendation. For example, in the case of ISDN, for nodes supporting H.243, the alternative node ID could be the H.243 site ID which is two octets in length.
Node Category (conditional)	This field indicates the Node Category for the node. Conventional and Counted nodes are the only valid categories for this field because Anonymous nodes do not appear in a Conference Roster.
User Data (conditional)	Optional user data which may be used for functions outside the scope of this Recommendation.

Table 7-23/T.124 – Contents of the Conference Roster parameter

Parameter	Description
List of Conference Nodes	A list of the Nodes joined to the conference along with information about each node. The contents of each entry in this list is shown in Table 7-22.
Instance Number	The instance number for the Conference Roster. This is a 16-bit number which is incremented modulo 2^{16} each time the contents of the Conference Roster changes.

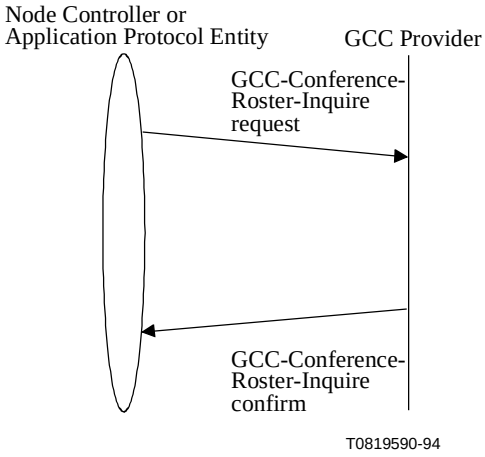

Figure 7-17/T.124 – GCC-Conference-Roster-Inquire – Sequence of primitives

7.3 The application roster

The Application Roster allows a node participating in a conference to learn what Application Protocol Entities are available at nodes in the conference, and provides sufficient information about these Application Protocol Entities to allow direct communication between Peer Application Protocol Entities to begin. The roster may include Application Protocol Entities that are based on standard Application Protocols as well as those based on non-standard Application Protocols.

7.3.1 Contents of the application roster

For each conference, a GCC Provider compiles a list of information associated with each Application Protocol Entity. These lists form the Local Application Roster which can be exchanged with other nodes as part of the Application Roster exchange procedure. This exchange will result in a list of information associated with all Application Protocol Entities in the entire conference collectively known as the Conference Application Roster.

The Local Application Roster is held by each GCC Provider and stored in its local database. At an appropriate point in the set-up of a conference, a node can exchange its Local Application Roster with all other conference nodes and the GCC Provider will receive the Conference Application Roster which it will report to all Application Protocol Entities as well as the Node Controller. This exchange occurs within GCC and is apparent to Application Protocol Entities only through primitives that may be generated. Whether or not a GCC Provider exchanges its local Application Roster depends on the Node Category into which a node falls. For Conventional and Counted nodes, the Local Application Roster is exchanged. For Anonymous nodes, the Local Application Roster is not exchanged.

The Local and Conference Application Rosters consist of the following components:

- *Local Application Roster* – For each Application Protocol Entity which has enrolled with the local GCC Provider, the Local Application Roster includes a Session Key, an Application Record, and an Application Capabilities List.
- *Conference Application Roster* – For each Application Protocol Session, the Conference Application Roster includes the Session Key for that Session, an Application Capabilities List containing the negotiated capability set for the session, and the list of Application Protocol Entities enrolled in the session which includes, for each, the Node ID of the node at which that Application Protocol Entity had enrolled, an identifier which identifies the particular Application Protocol Entity within its local node, and an Application Record.

Each set of Peer Application Protocol Entities – those within a single Application Protocol Session – are characterized by having enrolled using the same value of the Session Key. A Session Key is defined as follows:

- *The Session Key* – An identifier used to uniquely identify an Application Protocol Session. A Session Key consists of two components. One component identifies the Application Protocol. The second component, which is optional, identifies a particular session of that Application Protocol.

  The first component of a Session Key, the Application Protocol Key, identifies either a standard or non-standard Application Protocol specification. Keys are structured so that any Application Protocol, whether standard or non-standard, may be defined to have a unique key with no possibility of conflict. Keys may be specified in one of two forms. A key may be specified as an ASN.1 type Object Identifier. This form of Key may be used to specify standard Application Protocols. The assignment of Object Identifier components for Standards and Recommendations is described in Annex B/X.680 and Annex C/X.680, respectively. Object Identifiers may also be used to specify non-standard Application Protocols in the case of national or private administrative authorities which have been directly or indirectly authorized by ISO or ITU.

  A key may also be specified as an ASN.1 type OCTET STRING interpreted in a manner similar to the pattern adopted in Recommendation H.221 to designate non-standard commands and capabilities. In this case, the first two octets of the OCTET STRING define a country code, the second two octets define a manufacturer code. The first octet of the country code is assigned according to Recommendation T.35; the second octet and the manufacturer code are assigned nationally. Octets beyond this shall be freely chosen by the responsible manufacturer or national body.

  The second component of a Session Key is an optional Session ID. The Session ID is an MCS Channel ID which is used as the unique identifier of an Application Protocol Session and may be used as a communication channel by the Application Protocol Entities taking part in that session (as determined by the Application Protocol specification). To ensure uniqueness, this Channel ID shall remain allocated for the duration of the session. The lack of a Session ID in a Session Key of an active Application Protocol Entity identifies a distinct session – the Default Session.

  In the case of an inactive Application Protocol Session, the Session ID is not required. Its absence in this case may be interpreted as an indication of support for the indicated Application Protocol and the ability to invoke one or more Application Protocol Entities as part of any session, default or otherwise.

Each Application Record contains the following parameters, some of which are optional:

- *Active/Inactive Flag* – An Application Protocol Entity may enroll but not yet be ready to receive data (e.g. it has not yet joined the appropriate channels). To indicate this to other nodes, an Application Protocol Entity may enroll with this flag set to Inactive. This may be done to allow the Application Protocol Entity to make use of GCC services (such as the registry) in preparation for becoming active, or it may simply be to allow other nodes to become aware of the presence of this Application Protocol Entity without becoming active until it is known that there are like Application Protocol Entities at other nodes. When an inactive Application Protocol Entity becomes active, it may re-enroll setting the flag to Active. This flag may be changed by the Application Protocol Entity at any time by re-enrolling.
- *Application User ID* – The MCS User ID associated with the Session Key. The Application User ID is the only means by which a node can include another node as a participant when

convening a private channel for that Application Protocol Session to use. This parameter is optional for Application Protocol Entities enrolled as inactive, but mandatory for active Application Protocol Entities. If it is not included because the Application Protocol Entity has not yet attached to the MCS Domain, it may later be added, once the Application Protocol Entity has attached, by re-enrolling.

- *Conducting Operation Capable* – This is a flag which indicates whether the Application Protocol Entity is capable of operating as a conducting Application Protocol Entity if the corresponding Application Protocol specification defines the procedures for such an Application Protocol Entity to follow. The GCC Provider within each node chooses no more than one Peer Application Protocol Entity per Application Protocol Session to include with this flag set in the exchanged Application Roster. If a node becomes the conference conductor, the designated Application Protocol Entity, if any, for each Application Protocol Session becomes the designated conducting Application Protocol Entity. This flag only applies to Active Application Protocol Entities. If the Active/Inactive flag is set to Inactive when enrolling, the Conducting Operation Capable flag is ignored and assumed by the GCC Provider to be FALSE.
- *Non-Collapsing Capabilities List* – This is an optional parameter which allows Application Protocol Entities to list capabilities (either standard or non-standard) which are to be maintained in the roster as part of the Application Record of each Application Protocol Entity rather than being collapsed by a set of rules as the capabilities listed in the Application Capabilities List would be.

The Application Capabilities List is defined as follows:

- **Application Capabilities List**: This is an optional parameter list which may be used to specifically list the capabilities of the Application Protocol Entity. While the capabilities themselves are Application Protocol–specific, they are listed along with information which allows GCC to determine the common capability set for the Application Protocol Session and inform the Peer Application Protocol Entities of this set. This avoids the need for a complete exchange of a full capability list between all nodes. Each capability in the capability list is tagged with a class specifier. The class specifier determines the rule that is applied to determine the common capability set. Table 7-24 lists all capability classes. Note that neither Counted or Anonymous nodes can affect the Conference Application Roster capabilities.

Table 7-24/T.124 – Capability combination rules

Class	Description
Logical	If any Peer Application Protocol Entity in a conference indicates the use of this capability, the final capability list indicates the number of Peer Application Protocol Entities that have indicated this capability.
Unsigned-Minimum	The parameter octets are treated as a single unsigned integer. The final capability list indicates the minimum value among all Peer Application Protocol Entities which indicated this capability as well as the number of Peer Application Protocol Entities which have indicated this capability.
Unsigned-Maximum	The parameter octets are treated as a single unsigned integer. The final capability list indicates the maximum value among all Peer Application Protocol Entities which indicated this capability as well as the number of Peer Application Protocol Entities which have indicated this capability.

In addition to the basic classifications, it is also possible to nest capabilities. Nesting is not done explicitly, but rather is done by appropriate interpretation of the three capability classes. For example, an Application Protocol may define a particular capability, Y, to be conditional on a Logical capability, X. If the rule is applied within the Application Protocol specification that capability Y may only be issued if capability X had been issued, then the value of Y in the final capability set may be interpreted as, in the case of a Logical capability Y, the number of Peer Application Protocol Entities, among those which have capability X, which also have capability Y, and in the case of a numerical capability Y, the minimum or maximum value of Y among those Peer Application Protocol Entities which have capability X. Note that in the case of a numerical capability Y, if the number of Peer Application Protocol Entities which indicate the capability Y is less than the number of Peer Application Protocol Entities indicating the capability X, this indicates that some Peer Application Protocol Entities do not support a value of Y beyond its default value. In this case, the negotiated value of Y should be ignored. The negotiated value of Y is valid in this case only if the number of Peer Application Protocol Entities with both the X and Y capabilities are equal.

7.3.2 Description of the application roster exchange process

Each Application Protocol Entity makes its local GCC Provider aware of its presence through a GCCSAP over which it communicates with the GCC Provider. Creation and management of the GCCSAP is a local matter not covered by this Recommendation.

Application Protocol Entities are made aware of the existence of a conference to which the node is joined by the GCC-Application-Permission-To-Enroll indication. Once an Application Protocol Entity has been made aware of an existing conference, it shall issue a GCC-Application-Enroll request to the GCC Provider. This enroll request may indicate that the Application Protocol Entity is enrolling itself in the conference (the enroll/un-enroll flag set to enroll), or it may indicate that it is not enrolling itself into the conference (the enroll/un-enroll flag set to un-enroll). In the latter case, the Application Record parameters need not be included in the primitive. Any time thereafter, until permission to enroll is revoked (upon receiving a GCC-Application-Permission-To-Enroll primitive with the grant/revoke flag set to revoke), an Application Protocol Entity which did not choose to initially enroll itself into the conference may enroll by issuing a GCC-Application-Enroll request.

There are several parameters in the GCC-Application-Enroll request which, if used, require steps to be taken prior to issuing the GCC-Application-Enroll request. First is the Application User ID which is mandatory in the request if enrolling with the Active/Inactive flag is set to Active. To get an Application User ID, the Application Protocol Entity must first attach to the MCS Domain indicated in the GCC-Application-Permission-To-Enroll indication. This is done by issuing the MCS primitive MCS-Attach-User request. On reception of the MCS-Attach-User confirm, the Application Protocol Entity will have been allocated an Application User ID. This ID may be included in the GCC-Application-Enroll request. If the attachment to the domain should fail due to the connection to the domain being lost during the intervening time, the Application Protocol Entity shall assume that the node is no longer part of the conference and shall wait until receiving another GCC-Application-Permission-To-Enroll indication before proceeding further. When the node disconnects from the conference, the Application Protocol Entity will be notified directly by MCS that the attachment is no longer valid. The Application Protocol Entity shall assume that the node is no longer part of the conference and shall wait until receiving another GCC-Application-Permission-To-Enroll indication before proceeding further. If permission to enroll is revoked by a GCC-Application-Permission-To-Enroll primitive, the Application Protocol Entity shall not attempt to attach to the corresponding domain.

The second such parameter is the optional Session ID which is part of the Session Key. The Session ID shall be formed from an MCS Channel ID (which is not a User ID). When enrolling into a session being invoked, the Session ID is obtained from the GCC-Application-Invoke indication. If joining a

session already in progress, the Session ID may be obtained from the most recently received Application Roster. If creating a new session, then a new Channel ID may be allocated using an MCS-Channel-Join primitive or MCS-Channel-Convene primitive.

As a consequence of a GCC-Application-Enroll request primitive, the GCC Provider creates a new entry in its Local Application Roster. If an entry already exists for an Application Protocol Entity, the contents of the Local Application Roster entry are modified to contain the new information, or in the case of the un-enroll flag being set, the entry is removed from the roster.

At the start of a conference, the GCC Provider, after issuing the GCC-Application-Permission-To-Enroll primitive, waits for all Application Protocol Entities which are connected to the GCC Provider through a GCCSAP to issue a GCC-Application-Enroll primitive either enrolling, or indicating their explicit desire not to enroll. For Conventional and Counted nodes, once all Application Protocol Entities have responded, the GCC Provider exchanges the Local Application Roster with the other nodes in the conference. During a conference in progress, any changes to the Local Application Roster are immediately exchanged with other nodes. Note that only Conventional nodes can create new Application Protocol Sessions (by being the first node to exchange a Local Application Roster associated with a non-existent session). GCC Providers at Counted nodes must therefore wait until a session associated with a particular Application Protocol Entity exists before it can exchange its associated Local Application Roster information.

For Anonymous nodes, the Local GCC Provider will never exchange its Local Application Roster. Instead, the GCC Provider uses the enrollment information to determine which Application Protocol Sessions the local Application Protocol Entities are interested in. When a Conference Application Roster associated with one of these APEs is received by the local GCC Provider, that Conference Application Roster is immediately delivered to the interested APE.

In any of the cases above, whenever a new exchange takes place, initiated by any node, a new Conference Application Roster is generated and distributed to all interested nodes. The GCC Provider issues the entire roster to the Node Controller, and issues portions of this roster to enrolled Application Protocol Entities using the GCC-Application-Roster-Report indication primitive. To an Application Protocol Entity, the roster report includes the portion of the Conference Application Roster which specifically applies to that Application Protocol Session (and may include other portions as well). For each session, this is in the form of a list. Each entry in the list includes the Node ID and Application Protocol Entity ID which, together, identify the Application Protocol Entity, and the Application Record for that Application Protocol Entity. It may also include the Application Capabilities List for the Application Protocol Session. This is a list of the common capabilities for this Application Protocol Session based on application of the capability class rules to the capabilities announced by all Peer Conventional node Application Protocol Entities. In the case of the node controller, the GCC-Application-Roster-Report indication includes the entire Conference Application Roster. That is, for each entry in the list, it includes the node identifier, and the Application Records for all Application Protocols enrolled at that node. It also may include, for each Application Protocol, an Application Capabilities List.

An Application Protocol Entity is not considered part of a conference until it has received a GCC-Application-Roster-Report indication in which it is included in the Application Roster. If, at any time, an Application Protocol Entity which had previously been enrolled receives a GCC-Application-Roster-Report indication in which it is no longer included, that Application Protocol Entity shall be considered no longer enrolled. The Application Protocol Entity may attempt to re-enroll by issuing a GCC-Application-Enroll request. At Counted and Anonymous nodes, it is the responsibility of the local GCC Provider to insert an Application Record (associated with a local Application Protocol Entity that is trying to enroll with a session), into the roster before delivering the associated GCC-Application-Roster-Report indication.

At any time an Application Protocol Entity may remove itself from the Application Roster by issuing a GCC-Application-Enroll request primitive with the enroll/un-enroll flag set to un-enroll. This removes the Application Record for that Application Protocol Entity from the Local Application Roster as well as from the Conference Application Roster at all nodes in the conference.

At any time during a conference, an Application Protocol Entity or the Node Controller may request a portion of the Conference Application Roster. For each node in the conference, all Application Records which match the Session Key given in the GCC-Application-Roster-Inquire request are returned. Using a null key will result in the complete Conference Application Roster.

An Application Protocol Entity or Node Controller may attempt to remotely invoke an Application Protocol Entity at another node. This is done by issuing a GCC-Application-Invoke request. This request may optionally include a specified list of destination nodes. The corresponding GCC-Application-Invoke indication is received by the Node Controller at the specified destination nodes (or all nodes if no destination is specified). The node controller may optionally respond to this indication by invoking a Peer Application Protocol Entity which may then enroll itself in the conference.

7.3.3 Description of abstract services

The following is a list of the primitives defined in this subclause and a brief summary of the function of each:

- GCC-Application-Permission-To-Enroll – Generated by GCC and issued to all Application Protocol Entities which have made the GCC Provider aware of their presence whenever the local node has been joined to a conference. This indicates that the Application Protocol Entity may enroll. With the Revoke flag set, this revokes the ability to enroll when the node is no longer joined to the conference.
- GCC-Application-Enroll – Used by Application Protocol Entities to establish (or terminate) communications with other Peer Application Protocol Entities in a conference. Use of this primitive generates (or modifies, or removes) an entry in the Application Roster exchanged with other nodes.
- GCC-Application-Roster-Report – Generated by GCC in response to any change in the Application Roster due to Application Protocol Entities enrolling or un-enrolling or nodes entering or leaving a conference.
- GCC-Application-Roster-Inquire – Used by either the Node Controller or by Application Protocol Entities to retrieve all or a portion of the current Application Roster at any time during a conference.
- GCC-Application-Invoke – Used to signal a set of other nodes in a conference to invoke an Application Protocol Entity for a particular session of an Application Protocol.

7.3.3.1 GCC-Application-Permission-To-Enroll

The primitive GCC-Application-Permission-To-Enroll indication tells a local Application Protocol Entity that the local node is now joined to the specified conference and that the Application Protocol Entity may enroll itself with that conference. This primitive is also used to revoke such permission if the local node is no longer joined to a conference. While an MCS-attached Application Protocol Entity will receive notification from MCS when the conference terminates at that node, an Application Protocol Entity which has chosen not to attach (or to enroll) will not necessarily receive such notification. This primitive indicates to such an Application Protocol Entity that it may no longer enroll with that conference or attach to the corresponding MCS Domain. The GCC-Application-Permission-To-Enroll indication is issued by a GCC Provider to all Application Protocol Entities which have made the GCC Provider aware of their presence only after the node has

been successfully joined to a conference. Table 7-25 shows the parameters and types of this primitive. Figure 7-18 shows the sequence of events when using this primitive.

Table 7-25/T.124 – GCC-Application-Permission-To-Enroll – Types of primitives and their parameters

Parameter	Indication
Conference ID	M
Grant/Revoke Flag	M
Node Category	C

Conference ID: Identifier of the Conference to which the Application Protocol Entity may enroll. This parameter is equal to the MCS Domain to which the Application Protocol Entity may attach.

Grant/Revoke Flag: This flag indicates whether the Application Protocol Entity is being given permission to enroll, or if that permission is being revoked.

Node Category: This field informs the APE which Node Category was established for this node, within this conference, when the node joined.

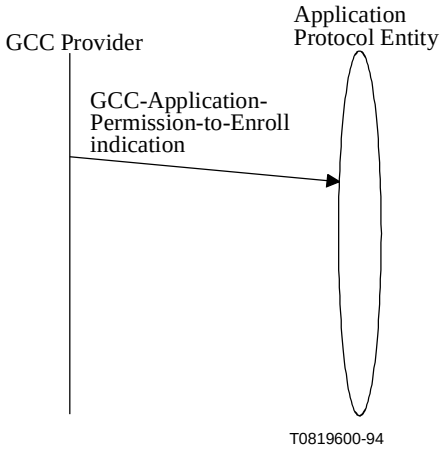

Figure 7-18/T.124 – GCC-Application-Permission-to-Enroll – Sequence of primitives

7.3.3.2 GCC-Application-Enroll

The GCC-Application-Enroll request primitive is issued by an Application Protocol Entity to establish itself as part of the specified conference. This primitive shall be issued in response to a GCC-Permission-To-Enroll indication, although the Application Protocol Entity may set the Enroll/Un-enroll flag to Un-enroll, indicating that it does not wish to enroll at this time. At any time until permission to enroll is revoked, the Application Protocol Entity may enroll, re-enroll (to change entries in its Application Record), or un-enroll. If an Application Protocol Entity which is already un-enrolled attempts to un-enroll, the request is accepted with a successful result, but no change is made to the status of that Application Protocol Entity. When an Application Protocol Entity is enrolled, the associated parameters form the Application Record which is broadcast to the conference during the Application Roster exchange. Application Protocol Entities that are not enrolled will not be available to any local or remote participants and will not receive GCC-Application-Roster-Report indications. Table 7-26 shows the parameters and types of this primitive. Figure 7-19 shows the sequence of events when using this primitive.

Table 7-26/T.124 – GCC-Application-Enroll –
Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Session Key	M	M(=)
Active or Inactive	M	
Application User ID	C	
Conducting Operation Capable	O	
Start-up Channel	O	
Non-Collapsing Capabilities List	O	
Application Capabilities List	O	
Enroll or Un-enroll	M	
Application Protocol Entity ID		C
Node ID		C
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Session Key: Unique identifier of a particular Application Protocol Session. This corresponds to the identifier of the Application Protocol, optionally concatenated with a Session ID.

Active or Inactive: This flag is used to indicate whether the enrolling Application Protocol Entity is fully active (indicating that it can perform tasks required by the specification of the corresponding Application Protocol), or inactive. An Application Protocol Entity might indicate itself to be inactive in the case that it must enroll to make use of certain GCC services, but is not yet fully operational (e.g. it has not yet joined dynamic channels, but must first check the registry to determine which channels to join). Alternatively, it might enroll inactively to indicate support for the corresponding Application Protocol without the need to become active until it is known that there are like Application Protocol Entities at other nodes. The specific meaning of this flag shall be defined in the specification of each Application Protocol.

Application User ID: MCS User ID given to this Application Protocol Entity. This parameter is optional in the case of inactive Application Protocol Entities, but is mandatory for active Application Protocol Entities. It is an error for an Application Protocol Entity that is enrolled with an MCS User ID to re-enroll with a different MCS User ID. This parameter is not needed in the case of un-enrolling.

Conducting Operation Capable: This is a flag which indicates whether the Application Protocol Entity is capable of operating as a conducting Application Protocol Entity if the corresponding Application Protocol specification defines the procedures for such an Application Protocol Entity to follow. The GCC Provider within each node chooses no more than one Peer Application Protocol Entity per Application Protocol Session to include with this flag set in the exchanged Application Roster. If a node becomes the conference conductor, the designated Application Protocol Entity, if any, for each Application Protocol Session becomes the designated conducting Application Protocol Entity. The enrolling Application Protocol Entity shall not assume itself to be in the designated conducting role unless it determines that its corresponding entry in the received Application Roster has this flag set. This flag only applies to Active Application Protocol Entities. If the Active/Inactive flag is set to Inactive when enrolling, the Conducting Operation Capable flag is ignored and assumed by the GCC Provider to be FALSE.

Start-up Channel: This is an optional parameter which may take on the values Static, Dynamic Multicast, Dynamic Private, or Dynamic UserId. This parameter specifies the type of MCS Channel the Application Protocol Entity will use for start-up sequencing. The exact interpretation of this parameter as well as any requirements for the use of this parameter are Application Protocol-specific. In some cases, certain of these channel types may not be valid for particular Application Protocols.

Non-Collapsing Capabilities List: This is an optional parameter which allows Application Protocol Entities to list capabilities (either standard or non-standard) which are to be maintained in the roster as part of the Application Record of each Application Protocol Entity rather than being collapsed by a set of rules as the capabilities listed in the Application Capabilities List would be. Each entry in this list includes a capability ID which may be either standard or non-standard, as well as a data field for Application Protocol-specific data.

Application Capabilities List: This is an optional list of Application Protocol–specific capabilities. Each capability is tagged with a class identifier indicating the rule to be applied by GCC in determining the common capability set for this Application Protocol Session. The class identifiers are one of the choices listed in Table 7-24.

Enroll or Un-enroll: This is a flag used to indicate whether the Application Protocol Entity wishes to add, or change its Application Record (enroll), or to remove its Application Record (un-enroll). When an Application Protocol Entity has received a GCC-Permission-To-Enroll indication, it shall respond with a GCC-Enroll-Request. If that Application Protocol Entity wishes not to enroll, it shall set this flag to Un-enroll.

Application Protocol Entity ID: Present only in the case of a successful new enroll (not in the case when already enrolled, in the case of an unsuccessful enroll, or in the case of an un-enroll). An identifier which uniquely identifies the Application Protocol Entity within the local node. This corresponds to the same parameter in each entry of the Application Roster. This allows an Application Protocol Entity to determine if it is included in subsequent updates of the Application Roster.

Node ID: Present only in the case of a successful enroll (not in an unsuccessful enroll or an un-enroll). This provides the enrolling Application Protocol Entity the Node ID of its local node. This helps to allow the Application Protocol Entity to find its corresponding entry in received Application Rosters to determine when it actually becomes enrolled into the conference. The combination of the Node ID and the Application Protocol Entity ID uniquely determine the Application Protocol Entity.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, not permitted to change MCS User ID.

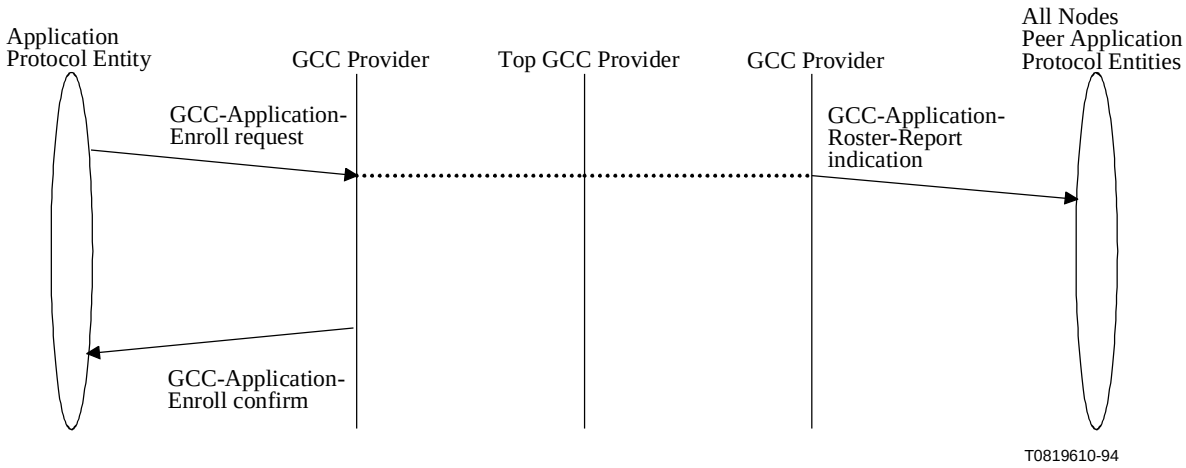

Figure 7-19/T.124 – GCC-Application-Enroll – Sequence of primitives

7.3.3.3 GCC-Application-Roster-Report

This primitive is used by a GCC Provider to send to each enrolled Application Protocol Entity as well as to the Node Controller some or all of the Conference Application Roster. A GCC-Application-Roster-Report indication may be generated automatically by a GCC Provider whenever the GCC Provider finds out that a change in any portion of the roster has occurred. This may happen, for example, as a result of a GCC-Application-Enroll request, or on detection of an Application Protocol Entity detaching from the conference, or a node leaving the conference altogether. When any portion of the roster has changed for any Application Protocol Entity, the GCC-Application-Roster-Report indication shall be issued to the Node Controller. The GCC Provider shall issue a GCC-Application-Roster-Report indication to an active Application Protocol Entity when any portion of the roster has changed for any Peer Application Protocol Entity. In this case it is required to include only the portion of the roster which applies to the corresponding Application Protocol Session. The GCC Provider may also issue a GCC-Application-Roster-Report indication to an Application Protocol Entity at other times (such as when entries other than for Peer Application Protocol Entities have changed) and may include in the indication primitive portions of the roster beyond those corresponding to Peer Application Protocol Entities. For inactive Application Protocol Entities, if the Application Protocol Entity has enrolled with a specific Session ID, the same rules apply as for an active Application Protocol Entity. For an inactive Application Protocol Entity which has enrolled with no Session ID, the GCC Provider shall issue a GCC-Application-Roster-Report when the contents of the roster have changed for all Application Protocol Sessions which are based on the same Application Protocol as the enrolled Application Protocol Entity. Table 7-27 shows the parameters and types of this primitive. When issued to the Node Controller, each of the Application Protocol–specific parameters (all except the Conference ID) is repeated separately for each Application Protocol Session. Figure 7-20 shows the sequence of events when using this primitive.

Table 7-27/T.124 – GCC-Application-Roster-Report – Types of primitives and their parameters

Parameter	Indication
Conference ID	M
Updated Application Roster	M

Conference ID: Identifier of the MCS Domain corresponding to the indicated conference.

Updated Application Roster: The updated Application Roster includes the information shown in Table 7-28 for each Application Protocol Session being indicated by the indication.

Table 7-28/T.124 – Contents of the updated Application Roster for each Application Protocol Session

Parameter	Description
Session Key	The Session Key (including the Session ID, if any) designating the particular Application Protocol Session.
Application Protocol Entity List Updated Flag	A flag indicating whether the Application Protocol Entity List for this Session Key has been updated in this report. If so, the Application Protocol Entity List is included as the following parameter.
Application Protocol Entity List (conditional)	A list of the Application Protocol Entities enrolled in the conference as part of this Application Protocol Session including an Application Record for each. The contents of each entry in this list is shown in Table 7-29.
Instance Number	The instance number for the Application Roster for this Session Key. This is a 16-bit number which is incremented modulo 2^{16} each time the contents of the Application Roster changes for this Session Key. This allows Application Protocol Entities to perform operations with respect to a particular capability set which may be in the process of changing avoiding any race conditions.
Peer Entities Added Flag	A flag indicating whether one or more Peer Application Protocol Entities have been added to the Application Roster at any node since the last instance. This flag is not mutually exclusive of the Peer Entities Removed Flag.
Peer Entities Removed Flag	A flag indicating whether one ore more Peer Application Protocol Entities have been removed from the Application Roster at any node since the last instance. This flag is not mutually exclusive of the Peer Entities Added Flag.
Application Capabilities List Updated Flag	A flag indicating whether the Application Capabilities List for this Session Key has been updated in this report. If so, the Application Capabilities List is included as the following parameter.
Application Capabilities List (conditional)	The fully collapsed Application Capabilities List for this Session Key.

Table 7-29/T.124 – Contents of each entry of the Application Protocol Entity List

Parameter	Description
Node ID	Node ID identifying the node at which the Application Protocol Entity has enrolled.
Application Protocol Entity ID	An identifier which uniquely identifies the Application Protocol Entity within the node specified by the Node ID.
Active/Inactive Flag	A flag indicating whether the enrolled Application Protocol Entity is inactive or active.
Application User ID (conditional)	The MCS User ID associated with the enrolled Application Protocol Entity.
Conducting Operation Capable	This is a flag which indicates whether the Application Protocol Entity is capable of operating as a conducting Application Protocol Entity if the corresponding Application Protocol specification defines the procedures for such an Application Protocol Entity to follow. There is no more than one Peer Application Protocol Entity per node with this flag set. If a node becomes the conference conductor, the designated Application Protocol Entity at that node, if any, for this Application Protocol Session becomes the designated conducting Application Protocol Entity.
Start-up Channel (conditional)	This parameter, if present, may take on the values Static, Dynamic Multicast, Dynamic Private, or Dynamic UserId. This parameter indicates the type of MCS Channel the Application Protocol Entity will use for start-up sequencing. The exact interpretation of this parameter is Application Protocol-specific. In some cases, certain of these channel types may not be valid for particular Application Protocols.
Non-Collapsing Capabilities List (conditional)	This is a list of Application Protocol-specific capabilities (either standard or non-standard) which are maintained in the roster as part of the Application Record of each Application Protocol Entity.

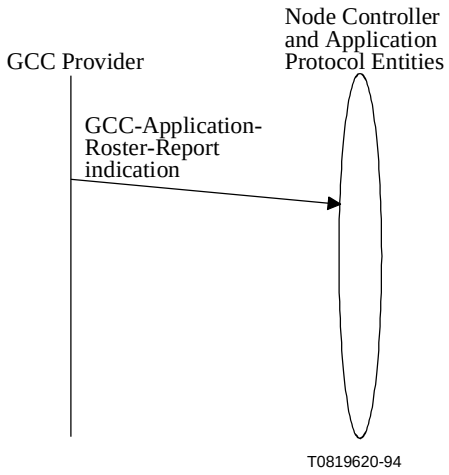

Figure 7-20/T.124 – GCC-Application-Roster-Report – Sequence of primitives

7.3.3.4 GCC-Application-Roster-Inquire

An Application Protocol Entity may request a portion of the Conference Application Roster corresponding to a single Application Protocol Session, a group of these, or all Application Protocol Entities, using the GCC-Application-Roster-Inquire request primitive. This information is returned by GCC in the GCC-Application-Roster-Inquire confirm primitive. Table 7-30 shows the parameters and types of this primitive. Figure 7-21 shows the sequence of events when using this primitive.

Table 7-30/T.124 – GCC-Application-Roster-Inquire – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Session Key	M	M(=)
Application Roster		M
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Session Key: Unique identifier of an Application Protocol Session. This may be a partial Session Key or a null key, indicating that the Application Information for all Application Protocol Entities is desired.

Application Roster: The Application Roster includes a list of roster entries for each Application Protocol Session specified by a Session Key whose octets match the Session Key in the request up to the length of the Session Key in the request, and if a Session ID is specified as part of the Session Key, those with the identical Session ID. For each matched entry, the information returned is shown in Table 7-31.

Table 7-31/T.124 – Contents of the Application Roster for each Application Protocol Session

Parameter	Description
Session Key	The Session Key (including the Session ID, if any) designating the particular Application Protocol Session.
Application Protocol Entity List	A list of the Application Protocol Entities enrolled in the conference as part of this Application Protocol Session including an Application Record for each. The contents of each entry in this list is shown in Table 7-28.
Instance Number	The instance number for the Application Roster for this Session Key. This is a 16-bit number which is incremented modulo 2^{16} each time the contents of the Application Roster changes. This allows Application Protocol Entities to perform operations with respect to a particular capability set which may be in the process of changing avoiding any race conditions.
Application Capabilities List	The fully collapsed Application Capabilities List for this Session Key.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

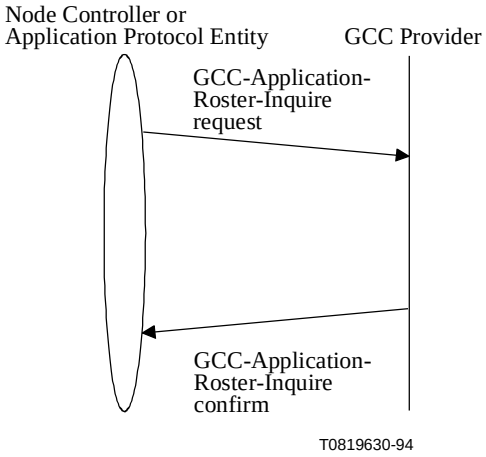

Figure 7-21/T.124 – GCC-Application-Roster-Inquire – Sequence of primitives

7.3.3.5 GCC-Application-Invoke

At the specified nodes, the indication form of this primitive is issued to the Node Controller which may invoke the specified Application Protocol Entities. For each listed Application Protocol Entity, this indication indicates the desire to create a new Application Protocol Entity with the Session Key as specified in the request which would then enroll actively into the conference, becoming part of the corresponding Application Protocol Session. Alternatively, if an inactive Application Protocol Entity already exists at a destination node with the identical Session Key, receipt of this indication indicates the desire to make that Application Protocol Entity re-enroll in the active state. If an Application Protocol Entity exists at a destination node with the identical Session Key which is already in the active state, this indication may be ignored. Table 7-32 shows the parameters and types of this primitive. Figure 7-22 shows the sequence of events when using this primitive. Note that Application Protocol Entities can only be invoked at Conventional and Counted nodes. Also, a GCC-Application-Invoke can only be initiated from a Conventional node. A Conventional node should have already created the session passed in the Invoke request before a Counted node is invoked.

Table 7-32/T.124 – GCC-Application-Invoke – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=)
Application Protocol Entity Invoke List	M	M(=)	M(=)
Destination Nodes (List of Node IDs or NULL)	O		
Invoking Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Application Protocol Entity Invoke List: This is a list of one or more Application Protocol Entities to be invoked. The contents of each entry in this list are shown in Table 7-33.

Table 7-33/T.124 – Contents of each entry in the Application Protocol Entity Invoke List

Parameter	Description
Session Key	The Session Key (including the Session ID, if any) designating the particular Application Protocol Session to invoke.
Expected Capability Set	This is an optional list of application capabilities (in the same form as the Application Capabilities List) which indicates the expected set of capabilities that the Application Protocol Entity to be invoked must have. If the Application Protocol Entity at a particular node cannot satisfy this, it shall not be invoked at that node. If this parameter is not included, no constraints are placed on the invoked Application Protocol Entity. Note that the capabilities class definitions are used to determine the interpretation of this expected capability set, but are not interpreted by GCC as they are in the case of the Application Capabilities List. For a capability of the Unsigned-MAX class, the invoked Application Protocol Entity must have a capability less-than-or-equal-to the specified capability, while for an Unsigned-MIN class, the invoked Application Protocol Entity must have a capability greater-than-or-equal-to the specified capability. For a capability in the Logical class, the invoked Application Protocol Entity simply must have the identical capability.
Start-up Channel (conditional)	Either Static, Dynamic Multicast, Dynamic Private, or Dynamic UserId. This parameter specifies the type of MCS Channel the Application Protocol Entity should assume for use when invoking. In some cases, certain of these channel types may not be valid for particular Application Protocols. If an invalid setting is used, the Application Protocol Entity shall not be invoked.
Mandatory/Optional Flag	This flag indicates whether this Application Protocol Entity must be invoked in order to invoke the other Application Protocol Entities in this list. This flag is used by the invoking node to indicate that the destination nodes should only invoke the Application Protocol Entities in this list if all of the Application Protocol Entities marked as Mandatory can successfully be invoked.

Destination Nodes (List of Node IDs or NULL): List of Node IDs identifying GCC Providers to which the request should go, or null, to indicate it should go to all nodes in the conference designated by the Conference ID.

Invoking Node ID: The Node ID of the node which initiated the corresponding GCC-Application-Invoke request.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

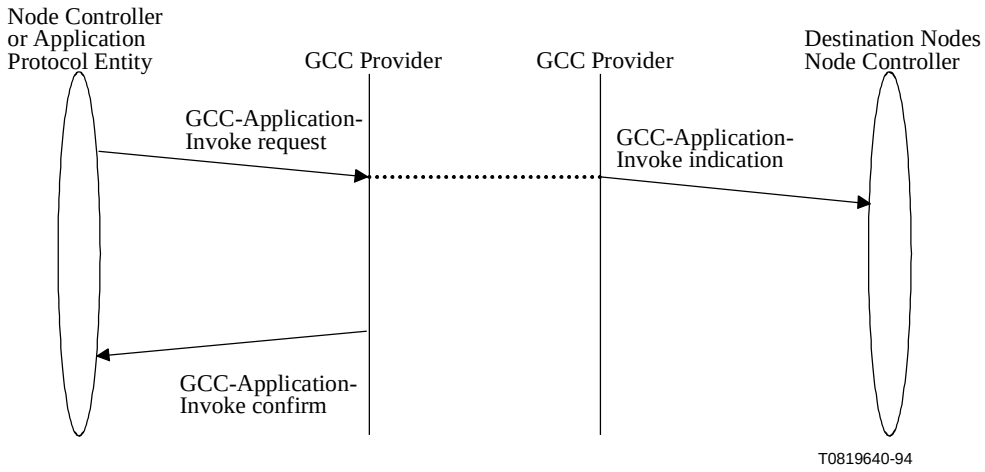

Figure 7-22/T.124 – GCC-Application-Invoke – Sequence of primitives

7.4 The Application Registry

The Application Registry is a functional component of GCC. The registry offers a set of functions to Application Protocol Entities which operate on a central database located at the Top GCC Provider. The contents of the registry are unique to a single conference. The significance of information stored in the registry database is defined by Application Protocols. The central repository can assist in establishing communication among Application Protocol Entities. It can help peers to discover a common Channel ID on which to communicate, a common Token ID to regulate exclusive access, or a common parameter setting (a parameter is a registry entry with an Application Protocol-specific use). It offers an alternative to the static reservation of Channel IDs, Token IDs, and other parameter values. By supporting dynamic discovery, the Application Registry can facilitate the introduction of new features, standardized or non-standardized, that enhance audiographic and audiovisual conferencing. The registry also includes a general purpose service for allocation of numeric handles which are globally unique within a conference.

7.4.1 Registry keys

The indexes used to store and retrieve entries in the registry database are Application Protocol-specified keys. Both standard and non-standard keys are allowed. Standard keys are allocated by Recommendations like this. Non-standard keys may be proprietary. Both are structured to avoid conflicts in the choice of key values.

Registry keys consist of a Session Key as defined in 7.3.1 in combination with a Resource ID. The Session Key used for any registry operation requiring a Registry Key shall be identical to the Session Key of an Application Protocol Session present in the Application Roster (with either active or inactive members). The Resource ID is of ASN.1 type OCTET STRING. The Resource ID allows a single Application Protocol Entity to make use of multiple Registry Keys. The value of the Resource IDs are defined by the specifications of the Application Protocols which make use of them.

7.4.2 Ownership and persistence

The registry yields ownership of a Registry Key to the first Application Protocol Entity that requests to store an item there via either the GCC-Registry-Register-Channel, GCC-Registry-Assign-Token, or GCC-Registry-Set-Parameter request primitives. A Conventional node is the only Node Category allowed to have ownership of a registry entry.

In the case of a Parameter registry item, when the parameter is created, its owner may specify the scope of Application Protocol Entities allowed to modify the value of that entry. The owner may specify either owner, session, or public modification rights – the parameter may be modifiable by the owner only (as in the case of Channels and Tokens), by all Peer Application Protocol Entities (those within the same Application Protocol Session), or by any Application Protocol Entity in the conference, respectively. Modification of the contents of a parameter by an Application Protocol Entity other than the owner does not alter the identity of the owner of that entry. The owner of a parameter may at any time re-define the scope of modification rights. If another node attempts to modify the modification rights as part of issuing a GCC-Registry-Set-Parameter request, the rights will not be modified, although the parameter will be set to the requested value if the requester had modification rights.

For all registry item types (Channel, Token, or Parameter), ownership is required to delete the entry. An entry, once deleted, has no owner and may be taken over by a different Application Protocol Entity.

Registry entries are not deleted automatically when the owner un-enrolls from the conference. Their content persists unchanged indefinitely. However, ownership of an entry is removed when the owner un-enrolls. This allows a surviving Application Protocol Entity to modify (for a Parameter entry only) or delete an orphaned entry if its usefulness has expired. The first Application Protocol Entity that requests to store an item in an orphaned Parameter entry becomes its new owner.

When all Application Protocol Entities in the session that corresponds to the Session Key used to form the Registry Key for a registry entry become un-enrolled, that registry entry is automatically deleted.

7.4.3 Dynamic allocation

MCS distinguishes between static and dynamic Channel IDs. The latter comprise User IDs, private Channel IDs, and assigned Channel IDs. Dynamic Channel IDs are created and deleted directly by Application Protocol Entities, separately from their use of registry services. The Application Registry stores dynamic Channel IDs in a central repository for retrieval by other Application Protocol Entities. It does not test or operate on these Channel IDs with MCS primitives.

MCS makes no distinction between static and dynamic Token IDs. To maintain similar semantics, an artificial division is imposed by GCC. Token IDs 1 to 16 383 are designated static and are reserved for assignment by other specifications. Token IDs 16 384 to 65 535 are designated dynamic and are allocated by the Top GCC Provider upon request, as part of creating an entry in the registry database. When the associated entry is deleted, the Token ID it held is made available for reassignment. The registry does not invoke MCS primitives, like grab and release, on the Token IDs it assigns. It merely chooses specific ID values and disseminates them. Application Protocol Entities are free to operate on assigned Token IDs according to their own logic.

7.4.4 Description of abstract services

The following is a list of the primitives defined in this subclause and a brief summary of the function of each:

- GCC-Registry-Register-Channel – Used by Application Protocol Entities to register the Channel ID of an MCS dynamic channel. Application Protocol Entities may examine the entry using GCC-Registry-Retrieve-Entry to determine if a node has already registered the Channel ID, and if so, retrieve the value of the Channel ID. Only Conventional nodes are allowed to register channels.

- GCC-Registry-Assign-Token – Used by Application Protocol Entities to allocate a dynamic token and register the assigned Token ID. Application Protocol Entities may examine the entry using GCC-Registry-Retrieve-Entry to determine if a node has already registered the Token ID, and if so, retrieve the value of the Token ID. Only Conventional nodes are allowed to assign a token.
- GCC-Registry-Set-Parameter – Used by Application Protocol Entities to set a value in the registry database which may be examined or modified from any node in a conference. Only Conventional nodes are allowed to set a registry parameter.
- GCC-Registry-Retrieve-Entry – Used by Application Protocol Entities to extract the current contents of any registry entry.
- GCC-Registry-Delete-Entry – Used by Application Protocol Entities to remove an entry in the registry database. Only Conventional nodes are allowed to delete a registry entry.
- GCC-Registry-Monitor – Used by Application Protocol Entities to enable (or disable) monitoring of a registry entry. Once enabled, the indication form of this primitive notifies the requesting Application Protocol Entities of any changes to the content of the entry (including deletion). Any Node Category can monitor the registry.
- GCC-Registry-Allocate-Handle – Used to generate a 32-bit Handle which is unique within the scope of a single conference. Any Node Category can allocate a handle.

For a particular requester, the order of registry request primitives, the resulting action at the Top GCC Provider (if successful), and their associated confirm primitives is preserved.

7.4.4.1 GCC-Registry-Register-Channel

The GCC-Registry-Register-Channel request primitive may be issued by an Application Protocol Entity at a Conventional node to inform Application Protocol Entities at other nodes that a particular MCS channel has been designated for use by the Application Protocol Entity in the manner indicated by the Registry Key. Once any Application Protocol Entity registers using a particular Registry Key, Application Protocol Entities at other nodes may find out if a channel has been registered for this key (and if so, the value of the channel ID) by issuing the GCC-Registry-Retrieve-Entry request primitive specifying the Registry Key in question. Once a channel is registered, if any Application Protocol Entity attempts to register using the same Registry Key (including the owner), their attempt will be rejected with an indication that this Registry Key has already been used. Table 7-34 shows the parameters and types of this primitive. Figure 7-23 shows the sequence of events when using this primitive.

Table 7-34/T.124 – GCC-Registry-Register-Channel – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Registry Key	M	M(=)
Channel ID	M	
Registry Item		C
Owner		C
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Registry Key: The database index at which the item is to be stored.

Channel ID: A dynamic channel ID (user ID, private, or assigned) specified by the Application Protocol Entity.

Registry Item: The value of the entry after the request has taken effect. If the request is successful, this is the value of the parameter, if not successful due to inconsistent type or index already owned, this is the value prior to the request. It is either a Channel ID, Token ID, Parameter Value, or None if the entry is vacant (both the type and the value, if any, are indicated by this parameter). This parameter is not present if the confirm is a locally-generated error condition.

Owner: This parameter indicates the current owner of this registry entry. If owned, this parameter includes the Node ID of the node at which the owner resides and the Entity ID of the owning Application Protocol Entity. This parameter also indicates if the entry is not owned. This parameter is not present if the confirm is a locally generated error condition.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, registry full, index already exists, inconsistent type, invalid requester.

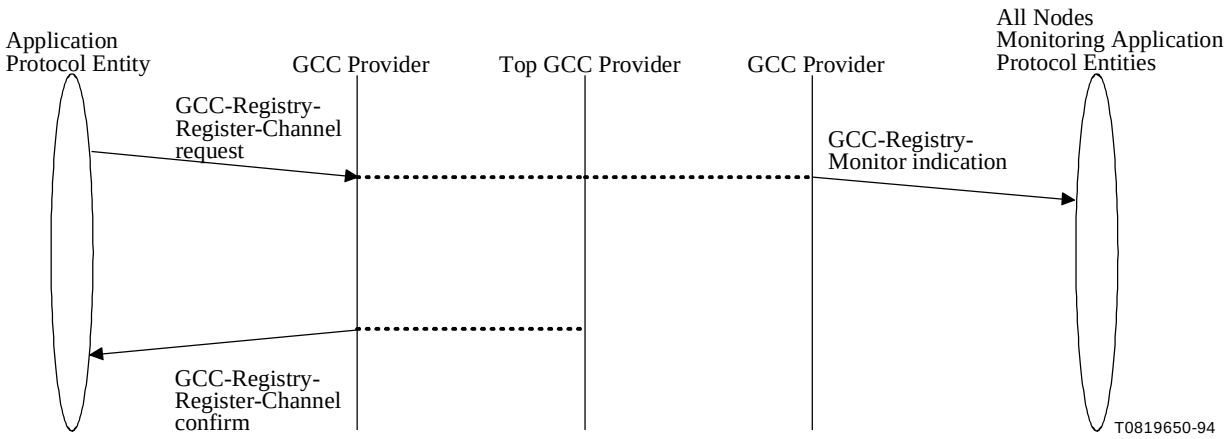

Figure 7-23/T.124 – GCC-Registry-Register-Channel – Sequence of primitives

7.4.4.2 GCC-Registry-Assign-Token

The GCC-Registry-Assign-Token request primitive may be issued by an Application Protocol Entity at a Conventional node to assign a token to be associated with a particular Registry Key. If the request is successful, the token ID is returned as a parameter in the confirm primitive. Once any Application Protocol Entity gets a token assigned using a particular Registry Key, Application Protocol Entities at other nodes may find out if a token has been assigned for this key (and if so, the value of the Token ID) by issuing the GCC-Registry-Retrieve-Entry request primitive specifying the Registry Key in question. Once the token is assigned, if any Application Protocol Entity attempts to get a token assigned using the same Registry Key (including the owner), their attempt will be rejected with an indication that this Registry Key has already been used. Table 7-35 shows the parameters and types of this primitive. Figure 7-24 shows the sequence of events when using this primitive.

Table 7-35/T.124 – GCC-Registry-Assign-Token –
Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Registry Key	M	M(=)
Token ID		C
Registry Item		C
Owner		C
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Registry Key: The database index at which the item is to be stored.

Token ID: A dynamic token ID (16 384 or greater) assigned by the Top GCC Provider. This parameter is not present if the confirm is a locally-generated error condition.

Registry Item: The value of the entry after the request has taken effect. If the request is successful, this is the value of the parameter, if not successful due to inconsistent type or index already owned, this is the value prior to the request. It is either a Channel ID, Token ID, Parameter Value, or None if the entry is vacant (both the type and the value, if any, are indicated by this parameter). This parameter is not present if the confirm is a locally-generated error condition.

Owner: This parameter indicates the current owner of this registry entry. If owned, this parameter includes the Node ID of the node at which the owner resides and the Entity ID of the owning Application Protocol Entity. This parameter also indicates if the entry is not owned. This parameter is not present if the confirm is a locally-generated error condition.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, registry full, index already exists, inconsistent type, invalid requester.

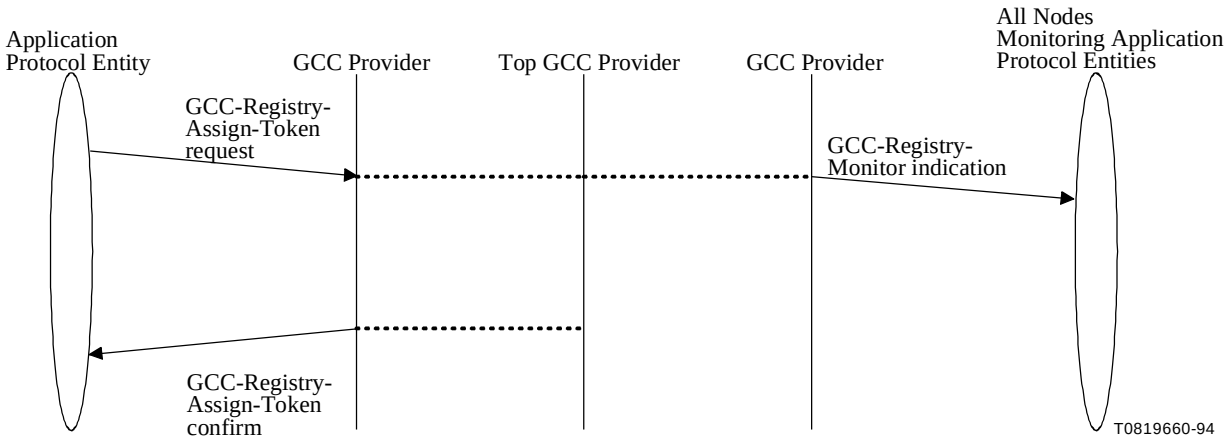

Figure 7-24/T.124 – GCC-Registry-Assign-Token – Sequence of primitives

7.4.4.3 GCC-Registry-Set-Parameter

The GCC-Registry-Set-Parameter request primitive may be issued by an Application Protocol Entity at a Conventional node to set or modify the value of a registry parameter. If the registry entry had been designated to be monitored using the GCC-Registry-Monitor request primitive, each successful GCC-Registry-Set-Parameter request results in a GCC-Monitor-Indication to Application Protocol Entities at all nodes in the conference which have enabled monitoring for this entry. If a registry entry exists for a particular key, a request to set a parameter shall only be accepted if the entry is already a parameter. Table 7-36 shows the parameters and types of this primitive. Figure 7-25 shows the sequence of events when using this primitive.

Table 7-36/T.124 – GCC-Registry-Set-Parameter – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Registry Key	M	M(=)
Parameter Value	M	
Registry Item		C
Owner		C
Modification Rights	O	C
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Registry Key: The database index at which the item is to be stored.

Parameter Value: An octet string specified by the Application Protocol Entity.

Registry Item: The value of the entry after the request has taken effect. If the request is successful, this is the value of the parameter; if not successful due to inconsistent type, this is the value prior to the request. It is either a Channel ID, Token ID, Parameter Value, or None if the entry is vacant (both the type and the value, if any, are indicated by this parameter). This parameter is not present if the confirm is a locally-generated error condition.

Owner: This parameter indicates the current owner of this registry entry. If owned, this parameter includes the Node ID of the node at which the owner resides and the Entity ID of the owning Application Protocol Entity. This parameter also indicates if the entry is not owned. If successful completion of this primitive results in the requester becoming the owner of this entry, the requester is indicated as the new owner in the confirm primitive. This parameter is not present if the confirm is a locally-generated error condition.

Modification Rights: This optional parameter specifies the scope of Application Protocol Entities allowed to make modifications to the value of this registry entry. The value of this parameter may be either Owner, Session, or Public. The Owner setting specifies that only the owner (as long as the entry is owned) may modify this entry. The Session setting specifies that any Application Protocol Entity which is part of the same Application Protocol Session as the owner may modify this entry. The Public setting specifies that any Application Protocol Entity enrolled in the conference may modify this entry. If this parameter is not present when the entry is first created, a value of Public is assumed. If not present at other times, its value is left unchanged. Only the owner of this registry entry (as long as the entry is owned) can change the modification rights. If a non-owner attempts to change the modification rights, this change will not take place; however, any modification to the

parameter entry itself will occur as long as the requester has modification rights. In the confirm primitive, this parameter is present independent of whether it was in the request primitive and indicates the actual modification rights. This parameter is not present if the confirm is a locally-generated error condition.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, registry full, inconsistent type, invalid requester.

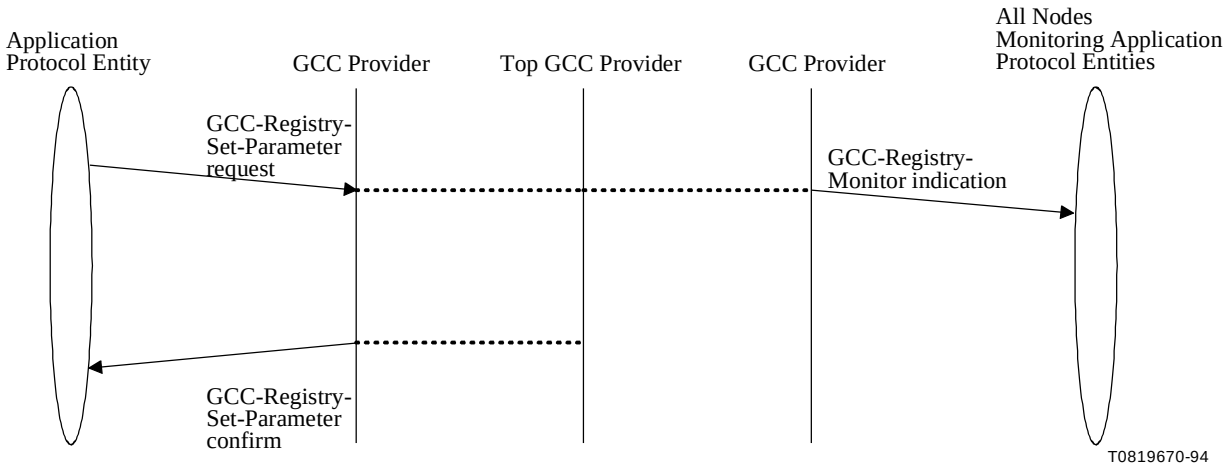

Figure 7-25/T.124 – GCC-Registry-Set-Parameter – Sequence of primitives

7.4.4.4 GCC-Registry-Retrieve-Entry

The GCC-Registry-Retrieve-Entry request primitive may be issued by an Application Protocol Entity to determine the contents of a single registry entry. This primitive may be issued at any time and will indicate the contents of the entry as well as whether the entry is a channel ID, a token ID, a parameter, or if the entry is empty. Table 7-37 shows the parameters and types of this primitive. Figure 7-26 shows the sequence of events when using this primitive.

Table 7-37/T.124 – GCC-Registry-Retrieve-Entry – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Registry Key	M	M(=)
Registry Item		C
Owner		C
Modification Rights		C
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Registry Key: The database index of the entry to be retrieved.

Registry Item: A Channel ID, Token ID, Parameter Value, or None if the entry is vacant (both the type and the value, if any, are indicated by this parameter). This parameter is not present if the confirm is a locally-generated error condition.

Owner: This parameter indicates the current owner of this registry entry. If owned, this parameter includes the Node ID of the node at which the owner resides and the Entity ID of the owning Application Protocol Entity. This parameter also indicates if the entry is not owned. This parameter is not present if the confirm is a locally-generated error condition.

Modification Rights: This parameter, included only in the case of Parameter entries, indicates the scope of Application Protocol Entities allowed to make modifications to the value of this registry entry. The value of this parameter may be either Owner, Session, or Public. The Owner setting specifies that only the owner (as long as the entry is owned) may modify this entry. The Session setting specifies that any Application Protocol Entity which is part of the same Application Protocol Session as the owner may modify this entry. The Public setting specifies that any Application Protocol Entity enrolled in the conference may modify this entry. If this parameter is not present when the entry is first created, a value of Public is assumed. This parameter is not present if the confirm is a locally-generated error condition.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, entry not found, invalid conference.

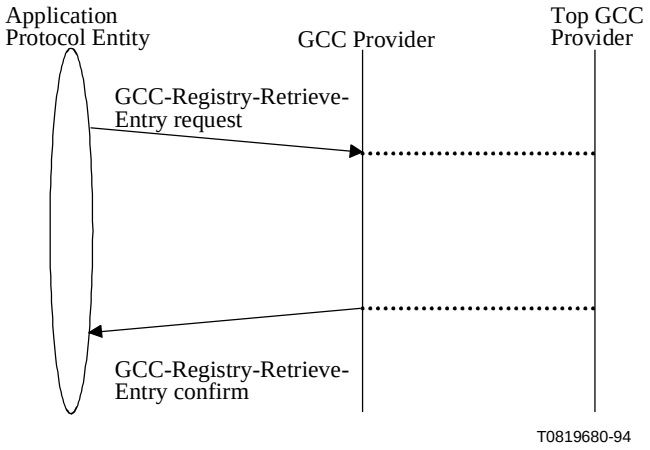

Figure 7-26/T.124 – GCC-Registry-Retrieve-Entry – Sequence of primitives

7.4.4.5 GCC-Registry-Delete-Entry

The GCC-Registry-Delete-Entry request primitive may be issued by an Application Protocol Entity to remove a registry entry. Removal of a registry entry is only allowed by the Application Protocol Entity which owns that entry (unless the last owner disconnected from the conference and no new owner has been assigned) and must be issued from a Conventional node. If the registry entry had been designated to be monitored using the GCC-Registry-Monitor request primitive, a successful GCC-Registry-Delete-Entry request results in a GCC-Monitor-Indication to Application Protocol Entity at all nodes in the conference which have enabled monitoring for this entry with a NULL Registry Item parameter to indicate that the entry has been deleted. Table 7-38 shows the parameters and types of this primitive. Figure 7-27 shows the sequence of events when using this primitive.

Table 7-38/T.124 – GCC-Registry-Delete-Entry – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Registry Key	M	M(=)
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Registry Key: The database index of the entry to be vacated, removing any item previously stored.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, index already owned by another GCC Provider.

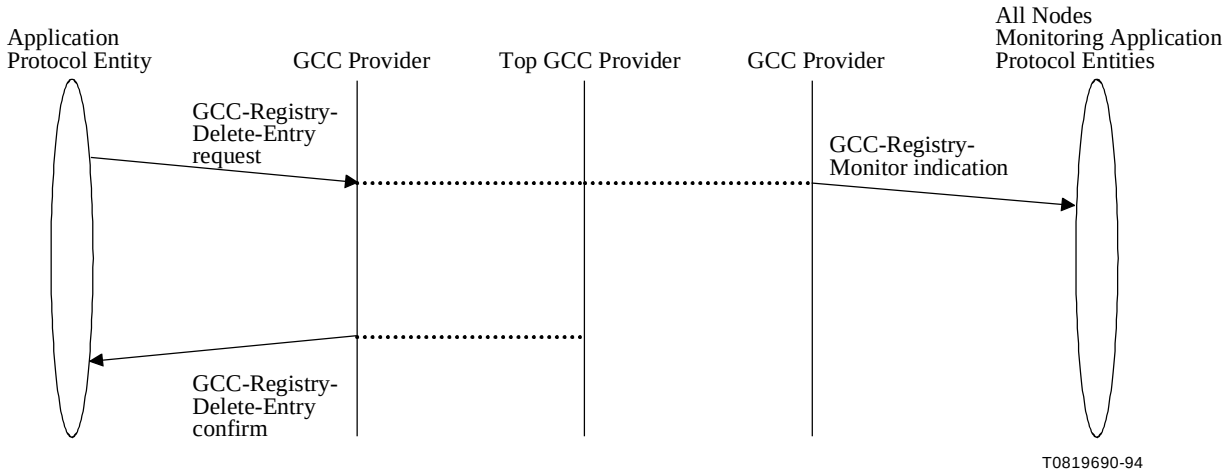

Figure 7-27/T.124 – GCC-Registry-Delete-Entry – Sequence of primitives

7.4.4.6 GCC-Registry-Monitor

GCC provides a mechanism for continuously monitoring particular registry entries of any type to determine if they have been changed (either altered, deleted, changed owners, or in the case of a parameter entry, changed modification rights) without the need for continuous polling. The GCC-Registry-Monitor request primitive may be used by an Application Protocol Entity to enable (or disable) monitoring of a particular registry entry. While enabled, the requesting Application Protocol Entity is notified of all changes to this entry via the GCC-Registry-Monitor indication primitive. An indication will be generated as a result of any modification of the contents of the registry entry or deletion of the entry. Only registry entries which exist may be monitored. Once an entry has been deleted, if it is recreated, the monitor request must be re-issued to begin monitoring again. Table 7-39 shows the parameters and types of this primitive. Figure 7-28 shows the sequence of events when using the request form of this primitive.

NOTE – Particular GCC Provider implementations may choose not to keep track of which Application Protocol Entities have enabled or disabled monitoring for each entry. In that case, Application Protocol Entities may receive GCC-Registry-Monitor indications for entries that they have not requested to be monitored, or for entries for which they specifically disabled monitoring.

Table 7-39/T.124 – GCC-Registry-Monitor – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Enable/Disable	M		M(=)
Registry Key	M	M	M(=RQ)
Registry Item		M	
Owner		M	
Modification Rights		C	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Enable/Disable: TRUE to deliver indications of registry contents; FALSE to suppress indications.

Registry Key: The database index of an entry to be monitored (in the request/confirm), or that has changed (in the indication).

Registry Item: A Channel ID, Token ID, Parameter Value, or None indicating that the entry has been deleted (both the type and the value, if any, are indicated by this parameter).

Owner: This parameter indicates the current owner of this registry entry. If owned, this parameter includes the Node ID of the node at which the owner resides and the Entity ID of the owning Application Protocol Entity. This parameter also indicates if the entry is not owned. This parameter is not present if the confirm is a locally-generated error condition.

Modification Rights: This parameter, included only in the case of Parameter entries, indicates the scope of Application Protocol Entities allowed to make modifications to the value of this registry entry. The value of this parameter may be either Owner, Session, or Public. The Owner setting specifies that only the owner (as long as the entry is owned) may modify this entry. The Session setting specifies that any Application Protocol Entity which is part of the same Application Protocol Session as the owner may modify this entry. The Public setting specifies that any Application Protocol Entity enrolled in the conference may modify this entry. If this parameter is not present when the entry is first created, a value of Public is assumed. This parameter is not present if the confirm is a locally-generated error condition.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, registry entry does not exist.

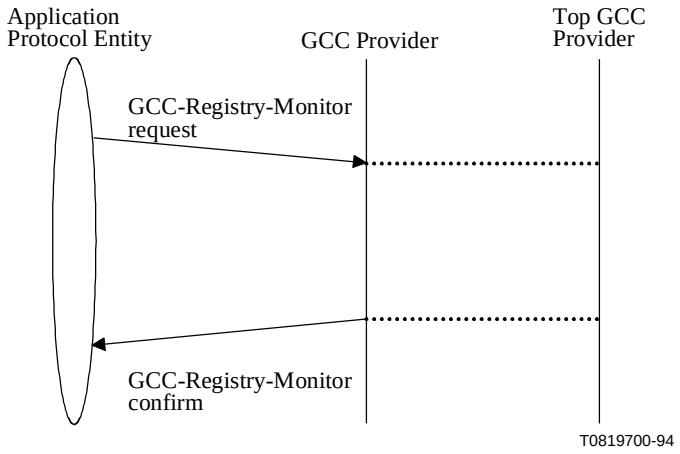

Figure 7-28/T.124 – GCC-Registry-Monitor – Sequence of primitives

7.4.4.7 GCC-Registry-Allocate-Handle

The GCC-Registry-Allocate-Handle request primitive may be issued by an Application Protocol Entity at a Conventional node to request that a numerical value (or list of values) be allocated to that Application Protocol Entity which is globally unique within the scope of a single conference. Handles are allocated by the Top GCC Provider in increasing numerical order in the order that requests are received. Blocks of handles are also allocated in increasing numerical order. As a result, only the first handle in a block is returned if the number of handles is more than one. Table 7-40 shows the parameters and types of this primitive. Figure 7-29 shows the sequence of events when using this primitive.

Table 7-40/T.124 – GCC-Registry-Allocate-Handle – Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Number of Handles	M	M(=)
First Handle		M
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Number of Handles: The number of handles requested to be allocated and returned in the confirm primitive. This may range from 1 to 1024.

First Handle: 32-bit unsigned integer value. If the number of requested handles is equal to one, this is the value of the allocated handle. If the number of handles is greater than one, the set of allocated handles are those contiguous values (modulo 2^{32}) that range from the value of First Handle, through the value (First Handle + Number of Handles) mod 2^{32}.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, no handles available, too many handles requested.

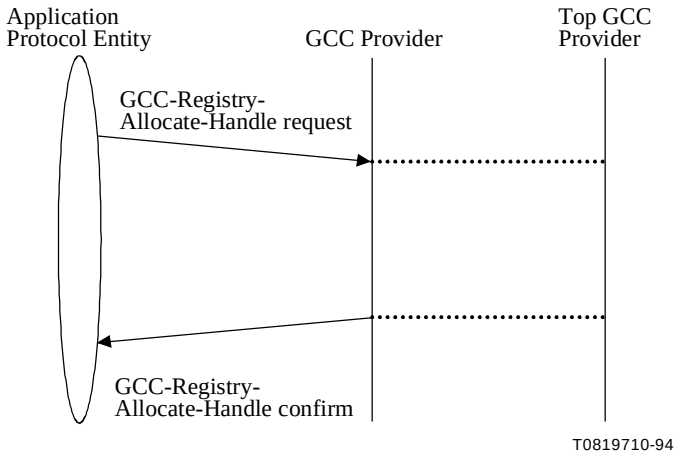

Figure 7-29/T.124 – GCC-Registry-Allocated-Handle – Sequence of primitives

7.5 Conference conductorship

GCC provides a method for allowing a Conventional node to become a conductor for a conference. A token is used by GCC to determine whether a conference is conducted or non-conducted. The Node Controller at a node may acquire the Conductor token by issuing a GCC-Conductor-Assign request primitive. When any node successfully acquires the Conductor token, the conference is placed in Conducted Mode. The Node Controller as well as all Application Protocol Entities at all nodes in the conference are made aware when a conference switches into Conducted Mode by means of a GCC-Conductor-Assign indication. This indication also indicates which node has become the conductor.

Whether a particular conference may operate in conducted mode or not is determined when the conference is created. If the Conference Conductible flag had been set, the conference may be placed in conducted mode. If not, any attempt to place the conference in conducted mode will be rejected.

A node may release conductorship and return the conference into Non-conducted Mode by issuing a GCC-Conductor-Release request primitive. The Node Controller as well as all Application Protocol Entities at all nodes in the conference are made aware when a conference switches into Non-conducted Mode by means of a GCC-Conductor-Release indication.

A node which is currently the conductor of a conference may pass conductorship directly to another node without placing the conference in Non-Conducted mode during the transition by issuing a GCC-Conductor-Give request specifying the desired recipient node. If the recipient accepts conductorship, the Node Controller as well as all Application Protocol Entities at all nodes in the conference are made aware of this transition by means of a GCC-Conductor-Assign indication which indicates which node is the new conductor. A node may explicitly request from the conductor that conductorship be given to it by issuing a GCC-Conductor-Please request primitive. The current conductor may choose to give the requesting node conductorship, or may ignore the request.

The Node Controller as well as Application Protocol Entities at any node may also inquire as to which node, if any, currently holds the Conductor token by using the GCC-Conductor-Inquire request primitive.

When an Application Protocol Entity is made aware that the conference to which it is joined is in Conducted Mode, it shall immediately begin operating according to its Conducted Mode behaviour as prescribed by the specification of the corresponding Application Protocol. When an Application Protocol Entity is made aware that the conference to which it is joined is in Non-conducted mode, it shall immediately begin operating according to its Non-conducted Mode behaviour as prescribed by

the specification of the corresponding Application Protocol. A typical Application Protocol specification may, for example, state that the Application Protocol Entity at any node when in Conducted Mode must request permission from the Peer Application Protocol Entity at the conductor before taking any action, while in Non-conducted Mode, no such permission is required. However, the behaviour defined by Application Protocol specifications while in Conducted or Non-conducted Mode is a matter outside of the scope of this Recommendation.

7.5.1 Description of abstract services

The following is a list of the primitives defined in this subclause and a brief summary of the function of each:

- GCC-Conductor-Assign – Used by a Node Controller at a Conventional node to request conductorship of a conference. When a new node becomes conductor, the indication form of this primitive is used to announce this to all Node Controllers in the conference as well as to all Application Protocol Entities at all nodes in the conference.
- GCC-Conductor-Release – Used by a Node Controller at a Conventional node to release conductorship of a conference. When any node releases conductorship, the indication form of this primitive is used to announce this to all Node Controllers in the conference as well as to all Application Protocol Entities at all nodes in the conference.
- GCC-Conductor-Please – Used by a Node Controller at a Conventional node to request that conductorship be given to it from the current conductor.
- GCC-Conductor-Give – Used by a Node Controller at a Conventional node to pass conductorship to specified node.
- GCC-Conductor-Inquire – Used by a Node Controller or an Application Protocol Entity to determine whether the conference is currently conducted nor non-conducted, and if conducted, the Node ID of the current conductor.
- GCC-Conductor-Permission-Ask – Used by a Node Controller to request permission for Application Protocol Entities at that node to take actions which, in conducted-mode, require permission from the conductor.
- GCC-Conductor-Permission-Grant – Used by the Node Controller at the conducting node to indicate which subset of nodes in a conference have been granted conducted-mode permission.

7.5.1.1 GCC-Conductor-Assign

During non-conducted mode the Node Controller at any Conventional node has the possibility to issue a request to become conductor, by issuing a GCC-Conductor-Assign request primitive. The receipt of a GCC-Conductor-Assign confirm primitive indicates whether the requester has become the conductor or not, depending on the result parameter within the primitive. A successful GCC-Conductor-Assign request shall be accompanied by GCC-Conductor-Assign indications to all Node Controllers in the conference as well as to all enrolled Application Protocol Entities at all nodes in the conference signalling that the conference has become conducted, and giving information about the identity of the conductor. The indication form of this primitive may also be issued to an Application Protocol Entity upon enrolling into a conference to inform it that the conference is in conducted mode. The order of GCC-Conductor-Assign and GCC-Conductor-Release indications represents the actual order of conductorship transitions. Table 7-41 shows the parameters and types of this primitive. Figure 7-30 shows the sequence of events when using this primitive.

NOTE – If the conductor disconnects from the conference for any reason, the conference reverts to non-conducted mode until another node issues a request to become conductor.

**Table 7-41/T.124 – GCC-Conductor-Assign –
Types of primitives and their parameters**

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Requesting Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Requesting Node ID: The Node ID of the requesting node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, non-conductible conference, token already owned.

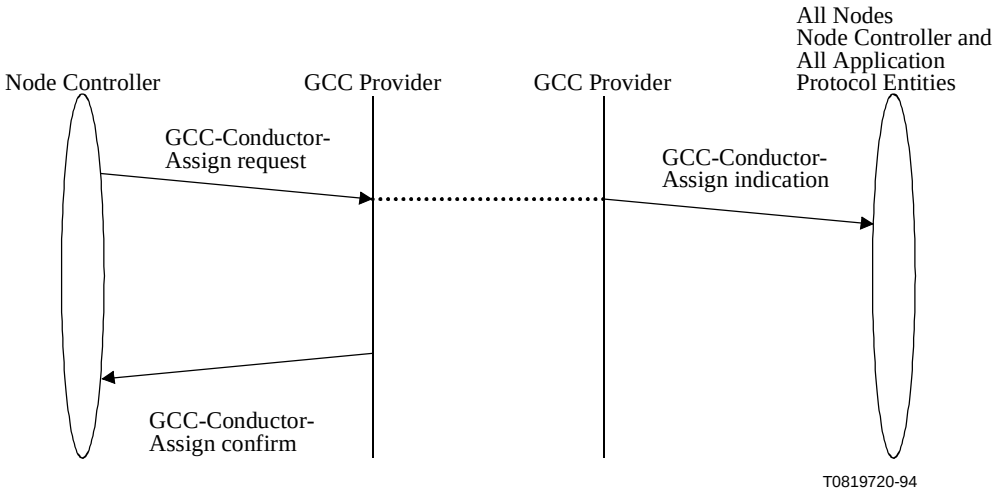

Figure 7-30/T.124 – GCC-Conductor-Assign – Sequence of primitives

7.5.1.2 GCC-Conductor-Release

In order to change its status from conductor to normal participant, the Node Controller at the conductor node issues the GCC-Conductor-Release request primitive. As soon as the conductor has requested the release of conductorship, all Node Controllers in the conference as well as all enrolled Application Protocol Entities at all nodes in the conference are informed of the change in operational mode by means of the GCC-Conductor-Release indication. The order of GCC-Conductor-Assign and GCC-Conductor-Release indications represents the actual order of conductorship transitions. Any participant issuing a GCC-Conductor-Release, but not being the current conductor, shall be answered by GCC with a GCC-Conductor-Release confirmation containing a negative result and reason description. In this situation, no indications shall be sent to the other participants. Apart from this user-initiated transition to non-conducted mode, the release of conductorship can also be initiated by GCC itself, for instance because the conductor disconnected from the conference. The indication form of this primitive may also be issued to an Application Protocol Entity upon enrolling into a conference to inform it that the conference is in non-conducted mode. Table 7-42 shows the parameters and types of this primitive. The sequences of primitives belonging to both situations are depicted in Figures 7-31 and 7-32, respectively.

Table 7-42/T.124 – GCC-Conductor-Release –
Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Accept/Reject: This flag indicates whether the GCC-Conductor-Release request was accepted or rejected.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, did not own token.

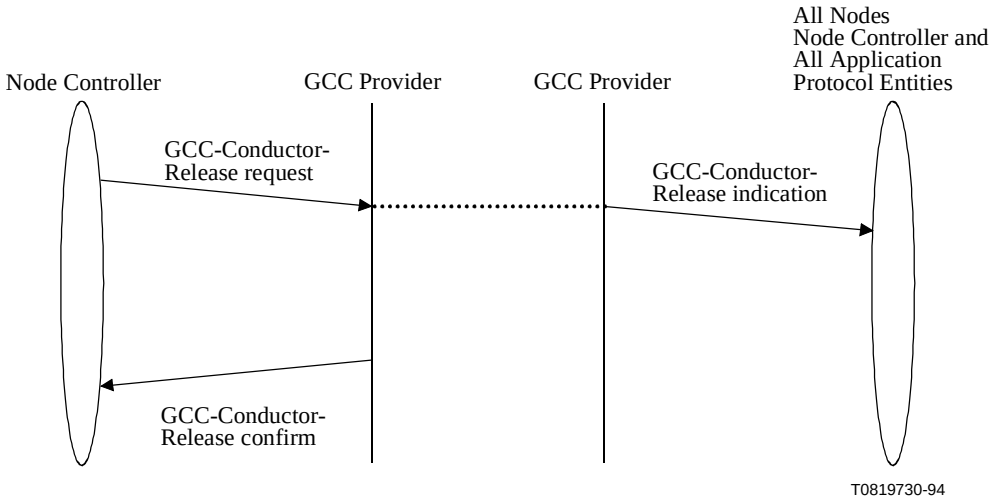

Figure 7-31/T.124 – GCC-Conductor-Release (user-initiated) – Sequence of primitives

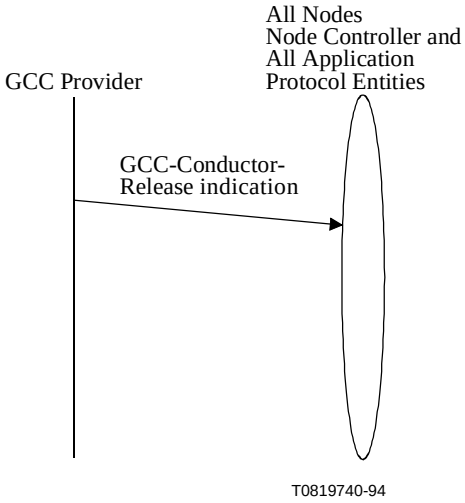

Figure 7-32/T.124 – GCC-Conductor-Release (GCC-initiated) – Sequence of primitives

7.5.1.3 GCC-Conductor-Please

The primitive GCC-Conductor-Please request may be issued by a Node Controller at a Conventional node to ask the current conductor to give conductorship to the requesting node. The GCC-Conductor-Please indication is forwarded to the current conducting node. The current conductor may then choose to give conductorship to the requester by using the GCC-Conductor-Give primitive. The confirm portion of this primitive is only local confirmation that the request was accepted by the local GCC Provider. No confirmation from the conductor is given directly. Table 7-43 shows the parameters and types of this primitive. Figure 7-33 shows the sequence of events when using this primitive.

Table 7-43/T.124 – GCC-Conductor-Please – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Requesting Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Requesting Node ID: The Node ID of the requesting node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, not in conducted mode.

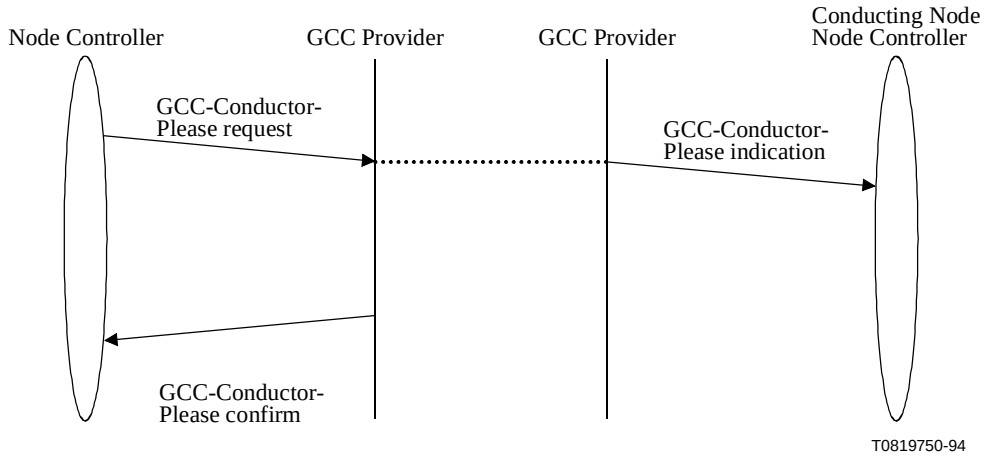

Figure 7-33/T.124 – GCC-Conductor-Please – Sequence of primitives

NOTE – It is possible to use this primitive (in conjunction with others) to ensure that a particular node becomes the conference conductor with no opportunity for another node to acquire conductorship. The procedure would be for the conference convener first to create a conference which is locked and to then acquire conductorship using the GCC-Conductor-Assign primitive. Once the convener has become the conference conductor, it may then allow other nodes into the conference either by unlocking it using GCC-Conference-Unlock and/or by adding the other nodes directly. Once the intended conducting node has joined the conference, conductorship may be passed directly to that node by use of the GCC-Conductor-Give primitive.

7.5.1.4 GCC-Conductor-Give

The primitive GCC-Conductor-Give request may be issued by a Node Controller to transfer conductorship to a specific node. If the conductorship is not accepted by the intended recipient, conference conductorship continues to be held by the original conductor. Table 7-44 shows the parameters and types of this primitive. Figure 7-34 shows the sequence of events when using this primitive.

Table 7-44/T.124 – GCC-Conductor-Give – Types of primitives and their parameters

Parameter	Request	Indication	Response	Confirm
Conference ID	M	M	M(=IN)	M(=RQ)
Recipient Node ID	M			M(=)
Result			M	M(=)

Conference ID: Identifier of the conference to which the primitive refers.

Recipient Node ID: The Node ID of the node to which conductorship is being transferred.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, not conductor, give not accepted.

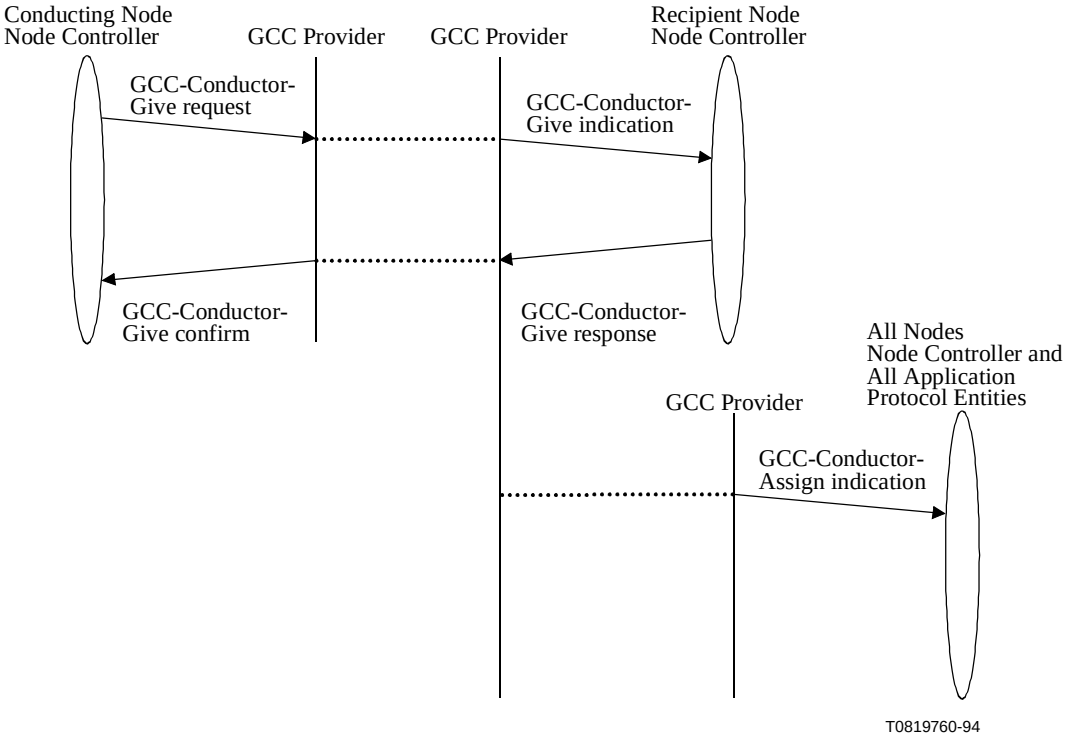

Figure 7-34/T.124 – GCC-Conductor-Give – Sequence of primitives

7.5.1.5 GCC-Conductor-Inquire

The primitive GCC-Conductor-Inquire request may be issued at any time by either a Node Controller or Application Protocol Entity in order to find out whether the conference is conducted or not, and if so, which node is the conductor, and if the requesting node has been granted conducted-mode

permission. Table 7-45 shows the parameters and types of this primitive. Figure 7-35 shows the sequence of events when using this primitive.

Table 7-45/T.124 – GCC-Conductor-Inquire –
Types of primitives and their parameters

Parameter	Request	Confirm
Conference ID	M	M(=)
Conducted vs. Non-Conducted Flag		M
Conductor Node ID		C
Permission flag		C
Result		M

Conference ID: Identifier of the conference to which the primitive refers.

Conducted vs. Non-Conducted Flag: A flag indicating whether the indicated conference is currently in conducted mode or non-conducted mode.

Conductor Node ID: The Node ID of the node that is currently conductor. Not present if currently in non-conducted mode.

Permission Flag: If in conducted mode, this flag indicates whether or not the local node has been granted conducted-mode permission.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

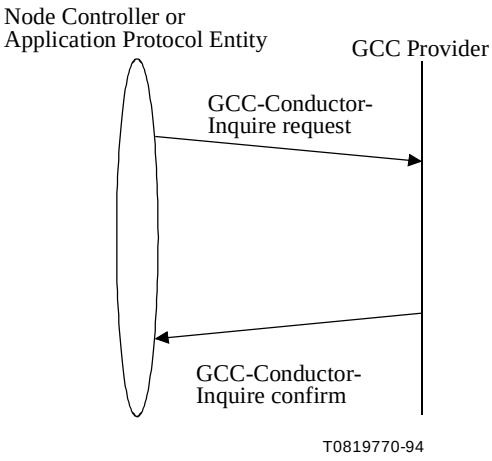

Figure 7-35/T.124 – GCC-Conductor-Inquire – Sequence of primitives

7.5.1.6 GCC-Conductor-Permission-Ask

The primitive GCC-Conductor-Permission-Ask request may be issued by a Node Controller to ask the current conductor to grant (or release) permission to allow Application Protocol Entities at the requesting node to perform any actions for which permission from the conductor is required. The definition of which specific actions require this permission is a matter for the individual Application Protocol specifications to define. The GCC-Conductor-Permission-Ask indication is forwarded to the current conducting node. The order of GCC-Conductor-Permission-Ask indications from a single

node represents the actual order of the requests from that node. The current conductor may then choose to grant (or release) conducted-mode permission to the requester by using the GCC-Conductor-Permission-Grant primitive, or to ignore the request. The confirm portion of this primitive is only local confirmation that the request was accepted by the local GCC Provider. No confirmation from the conductor is given directly. Table 7-46 shows the parameters and types of this primitive. Figure 7-36 shows the sequence of events when using this primitive.

Table 7-46/T.124 – GCC-Conductor-Permission-Ask – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Grant/Release Flag	M	M(=)	M(=)
Requesting Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Give/Release Flag: This flag indicates whether the requester desires to be granted conducted-mode permission, or if the requester desires to release conducted-mode permission.

Requesting Node ID: The Node ID of the requesting node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, not in conducted mode.

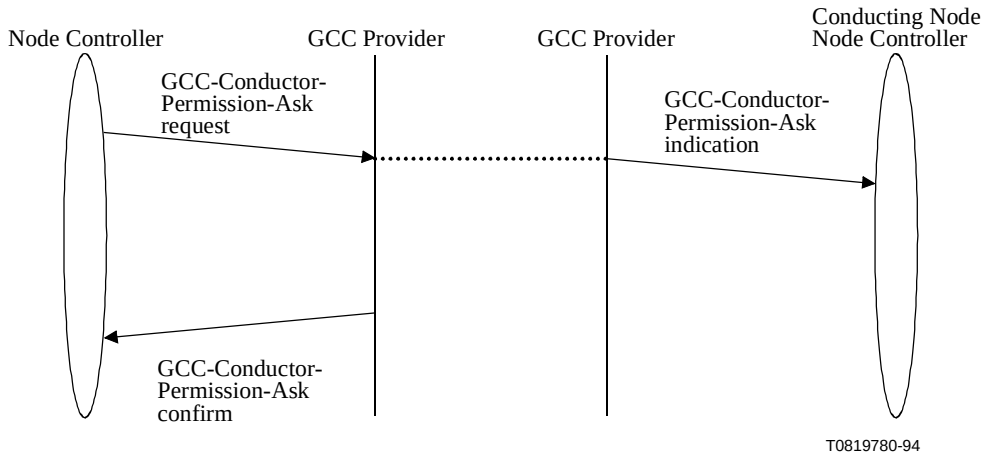

Figure 7-36/T.124 – GCC-Conductor-Permission-Ask – Sequence of primitives

7.5.1.7 GCC-Conductor-Permission-Grant

The primitive GCC-Conductor-Permission-Grant request may be issued by a Node Controller of the conducting node to grant or revoke conducted-mode permission from one or more nodes in a conference. The corresponding GCC-Conductor-Permission-Grant indication is broadcast to every node in the conference and indicates which nodes currently have conducted-mode permission, and also which have requested, but are still waiting for permission. The latter list may be given in the order that the conductor believes permission will ultimately be granted. The indication is given to the

Node Controller as well as all enrolled Application Protocol Entities at each node. The order of multiple GCC-Conductor-Permission-Grant indications represents the actual order of the requests from the conductor – that is, the most recently received indication applies. Conducted-mode permission is typically given in response to a GCC-Conductor-Permission-Ask, but may also be given unsollicited by the conductor. The conducted node, itself, is assumed to have conducted-mode permission whether or not it is explicitly listed in the list of nodes granted permission. When a conference first becomes conducted it is to be assumed that no nodes have permission – this is true even if the conference had previously been in conducted mode. If the conductorship changes hands via a successful GCC-Conductor-Give operation, the states of permission are left as they were last broadcast by the original conductor. When a new node joins a conference, the node shall assume that it has no permission. In this case, the Node Controller at the conducting node may re-broadcast the permission list by re-issuing a CC-Conductor-Permission-Grant request so that the new node is made aware of the permission status of other nodes in the conference. The confirm portion of this primitive is only local confirmation that the request was accepted by the local GCC Provider. Table 7-47 shows the parameters and types of this primitive. Figure 7-37 shows the sequence of events when using this primitive.

Table 7-47/T.124 – GCC-Conductor-Permission-Grant – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
List of Nodes Granted Permission	M	M(=)	
List of Nodes Waiting for Permission	O	O(=)	
Permission Flag		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

List of Nodes Granted Permission: A list of Node IDs, one for each node which the conductor has granted conducted-mode permission. If this list is empty, then no nodes have conducted-mode permission.

List of Nodes Waiting for Permission: An ordered list of Node IDs, one for each node which is being considered by the conductor to receive conducted-mode permission, but has not yet received it. The conductor may order the list in the order in which it is expected that permission will be granted. If so, the first item in the list is to be considered the next node likely to receive permission.

Permission Flag: This flag indicates whether or not the local node is present on the list of nodes granted permission. This is primarily to allow Application Protocol Entities to easily determine their mode of operation without requiring them to search through the returned lists.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, not conductor.

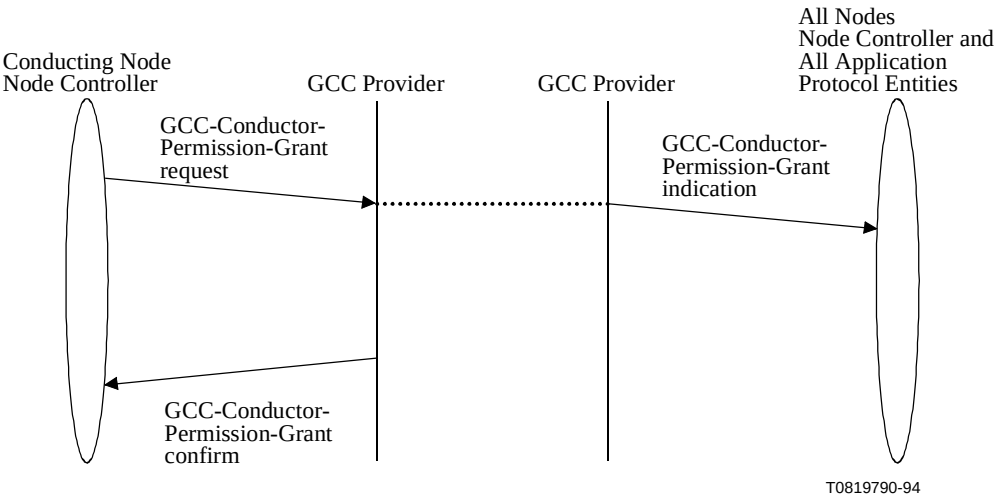

Figure 7-37/T.124 – GCC-Conductor-Permission-Grant – Sequence of primitives

7.6 Miscellaneous functions

7.6.1 Description of abstract services

The following is a list of the primitives defined in this subclause and a brief summary of the function of each:

- GCC-Conference-Time-Remaining – Allows the Node Controller at the conference convener to notify all nodes that a timed conference is scheduled to end at a particular time.
- GCC-Conference-Time-Inquire – Allows the Node Controller at any node to find out how much time is remaining in a timed conference.
- GCC-Conference-Extend – Allows the Node Controller at any node to request that the convener extend a timed conference beyond its allocated time duration.
- GCC-Conference-Assistance – Used to request some unspecified form of assistance from a conference operator.
- GCC-Text-Message – Used to send an arbitrary text message to a set of other nodes for display to the user or users at those nodes.

7.6.1.1 GCC-Conference-Time-Remaining

The GCC-Conference-Time-Remaining request primitive may be used by the Node Controller to announce to all nodes that a certain amount of time is remaining in a timed conference. It may also be issued indicating the time remaining for a specific node, rather than for all nodes. This request is intended to be issued by the Convener, but may be issued by other nodes as well. Typically, this primitive would be issued once, near the end of a timed conference, to indicate that the conference is almost over. Table 7-48 shows the parameters and types of this primitive. Figure 7-38 shows the sequence of events when using this primitive.

Table 7-48/T.124 – GCC-Conference-Time-Remaining – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Time Remaining	M	M(=)	
Node ID	O	O(=)	
Source Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Time Remaining: Indication of time remaining in conference in one second increments.

Node ID: Optional parameter which, if present, indicates that the indicated time-remaining only applies to the specific node listed. If not included, the time-remaining applies to all nodes in the conferences.

Source Node ID: Node ID of the node which issued the request.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

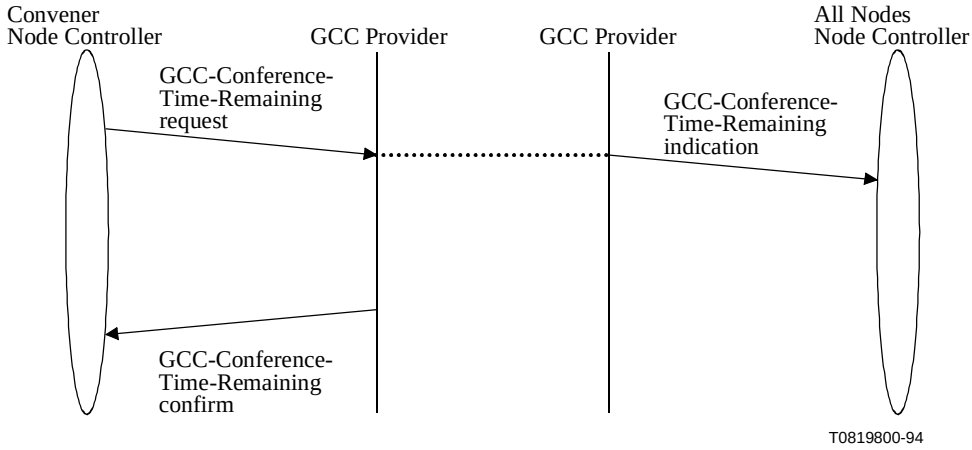

Figure 7-38/T.124 – GCC-Conference-Time-Remaining – Sequence of primitives

7.6.1.2 GCC-Conference-Time-Inquire

The GCC-Conference-Time-Inquire request primitive may be used by any node to find out from the Convener how much time is remaining in the conference. Receipt of the indication form of this primitive by the Convener (if the Convener supports this primitive) results in the time remaining to be broadcast to all nodes in the conference using the GCC-Conference-Time-Remaining primitive. Table 7-49 shows the parameters and types of this primitive. Figure 7-39 shows the sequence of events when using this primitive.

Table 7-49/T.124 – GCC-Conference-Time-Inquire – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Conference-Wide vs. Node-Specific Time Flag	M	M(=)	
Requesting Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Conference-Wide vs. Node-Specific Time Flag: This flag indicates if the request is to find out the time remaining for the entire conference, or, if different, the time remaining for the requesting node. If the Convener only considers a single conference-wide end time, it may ignore this flag.

Requesting Node ID: Node ID of the requesting node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

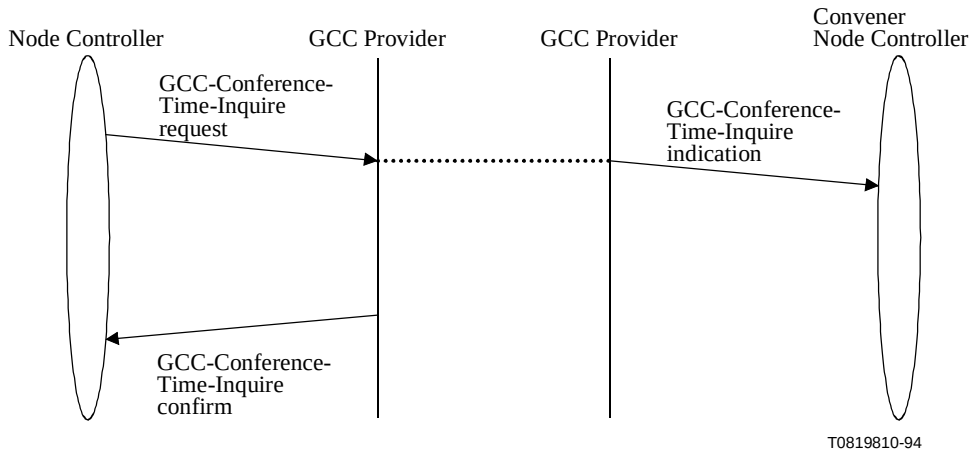

Figure 7-39/T.124 – GCC-Conference-Time-Inquire – Sequence of primitives

7.6.1.3 GCC-Conference-Extend

The GCC-Conference-Extend request primitive may be used by the Node Controller at a node to request from the Convener that more time be added to a timed conference. Receipt of the indication form of this primitive by the Convener (if the Convener supports this primitive) results in a broadcast of the new time remaining in the conference to all nodes using the GCC-Conference-Time-Remaining primitive (even if the time was not actually extended). There is no requirement that the actual amount of time the convener adds to the time-remaining equal the requested time. Table 7-50 shows the parameters and types of this primitive. Figure 7-40 shows the sequence of events when using this primitive.

Table 7-50/T.124 – GCC-Conference-Extend –
Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Time Requested	M	M(=)	M(=)
Conference-Wide vs. Node-Specific Time Flag	M	M(=)	
Requesting Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Time Requested: In the request and indication, this parameter indicates the desired amount of time to extend the conference in one second increments.

Conference-Wide vs. Node-Specific Time Flag: This flag indicates if the request is to extend the time remaining for the entire conference, or the time remaining for the requesting node. If the Convener only considers a single conference-wide end time, it may ignore this flag.

Requesting Node ID: Node ID of the requesting node.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference, not conductor.

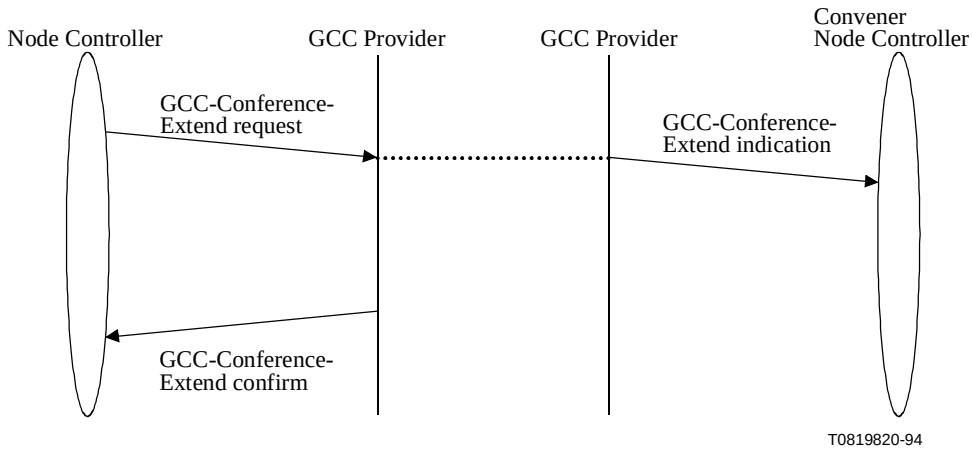

Figure 7-40/T.124 – GCC-Conference-Extend – Sequence of primitives

7.6.1.4 GCC-Conference-Assistance

The GCC-Conference-Assistance primitive provides a simple means to request some form of assistance from a conference operator. Issuing a GCC-Conference-Assistance request results in a GCC-Conference-Assistance indication to be broadcast to the Node Controller at all nodes in the specified conference which support this primitive. The intended response to this primitive is unspecified and outside the scope of this Recommendation. Table 7-51 shows the parameters and types of this primitive. Figure 7-41 shows the sequence of events when using this primitive.

Table 7-51/T.124 – GCC-Conference-Assistance – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
User Data	O	O(=)	
Source Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

User Data: Unspecified user data.

Source Node ID: The Node ID of the source of the assistance request.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

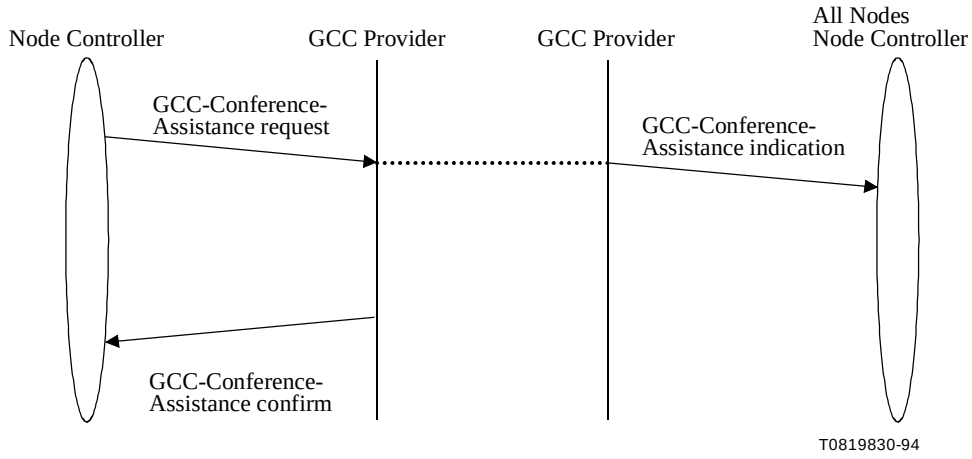

Figure 7-41/T.124 – GCC-Conference-Assistance – Sequence of primitives

7.6.1.5 GCC-Text-Message

The GCC-Text-Message primitive provides a simple means to communicate unspecified text messages. Issuing a GCC-Text-Message request results in a GCC-Text-Message indication to be either broadcast to the Node Controller at all nodes in the specified conference which support this primitive, or to be sent to a single node. The intended response to this primitive is to display the text message to the conference participants by some means unspecified by this Recommendation. Table 7-52 shows the parameters and types of this primitive. Figure 7-42 shows the sequence of events when using this primitive.

Table 7-52/T.124 – GCC-Text-Message – Types of primitives and their parameters

Parameter	Request	Indication	Confirm
Conference ID	M	M	M(=RQ)
Text Message	M	M(=)	
Destination Node ID	O		
Source Node ID		M	
Result			M

Conference ID: Identifier of the conference to which the primitive refers.

Text Message: Unicode text message.

Destination Node ID: The Node ID of a single node to receive the text message. If no node is specified, the message is broadcast to all nodes in the specified conference.

Source Node ID: The Node ID of the source of the text message.

Result: An indication of whether the request was accepted or rejected, and if rejected, the reason why. It contains one of a list of possible results: successful, invalid conference.

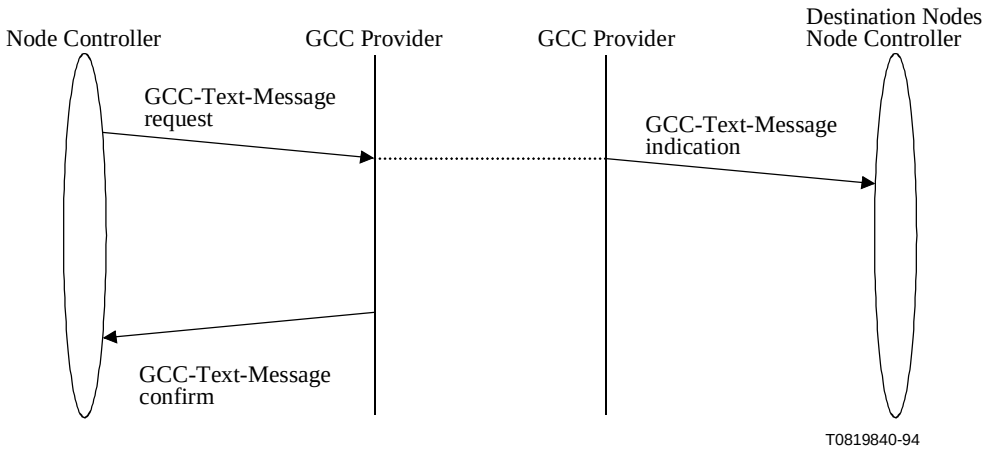

Figure 7-42/T.124 – GCC-Text-Message – Sequence of primitives

8 GCC Protocol Specification

8.1 General operation

A GCC Provider in any node is the MCS Control application, communicating with MCS via the Control MCSAP. On initialization, the GCC Provider shall establish its communication link with MCS through a Control MCSAP through local means.

In all cases, a Conference is established by creating an MCS Domain. A Conference has a one-to-one correspondence with a single MCS Domain. The actual creation of an MCS domain is done through local means. Conferences are created or joined by use of the MCS-Connect-Provider primitives.

For all request primitives which refer to a particular conference (via the Conference ID), the local GCC Provider to which the primitive is issued shall determine if the specified conference is one to which the node is currently joined. If so, the GCC Provider proceeds as described below for each

primitive. If not, the request is rejected and the corresponding confirm primitive is issued specifying invalid-conference as the reason for rejection. Any response primitive issued which refers to an invalid conference is ignored by the GCC Provider (or the error is handled by some unspecified local means).

GCC Providers communicate with each other via GCC Protocol Data Units (GCCPDUs). The GCCPDUs are transmitted either via the two MCS data service primitives (MCS-Send-Data or MCS-Uniform-Send-Data; see Table 9-4) or via MCS-Connect-Provider primitives for GCCPDUs used during connection set-up.

All GCCPDUs are categorized into either request, response, or indication classes. Request PDUs are defined to be those that require a corresponding response PDU in return. Indication PDUs are those that do not require a response (or in some cases, where the response is provided indirectly). For request PDUs which do not correspond to mandatory functionality, a generic response PDU is provided (FunctionNotSupportedResponse), which allows the GCC Provider receiving a request to respond to the requesting node without needing to have knowledge of the format of the specific response PDU. To allow the requester to know what this PDU is in response to, the entire request PDU is included within the FunctionNotSupportedResponse PDU. The FunctionNotSupportedResponse PDU shall be sent at the same priority level as that of the received request PDU.

NOTE – This terminology (request, response, and indication) does not have a one-to-one correspondence to the definition of request, indication, response, and confirm used in the definition of the primitives. The terminology has been chosen to relate to the primary purpose of the PDU types with respect to the functions that they will perform. For this reason, the term confirm was not needed to describe PDUs – response PDUs are those which are sourced from a response primitive and result in a confirm primitive.

The GCC protocol includes support for non-standard extensions. On receipt of a nonStandardRequest PDU, a GCC Provider which does not understand the request, shall issue a FunctionNotSupportedResponse PDU in return. nonStandardResponse and nonStandardIndication PDUs may be ignored by a GCC Provider.

8.2 Conference establishment and termination

8.2.1 Conference creation

On receipt of the primitive GCC-Conference-Create request, a GCC Provider shall issue an MCS-Connect-Provider request primitive with the parameters shown in Table 8-1. The local GCC Provider shall allocate the Conference ID, which shall be used as the local MCS Domain Selector associated with the created conference. The Conference ID is included as the Calling Domain Selector. It is also maintained by the local GCC Provider as the means identifying this conference in future primitives.

If the combination of the Conference Name and Conference Name Modifier parameters are identical to those of a conference to which the local node is already joined (either the numerical or text forms of the name), the request is instead immediately rejected by issuing a GCC-Conference-Create response with conference-name-already-exists as the result. Otherwise, the GCC Provider shall retain the Conference Name and Conference Name Modifier (in addition to including the Conference Name in the ConferenceCreateRequest PDU) to be used in the procedures for responding to a conference query, conference join, or in initiating a conference invite.

Table 8-1/T.124 – MCS-Connect-Provider request parameters for ConferenceCreateRequest PDU

Parameter	Contents
Calling Address	From request primitive
Calling Domain Selector	Conference ID as chosen by the GCC Provider
Called Address	From request primitive
Called Domain Selector	NULL
Upward/Downward Flag	Up
Domain Parameters	From request primitive
Quality of Service	From request primitive
User Data	T.124 Object Identifier
	ConferenceCreateRequest PDU (see Table 8-2)

The User Data parameter of the MCS-Connect-Provider request contains a structure which includes an Object Identifier identifying the contained PDU as adhering to this Recommendation followed by the PDU itself. The details of this structure are defined in 9.6. The contents of this PDU are shown in Table 8-2.

Table 8-2/T.124 – ConferenceCreateRequest GCCPDU

Content	Source	Sink
Conference Name	Request	Indication and Destination GCC Provider
Convener Password (optional)	Request	Indication
Password (optional)	Request	Indication
Locked Conference Flag	Request	Indication and Destination GCC Provider
Listed Conference Flag	Request	Indication and Destination GCC Provider
Conductible Conference Flag	Request	Indication and Destination GCC Provider
Termination Method	Request	Indication and Destination GCC Provider
Conductor Privilege List (optional)	Request	Indication and Destination GCC Provider
Conducted-Mode Conference Privilege List (optional)	Request	Indication and Destination GCC Provider
Non-Conducted-Mode Conference Privilege List (optional)	Request	Indication and Destination GCC Provider
Conference Description (optional)	Request	Indication and Destination GCC Provider
Caller Identifier (optional)	Request	Indication
Conference Mode (optional)	Request	Indication
User Data (optional)	Request	Indication

On receipt of the MCS-Connect-Provider indication that includes a T.124 Object Identifier and the ConferenceCreateRequest PDU, a GCC Provider shall generate a GCC-Conference-Create indication primitive with the parameters as specified in the included ConferenceCreateRequest PDU. It shall issue this primitive to the Control GCCSAP. The GCC Provider shall also allocate a Conference ID, a locally unique string, which shall also be included in this primitive. If the conference is

successfully created, the GCC Provider shall use this Conference ID as the means of identifying this conference in future primitives. If GCC does not have the resources necessary to create a new conference, it may generate the negative response automatically without generating the GCC-Conference-Create indication. Otherwise, on receipt of a successful GCC-Conference-Create response from the Control GCCSAP, the GCC Provider (now the Top GCC Provider for this conference) shall issue an MCS-Attach-User request. On receipt of the MCS-Attach-User confirm which contains the allocated Node ID, the GCC Provider shall then join the corresponding Node ID Channel by issuing an MCS-Channel-Join request. The GCC Provider shall also join the appropriate GCC Broadcast Channel(s) by issuing one or two MCS-Channel-Join requests (successive such requests may be issued prior to receiving the previous confirm). It may be necessary, prior to attaching, to locally indicate to the MCS Provider that a new domain has been created. Any exchange necessary to do this is considered a local matter not covered by this Recommendation.

There are three different GCC Broadcast Channels, each of which broadcasts similar, but different, sets of information. Earlier versions of GCC used a single broadcast channel known as the GCC-Broadcast-Channel. On this channel, older Conventional nodes that existed prior to the introduction of Node Categories receive GCC control messages and Full Roster Refreshes. To support all possible GCC nodes that may participate in a conference, it is imperative that nodes continue to support the original GCC-Broadcast-Channel even though no nodes may be joined to it (an exception to this may be Terminal Nodes which are Anonymous). A second GCC Broadcast Channel, known as the GCC-Conventional-Broadcast-Channel, is used by nodes that are Node Category-aware. All GCC control PDUs and Roster delta updates that are generated when Conventional Nodes join, leave, or change a roster record are broadcasted on this channel. All nodes (except for older protocol nodes) must join this broadcast channel. Anonymous nodes may ignore the roster deltas received on this channel, but they must process the control messages received. The last GCC Broadcast Channel is known as the GCC-Counted-Broadcast-Channel. On this channel, the Top Provider broadcasts Roster delta updates which are generated when Counted Nodes join, leave, or change a roster record. Only Conventional Nodes join this GCC Broadcast Channel which reduces the amount of network traffic incurred when a large number of Counted Nodes are participating in a conference.

If the GCC-Conference-Create response includes a Conference Name Modifier parameter, the GCC Provider (now the Top GCC Provider) shall retain this name modifier for later use in handling the conference query, conference join, and conference invite procedures.

The GCC Provider shall generate an MCS-Connect-Provider response which includes a result which is either success, or user-rejected depending on whether or not the Result parameter in the GCC-Conference-Create response primitive indicated success or failure. The User Data parameter includes the T.124 Object Identifier as well as the ConferenceCreateResponse PDU. The contents of the connect provider primitive are shown in Table 8-3. In the case of successful conference creation, the GCC Provider at the node receiving the MCS-Connect-Provider indication shall become the Top GCC Provider for the conference.

Table 8-3/T.124 – MCS-Connect-Provider response parameters for ConferenceCreateResponse PDU

Parameter	Contents
Domain Parameters	From response primitive
Quality of Service	From response primitive
Result	As specified in Rec. T.122
User Data	T.124 Object Identifier
	ConferenceCreateResponse PDU (see Table 8-4)

The ConferenceCreateResponse PDU is shown in Table 8-4. The Node ID parameter, which is the User ID assigned by MCS in response to the MCS-Attach-User request issued by the GCC Provider, shall be supplied by the GCC Provider sourcing this PDU. The Tag parameter is assigned by the source GCC Provider to be locally unique. It is used to identify the returned UserIDIndication PDU. The Result parameter includes GCC-specific failure information sourced directly from the Result parameter in the GCC-Conference-Create response primitive. If the Result parameter is anything except successful, the Result parameter in the MCS-Connect-Provider response is set to user-rejected.

Table 8-4/T.124 – ConferenceCreateResponse GCCPDU

Content	Source	Sink
Node ID	Top GCC Provider	Destination GCC Provider
Tag	Top GCC Provider	Destination GCC Provider
Result	Response	Confirm
User Data (optional)	Response	Confirm

On receipt of the ConferenceCreateResponse PDU, if the PDU indicated a successful result, a GCC Provider shall first issue an MCS-Attach-User request. On receipt of the MCS-Attach-User confirm which contains the allocated Node ID, the GCC Provider shall then join the Node ID Channel by issuing an MCS-Channel-Join request. The GCC Provider shall also join the GCC-Conventional-Broadcast-Channel and the GCC-Counted-Broadcast-Channel by issuing two MCS-Channel-Join requests (Convener nodes are always considered to be Conventional; see explanation of Broadcast Channels above). The GCC Provider may also join the GCC-Convener-Channel (if it supports any of the functions which require use of this channel) by issuing an MCS-Channel-Join request. Once the GCC Provider has received an MCS-Channel-Join confirm from each of the channel join requests (successive requests may be issued prior to receiving the previous confirm), it shall send a UserIDIndication PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying as the Channel ID the Node ID of the Top GCC Provider as contained in the received ConferenceCreateResponse PDU, specifying Top data priority, and including the PDU in the Data field. The content of the UserIDIndication PDU is shown in Table 8-5. The Tag parameter is filled in with the value of the corresponding parameter received in the ConferenceCreateResponse PDU.

Table 8-5/T.124 – UserIDIndication GCCPDU

Content	Source	Sink
Tag	Source GCC Provider	Destination GCC Provider

The GCC Provider shall then generate a GCC-Conference-Create confirm primitive and issue it to the Control GCCSAP. This primitive shall include the Conference Name from the original request primitive, the Modified Conference Name (if any) and Result parameters from the received PDU, as well as the locally-allocated Conference ID. If the received PDU had indicated an unsuccessful result, or if the GCC Provider receives an MCS-Disconnect-Provider indication for this connection prior to having issued a successful GCC-Conference-Create confirm, the GCC-Conference-Create confirm primitive is issued immediately, indicating an unsuccessful result, without issuing the attach-user or channel-join requests, and without sending the UserIDIndication PDU. The Result parameter in the PDU as well as the Result parameter reported in the MCS-Connect-Provider confirm (or the Reason parameter of the MCS-Disconnect-Provider) is used to generate the result reported in the GCC-Conference-Create confirm primitive. If the Result parameter of the

MCS-Connect-Provider confirm is user-rejected, the Result parameter in the PDU is used to determine the reported result. Otherwise, the Result parameter in the MCS-Connect-Provider is used directly.

On receipt of a UserIDIndication, the node at which the conference was created shall compare the Tag parameter to its list of outstanding Tags. If the Tag matches one of these, it shall save the User ID of the source node (extracted from the MCS-Send-Data indication) in its database of Node IDs of nodes which are directly below it in the connection hierarchy. If the Tag does not match any outstanding Tags, the PDU shall be ignored.

If the newly-created Top GCC Provider for this conference receives an MCS-Disconnect-Provider indication for the connection being established at any time during the process of creating the conference (this includes the time that the GCC-Conference-Create indication has been issued to the Control GCCSAP until the time that the UserIDIndication has been received), it shall issue a GCC-Conference-Terminate indication to the Control GCCSAP indicating the requested normal termination as the reason if user-initiated was the reason provided in the MCS-Disconnect-Provider indication, and error termination otherwise. It shall then stop further processing for this connection establishment procedure. If the MCS-Disconnect-Provider indication was received prior to issuing the GCC-Conference-Create indication to the Control GCCSAP, the GCC Provider shall stop further processing for this connection establishment procedure and take no further action.

The sequence of events for successful conference creation is shown in Figure 8-1.

NOTE – If a conference is created by local initiation rather than through a request from another node, the GCC Provider at that node shall perform the sequence of events performed by the Top GCC Provider shown in the figure beginning from the MCS-Attach-User request. That is, it shall attach to MCS by issuing an MCS-Attach-User request and wait for the confirm, then it shall join its Node ID channel, and the appropriate GCC Broadcast Channels. It may optionally join the GCC-Convener-Channel since this node is also the Convener in this case. In this case, if another node joins the conference at a later time indicating itself to be the convener (having issued the correct Convener Password), the Top GCC Provider may choose to accept the connection, relinquishing convener privileges to the new node. Note also that it is assumed that in this case, MCS Domain Parameters are set at the time of creation rather than at the time the first connection is established.

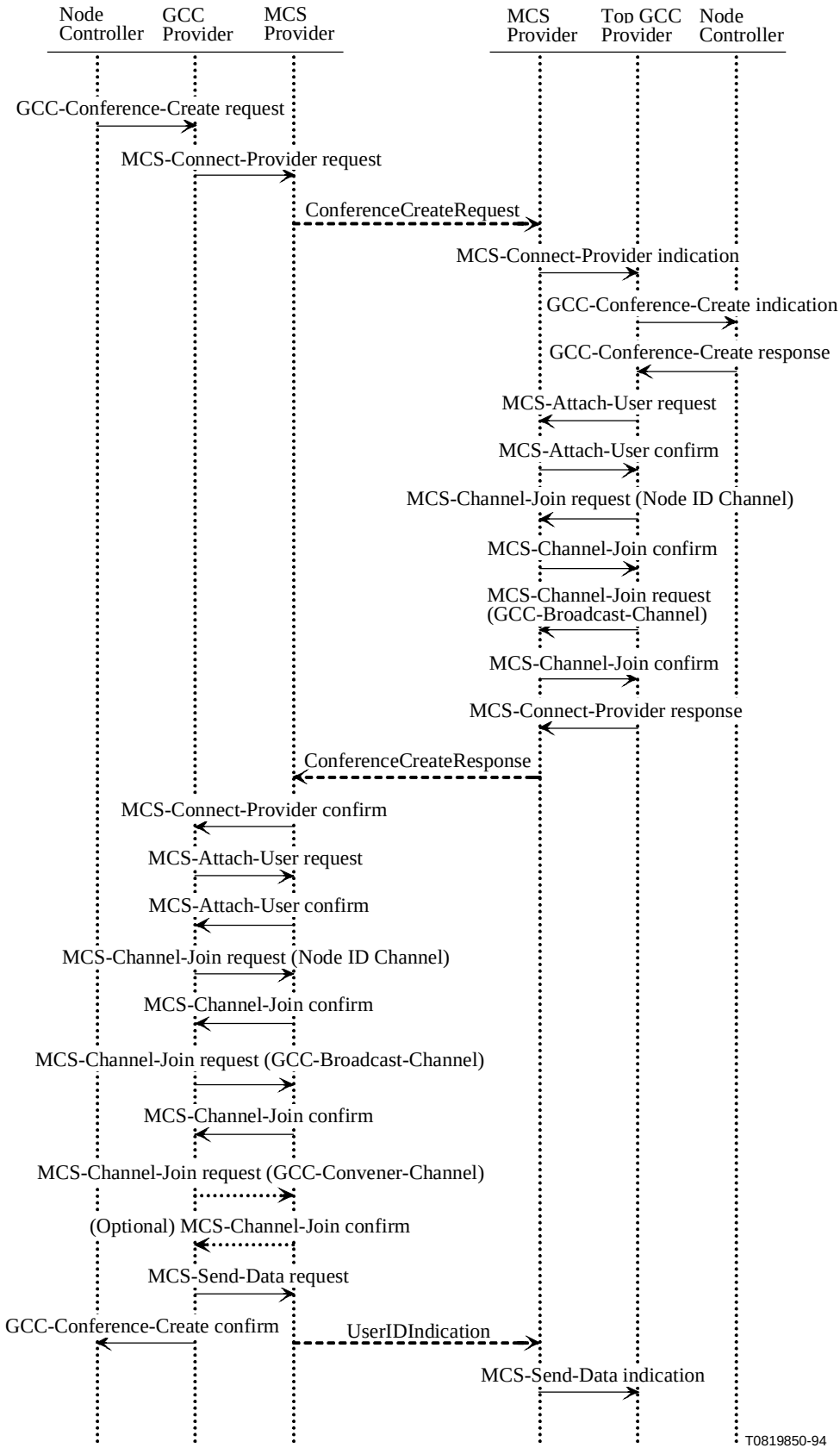

Figure 8-1/T.124 – Creating a conference

8.2.2 Querying conferences

On receipt of a GCC-Conference-Query request primitive, a GCC Provider shall issue an MCS-Connect-Provider request primitive with the parameters shown in Table 8-6.

Table 8-6/T.124 – MCS-Connect-Provider request parameters for ConferenceQueryRequest PDU

Parameter	Contents
Calling Address	From request primitive
Calling Domain Selector	NULL
Called Address	From request primitive
Called Domain Selector	NULL
Upward/Downward Flag	Up
Domain Parameters	Default Domain Parameters
Quality of Service	Default Quality of Service Parameters
User Data	T.124 Object Identifier
	ConferenceQueryRequest PDU (see Table 8-7)

The User Data parameter of the MCS-Connect-Provider request contains a structure which includes an Object Identifier identifying the contained PDU as adhering to this Recommendation followed by the PDU itself. The details of this structure are defined in 9.6. The contents of this PDU are shown in Table 8-7.

Table 8-7/T.124 – ConferenceQueryRequest GCCPDU

Content	Source	Sink
Node Type	Request	Indication
Asymmetry Indicator (conditional)	Request	Indication
User Data (optional)	Request	Indication

On receipt of the MCS-Connect-Provider indication that includes a T.124 Object Identifier and the ConferenceQueryRequest PDU, a GCC Provider shall generate a GCC-Conference-Query indication primitive with the parameters as specified in the ConferenceQueryRequest PDU. It shall issue this primitive to the Control GCCSAP. On receipt of a GCC-Conference-Query response from the Control GCCSAP, the GCC Provider shall send an MCS-Connect-Provider response which is rejected (i.e. no conference is set up), but includes the ConferenceQueryResponse PDU in the User Data field. The parameters of the MCS-Connect-Provider response are shown in Table 8-8.

Table 8-8/T.124 – MCS-Connect-Provider response parameters for ConferenceQueryResponse PDU

Parameter	Contents
Domain Parameters	Default Domain Parameters
Quality of Service	Default Quality of Service Parameters
Result	User-rejected
User Data	T.124 Object Identifier
	ConferenceQueryResponse PDU (see Table 8-9)

The ConferenceQueryResponse PDU is shown in Table 8-9. The Conference Descriptor List contains an entry for each listed conference to which the queried node is currently joined. The Conference Descriptor List is generated by the GCC Provider sourcing the ConferenceQueryResponse PDU. The descriptor is filled in with the Conference Name of each conference to which the GCC Provider is joined, the locally-maintained Conference Name Modifier for each conference, if any, the Conference Description, if any, the Locked/Unlocked flag, the Password In The Clear Required indicator, the Network Address field (if one is available), the default Conference Flag, and the Conference Mode. The Network Address field, if one exists, is sourced from the Local Network Address parameter of the GCC-Conference-Create request, the GCC-Conference-Create response, the GCC-Conference-Join request, or the GCC-Conference-Invite response, depending on how the conference was joined at this node. The Result parameter shall indicate success if the query request is able to be fulfilled (even if the Conference List is empty), or user-rejected if so indicated in the response primitive.

If the GCC Provider processing the MCS-Connect-Provider indication receives an MCS-Disconnect-Provider indication for the same connection at any time during the process, the GCC Provider shall stop further processing for this procedure and take no further action.

Table 8-9/T.124 – ConferenceQueryResponse GCCPDU

Content	Source	Sink
Node Type	Request	Confirm
Asymmetry Indicator (conditional)	Response	Confirm
Conference Descriptor List	Source GCC Provider	Confirm
Result	Response	Confirm
User Data (optional)	Response	Confirm

On receipt of an MCS-Connect-Provider response which includes the T.124 Object Identifier and a ConferenceQueryResponse PDU, a GCC Provider shall generate a GCC-Conference-Query confirm primitive and issue it to the Control GCCSAP. The content of the confirm primitive shall be obtained from the parameters of the ConferenceQueryResponse PDU. The Result parameter in the primitive, in particular, is obtained strictly from the Result parameter in the received PDU. The Result parameter of the MCS-Connect-Provider confirm is ignored since it would be set to user-rejected even in the case of a successful operation.

If the received PDU had indicated an unsuccessful result, or if the GCC Provider receives an MCS-Disconnect-Provider indication for this connection prior to having issued a successful GCC-Conference-Query confirm, the GCC-Conference-Query confirm primitive is issued immediately, indicating an unsuccessful result. The Result parameter in the PDU as well as the Result parameter reported in the MCS-Connect-Provider confirm (or the Reason parameter of the

MCS-Disconnect-Provider) is used to generate the result reported in the GCC-Conference-Query confirm primitive. If the Result parameter of the MCS-Connect-Provider confirm is user-rejected, the Result parameter in the PDU is used to determine the reported result. Otherwise, the Result parameter in the MCS-Connect-Provider is used directly.

8.2.3 Joining a conference

On receipt of the primitive GCC-Conference-Join request, a GCC Provider shall issue an MCS-Connect-Provider request primitive with the parameters shown in Table 8-10. The local GCC Provider allocates the Conference ID, which shall be used as the local MCS Domain Selector associated with the created conference. The Conference ID is sent as part of the MCS-Connect-Provider request as the Calling Domain Selector.

Table 8-10/T.124 – MCS-Connect-Provider request parameters for ConferenceJoinRequest PDU

Parameter	Contents
Calling Address	From request primitive
Calling Domain Selector	Conference ID as chosen by the GCC Provider
Called Address	From request primitive
Called Domain Selector	NULL
Upward/Downward Flag	Up
Domain Parameters	From request primitive
Quality of Service	From request primitive
User Data	T.124 Object Identifier
	ConferenceJoinRequest PDU (see Table 8-11)

The User Data parameter of the MCS-Connect-Provider request contains a structure which includes an Object Identifier identifying the contained PDU as adhering to this Recommendation followed by the PDU itself. The details of this structure are defined in 9.6. The contents of this PDU are shown in Table 8-11. From the joining node, the Tag parameter shall not be included. The Conference Name and Conference Name Modifier are sourced from the request primitive – the Conference Name Modifier parameter in the PDU is sourced from the Called Node Conference Name Modifier parameter of the primitive. The Calling Node Conference Name Modifier (also from the request primitive) is maintained at the local GCC Provider as the identifiers of the conference to be used in the procedures for response to a conference join request and conference query request.

Table 8-11/T.124 – ConferenceJoinRequest GCCPDU

Content	Source	Sink
Conference Name (conditional)	Request	GCC Provider receiving MCS-Connect-Provider indication
Conference Name Modifier (optional)	Request	GCC Provider receiving MCS-Connect-Provider indication
Tag (conditional)	GCC Provider receiving MCS-Connect-Provider indication	Top GCC Provider
Password (optional)	Request	Indication
Convener Password (optional)	Request	Indication
Caller Identifier (optional)	Request	Indication
Node category (optional)	Request	Indication
User Data (optional)	Request	Indication

On receipt of the MCS-Connect-Provider indication that includes a T.124 Object Identifier and the ConferenceJoinRequest PDU, the action of the GCC Provider depends on whether or not it is the Top GCC Provider of the conference specified by the Conference Name and Conference Name Modifier (if any). The GCC Provider determines which conference is to be joined by comparing the Conference Name and Conference Name Modifier to the list of conferences to which this node is joined. The Conference Name received in the ConferenceJoinRequest PDU may contain either the numerical or text forms of the Conference Name. The GCC Provider shall attempt to match the indicated name with the corresponding portion of the names of existing conferences. For a conference to be considered matched, both the Conference Name and Conference Name Modifier must match those of a current conference. If no Conference Name Modifier is given in the request, the GCC Provider shall match it only with an existing conference which also has no Conference Name Modifier.

If the GCC Provider which received the MCS-Connect-Provider indication is not the Top GCC Provider for the conference to which the requesting node wishes to join, and if the node is joined to the specified conference, the GCC Provider shall forward the ConferenceJoinRequest PDU to the Top-GCC-Provider of the specified conference by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying Top data priority, and including the PDU in the Data field. In this case, it shall assign a locally unique identifier and include it in the Tag parameter of the PDU. This number is used to identify the corresponding response PDU when it is returned from the Top GCC Provider. It may also leave out the Conference Name and Conference Name Modifier parameters from the received PDU as they are not needed by the Top GCC Provider.

When the Top-GCC-Provider receives this PDU, if the conference is not locked, it shall generate a GCC-Conference-Join indication primitive with the parameters as specified in the ConferenceJoinRequest PDU, as well as the Conference ID for the corresponding conference. It shall issue this primitive to the Control GCCSAP. Note that the Conference Name and Conference Name Modifier parameters are used only by the node receiving the MCS-Connect-Provider indication, and not by the Top GCC Provider. If the conference had been locked, the Top GCC Provider shall instead send a ConferenceJoinResponse PDU to the originator of the MCS-Send-Data containing the ConferenceJoinRequest PDU (not the original requester) by issuing an MCS-Send-Data request specifying the User ID of that node as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The PDU shall indicate locked-conference as the result code. Otherwise, on receipt of the GCC-Conference-Join response, the GCC Provider shall send a

ConferenceJoinResponse PDU to the originator of the MCS-Send-Data containing the ConferenceJoinRequest PDU (not the original requester) by issuing an MCS-Send-Data request specifying the User ID of that node as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The contents of the ConferenceJoinResponse PDU are shown in Table 8-13. The Node ID is not included in this portion of the response, but the Top Node ID parameter, the Tag, and the parameters associated with the Conference Profile are included, as well as the Result parameter. The Tag shall have the same value as the corresponding parameter in the received ConferenceJoinRequest PDU.

On receipt of the MCS-Send-Data indication containing this PDU, if the Tag parameter matches that of its locally-stored list of outstanding join requests (which allows it to identify the connection over which to send the MCS-Connect-Provider response), the GCC Provider shall generate an MCS-Connect-Provider response which includes a result which is either success, or user-rejected depending on whether or not the Result parameter in the received PDU indicated success or failure. The User Data parameter includes the T.124 Object Identifier as well as the ConferenceJoinResponse PDU. The Node ID parameter of the ConferenceJoinResponse PDU is filled in at this time indicating the Node ID of the node directly connected to the joining node. The contents of the connect provider primitive are shown in Table 8-12.

If the receiver of the original MCS-Connect-Provider indication is the Top-GCC-Provider, if the conference is not locked, it shall instead generate a GCC-Conference-Join indication primitive with the parameters as specified in the ConferenceJoinRequest PDU, as well as the Conference ID for the designated conference. It shall issue this primitive to the Control GCCSAP. If the conference had been locked, the Top GCC Provider shall instead generate an MCS-Connect-Provider response which indicates in the included ConferenceJoinResponse PDU locked-conference as the result code. The PDU shall indicate locked-conference as the result code. Otherwise, on receipt of the GCC-Conference-Join response, the GCC Provider generates an MCS-Connect-Provider response which includes the ConferenceJoinResponse PDU with a result which is either success, or user-rejected depending on whether or not the Result parameter in the GCC-Conference-Join response primitive indicated success or failure.

If the receiver of the original MCS-Connect-Provider indication is not currently joined to any conference (or if it does not support the GCC-Conference-Join indication primitive at all), it may reject the join request by immediately issuing an MCS-Connect-Provider response containing user-rejected as the result code, and including a ConferenceJoinResponse with invalid-conference respectively as the reason code in the PDU.

If the receiver of the original MCS-Connect-Provider indication is not joined to the conference specified in the Conference Name and Conference Name Modifier parameter of the ConferenceJoinRequest PDU, it shall reject the request by issuing an MCS-Connect-Provider response with the reason of no-such-domain.

Table 8-12/T.124 – MCS-Connect-Provider response parameters for ConferenceJoinResponse PDU

Parameter	Contents
Domain Parameters	From GCC Provider as previously saved
Quality of Service	From GCC Provider as previously saved
Result	As specified in Rec. T.122
User Data	T.124 Object Identifier
	ConferenceJoinResponse PDU (see Table 8-13)

The ConferenceJoinResponse PDU is shown in Table 8-13. The Node ID parameter, which is the User ID assigned by MCS in response to the MCS-Attach-User request issued by the GCC Provider, shall be supplied by the GCC Provider at the node directly connected to the joining node. The Tag parameter is filled in by the GCC Provider at the node directly connected to the joining node with a locally unique value. It is used to identify the returned UserIDIndication PDU. The value of this parameter may be set to the value of the same parameter received from the Top GCC Provider since this parameter was allocated by this node originally as sent in the ConferenceJoinRequest PDU. If this is done, it requires that Tags used for either purpose are locally unique. The other parameters in this PDU are filled out by the Top GCC Provider. This includes the Top Node ID, which is the Node ID of the Top GCC Provider, as well as the parameters associated with the Conference Profile. Also filled in by the Top GCC Provider is the Conference Name Alias. This is conditionally included depending on whether the Conference Name included both numeric and text forms. If so, the Conference Name Alias is whichever form of the Conference Name was not included in the ConferenceJoinRequest PDU. The Result parameter includes GCC-specific failure information if the Result parameter in the MCS-Connect-Provider message is set to user-rejected. This information is from the Result parameter in the GCC-Conference-Create response PDU.

Table 8-13/T.124 – ConferenceJoinResponse GCCPDU

Content	Source	Sink
Node ID (conditional)	GCC Provider of node directly connected to joining node	Destination GCC Provider
Top Node ID	Top GCC Provider	Destination GCC Provider
Tag	Top GCC Provider	Destination GCC Provider
Conference Name Alias (conditional)	Top GCC Provider	Confirm and Destination GCC Provider
Password In The Clear Required Flag	Top GCC Provider	Confirm and Destination GCC Provider
Locked Conference Flag	Top GCC Provider	Confirm and Destination GCC Provider
Listed Conference Flag	Top GCC Provider	Confirm and Destination GCC Provider
Conductible Conference Flag	Top GCC Provider	Confirm and Destination GCC Provider
Termination Method	Top GCC Provider	Confirm and Destination GCC Provider
Conductor Privilege List (optional)	Top GCC Provider	Confirm and Destination GCC Provider
Conducted-Mode Conference Privilege List (optional)	Top GCC Provider	Confirm and Destination GCC Provider
Non-Conducted-Mode Conference Privilege List (optional)	Top GCC Provider	Confirm and Destination GCC Provider
Conference Description (optional)	Top GCC Provider	Confirm and Destination GCC Provider
Password (optional)	Request	Indication
Node Category (optional)	Request	Indication
Result	Response	Confirm
User Data (optional)	Response	Confirm

On receipt of the ConferenceJoinResponse PDU, if the PDU indicated a successful result, a GCC Provider shall first issue an MCS-Attach-User request. On receipt of the MCS-Attach-User confirm which contains the allocated Node ID, the GCC Provider shall then join the corresponding Node ID Channel by issuing an MCS-Channel-Join request. The GCC Provider shall also join the appropriate GCC Broadcast Channel(s) by issuing one or two MCS-Channel-Join requests. All nodes are required to join the GCC-Conventional-Broadcast-Channel. All GCC control PDUs and Roster delta updates that are generated when Conventional Nodes join, leave, or change a roster record are broadcasted on this channel. Anonymous nodes may ignore the roster deltas received on this channel, but they must process the control messages received. Conventional Nodes must also join the GCC-Counted-Broadcast-Channel. On this channel, the Top Provider broadcasts Roster delta updates which are generated when Counted Nodes join, leave, or change a roster record. Because only Conventional Nodes join the GCC-Counted-Broadcast-Channel, the use of this channel reduces the amount of network traffic incurred when a large number of Counted Nodes are participating in a conference. Note that older protocol nodes join the GCC-Broadcast-Channel at this point.

If the original GCC-Conference-Join request had specified the Convener Password, indicating that the node was to regain its role as the Conference Convener, the GCC Provider may also join the GCC-Convener-Channel (if it supports any of the functions which require use of this channel) by issuing an MCS-Channel-Join request. Once the GCC Provider has received an MCS-Channel-Join confirm from each of the channel join requests (successive requests may be issued prior to receiving the previous confirm), it shall send a UserIDIndication PDU to the GCC Provider of the directly connected node by issuing an MCS-Send-Data request specifying the Node ID of the directly connected node as contained in the received ConferenceJoinResponse PDU, specifying Top data priority, and including the PDU in the Data field. The content of the UserIDIndication PDU is shown in Table 8-5. The Tag parameter is filled in with the value of the corresponding parameter received in the ConferenceJoinResponse PDU.

The GCC Provider shall then generate a GCC-Conference-Join confirm primitive and issue it to the Control GCCSAP. This primitive shall include the Result parameters from the received PDU as well as the locally-allocated Conference ID. If the received PDU had indicated an unsuccessful result, or if the GCC Provider receives an MCS-Disconnect-Provider indication for this connection prior to having issued a successful GCC-Conference-Join confirm, the GCC-Conference-Join confirm primitive is issued immediately, indicating an unsuccessful result, without issuing the attach-user or channel-join requests, or sending the UserIDIndication PDU. The Result parameter in the PDU as well as the Result parameter reported in the MCS-Connect-Provider confirm (or the Reason parameter of the MCS-Disconnect-Provider) is used to generate the result reported in the GCC-Conference-Join confirm primitive. If the Result parameter of the MCS-Connect-Provider confirm is user-rejected, the Result parameter in the PDU is used to determine the reported result. Otherwise, the Result parameter in the MCS-Connect-Provider is used directly.

The Conference Name Alias received in the ConferenceJoinResponse PDU, if any, is appended with the Conference Name included in the ConferenceJoinRequest PDU to form the full Conference Name returned in the GCC-Conference-Join confirm. This full conference name (which includes the numerical form as well as text form of the Conference Name, if any) shall be maintained at the local GCC Provider as the identifiers of this conference to be used in the procedures for response to a conference join request, conference query request, and in initiation of a conference invite.

On receipt of a UserIDIndication, the node directly connected to the joining node shall compare the Tag parameter to its list of nodes in the conference from which it is expecting this PDU. If the Tag matches one of these, it shall save the User ID of the source node (extracted from the MCS-Send-Data indication) in its database of Node IDs of nodes which are directly below it in the connection hierarchy. If the Tag does not match any outstanding Tags, the PDU shall be ignored.

If the Top GCC Provider receives a UserIDIndication as a result of a successful join operation in which the Convener Password parameter had been included in the ConferenceJoinRequest PDU, the Top GCC Provider shall retain the Node ID of the joining node as indicated in the UserIDIndication as the valid conference convener. It shall use this to verify later requests which only the convener may perform.

If the GCC Provider at the node directly connected to the joining node receives an MCS-Disconnect-Provider indication for the connection being established at any time during the joining process, the GCC Provider shall stop further processing for this procedure and take no further action.

The sequence of events for successfully joining a conference from a node directly connected to the Top GCC Provider is shown in Figure 8-2. The case that the joining node is not directly connected to the Top GCC Provider is shown in Figure 8-3.

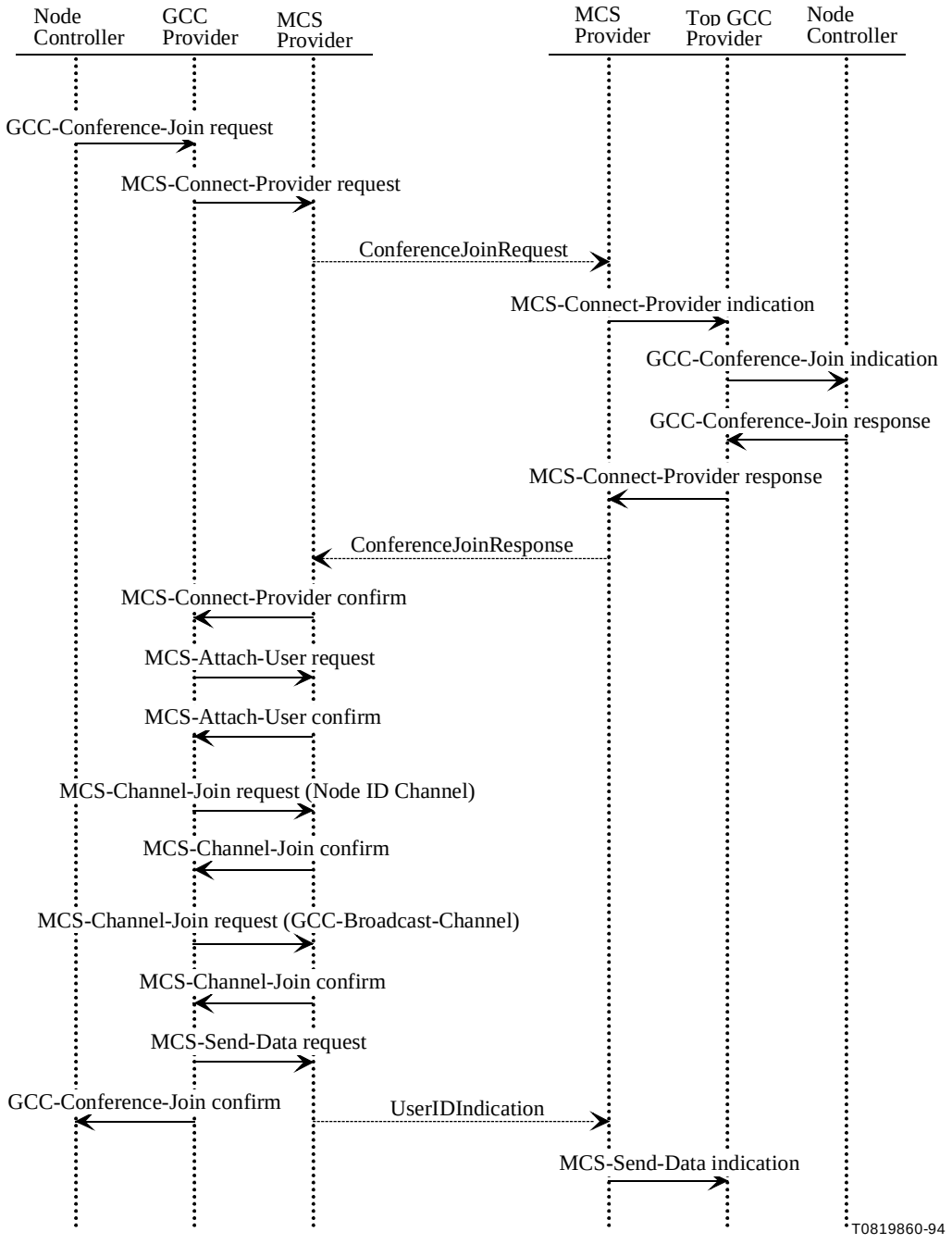

Figure 8-2/T.124 – Joining a conference when directly connected to Top GCC Provider

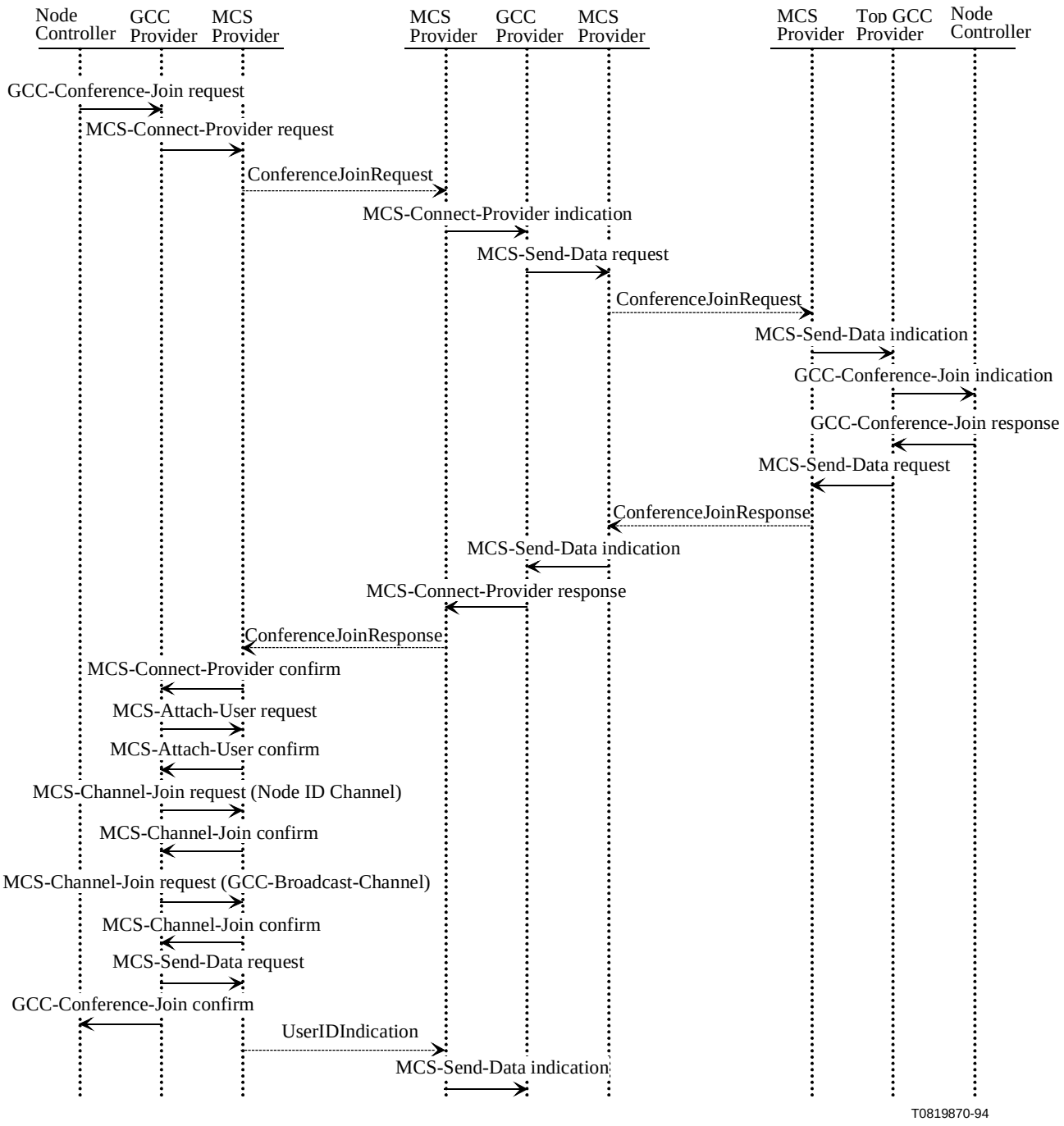

Figure 8-3/T.124 – Joining a conference when not directly connected to Top GCC Provider

8.2.4 Inviting a node to a conference

On receipt of a GCC-Conference-Invite request primitive, a GCC Provider shall issue an MCS-Connect-Provider request primitive with the parameters shown in Table 8-14.

Table 8-14/T.124 – MCS-Connect-Provider request parameters for ConferenceInviteRequest PDU

Parameter	Contents
Calling Address	From request primitive
Calling Domain Selector	Conference ID from request primitive
Called Address	From request primitive
Called Domain Selector	NULL
Upward/Downward Flag	Down
Domain Parameters	From GCC Provider as previously saved
Quality of Service	From GCC Provider as previously saved
User Data	T.124 Object Identifier
	ConferenceInviteRequest PDU (see Table 8-15)

The User Data parameter of the MCS-Connect-Provider request contains a structure which includes an Object Identifier identifying the contained PDU as adhering to this Recommendation followed by the PDU itself. The details of this structure are defined in 9.6. The contents of this PDU are shown in Table 8-15. The Conference Name is the name of the conference specified by the Conference ID in the request primitive as stored in the local Conference Profile. The Node ID parameter, which is the User ID assigned by MCS in response to the MCS-Attach-User request issued by the GCC Provider, shall be supplied by the source GCC Provider. The Top Node ID is the Node ID of the Top GCC Provider, previously saved by the inviting GCC Provider at the inviting node. The Tag parameter is assigned by the source GCC Provider to be locally unique. It is used to identify the returned UserIDIndication PDU.

Table 8-15/T.124 – ConferenceInviteRequest GCCPDU

Content	Source	Sink
Conference Name	Source GCC Provider	Destination GCC Provider and Indication
Node ID	Source GCC Provider	Destination GCC Provider
Top Node ID	Source GCC Provider	Destination GCC Provider
Tag	Source GCC Provider	Destination GCC Provider
Password In The Clear Required Flag	Source GCC Provider	Indication and Destination GCC Provider
Locked Conference Flag	Source GCC Provider	Indication and Destination GCC Provider
Listed Conference Flag	Source GCC Provider	Indication and Destination GCC Provider
Conductible Conference Flag	Source GCC Provider	Indication and Destination GCC Provider
Termination Method	Source GCC Provider	Indication and Destination GCC Provider
Conductor Privilege List (optional)	Source GCC Provider	Indication and Destination GCC Provider
Conducted-Mode Conference Privilege List (optional)	Source GCC Provider	Indication and Destination GCC Provider

Table 8-15/T.124 – ConferenceInviteRequest GCCPDU (*concluded*)

Content	Source	Sink
Non-Conducted-Mode Conference Privilege List (optional)	Source GCC Provider	Indication and Destination GCC Provider
Conference Description (optional)	Source GCC Provider	Indication and Destination GCC Provider
Caller Identifier (optional)	Request	Indication
Node Category	Request	Indication
User Data (optional)	Request	Indication

On receipt of the MCS-Connect-Provider indication that includes a T.124 Object Identifier and the ConferenceInviteRequest PDU, a GCC Provider shall generate a GCC-Conference-Invite indication primitive and issue it to the Control GCCSAP. The Conference ID in this primitive shall be assigned locally by the GCC Provider and shall be used as the local MCS Domain Selector. If GCC does not have the resources necessary to join a conference, it may generate the negative response automatically without generating the GCC-Conference-Invite indication. Otherwise, on receipt of the GCC-Conference-Invite response, the GCC Provider shall generate an MCS-Connect-Provider response which includes a result which is either success, or user-rejected depending on whether or not the Result parameter in the GCC-Conference-Invite response primitive indicated success or failure (and shall include the result in the ConferenceInviteResponse PDU). The User Data parameter includes the T.124 Object Identifier as well as the ConferenceInviteResponse PDU. The contents of the connect provider primitive are shown in Table 8-16.

If the successful response includes the Conference Name Modifier parameter, the GCC Provider shall maintain this as well as the Conference Name parameter from the received ConferenceInviteRequest PDU (which includes both the numerical form as well as the text form of the Conference Name, if any) together as the local identifier of the conference to be used in the procedure for responding to a conference join request, a conference query request, or in initiating a conference invite request. If no Conference Name Modifier is present, only the Conference Name is maintained for this purpose.

After sending the MCS-Connect-Provider response, if the invitation was acknowledged as successful, the GCC Provider shall issue an MCS-Attach-User request. On receipt of the MCS-Attach-User confirm which contains the allocated Node ID, the GCC Provider shall then join the corresponding Node ID Channel by issuing an MCS-Channel-Join request. The GCC Provider shall also join the appropriate GCC Broadcast Channel(s) by issuing one or two MCS-Channel-Join requests. All nodes are required to join the GCC-Conventional-Broadcast-Channel. All GCC control PDUs and Roster delta updates that are generated when Conventional Nodes join, leave, or change a roster record are broadcasted on this channel. Anonymous nodes may ignore the roster deltas received on this channel, but they must process the control messages received. Conventional Nodes must also join the GCC-Counted-Broadcast-Channel. On this channel, the Top Provider broadcasts Roster delta updates which are generated when Counted Nodes join, leave, or change a roster record. Because only Conventional Nodes join the GCC-Counted-Broadcast-Channel, the use of this channel reduces the amount of network traffic incurred when a large number of Counted Nodes are participating in a conference. Note that older protocol nodes join the GCC-Broadcast-Channel at this point.

Once the GCC Provider has received an MCS-Channel-Join confirm from each of the channel join requests (successive requests may be issued prior to receiving the previous confirm), it shall send a UserIDIndication PDU to the GCC Provider of the directly connected node by issuing an MCS-Send-Data request specifying the Node ID of the directly connected node as contained in the received ConferenceInviteRequest PDU, specifying Top data priority, and including the PDU in the Data field. The content of the UserIDIndication PDU is shown in Table 8-5. The Tag parameter is filled in with the value of the corresponding parameter received in the ConferenceInviteRequest PDU.

If the GCC Provider at the invited node receives an MCS-Disconnect-Provider indication for the connection being established at any time during the process of being joined to the conference (this includes the time that the GCC-Conference-Invite indication has been issued to the Control GCCSAP until the time that the UserIDIndication has been transmitted), it shall issue a GCC-Conference-Terminate indication to the Control GCCSAP indicating the requested normal termination as the reason if user-initiated was the reason provided in the MCS-Disconnect-Provider indication, and error termination otherwise. It shall then stop further processing for this connection establishment procedure. If the MCS-Disconnect-Provider indication was received prior to issuing the GCC-Conference-Invite indication to the Control GCCSAP, the GCC Provider shall stop further processing for this connection establishment procedure and take no further action.

Table 8-16/T.124 – MCS-Connect-Provider response parameters for ConferenceInviteResponse PDU

Parameter	Contents
Domain Parameters	From response primitive
Quality of Service	From response primitive
Result	As specified in Rec. T.122
User Data	T.124 Object Identifier
	ConferenceInviteResponse PDU (see Table 8-17)

The ConferenceInviteResponse PDU is shown in Table 8-17.

Table 8-17/T.124 – ConferenceInviteResponse GCCPDU

Content	Source	Sink
Result	Response	Confirm
User Data (optional)	Response	Confirm

On receipt of the ConferenceInviteResponse PDU, if the result parameter of the MCS-Connect-Provider was successful, a GCC Provider shall record the User Data parameter to fill in to the GCC-Conference-Invite confirm primitive once it is generated on receipt of the pending UserIDIndication. If the result parameter was unsuccessful, or if the GCC Provider receives an MCS-Disconnect-Provider indication for this connection prior to having issued a successful GCC-Conference-Invite confirm, the GCC Provider shall immediately generate a GCC-Conference-Invite confirm primitive and issue it to the Control GCCSAP. The Result parameter in the PDU as well as the Result parameter reported in the MCS-Connect-Provider confirm (or the Reason parameter of the MCS-Disconnect-Provider) is used to generate the result reported in the GCC-Conference-Invite confirm primitive. If the Result parameter of the MCS-Connect-Provider

confirm is user-rejected, the Result parameter in the PDU is used to determine the reported result. Otherwise, the Result parameter in the MCS-Connect-Provider is used directly.

On receipt of a UserIDIndication, the node directly connected to the invited node shall compare the Tag parameter to its list of nodes in the conference from which it is expecting this PDU. If the Tag matches one of these, it shall save the User ID of the source node (extracted from the MCS-Send-Data indication) in its database of Node IDs of nodes which are directly below it in the connection hierarchy. If the Tag does not match any outstanding Tags, the PDU shall be ignored.

If the Tag was from an outstanding conference invite, on receipt of the UserIDIndication, the GCC Provider shall generate a GCC-Conference-Invite confirm primitive and issue it to the Control GCCSAP with a successful result parameter.

The sequence of events for successfully inviting a node to a conference is shown in Figure 8-4.

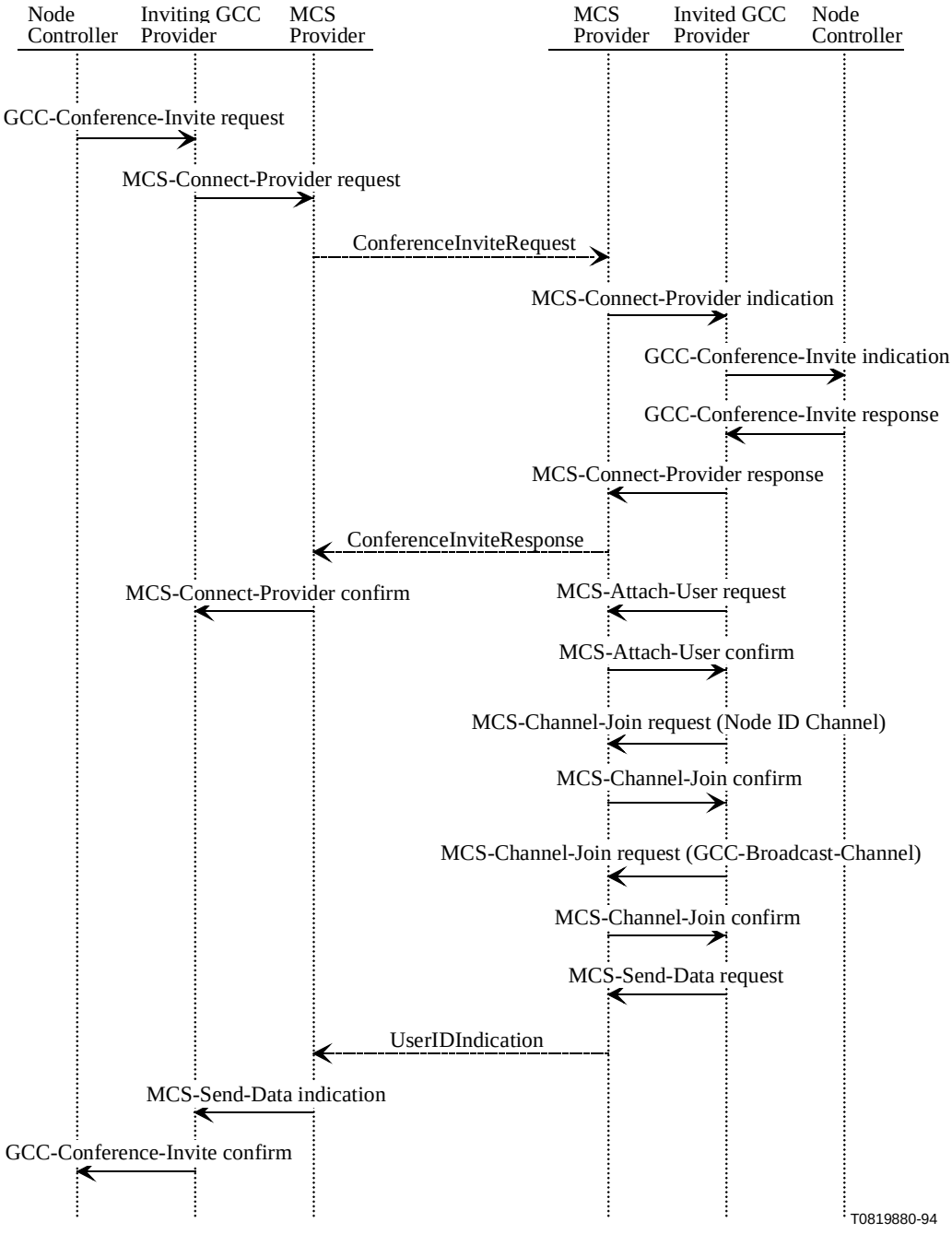

Figure 8-4/T.124 – Inviting a node to a conference

8.2.5 Requesting to add a node to a conference

On receipt of a GCC-Conference-Add request primitive, a GCC Provider shall send a ConferenceAddRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The contents of the ConferenceAddRequest PDU are shown in Table 8-18. The contents are filled in from the parameters passed in the GCC-Conference-Add request primitive. The Tag parameter is assigned by the requesting GCC Provider to be locally unique. This number is used to identify the corresponding response PDU when it is returned.

Table 8-18/T.124 – ConferenceAddRequest GCCPDU

Content	Source	Sink
Network Address	Request	Indication
Requesting Node	Requesting GCC Provider	Indication
Tag	Requesting GCC Provider	Destination GCC Provider
Adding MCU (optional)	Request	Top GCC Provider
Node Category (optional)	Request	Indication
User Data (optional)	Request	Indication
Network Address V2 (see Annex B)	Request	Indication

On receipt of a ConferenceAddRequest PDU, the Top GCC Provider shall first check whether the requesting node had the necessary privilege to add a node to the conference as indicated in the Conference Profile. If not, it shall reject the request by sending a ConferenceAddResponse PDU to the requesting node indicating invalid-requester as the result. The content of the ConferenceAddResponse PDU is shown in Table 8-19. If the requesting node had sufficient privilege, the Top GCC Provider shall then examine whether the Adding MCU parameter is present. If not present, or if the Adding MCU identifier is equal to the Node ID of the Top GCC Provider, the Top GCC Provider, if it supports the GCC-Conference-Add function, shall generate a GCC-Conference-Add indication primitive and issue it to the Control GCCSAP. The contents of the primitive are filled in from the contents of the received PDU. If the node does not support this function, it may reject the request by issuing a FunctionNotSupportedResponse PDU including the received ConferenceAddRequest in the response.

If the optional Adding MCU parameter is present and set to a value other than that of the Node ID of the Top GCC Provider, the Top GCC Provider shall send a ConferenceAddRequest PDU to the Adding MCU by issuing an MCS-Send-Data request specifying the Node ID of the Adding MCU as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the PDU is the same as that of the received request PDU. On receipt of a ConferenceAddRequest PDU, a node which is not the Top GCC Provider, if it supports the GCC-Conference-Add function, shall first check that it has been received from the Top GCC Provider by examining the User ID from the received MCS-Send-Data indication. If it has been received from the Top GCC Provider, it shall generate a GCC-Conference-Add indication primitive and issue it to the Control GCCSAP. The contents of the primitive are filled in from the contents of the received PDU. If the User ID of the received PDU does not match the Node ID of the Top GCC Provider, the received PDU is ignored and no further action is taken. If the node does not support this function, it may reject the request by sending to the Top GCC Provider a FunctionNotSupportedResponse PDU including the received ConferenceAddRequest in the response.

On receipt of a GCC-Conference-Add response primitive, a GCC Provider shall send a ConferenceAddResponse PDU to the requesting node by issuing an MCS-Send-Data request specifying the Node ID of the requesting node, specifying High data priority, and including the PDU in the Data field. In the case that the request had been routed through the Top GCC Provider to another node, the response is issued to the original requesting node as indicated by the Requesting Node parameter of the request PDU. The contents of the ConferenceAddResponse PDU are shown in Table 8-19. The contents are filled in from the parameters passed in the GCC-Conference-Add response primitive. The Tag shall have the same value as the corresponding parameter in the received ConferenceAddRequest PDU.

Table 8-19/T.124 – ConferenceAddResponse GCCPDU

Content	Source	Sink
Tag	Source GCC Provider	Destination GCC Provider
Result	Response	Confirm
User Data (optional)	Response	Confirm

On receipt of a ConferenceAddResponse PDU, a GCC Provider shall generate a GCC-Conference-Add confirm primitive and issue it to the Control GCCSAP. The contents of the primitive are obtained from the parameters of the received PDU.

8.2.6 Locking a conference

On receipt of a GCC-Conference-Lock request primitive, a GCC Provider shall send a ConferenceLockRequest by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceLockRequest PDU is shown in Table 8-20. There are no parameters in this PDU.

Table 8-20/T.124 – ConferenceLockRequest GCCPDU

Content	Source	Sink
-- No parameters --		

On receipt of a ConferenceLockRequest PDU, if the Top GCC Provider supports the conference lock capability, it shall first determine if the requesting node has the privilege necessary to lock the conference based on the lock-unlock-privileges defined when the conference was created. If so, the Top GCC Provider shall generate a GCC-Conference-Lock indication primitive and issue it to the Control GCCSAP. The Source Node specified in the primitive shall be obtained from the Sender User ID in the MCS-Send-Data indication. On receipt of a GCC-Conference-Lock response, the GCC Provider shall send a ConferenceLockResponse PDU by issuing an MCS-Send-Data request specifying the Source Node indicated in the response as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceLockResponse PDU is shown in Table 8-21. The Result parameter is generated from the result returned in the GCC-Conference-Lock response.

If the requesting node did not have the proper privilege to support this operation, the request is immediately rejected without generating a GCC-Conference-Lock indication. This is done by generating a ConferenceLockResponse PDU specifying invalid-requester as the Result.

The Top GCC Provider shall preserve the order of received ConferenceLockRequest PDUs and ConferenceUnlockRequest PDUs and their corresponding indication primitives issued to the Control GCCSAP, as well as between the response primitives from the Control GCCSAP and the ConferenceLockResponse and ConferenceUnlockResponse PDUs transmitted.

At the requesting node, order shall also be preserved between the request primitives and the corresponding transmitted ConferenceLockRequest PDUs and ConferenceUnlockRequest PDUs as well as between the received ConferenceLockResponse and ConferenceUnlockResponse PDUs and the corresponding confirm primitives.

If the Top GCC Provider does not support the conference lock capability, on receipt of a ConferenceLockRequest PDU, it shall immediately generate a FunctionNotSupportedResponse PDU including the received ConferenceLockRequest PDU in the response.

Table 8-21/T.124 – ConferenceLockResponse GCCPDU

Content	Source	Sink
Result	Response	Confirm

On receipt of a ConferenceLockResponse PDU, a GCC Provider shall generate a GCC-Conference-Lock confirm primitive and issue it to the Control GCCSAP. The result parameter in the confirm primitive is obtained from the Result field in the PDU.

If the Top GCC Provider sends a ConferenceLockResponse which indicates a successful result, it shall also generate a ConferenceLockIndication PDU and send it to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceLockIndication PDU is shown in Table 8-22. This PDU contains no parameters.

Table 8-22/T.124 – ConferenceLockIndication GCCPDU

Content	Source	Sink
-- No parameters --		

On receipt of a ConferenceLockIndication PDU, a GCC Provider which supports the optional lock indication may generate a GCC-Conference-Lock-Report indication and issue it to the Control GCCSAP. Before it does so, it shall examine the User ID as indicated in the received MCS-Uniform-Send-Data indication and compare it to the Node ID of the Top GCC Provider. The GCC-Conference-Lock-Report primitive may only be generated if the received User ID matches the Node ID of the Top GCC Provider. Otherwise, the received PDU is ignored, and no further action is taken.

If the Top GCC Provider becomes aware of a new node entering a conference by its presence in the Conference Roster and the conference is currently locked and if the possibility exists that the conference may have been unlocked when the node joined (or was invited to) the conference, the Top GCC Provider shall generate a ConferenceLockIndication PDU and send it to the new node by issuing MCS-Send-Data request specifying the Node ID of that node as the Channel ID, specifying High data priority, and including the PDU in the Data field. Alternatively, it may send this PDU to all nodes by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. Although the flag indicating whether the conference is locked or unlocked is included in the information provided to the newly-joined node as part of the connection establishment process, this ensures that if the conference changed its lock state since the connection had been established, that the node is properly

notified of the change. If there is no possibility that the lock state has changed since the node was joined (or was invited) into the conference (e.g. if the lock state has not changed since the creation of the conference), the Top GCC Provider need not send this PDU.

8.2.7 Unlocking a conference

On receipt of a GCC-Conference-Unlock request primitive, a GCC Provider shall send a ConferenceUnlockRequest by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceUnlockRequest PDU is shown in Table 8-23. There are no parameters in this PDU.

Table 8-23/T.124 – ConferenceUnlockRequest GCCPDU

Content	Source	Sink
-- No parameters --		

On receipt of a ConferenceUnlockRequest PDU, if the Top GCC Provider supports the conference unlock capability, it shall first determine if the requesting node has the privilege necessary to unlock the conference based on the lock-unlock-privileges defined when the conference was created. If so, the Top GCC Provider shall generate a GCC-Conference-Unlock indication primitive and issue it to the Control GCCSAP. The Source Node specified in the primitive shall be obtained from the Sender User ID in the MCS-Send-Data indication. On receipt of a GCC-Conference-Unlock response, the GCC Provider shall send a ConferenceUnlockResponse PDU by issuing an MCS-Send-Data request specifying the Source Node indicated in the response as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceUnlockResponse PDU is shown in Table 8-24. The Result parameter is generated from the result returned in the GCC-Conference-Unlock response.

If the requesting node did not have the proper privilege to support this operation, the request is immediately rejected without generating a GCC-Conference-Unlock indication. This is done by generating a ConferenceUnlockResponse PDU specifying invalid-requester as the Result.

The Top GCC Provider shall preserve the order of received ConferenceLockRequest PDUs and ConferenceUnlockRequest PDUs and their corresponding indication primitives issued to the Control GCCSAP, as well as between the response primitives from the Control GCCSAP and the ConferenceLockResponse and ConferenceUnlockResponse PDUs transmitted.

At the requesting node, order shall also be preserved between the request primitives and the corresponding transmitted ConferenceLockRequest PDUs and ConferenceUnlockRequest PDUs as well as between the received ConferenceLockResponse and ConferenceUnlockResponse PDUs and the corresponding confirm primitives.

If the Top GCC Provider does not support the conference unlock capability, on receipt of a ConferenceUnlockRequest PDU, it shall immediately generate a FunctionNotSupportedResponse PDU including the received ConferenceUnlockRequest PDU in the response.

Table 8-24/T.124 – ConferenceUnlockResponse GCCPDU

Content	Source	Sink
Result	Response	Confirm

On receipt of a ConferenceUnlockResponse PDU, a GCC Provider shall generate a GCC-Conference-Unlock confirm primitive and issue it to the Control GCCSAP. The result parameter in the confirm primitive is obtained from the Result field in the PDU.

If the Top GCC Provider sends a ConferenceUnlockResponse which indicates a successful result, it shall also generate a ConferenceUnlockIndication PDU and send it to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceUnlockIndication PDU is shown in Table 8-25. This PDU contains no parameters.

Table 8-25/T.124 – ConferenceUnlockIndication GCCPDU

Content	Source	Sink
-- No parameters --		

On receipt of a ConferenceUnlockIndication PDU, a GCC Provider which supports the optional unlock indication may generate a GCC-Conference-Lock-Report indication and issue it to the Control GCCSAP. Before it does so, it shall examine the User ID as indicated in the received MCS-Uniform-Send-Data indication and compare it to the Node ID of the Top GCC Provider. The GCC-Conference-Lock-Report primitive may only be generated if the received User ID matches the Node ID of the Top GCC Provider. Otherwise, the received PDU is ignored, and no further action is taken.

If the Top GCC Provider becomes aware of a new node entering a conference by its presence in the Conference Roster and the conference is currently unlocked and if the possibility exists that the conference may have been locked when the node joined (or was invited to) the conference, the Top GCC Provider shall generate a ConferenceUnlockIndication PDU and send it to the new node by issuing MCS-Send-Data request specifying the Node ID of that node as the Channel ID, specifying High data priority, and including the PDU in the Data field. Alternatively, it may send this PDU to all nodes by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. Although the flag indicating whether the conference is locked or unlocked is included in the information provided to the newly-joined node as part of the connection establishment process, this ensures that if the conference changed its lock state since the connection had been established, that the node is properly notified of the change. If there is no possibility that the lock state has changed since the node was joined (or was invited) into the conference (e.g. if the lock state has not changed since the creation of the conference), the Top GCC Provider need not send this PDU.

8.2.8 Disconnecting from a conference

On receipt of a GCC-Conference-Disconnect request, a GCC Provider shall first attempt to eject the nodes directly below it in the connection hierarchy, if any. It shall do this by sending, for each such node, a ConferenceEjectUserIndication PDU with the Node To Eject parameter set to the Node ID of the particular subordinate node and specifying higher-node-disconnected as the reason. This is done by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the data field. The content of the ConferenceEjectUserIndication PDU is shown in Table 8-31. The GCC Provider shall then wait until

it has received MCS-Disconnect-Provider indications from each subordinate connection. Following this, it shall disconnect from the conference by issuing first an MCS-Detach-User request followed by an MCS-Disconnect-Provider request directed at the upward connection (the only remaining connection). If, for some reason, the GCC Provider has not received MCS-Disconnect-Provider indications from each of the lower nodes within a reasonable period of time (determined locally), the GCC Provider may proceed to disconnect those connections itself by issuing MCS-Disconnect-Provider requests directed at each remaining lower connection, followed by an MCS-Detach-User request and an MCS-Disconnect-Provider request directed at the upward connection as in the normal case. In either case, or if there had been no subordinate nodes, the GCC Provider shall then generate a GCC-Conference-Disconnect confirm primitive and issue it to the Control GCCSAP. The GCC Provider shall then remove all database information associated with this conference.

On receipt of an MCS-Detach-User indication, each GCC Provider in the conference shall examine the User ID indicated in the indication and compare it to its list of Node IDs in its local copy of the Conference Roster. If the User ID corresponds to a Node ID, different from its own Node ID, the GCC Provider shall generate a GCC-Conference-Disconnect indication and issue it to the Control GCCSAP. The Disconnecting Node parameter in the indication shall correspond to the User ID in the received indication. If the reason code in the received indication is user-initiated, the reason code in the GCC-Conference-Disconnect indication shall be either user-initiated or ejected node, depending on whether or not a ConferenceEjectUserIndication PDU was received earlier containing in its Node To Eject field the same User ID as the MCS-Detach-User indication. Otherwise, the reason is indicated as unknown.

8.2.9 Terminating a conference

On receipt of a GCC-Conference-Terminate request, a GCC Provider shall send a ConferenceTerminateRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTerminateRequest PDU is shown in Table 8-26. The reason code is obtained from the corresponding parameter in the request primitive.

Table 8-26/T.124 – ConferenceTerminateRequest GCCPDU

Content	Source	Sink
Reason	Request	Top GCC Provider

On receipt of a ConferenceTerminateRequest PDU, the Top GCC Provider shall first determine if the requesting node has the privilege necessary to terminate the conference based on the terminate-privileges defined when the conference was created. If not, the request is rejected and a ConferenceTerminateResponse is sent back to the requester by issuing an MCS-Send-Data request specifying the Node ID Channel of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTerminateResponse PDU is shown in Table 8-27. In this case, the result parameter is set to indicate invalid-requester as the reason for rejection.

If the requester did have the proper privilege to terminate the conference, then a ConferenceTerminateResponse shall be sent back to the requester indicating a successful result. In addition, a ConferenceTerminateIndication is sent to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the

ConferenceTerminateIndication PDU is shown in Table 8-28. The reason code is obtained from the reason code in the request PDU.

Table 8-27/T.124 – ConferenceTerminateResponse GCCPDU

Content	Source	Sink
Result	Top GCC Provider	Confirm

Table 8-28/T.124 – ConferenceTerminateIndication GCCPDU

Content	Source	Sink
Result	Top GCC Provider	Confirm

On receipt of a ConferenceTerminateResponse PDU, a GCC Provider shall generate a GCC-Conference-Terminate confirm primitive and issue it to the Control GCCSAP. The result indicated in the primitive is obtained directly from the Result parameter in the PDU.

On receipt of a ConferenceTerminateIndication PDU sent by the Top GCC Provider, a GCC Provider shall first wait until it has received MCS-Disconnect-Provider indications from each connection directly below it in the connection hierarchy. Following this, it shall disconnect from the conference by issuing an MCS-Disconnect-Provider request directed at the upward connection (the only remaining connection). If, for some reason, the GCC Provider has not received MCS-Disconnect-Provider indications from each of the lower nodes within a reasonable period of time (determined locally), the GCC Provider may proceed to disconnect those connections itself by issuing MCS-Disconnect-Provider requests directed at each remaining lower connection, followed by an MCS-Disconnect-Provider request directed at the upward connection as in the normal case. In either case, the GCC Provider shall then generate a GCC-Conference-Terminate indication primitive and issue it to the Control GCCSAP. The reason indicated in the primitive is obtained directly from the Reason parameter in the PDU.

If a GCC Provider receives an MCS-Disconnect-Provider indication from the local MCS provider corresponding to its upward MCS connection, this is an indication that the MCS connection has been terminated due to an abnormal condition within MCS, the GCC Provider shall generate a GCC-Conference-Terminate indication primitive and issue it to the Control GCCSAP. The reason code shall indicate that this is an error termination.

If the Top GCC Provider receives a disconnect indication (either via PDU or MCS-Detach-User indication) which results in no nodes listed in the Conference Roster with the exception of the local node, the provider shall check the Conference Profile to determine if the conference is manually or automatically terminating. If it was manually terminating, no further action is taken. If it was automatically terminating, the GCC Provider shall indicate to the local node controller that the conference has been terminated by generating a GCC-Conference-Terminate indication and issuing it to the Control GCCSAP. The reason code shall indicate that there are no more nodes joined to an automatically terminating conference. The GCC Provider shall then remove all database information associated with this conference.

8.2.10 Ejecting a node from a conference

On receipt of a GCC-Conference-Eject-User request primitive, a GCC Provider shall first compare the Node To Eject parameter to the Node IDs of the nodes immediately below it in the connection hierarchy, if any. If the Node To Eject is a node other than a directly subordinate node, the GCC Provider shall send a ConferenceEjectUserRequest PDU to the Top GCC Provider by issuing an

MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The content of the ConferenceEjectUserRequest PDU is shown in Table 8-29. The contents of the PDU are obtained from the request primitive.

If the Node To Eject is a node directly below the local node in the connection hierarchy, the GCC Provider shall instead send a ConferenceEjectUserIndication PDU to all nodes specifying the Node ID of the node to be ejected in the PDU and the reason as indicated in the request primitive. It shall do this by issuing an MCS-Uniform-Data request specifying both the GCC-Broadcast-Channel (for older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the data field. The content of the ConferenceEjectUserIndication is shown in Table 8-31. The GCC Provider may then wait until it has received an MCS-Disconnect-Provider indication from the connection corresponding to the ejected node. If, for some reason, the GCC Provider has not received MCS-Disconnect-Provider indications from the ejected node within a reasonable period of time (determined locally), the GCC Provider may proceed to disconnect those connections itself by issuing an MCS-Disconnect-Provider request directed at the ejected node. The GCC Provider shall then generate a GCC-Conference-Eject-User confirm PDU indicating a successful result.

Table 8-29/T.124 – ConferenceEjectUserRequest GCCPDU

Content	Source	Sink
Node To Eject	Request	Top GCC Provider
Reason	Request	Top GCC Provider

On receipt of a ConferenceEjectUserRequest PDU, the Top GCC Provider shall first determine if the requesting node has the privilege necessary to eject a user based on the eject-user-privileges defined when the conference was created. If not, the request is rejected and a ConferenceEjectUserResponse is sent back to the requester by issuing an MCS-Send-Data request specifying the Node ID Channel of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceEjectUserResponse PDU is shown in Table 8-30. In this case, the result parameter is set to indicate invalid-requester as the reason for rejection.

If the requester did have the proper privilege to eject a user, then a ConferenceEjectUserIndication primitive is broadcast to all nodes by issuing an MCS-Uniform-Send-Data request. This request specifies both the GCC-Broadcast-Channel (for older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The content of the ConferenceEjectUserIndication PDU is shown in Table 8-31. The Reason parameter in this PDU is obtained from the request PDU. If the node to be ejected is valid, a response is sent back to the requester indicating a successful result. If it is not possible to eject the requested node, a response is sent which includes a negative result.

Table 8-30/T.124 – ConferenceEjectUserResponse GCCPDU

Content	Source	Sink
Node To Eject	Top GCC Provider	Confirm
Result	Top GCC Provider	Confirm

On receipt of a ConferenceEjectUserResponse PDU, a GCC Provider shall generate a GCC-Conference-Eject-User confirm primitive and issue it to the Control GCCSAP. The contents of the confirm primitive are obtained from the ConferenceEjectUserResponse PDU.

Table 8-31/T.124 – ConferenceEjectUserIndication GCCPDU

Content	Source	Sink
Node To Eject	Top GCC Provider	Confirm
Reason	Top GCC Provider	Indication

On receipt of a ConferenceEjectUserIndication PDU, a GCC Provider shall compare the Node To Eject parameter to its own Node ID. If they are the same, it shall then compare the User ID as indicated in the MCS-Send-Data indication to the Node ID of the Top GCC Provider and to that of the node directly above it in the connection hierarchy. If the source Node ID is the same as either of these, it shall immediately disconnect from the conference by first attempting to eject the nodes directly below it in the connection hierarchy, if any. It shall do this by sending, for each such node, a ConferenceEjectUserIndication PDU with the Node To Eject parameter set to the Node ID of the particular subordinate node and specifying higher-node-ejected as the reason. This is done by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (for older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the data field. The GCC Provider shall then wait until it has received MCS-Disconnect-Provider indications from each subordinate connection. Following this, it shall disconnect from the conference by issuing first an MCS-Detach-User request followed by an MCS-Disconnect-Provider request directed at the upward connection (the only remaining connection). If, for some reason, the GCC Provider has not received MCS-Disconnect-Provider indications from each of the lower nodes within a reasonable period of time (determined locally), the GCC Provider may proceed to disconnect those connections itself by issuing MCS-Disconnect-Provider requests directed at each remaining lower connection, followed by first an MCS-Detach-User request followed by an MCS-Disconnect-Provider request directed at the upward connection as in the normal case. In either case, or if there had been no subordinate nodes, the GCC Provider shall then generate a GCC-Conference-Eject-User indication primitive and issue it to the Control GCCSAP. If the PDU is received with a User ID not matching the Node ID of the Top GCC Provider or the node directly above in the connection hierarchy, the PDU shall be ignored with no further action taken.

If the receiving GCC Provider is an MCU which is directly connected above the node to be ejected in the connection hierarchy, it may optionally disconnect the node to be ejected from the conference by issuing an MCS-Disconnect-Provider request for the corresponding MCS connection. Prior to taking such action, it shall verify that the User ID indicated in the received PDU is the same as the Node ID of the Top GCC Provider.

On receipt of a ConferenceEjectUserIndication PDU sent either by the Top GCC Provider or by the Node ID of the node listed in the conference roster as above the node to be ejected, all nodes except the node to be ejected shall make a note of this event and consult it later if an MCS-Detach-User indication should arrive for the node to be ejected. At that time, as specified in 8.2.8, a node shall generate a GCC-Conference-Disconnect indication and issue it to the Control GCCSAP. The reason indicated shall be ejected-node.

8.2.11 Transferring nodes between conferences

On receipt of a GCC-Conference-Transfer request primitive, a GCC Provider shall send a ConferenceTransferRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTransferRequest PDU is shown in Table 8-32. All of the parameters in this PDU are obtained directly from the request primitive.

Table 8-32/T.124 – ConferenceTransferRequest GCCPDU

Content	Source	Sink
Conference Name	Request	Top GCC Provider
Conference Name Modifier (optional)	Request	Top GCC Provider
Network Address (optional)	Request	Top GCC Provider
Transferring Nodes (optional)	Request	Top GCC Provider
Password (optional)	Request	Top GCC Provider
Network Address V2 (see Annex B)	Request	Indication

On receipt of a ConferenceTransferRequest PDU, the Top GCC Provider shall first determine if the requesting node has the privilege necessary to request a transfer based on the transfer-privileges defined when the conference was created. If not, the request is rejected and a ConferenceTransferResponse is sent back to the requester by issuing an MCS-Send-Data request specifying the Node ID Channel of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTransferResponse PDU is shown in Table 8-33. In this case, the result parameter is set to indicate invalid-requester as the reason for rejection.

If the requester did have the proper privilege to request a transfer, then a the Top GCC Provider shall send a ConferenceTransferResponse PDU back to the requester as described above, but with the result parameter indicating success. It shall then broadcast a ConferenceTransferIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (for older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTransferIndication PDU is shown in Table 8-34. The parameters in this PDU are obtained from the received ConferenceTransferRequest PDU.

Table 8-33/T.124 – ConferenceTransferResponse GCCPDU

Content	Source	Sink
Conference Name	Top GCC Provider	Confirm
Conference Name Modifier (optional)	Top GCC Provider	Confirm
Transferring Nodes (optional)	Top GCC Provider	Confirm
Result	Top GCC Provider	Confirm

On receipt of a ConferenceTransferResponse PDU, a GCC Provider shall generate a GCC-Conference-Transfer confirm primitive and issue it to the Control GCCSAP. The contents of the confirm primitive are obtained from the ConferenceTransferResponse PDU.

Table 8-34/T.124 – ConferenceTransferIndication GCCPDU

Content	Source	Sink
Conference Name	Top GCC Provider	Indication
Conference Name Modifier (optional)	Top GCC Provider	Indication
Network Address (optional)	Top GCC Provider	Indication
Transferring Nodes (optional)	Top GCC Provider	Indication
Password (optional)	Top GCC Provider	Indication
Network Address V2 (see Annex B)	Request	Indication

On receipt of a ConferenceTransferIndication PDU, a GCC Provider which supports the GCC-Conference-Transfer indication primitive shall check the list of destination nodes. If the local Node ID is found on the list of destination nodes, or if the list of destination nodes is NULL, the GCC Provider shall then check that the User ID indicated in the MCS-Uniform-Send-Data indication matches the Node ID of the Top GCC Provider. If they match, it shall generate a GCC-Conference-Transfer indication primitive and issue it to the Control GCCSAP. If the GCC Provider does not support the GCC-Conference-Transfer indication primitive, if the local node is not on the list of destination nodes, or if the received PDU is not from the Top GCC Provider, the PDU is ignored and no further action is taken.

8.3 The conference and application rosters

8.3.1 Original Roster Protocol vs. Scalable Roster Protocol

At one time, the exchange of roster information was the single largest inhibitor of scaled conferences within T.124. Because of the features added to GCC to support scalable conferences, including Node Categories, the GCC-Conventional-Broadcast-Channel, and the GCC-Counted-Broadcast-Channel, the protocol associated with the exchange of Conference and Application Rosters underwent some dramatic changes. To ensure that backward compatibility is maintained with older protocol nodes, it is necessary to describe both the Original Roster Protocol and the Scalable Roster Protocol in separate sections. It is important that all T.124 nodes support both the Original Roster Protocol and the Scalable Roster Protocol to guarantee backward compatibility. Note that the description of the Original Roster Protocol provides all the information about the roster protocol necessary to support legacy nodes. All the information describing the Scalable Roster Protocol is contained in the subclause following the description of the Original Roster Protocol (see 8.3.3).

8.3.2 Original Roster Protocol

This subclause describes in detail all aspects of the Original Roster Protocol. The original protocol is not considered to be scalable due to the use of Full Roster Refreshes, which are propagated to all nodes in a conference any time a node joins, leaves, or changes a roster record. Nodes attempting to support legacy nodes must join the original GCC-Broadcast-Channel and implement the protocol detailed below. Note that nodes that only support the Original Roster Protocol are not aware of Node Categories and, therefore, only support Conventional nodes.

8.3.2.1 Overview

Both the Conference and Application Rosters are communicated among nodes using the same set of PDUs. A single PDU, the RosterUpdateIndication PDU, is associated with all aspects of this exchange. This PDU is used to send complete or partial roster information to other nodes in the conference.

When each node (other than the top node) first announces its roster information upon joining a conference, it is done by sending a RosterUpdateIndication to the node directly above it in the hierarchy. Subsequent updates to any portion of either the Conference or Application Roster information are announced by re-issuing a RosterUpdateIndication containing the new information.

When a node updates its portion of the roster (or announces it for the first time), that information is propagated from node-to-node up the connection hierarchy until it reaches the Top GCC Provider which is responsible for forming the full Conference and Application rosters and distributing them to all nodes in the conference.

As the roster information propagates up the connection hierarchy, each intermediate node (MCU) is responsible for forming a subset of the full Conference and Application Rosters. That subset includes that node as well as all nodes below it in the connection hierarchy. That is, the Conference Roster includes the Node Record for all such nodes, the Application Roster entries for each Application Protocol Session includes the Application Records for each such node with that Peer Application Protocol Entity enrolled, and the Application Capabilities List for each Application Protocol Session includes the collapsed capabilities information for all such nodes. On receipt of a RosterUpdateIndication from a node below, an intermediate node makes the appropriate changes to its subset of the rosters, then passes this information up to the next higher node by issuing a RosterUpdateIndication.

If the update from a lower node included a change which caused the Application Capabilities Lists for at least one Application Protocol Session to have to be re-computed, the new roster information for the roster subset can only be generated if the individual Application Capabilities Lists for each node directly below in the connection hierarchy is known (to allow the collapse rules to be re-applied with the new updated information). The GCC Provider at each node shall maintain this information locally.

When the roster updates have reached the Top GCC Provider, it shall then broadcast the new roster information to all nodes in the conference. This is done by broadcasting a RosterUpdateIndication PDU to all nodes.

Roster information sent using the RosterUpdateIndication PDU may be sent in one of three ways. The roster may be sent as refresh of the full Conference and Application Rosters, replacing all existing roster entries. Alternatively, portions of the roster may be sent whereby the Conference Roster, and/or portions of the Application Roster associated one or more Session Keys may be refreshed. In this case, all entries in the transmitted portion of the roster are replaced, but portions that are not sent are left unchanged. Finally, changes to portions of the roster may be sent as updates whereby only those particular elements of the roster (e.g. individual Application Records) which have been added, modified, or removed are sent, and all other entries are left unchanged. When the Top GCC Provider broadcasts new roster information, only the first two methods are used. When the Top GCC Provider broadcasts new roster information which has changed due to at least one node having joined the conference, it shall broadcast this information only as a full refresh since the new node or nodes have no prior roster information. When propagating roster information up the connection hierarchy, any of these three methods may be used. The method chosen depends on the scope of the information changed, and is ideally chosen on the basis of minimizing the size of the PDU in order to minimize transmission time.

NOTE – The described mechanism requires that all GCC Providers in a conference store the subset of the Conference and Application Rosters for their own node as well as nodes connected directly below them in the connection hierarchy. In addition, various functions require that the GCC Providers also store the full Conference and Application Rosters as broadcast from the Top GCC Provider. If for some reason a GCC Provider fails to retain some or all of this information, the only recourse is to disconnect from the conference and, if possible, to rejoin at a later time.

8.3.2.2 Nodes entering a conference

When a node is joined to a conference, either via conference creation, joining, or invitation, a GCC Provider shall issue a GCC-Application-Permission-To-Enroll indication to the GCCSAP for all Application Protocol Entities which have locally indicated their presence to the GCC Provider. In the case that a GCC Provider becomes aware of an additional Application Protocol Entity while it is already joined to a conference, the GCC Provider shall issue a GCC-Application-Permission-To-Enroll indication to the corresponding GCCSAP indicating the existence of this conference.

Before taking any further action, the GCC Provider shall wait until it has received a GCC-Conference-Announce-Presence request from the Control GCCSAP as well as a GCC-Application-Enroll request from all Application Protocol Entities which have been sent GCC-Application-Permission-To-Enroll indications. Some of these enroll requests may indicate that that Application Protocol Entity does not intend to enroll (the enroll/un-enroll flag is set to un-enroll). For each received GCC-Application-Enroll request with the enroll flag set, the information provided in the primitive for that Application Protocol Entity is added to the Local Application Roster.

The GCC Provider shall also assign a locally-allocated Entity ID to each enrolled Application Protocol Entity. This shall be used by the GCC Provider as an identifier of each corresponding Application Protocol Entity in the RosterUpdateIndication PDU. The assigned Application Protocol Entity ID is also included in the GCC-Application-Enroll confirm primitive returned to each enrolling Application Protocol Entity along with the Node ID of the local node. The Entity ID is a 16-bit integer value which is to be unique among all Application Protocol Entities enrolled at a node. If an Application Protocol Entity un-enrolls, the value of its Entity ID shall not be re-used unless all other values not assigned to an Application Protocol Entity at the time of the un-enroll have been assigned.

The GCC Provider shall examine the Conducting Operation Capable Flag of each enrolled Application Protocol Entity with the Active/Inactive flag set to Active (inactive Application Protocol Entities are assumed not to be capable of conducting operation). If there is more than one such enrolled Application Protocol Entity with this flag set, the GCC Provider shall choose one of them for which to set this flag in the Application Roster information which is to be sent in the RosterUpdateIndication PDU. The rule for which of these to choose is a local matter not specified in this Recommendation. A typical rule may be to choose the first such Application Protocol Entity that enrolls.

Upon receiving the GCC-Conference-Announce-Presence request from the Control GCCSAP and a GCC-Application-Enroll request from all Application Protocol Entities, a GCC Provider which is not the Top GCC Provider shall send a RosterUpdateIndication to the GCC Provider directly above it in the connection hierarchy. This is done by issuing an MCS-Send-Data request specifying the Node ID of the destination node as the Channel ID, specifying High data priority, and including the PDU in the Data field. The Node ID of the node directly above in the connection hierarchy is determined at the time of connection to the conference from either the ConferenceCreateResponse, ConferenceJoinResponse, or ConferenceInviteRequest PDUs, depending on how the conference was joined. The value of the Node ID of the above node shall also be included in the conference roster portion of the PDU as the Superior Node parameter. The content of the RosterUpdateIndication PDU is shown in Table 8-35. In this case, the roster is sent as a full refresh, including the local Node Record as well as the Application Records for all enrolled Application Protocol Entities and the Application Capabilities Lists for all Application Protocol Sessions corresponding to enrolled Application Protocol Entities. If there had been more than one Application Protocol Entity locally enrolled for a given Application Protocol Session, the GCC Provider shall perform a collapse of the Application Capabilities List among the Peer Application Protocol Entities to produce the

Application Capabilities List which is to be included in the RosterUpdateIndication PDU for that Application Protocol Session. This collapse shall be done by the procedure described in 8.3.2.8. The action taken by a node receiving a RosterUpdateIndication is described in 8.3.2.5.

A node which is the Top GCC Provider having just entered a conference, and received a GCC-Conference-Announce-Presence request shall include the information contained in its Conference Roster database, but may wait until it has received any RosterUpdateIndication PDUs from at least one other node in the conference before transmitting any PDUs. On receiving such indication, it shall update its Conference and Application Roster database and then broadcast the full Conference and Application Rosters by sending a RosterUpdateIndication to all nodes in the conference. This is done by issuing an MCS-Uniform-Send-Data request, specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. In the case of new nodes entering the conference, the RosterUpdateIndication is sent as a full refresh, including the Node Records from all nodes joined to the conference, as well as for each Application Protocol Session in the conference, a collapsed Application Capabilities List, and a list of Peer Application Protocol Entities which are part of that session along with the Application Record for each. The rules for generating the full Conference and Application Rosters, including the collapsed Application Capabilities List are described in 8.3.2.5.

Table 8-35/T.124 – RosterUpdateIndication GCCPDU

Content	Source	Sink
Full Refresh Flag	Source GCC Provider	Destination GCC Provider
Conference Information (optional)	Source GCC Provider from information in Request or received PDU	Destination GCC Provider
Application Information (optional)	Source GCC Provider from information in Request or received PDU	Destination GCC Provider

8.3.2.3 Enrolling Application Protocol Entities

On receipt of a GCC-Application-Enroll request primitive, a GCC Provider shall first determine if there is already an existing entry in the Local Application Roster for the specified conference associated with the GCCSAP of the requester. If not, and if the Enroll/un-enroll flag in the request is set to Enroll, a new entry is created in the Local Application Roster containing the information specified in the request primitive. In this, case the GCC Provider shall also assign an Entity ID to the newly-enrolled Application Protocol Entity as described in 8.3.2.2. If the Enroll/un-enroll flag had been set to Un-enroll, the request is confirmed immediately by generating a GCC-Application-Enroll confirm and issuing it to the GCCSAP of the requester. In this case, no further action is taken.

If the entry in the Local Application Roster already exists and the Enroll/un-enroll flag is set to Enroll, the contents of the existing entry are modified to reflect the new values specified. If the Enroll/un-enroll flag had been set to Un-enroll, the GCC Provider shall remove the corresponding entry from the Local Application Roster.

If the GCC Provider is not the Top GCC Provider for the specified conference, the GCC Provider shall then send its new Local Application Roster to the node directly above it in the connection hierarchy, using a RosterUpdateIndication PDU. It shall do this by issuing an MCS-Send-Data request specifying the Node ID of the higher node as the Channel ID, specifying High data priority, and including the selected PDU in the Data field. The Node ID of the GCC Provider directly above in the connection hierarchy is determined at the time of connection to the conference from either the ConferenceCreateResponse, ConferenceJoinResponse, or ConferenceInviteRequest PDUs,

depending on how the conference was joined. The content of the RosterUpdateIndication PDU is shown in Table 8-35.

If the enroll had occurred prior to the initial transmission of a RosterUpdateIndication PDU, then the new or altered Record is included in the initial roster information, sent as a full refresh as described in 8.3.2.2. If an Application Protocol Entity un-enrolled prior to the initial transmission, its record entry is removed from the roster and never included in any transmitted PDU.

If the enroll occurred after the initial transmission of a RosterUpdateIndication PDU, the new, modified, or removed Record is treated as an update to the existing roster. In this case, the GCC Provider may either choose to re-send its portion of the Conference and Application Rosters in their entirety, or it may send the information as an update – only sending information regarding the record that has changed. It is preferred that the latter method be used in order to minimize the size of the PDU, and therefore, its transmission time. If more than one change has been made (more than one GCC-Application-Enroll request had been received) since the last update, these changes may be concatenated into a single PDU. For a given Application Protocol Session, if only the Application Record information has changed since the last instance of that Application Protocol Session, the Application Capabilities List need not be included in the RosterUpdateIndication PDU, only the modified Application Record. For a given Application Protocol Session, the Application Capabilities List requires updating if its contents from an already enrolled Application Protocol Entity had been changed, or if a newly-enrolled Application Protocol Entity is part of the Application Protocol Session for which Peer Application Protocol Entities had already been enrolled at the local node or at nodes below the local node in the connection hierarchy, or if a node is un-enrolled in the case that there are other Peer Application Protocol Entities which remain enrolled at the local node or at nodes below the local node in the connection hierarchy. In the case where there is more than one Peer Application Protocol Entity either after an enroll, or before an un-enroll, the GCC Provider shall recreate the collapsed Application Capabilities List for that Application Protocol Session prior to including it in the RosterUpdateIndication PDU. This collapse shall be done by the procedure described in 8.3.4.

Before sending the RosterUpdateIndication PDU, if the Active/Inactive flag for the enrolling Application Protocol Entity is set to Active, the GCC Provider shall examine the Conducting Operation Capable Flag. If this flag is set, the GCC Provider shall ensure that only one Peer Application Protocol Entity from this node for each Application Protocol Session is included in the Application Roster with this flag set. If no such Peer Application Protocol Entity has been chosen indicated in the Application Roster so far, the newly enrolling Application Protocol Entity may be included in the transmitted RosterUpdateIndication PDU with this flag set. If there has already been a Peer Application Protocol Entity at this node which has been included in the Application Roster with this flag set, the GCC Provider may either include the newly enrolling Application Protocol Entity in the RosterUpdateIndication PDU without this flag set, or it may include the newly enrolling Application Protocol Entity with this flag set, and the previous designated conducting-capable Application Protocol Entity with this flag now set to FALSE. The rule for choosing which Peer Application Protocol Entity to include in the Application Roster with this flag set is a local matter beyond the scope of this Recommendation. If the Active/Inactive flag for the enrolling Application Protocol Entity is set to Inactive, the Conducting Operation Capable Flag indicated in the request primitive shall be ignored, and the corresponding field in the updated roster shall be set to FALSE.

In the case the Top GCC Provider, the successful change of an entry in the Local Application Roster results in a direct modification to the Conference Application Roster and the associated procedure for notification of all other nodes in the conference of the new Conference Application Roster as described in 8.3.2.5.

In all of the successful cases, the GCC Provider shall generate a GCC-Application-Enroll confirm primitive indicating a successful result and issue it to the GCCSAP of the requester.

8.3.2.4 Updating a Conference Roster Entry

If a GCC-Conference-Announce-Presence request is received after having already transmitted an initial roster, the new Node Record contained in this request shall be treated as an update to the Conference Roster. As in the case of an updated Application Record, the GCC Provider shall then send its new Node Record to the node directly above it in the connection hierarchy, using a RosterUpdateIndication PDU. It shall do this by issuing an MCS-Send-Data request specifying Node ID of higher node as the Channel ID, specifying High data priority, and including the selected PDU in the Data field. The content of the RosterUpdateIndication PDU is shown in Table 8-35.

As in the case of the enroll, the GCC Provider may either choose to re-send its portion of the Conference and Application Rosters in their entirety, or it may send only the new Node Record as an update. It is preferred that the later method be used in order to minimize the size of the PDU, and therefore, its transmission time.

If a GCC provider receives an MCS-Detach-User indication for which the User ID is that of a locally enrolled Application Protocol Entity, it shall update the roster to remove that Application Protocol Entity, and it shall issue a GCC-Permission-To-Enroll indication primitive to the GCCSAP corresponding to that Application Protocol Entity.

8.3.2.5 Propagation of Roster Updates to the Top GCC Provider

On receipt of a RosterUpdateIndication PDU from a node directly below in the connection hierarchy, a GCC Provider shall first update its subset of the Conference and Application Rosters. Each node in the conference shall maintain that subset of the Conference and Application Rosters which corresponds to that node as well as all nodes below it in the connection hierarchy. For each Application Protocol Entity which has been enrolled at any of these nodes, a list of Application Records including the Node ID of the node to which that record corresponds, and a partially collapsed Application Capabilities List are maintained. This is the Application Capabilities List which corresponds to the outcome of the set of rules to be applied to the Application Capabilities Lists of the Nodes directly below this node in the connection hierarchy (which are themselves partially collapsed).

The procedure that the GCC Provider shall use to update its roster subset depends on the update method used in the received RosterUpdateIndication. If the update was indicated to be a full refresh (as indicated by the full-refresh flag in the PDU), the following procedure is used. First, all Node Records and Application Records corresponding to the node from which the PDU was received, or nodes which were previously known to be below that node in the connection hierarchy are removed from the local subset of the Conference and Application Rosters. All Conference and Application Records listed in the PDU are then added to the roster. The entire Application Capabilities List, corresponding to the requesting node below, which may have been stored at this node is removed. The Application Capabilities List specified in the PDU replaces it. For all Application Protocol Entities either currently known to be enrolled at or below the current node, as well as any new Application Protocol Entities indicated in the newly received PDU (including ones not specified in this PDU), a new collapsed Application Capabilities List shall be computed from each of the Application Capabilities Lists from the nodes directly below. This collapse shall be done by the procedure described in 8.3.4.

In the case that the RosterUpdateIndication did not indicate a full refresh, a slightly different procedure is followed. First, if the Node Record information was indicated as changed, if the change was done as a refresh of the list of Node Records, all Node Records corresponding to the node from which the PDU was received, or nodes which were previously known to be below that node in the

connection hierarchy are removed from the local subset of the Conference Roster. All Node Records listed in the PDU are then added to the roster. If the change was done as a series of updates to Node Records, any records indicated as added are added to the roster subset, any indicated as replaced are used to replace the existing record, and any indicated for removal are removed. An attempt to change a record which already exists, or to modify or remove a record which does not exist is ignored. In any case, for any nodes which were previously in the roster subset and have been removed, all Application Records corresponding to those nodes are also removed (regardless of whether the corresponding update information was included in the Application Roster portion of the PDU). If any Application Record information was indicated as changed, for each Application Protocol Session, a similar procedure is followed. For each Application Protocol Session, if the change was done as a refresh, all Application Records corresponding to the node from which the PDU was received, or nodes which were previously known to be below that node in the connection hierarchy are removed from the local subset of the Application Roster. All Application Records listed in the PDU are then added to the roster. It is possible that for a given Application Protocol Session, it is indicated that there are no nodes which have that Application Protocol Entity enrolled. If the change was done as a series of updates to Application Records, any records indicated as added are added to the roster subset, any indicated as replaced are used to replace the existing record, and any indicated for removal are removed. An attempt to change a record which already exists, or to modify or remove a record which does not exist is ignored. For Application Capabilities List, the same procedure is followed as described for the full-refresh case. In this case, however, the new Application Capabilities Lists are only re-computed for sets of Peer Application Protocol Entities which have indicated a change.

In either of the above cases, the updated information for the Conference Roster and for each Application Protocol Session in the Application Roster includes an instance number. If the conference information has changed due to the received update, the instance number for the conference information shall be incremented by one modulo 2^{16}. Similarly, for each Application Protocol Session for which any information was modified (either Application Records or the Application Capabilities List), this instance number shall be incremented by one modulo 2^{16}. If multiple changes are made to be forwarded to other nodes as a single update, these changes may be counted as a single increment to the instance number. The instance numbers are maintained locally and corresponds to its local subset of the roster. That is, that node plus the nodes located below it in the connection hierarchy. In the case of the Top GCC Provider, the instance numbers apply to the Conference and Application Rosters which are broadcast to all nodes and reported in the GCC-Application-Roster-Report primitive. The conference information also includes a flag indicating whether nodes have been added and/or removed since the last instance. Similarly, for each Application Protocol Session, a flag indicates whether nodes have been added and/or removed since the last instance of the information for that set of Application Protocol Entities. In this case, nodes being added or removed may indicate that an Application Protocol Entity has been enrolled or un-enrolled at a node, respectively, not necessarily that the entire node has been added or removed from the conference.

Once the Conference and/or Application Roster information has been re-computed, a GCC Provider which is not the Top GCC Provider shall generate a RosterUpdateIndication PDU which shall then be sent to the GCC Provider directly above it in the connection hierarchy. The format of the information contained in this PDU may be either a full refresh, a refresh of some sets of Peer Application Protocol Entities and/or the conference information, or as individual updates. The choice of format to send the update information is left to the GCC Provider. It is preferred that a choice which minimizes the size of the PDU, and therefore minimizes transmission time, be used.

8.3.2.6 Distribution of the conference and application rosters

When the Top GCC Provider has received a RosterUpdateIndication, it shall modify the Conference and Application Roster information which it maintains (which, in this case, are the full Conference and Application Rosters rather than a subset) in a manner identical to that described for an intermediate MCU updating its subset of the roster. Once the complete roster has been updated, the Top GCC Provider shall broadcast the updated roster information to all nodes by sending a RosterUpdateIndication PDU. This is done by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel-ID, specifying High data priority, and including the PDU in the Data field.

If the case that the Conference Roster has been modified to include new nodes, the RosterUpdateIndication shall be broadcast as a full refresh. That is, the Conference and Application Rosters shall be transmitted in full to all nodes. In the case that a modification had been made to the roster which did not involve the addition of new nodes, the Top GCC Provider may choose to send the update either as a full refresh, or as a refresh to the list of Node Records, and/or the list of Application Records for some or all sets of Peer Application Protocol Entities, and/or the Application Capabilities List for some or all Application Protocol Sessions.

On receipt of a RosterUpdateIndication on the GCC-Broadcast-Channel, each GCC Provider shall generate a GCC-Conference-Roster-Report indication and issue it to the Control GCCSAP if the received PDU indicated any change to the Conference Roster. If the received PDU had indicated a change to some or all of the Application Roster, the GCC Provider shall generate a series of GCC-Application-Roster-Report indications and issue them to the GCCSAP associated with each enrolled Application Protocol Entity corresponding to an Application Protocol Session for which an Application Roster update has been received in the PDU. It may also issue GCC-Application-Roster-Report indications to other GCCSAPs, although the need to do so is considered a local matter beyond the scope of this Recommendation. Only that portion of the Conference Application Roster associated with the Session Key for that Application Protocol Entity must be included in the corresponding primitive. The GCC Provider may choose to include portions of the roster corresponding to other Session Keys, although the need to do this is considered a local matter beyond the scope of this Recommendation. In the case of an Application Protocol Entity enrolled inactively with no Session ID, the GCC Provider shall issue a GCC-Application-Roster-Report indication to the corresponding GCCSAP for portions of the roster corresponding to any Application Protocol Session with the same base Application Protocol as the Application Protocol Entity. The GCC Provider shall also issue a GCC-Application-Roster-Report indication to the Control GCCSAP, including Application Roster updates for all sets of Peer Application Protocol Entities which have been indicated as changed by the received PDU.

8.3.2.7 Nodes leaving a conference

On receiving an MCS-Detach-User indication, MCU nodes which have nodes below them in the connection hierarchy shall check the User ID indicated and determine if it corresponds to the Node ID of a node directly below it in the connection hierarchy. If so, it shall remove all entries corresponding to this node, as well as any nodes known to be connected below that node, from its subset of the Conference and Application Rosters. It shall then re-compute the Application Capabilities Lists for all sets of Peer Application Protocol Entities in the manner described in 8.3.4. Once the roster subset has been fully updated to reflect the leaving node, that GCC Provider shall follow the procedure described in 8.3.2.5 to propagate this update to the other nodes in the conference.

At a node which has disconnected from a conference (via either disconnection, termination of the conference, or ejection from the conference), the GCC Provider shall generate a GCC-Permission-To-Enroll indication which revokes permission to enroll in the corresponding conference. It shall issue this to all non-Control GCCSAPs.

8.3.2.8 An example of a roster update

Figure 8-5 shows an example of a GCC-Application-Enroll request issued during a conference causing an update of the Conference Application Roster. In this example, the node is in the third layer of the connection hierarchy. It issues the update indication to the next higher node which then formats and sends an update indication to the next higher node, which in this example, is the top node in the hierarchy. The top node assembles the full Application Roster and broadcasts the portions associated with the updated Application Protocol Session to all nodes in the conference, resulting in GCC-Application-Report primitives being issued at all nodes to the Node Controller, and if present, the Peer Application Protocol Entities.

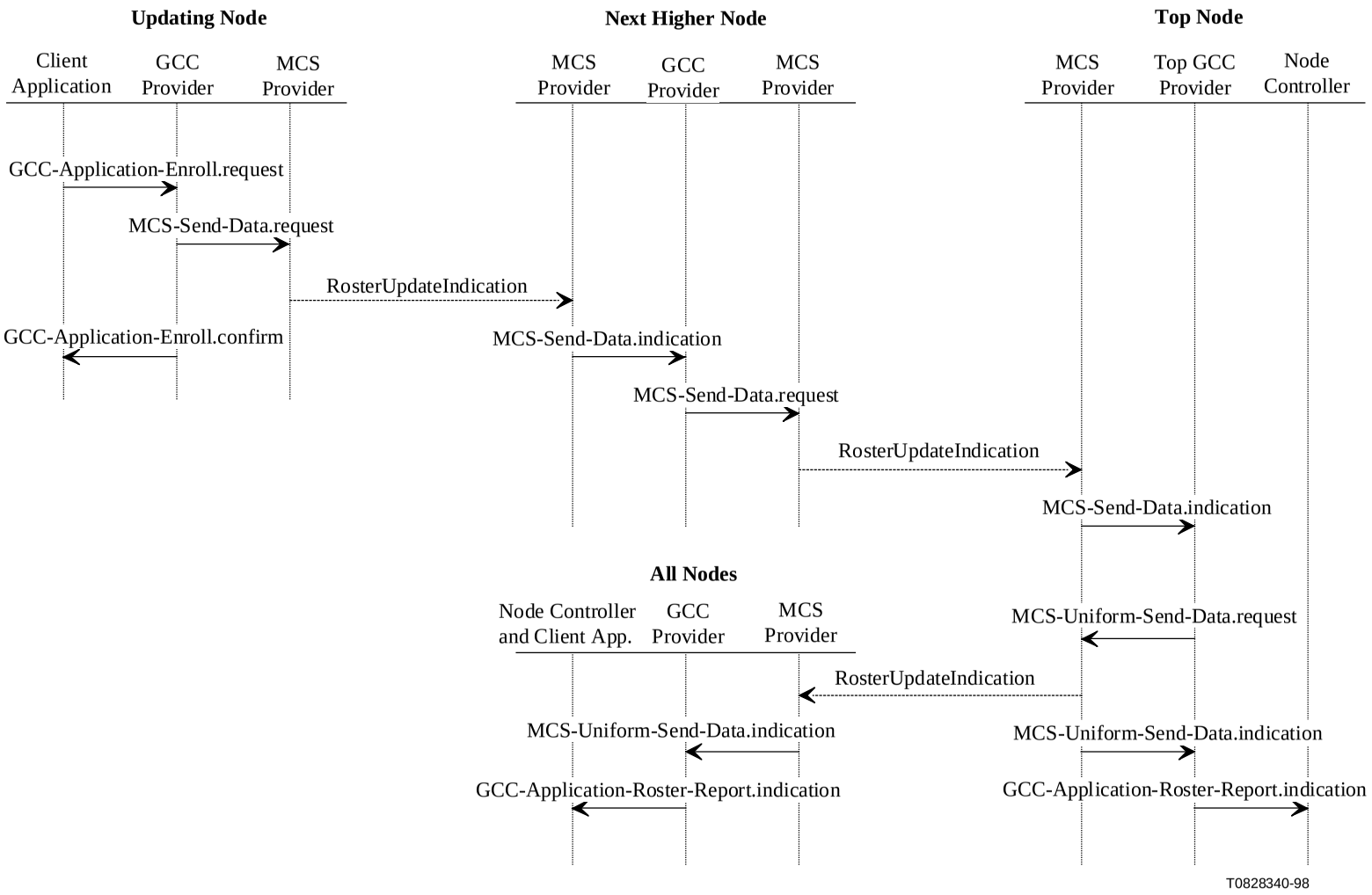

Figure 8-5/T.124 – An example of updating the Application Roster

8.3.3 Scalable Roster Protocol

The Scalable Roster Protocol was introduced to support various levels of scalability within a conference. The protocol relies on the use of Roster deltas instead of Full Roster Refreshes (which the Original Roster Protocol relied on) to exchange the most dynamically changing roster information. The protocol also relies on the use of Node Categories. The protocol specifies a markedly different behavior depending on a node's assigned Node Category. The subclauses below describe the Scalable Roster protocol in detail.

8.3.3.1 Overview

Both the Conference and Application Rosters are communicated among nodes using the same set of PDUs. The RosterUpdateIndication PDU is used to send complete or partial roster information to other nodes in the conference. In addition, the RosterRefreshRequest PDU is used to make requests for a full refresh of a Conference and/or Application roster from a Parent node.

When each node (other than the top node) first announces its roster information upon joining a conference, the node sends a RosterUpdateIndication to the node directly above it in the hierarchy. This is the case for all three Node Categories. Subsequent updates to any portion of either the Conference or Application Roster information are announced by reissuing a RosterUpdateIndication containing the new information.

When a node updates its portion of the roster (or announces it for the first time), that information is propagated from node-to-node up the connection hierarchy until it reaches the Top GCC Provider. The Top Provider is responsible for forming the full Conference and Application rosters and then distributing deltas of the rosters to all nodes in the conference. Whether or not a node is added to the roster depends on the Node Category of the node joining the conference. For instance, Conventional nodes will always be added to the roster while Anonymous nodes are never added. Counted nodes are only added to rosters maintained by Conventional nodes.

If the roster information propagating up the connection hierarchy is associated with a Conventional node, each intermediate node (MCU) is responsible for forming a subset of the full Conference and Application Rosters by storing this node's roster information. The Conference Roster is a subset that includes Node Records for the receiving node and all other Conventional nodes below it in the connection hierarchy. The Application Roster includes a separate list of Application Records for each Application Protocol Session in existence (at a Conventional Node) at or below the node that receives the roster update. It also includes the Application Capabilities List for these sessions. On receiving a RosterUpdateIndication from a Conventional node below, an intermediate node makes the appropriate changes to its subset of its rosters, then passes the roster update up to the next higher node by issuing a RosterUpdateIndication. Note that regardless of Node Category, the RosterUpdateIndication is always forwarded. The only difference here is that Counted and Anonymous nodes are not maintained in the roster subsets held at intermediate nodes.

The primary reason for maintaining subsets of roster information at intermediate nodes is to maintain the collapsed set of capabilities. Since Counted and Anonymous nodes do not affect capabilities, there is no need to maintain their records in the roster subsets. If an update from a lower Conventional node included a change that caused the Application Capabilities Lists for at least one Application Protocol Session to have to be recomputed, the new roster information for the roster subset can only be generated if the individual Application Capabilities Lists for each Conventional node directly below it in the connection hierarchy is known (to allow the collapse rules to be reapplied with the new updated information). The GCC Provider at each node shall maintain this information locally.

When a roster update associated with a Conventional node reaches the Top GCC Provider, it shall then broadcast the new roster information to all nodes in the conference (including Counted and Anonymous nodes). This is done by broadcasting only the delta information in a RosterUpdateIndication PDU to all nodes. If the roster update information that reaches the Top GCC Provider is associated with a Counted node, a delta update is broadcasted to both the Conventional nodes participating in the conference and to the joining Counted node that initiated the original roster update. No other Counted or Anonymous nodes will receive a roster update when this occurs. Anonymous node updates received at the Top GCC Provider do not cause an update to be broadcasted. Instead, the local Top GCC Provider may maintain a separate list of Anonymous nodes that are currently listening in (or have listened in) on the conference for informational or statistical purposes (this is up to the local implementation).

Roster information sent using the RosterUpdateIndication PDU may be sent in one of three ways. The roster may be sent as a refresh of the full Conference and Application Rosters, replacing all existing roster entries. Alternatively, portions of the roster may be sent so that the Conference Roster and/or portions of the Application Roster associated with one or more Session Keys may be refreshed. In this case, all entries in the transmitted portion of the roster are replaced, but portions that are not sent are left unchanged. Finally, changes to portions of the roster may be sent as updates so that only those particular elements of the roster (e.g. individual Application Records) which have been added, modified, or removed are sent and all other entries are left unchanged.

Since only delta RosterUpdateIndication PDUs are ever broadcasted from the Top Provider, the burden of assimilating a complete Conference or Application Roster is placed on the joining node and its Parent node. To do this, the joining node forwards a RosterRefreshRequest PDU to its Parent node requesting a Full Roster Refresh. The point at which this request is issued depends on the joining node's Node Category. For Conventional and Counted nodes, this request is issued after it receives a roster update from the Top Provider for itself (which ensures that its presence in the conference is known). For Anonymous nodes, the request can be made immediately following the reception of its Parent node's User ID.

After the RosterRefreshRequest PDU is received by the Parent node, it must either process the request (if it has access to the requested rosters) or it must forward the request up to the next node in the connection hierarchy. When a node is reached that contains the requested information, that node directs a RosterUpdateIndication containing a Full Roster Refresh directly to the requesting node. Note that the response to a RosterRefreshRequest will look different depending on the Node Category to which the requesting node belongs. For instance, if a Conventional node initiated the request, a full refresh which contains both Conventional and Counted nodes must be delivered. Therefore, a Conventional Parent node must be reached before the request can be processed (because only Conventional nodes maintain a list of Counted nodes). If the requesting node is either Anonymous or Counted, the request can be handled by the first node that contains the Conventional node roster.

It is necessary for joining nodes to buffer incoming RosterUpdateIndications received from the Top GCC Provider prior to receiving the response from their initial RosterRefreshRequest. This is necessary to ensure that no updates get lost which guarantee the synchronization of all Rosters in the conference. All RosterUpdateIndications include an instance number that is sequentially incremented every time the Top Provider detects that a roster changed. This roster instance number is used to determine the order that RosterUpdateIndications should be processed at a joining node. This can be especially complicated for Conventional nodes that are participating in a conference which can include both Conventional and Counted nodes. This is because RosterUpdateIndications will be received on both the GCC-Conventional-Broadcast-Channel and the GCC-Counted-Broadcast-Channel. It is up to the Conventional node to process the RosterUpdateIndications in the correct order based on its prior knowledge that the roster instance numbers are incremented sequentially.

8.3.3.2 Nodes entering a conference

When a node is joined to a conference, either via a conference creation, join, or invitation, a GCC Provider shall issue a GCC-Application-Permission-To-Enroll indication to the GCCSAP for all Application Protocol Entities (APEs) that have locally indicated their presence to the GCC Provider. In the case that a GCC Provider becomes aware of an additional Application Protocol Entity while it is already joined to a conference, the GCC Provider shall issue a GCC-Application-Permission-To-Enroll indication to the corresponding GCCSAP to indicate the existence of this conference. The Application Protocol Entity is informed of its Node Category through the GCC-Application-Permission-To-Enroll.

Before taking any further action, the GCC Provider shall wait until it has received a GCC-Conference-Announce-Presence request from the Control GCCSAP as well as a GCC-Application-Enroll request from all Application Protocol Entities which have been sent GCC-Application-Permission-To-Enroll indications. Some of these enroll requests may indicate that that Application Protocol Entity does not intend to enroll (the enroll/un-enroll flag is set to un-enroll). For each received GCC-Application-Enroll request with the enroll flag set, the information provided in the primitive for that Application Protocol Entity is added to the Local Application Roster.

The GCC Provider shall also assign a locally allocated Entity ID to each enrolled Application Protocol Entity. This shall be used by the GCC Provider as an identifier of each corresponding Application Protocol Entity in the RosterUpdateIndication PDU. The assigned Application Protocol Entity ID is also included in the GCC-Application-Enroll confirm primitive that is returned to each enrolling Application Protocol Entity along with the Node ID of the local node. The Entity ID is a 16-bit integer value that is to be unique among all Application Protocol Entities enrolled at a node. If an Application Protocol Entity un-enrolls, the value of its Entity ID shall not be re-used unless all other values not assigned to an Application Protocol Entity at the time of the un-enroll have been assigned.

The GCC Provider shall examine the Conducting Operation Capable Flag of each enrolled Application Protocol Entity with the Active/Inactive flag set to Active (inactive Application Protocol Entities are assumed not to be capable of conducting operation). If there is more than one such enrolled Application Protocol Entity with this flag set, the GCC Provider shall select one of them for which to set this flag in the Application Roster information. This information is to be sent in the RosterUpdateIndication PDU. The rule for which of these to select is a local matter not specified in this Recommendation. A typical rule may be to select the first such Application Protocol Entity that enrolls.

Depending on a node's Node Category, the local GCC Provider must determine the proper course of action to take after receiving the GCC-Conference-Announce-Presence request from the Control GCCSAP and a GCC-Application-Enroll request from all Application Protocol Entities. Since only Conventional nodes are allowed to create application sessions, GCC providers at Counted and Anonymous nodes must hold back any RosterUpdateIndication information associated with non-existing APE sessions until a RosterUpdateIndication PDU is received from a node above. The above node notifies that the session being enrolled with actually exists. The local GCC Providers at Counted and Anonymous nodes can also use the Application Protocol Keys received in the initial enrollments to determine local interest in incoming RosterUpdateIndications. For instance, any time a local GCC Provider is informed of a new session through a RosterUpdateIndication received from either the Top Provider or its Parent node, it must decide which APEs are interested in the occurrence. The local provider uses the Application Protocol Key associated with the new session to decide which APEs to send a Roster Update Indication to. Detailed procedures are described below for each Node Category.

A GCC Provider at a Conventional node that is not the Top GCC Provider shall send a RosterUpdateIndication to the GCC Provider directly above it in the connection hierarchy upon receiving the GCC-Conference-Announce-Presence request from the Control GCCSAP and a GCC-Application-Enroll request from all Application Protocol Entities. This is done by issuing an MCS-Send-Data request specifying the Node ID of the destination node as the Channel ID, specifying High data priority, and including the PDU in the Data field. The Node ID of the node directly above in the connection hierarchy is determined at the time of connection to the conference from either the ConferenceCreateResponse, ConferenceJoinResponse, or ConferenceInviteRequest PDUs, depending on how the conference was joined. The value of the Node ID of the above node shall also be included in the conference roster portion of the PDU as the Superior Node parameter. The content of the RosterUpdateIndication PDU is shown in Table 8-35. In this case, the roster is sent as a full refresh, including the local Node Record, as well as the Application Records for all enrolled Application Protocol Entities and the Application Capabilities Lists for all Application Protocol Sessions corresponding to enrolled Application Protocol Entities. If there had been more than one Application Protocol Entity locally enrolled for a given Application Protocol Session, the GCC Provider shall perform a collapse of the Application Capabilities List among the Peer Application Protocol Entities. This would produce the Application Capabilities List that is to be included in the RosterUpdateIndication PDU for that Application Protocol Session. This collapse shall be done by the procedure described in 8.3.4.

A GCC Provider at a Counted node shall send a RosterUpdateIndication to the GCC Provider directly above it in the connection hierarchy upon receiving the GCC-Conference-Announce-Presence request from the Control GCCSAP and a GCC-Application-Enroll request from all Application Protocol Entities. This is done by issuing an MCS-Send-Data request specifying the Node ID of the destination node as the Channel ID, specifying High data priority, and including the PDU in the Data field. The value of the Node ID of the above node shall again be included in the conference roster portion of the PDU as the Superior Node parameter. In this case, only the local Node Record is sent. The Application Records are held back until the local GCC Provider receives a RosterUpdateIndication from an above node informing it that the locally enrolled sessions actually exist.

A GCC Provider at an Anonymous node shall send a RosterUpdateIndication to the GCC Provider directly above it in the connection hierarchy upon receiving the GCC-Conference-Announce-Presence request from the Control GCCSAP and a GCC-Application-Enroll request from all Application Protocol Entities. This is done by issuing an MCS-Send-Data request specifying the Node ID of the destination node as the Channel ID, specifying High data priority, and including the PDU in the Data field. The value of the Node ID of the above node shall again be included in the conference roster portion of the PDU as the Superior Node parameter. Again, only the local Node Record is sent. Note that this RosterUpdateIndication is used only for informational purposes and does not affect the Conference Roster. An Anonymous node will never send a RosterUpdateIndication that includes roster information associated with an Application Protocol Entity. An Anonymous node should also never send more than a single RosterUpdateIndication.

A node that is the Top GCC Provider and just entered a conference, and that received a GCC-Conference-Announce-Presence request, shall include the information contained in its Conference Roster database. But the node may wait until it has received any RosterUpdateIndication PDUs from at least one other node in the conference before transmitting any PDUs. If an indication is received from a Conventional node, it shall update its Conference and Application Roster database, then broadcast back out the roster information associated with the update indication by sending a RosterUpdateIndication to all nodes in the conference. This is done by issuing an MCS-Send-Data request, specifying the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. Updates sent from the Top

GCC Provider to every node in the conference because of a newly joining node are never sent as a full refresh, which holds down the network traffic incurred by adding new nodes to a conference.

8.3.3.3 Enrolling Application Protocol Entities

On receiving a GCC-Application-Enroll request primitive, a GCC Provider shall first determine if there is already an existing entry in the Local Application Roster for the specified conference associated with the GCCSAP of the requester. If not, and if the Enroll/Un-enroll flag in the request is set to Enroll, a new entry is created in the Local Application Roster that contains the information specified in the request primitive. In this case, the GCC Provider shall also assign an Entity ID to the newly enrolled Application Protocol Entity as described in 8.3.3.2. If the Enroll/Un-enroll flag had been set to Un-enroll, the request is confirmed immediately by generating a GCC-Application-Enroll confirm and issuing it to the GCCSAP of the requester. In this case, no further action is taken.

If the entry in the Local Application Roster already exists and the Enroll/Un-enroll flag is set to Enroll, the contents of the existing entry are modified to reflect the new values specified. If the Enroll/Un-enroll flag had been set to Un-enroll, the GCC Provider shall remove the corresponding entry from the Local Application Roster.

If the GCC Provider is not the Top GCC Provider for the specified conference and is a Conventional node, the GCC Provider shall then send its new Local Application Roster to the node directly above it in the connection hierarchy, using a RosterUpdateIndication PDU. The GCC Provider shall do this by issuing an MCS-Send-Data request specifying the Node ID of the higher node as the Channel ID, specifying High data priority, and including the selected PDU in the Data field. The Node ID of the GCC Provider directly above it in the connection hierarchy is determined at the time of connection to the conference from either the ConferenceCreateResponse, ConferenceJoinResponse, or ConferenceInviteRequest PDUs, depending on how the conference was joined. The content of the RosterUpdateIndication PDU is shown in Table 8-35.

If the GCC Provider is a Counted node, the procedure is a bit more complicated. Before sending its new Local Application Roster to the node directly above it in the connection hierarchy, the local GCC Provider must determine if the sessions associated with each enroll request actually exist. Only the enroll requests that correspond to sessions that exist will be forwarded. This is due to the requirement that Counted nodes are not allowed to create sessions. The local GCC Provider is made aware of new sessions in one of two ways: either through a RosterUpdateIndication PDU received from the Top Provider or through a response to a RosterRefreshRequest received from a Parent node. Once it has been determined which enroll requests are associated with existing sessions, the GCC Provider will send a RosterUpdateIndication PDU containing the appropriate Application Records to the node directly above it in the connection hierarchy. The GCC Provider shall do this by issuing an MCS-Send-Data request specifying the Node ID of the higher node as the Channel ID, specifying High data priority, and including the selected PDU in the Data field. RosterUpdateIndications sent from a Counted node should never include capability information since a Counted node cannot affect the collapsed capability list. Each APE must make a decision on whether or not it can support the established capabilities within a conference.

If the GCC Provider is an Anonymous node, the local GCC Provider simply stores the information in its Local Application Roster and uses it to determine the Application Protocol Sessions that a particular GCC Service Access Point (SAP) is interested in. No APE information will ever be sent from an Anonymous node inside a RosterUpdateIndication. It is important that the local GCC Provider maintains this information base so that it can relay all information about new or changing Application Rosters to the appropriate SAPs. This guarantees that all the APEs running on an Anonymous node have access to all the information necessary to join and listen in on a particular session.

In general, if the enroll had occurred prior to the initial transmission of a RosterUpdateIndication PDU, then the new or altered Record is included in the initial roster information, sent as a full refresh as described in 8.3.3.2. If an Application Protocol Entity un-enrolled prior to the initial transmission, its record entry is removed from the roster and is never included in any transmitted PDU.

If the enroll occurred after the initial transmission of a RosterUpdateIndication PDU, the new, modified, or removed Record is treated as an update to the existing roster. In this case, the GCC Provider may either choose to resend its portion of the Conference and Application Rosters in their entirety, or the GCC Provider may send the information as an update – only sending information regarding the record that has changed. It is preferred that the latter method be used in order to minimize the size of the PDU and, therefore, its transmission time. If more than one change has been made (more than one GCC-Application-Enroll request had been received) since the last update, these changes may be concatenated into a single PDU. For a given Application Protocol Session, if only the Application Record information has changed since the last instance of that Application Protocol Session, the Application Capabilities List need not be included in the RosterUpdateIndication PDU, only the modified Application Record. For a given Application Protocol Session at a Conventional node, the Application Capabilities List requires updating if any of the following occur:

- Its contents from an already enrolled Application Protocol Entity had been changed.
- A newly-enrolled Application Protocol Entity is part of the Application Protocol Session for which Peer Application Protocol Entities had already been enrolled at the local node or at nodes below the local node in the connection hierarchy.
- A node is un-enrolled in the case that there are other Peer Application Protocol Entities that remain enrolled at the local node or at nodes below the local node in the connection hierarchy.

In the case where there is more than one Peer Application Protocol Entity either after an enroll or before an un-enroll, the GCC Provider at a Conventional node shall re-create the collapsed Application Capabilities List for that Application Protocol Session prior to including it in the RosterUpdateIndication PDU. This collapse shall be done by the procedure described in 8.3.4.

If the Active/Inactive flag for the enrolling Application Protocol Entity is set to Active before sending the RosterUpdateIndication PDU, the GCC Provider shall examine the Conducting Operation Capable Flag. If this flag is set, the GCC Provider shall ensure that only one Peer Application Protocol Entity from this node for each Application Protocol Session is included in the Application Roster with this flag set. If no such Peer Application Protocol Entity has been selected in the Application Roster so far, the newly enrolling Application Protocol Entity may be included in the transmitted RosterUpdateIndication PDU with this flag set. If there has already been a Peer Application Protocol Entity at this node, which has been included in the Application Roster with this flag set, the GCC Provider may either:

- Include the newly enrolling Application Protocol Entity in the RosterUpdateIndication PDU without this flag set.
- Include the newly enrolling Application Protocol Entity with this flag set, and the previous designated conducting-capable Application Protocol Entity with this flag now set to FALSE.

The rule for selecting which Peer Application Protocol Entity to include in the Application Roster with this flag set is a local matter beyond the scope of this Recommendation. If the Active/Inactive flag for the enrolling Application Protocol Entity is set to Inactive, the Conducting Operation Capable Flag indicated in the request primitive shall be ignored, and the corresponding field in the updated roster shall be set to FALSE. The Conducting Operation Capable Flag should always be set to FALSE for APEs residing on Conducted and Anonymous nodes.

In the case of the Top GCC Provider, the successful change of an entry in the Local Application Roster results in a direct modification to the Conference Application Roster and the associated procedure for notifying all other nodes in the conference of the new Conference Application Roster as described in 8.3.3.6.

In all of the successful cases, the GCC Provider shall generate a GCC-Application-Enroll confirm primitive indicating a successful result. The GCC Provider shall issue it to the GCCSAP of the requester.

8.3.3.4 Updating a Conference Roster Entry

If a GCC-Conference-Announce-Presence request is received after having already transmitted an initial roster, the new Node Record contained in this request shall be treated as an update to the Conference Roster. As in the case of an updated Application Record, the GCC Provider shall then send its new Node Record to the node directly above it in the connection hierarchy by using a RosterUpdateIndication PDU. The GCC Provider shall do this by issuing an MCS-Send-Data request specifying the Node ID of the higher node as the Channel ID, specifying High data priority, and including the selected PDU in the Data field. The content of the RosterUpdateIndication PDU is shown in Table 8-35. Note that a GCC-Conference-Announce-Presence request should never be received after having already transmitted an initial roster from Anonymous nodes.

If a GCC Provider receives an MCS-Detach-User indication for which the User ID is that of a locally-enrolled Application Protocol Entity, the GCC Provider shall update the roster to remove that Application Protocol Entity and it shall also issue a GCC-Permission-To-Enroll indication primitive to the GCCSAP corresponding to that Application Protocol Entity.

8.3.3.5 Propagation of Roster Updates to the Top GCC Provider

On receiving a RosterUpdateIndication PDU from a node directly below in the connection hierarchy, a GCC Provider shall first update its subset of the Conference and Application Rosters. Each node in the conference shall maintain this subset of the Conference and Application Rosters that corresponds to that node as well as all nodes below it in the connection hierarchy. This subset should never include either Node Records or Application Records associated with Anonymous nodes. Each Node Record contains a flag that indicates a node's Node Category.

For each Application Protocol Entity that has been enrolled at any of these nodes, a list of Application Records, including the Node ID of the node to which that record corresponds and a partially-collapsed Application Capabilities List, are maintained. This is the Application Capabilities List that corresponds to the outcome of the set of rules to be applied to the Application Capabilities Lists of the Nodes directly below this node in the connection hierarchy (which are themselves partially collapsed).

The procedure that the GCC Provider shall use to update its roster subset depends on the update method used in the received RosterUpdateIndication. If the update was indicated to be a full refresh (as indicated by the Full-Refresh flag in the PDU), the following procedure is used. First, all Node Records and Application Records corresponding to the node from which the PDU was received, or nodes which were previously known to be below that node in the connection hierarchy are removed from the local subset of the Conference and Application Rosters. All Conference and Application Records listed in the PDU are then added to the roster. The entire Application Capabilities List corresponding to the requesting node below, which may have been stored at this node, is removed. The Application Capabilities List specified in the PDU replaces it. For all Application Protocol Entities, either currently known to be enrolled at or below the current node, as well as any new Application Protocol Entities indicated in the newly-received PDU (including ones not specified in this PDU), a new collapsed Application Capabilities List shall be computed from each of the

Application Capabilities Lists from the nodes directly below. This collapse shall be done by the procedure described in 8.3.4.

In the case that the RosterUpdateIndication did not indicate a full refresh, a slightly different procedure is followed. First, if the Node Record information was indicated as changed and the change was done as a refresh of the list of Node Records, all Node Records corresponding to the node from which the PDU was received (or nodes which were previously known to be below that node in the connection hierarchy) are removed from the local subset of the Conference Roster. All Node Records listed in the PDU are then added to the roster. If the change was done as a series of updates to Node Records, any records indicated as added are added to the roster subset; any records indicated as replaced are used to replace the existing records; and any records indicated for removal are removed. An attempt to change a record which already exists, or to modify or remove a record which does not exist, is ignored. In any case, for any nodes which were previously in the roster subset and have been removed, all Application Records corresponding to those nodes are also removed (regardless of whether the corresponding update information was included in the Application Roster portion of the PDU). If any Application Record information was indicated as changed, for each Application Protocol Session, a similar procedure is followed. For each Application Protocol Session, if the change was done as a refresh, all Application Records corresponding to the node from which the PDU was received (or nodes which were previously known to be below that node in the connection hierarchy) are removed from the local subset of the Application Roster. All Application Records listed in the PDU are then added to the roster. It is possible that for a given Application Protocol Session, it is indicated that there are no nodes which have that Application Protocol Entity enrolled. If the change was done as a series of updates to Application Records, any records indicated as added are added to the roster subset; any records indicated as replaced are used to replace the existing record; and any records indicated for removal are removed. An attempt to change a record that already exists, or to modify or remove a record which does not exist, is ignored. For Application Capabilities List, the same procedure is followed as described for the full refresh case. In this case, however, the new Application Capabilities Lists are only recomputed for sets of Peer Application Protocol Entities that have indicated a change.

In either of the above cases, the updated information for the Conference Roster and for each Application Protocol Session in the Application Roster includes an instance number. If the conference information has changed due to the received update, the instance number for the conference information shall be incremented by one modulo 2^{16}. Similarly, for each Application Protocol Session for which any information was modified (either Application Records or the Application Capabilities List), this instance number shall be incremented by one modulo 2^{16}. If multiple changes are made to be forwarded to other nodes as a single update, these changes may be counted as a single increment to the instance number. The instance numbers are maintained locally and correspond to the local subset of the roster. That is, that node plus the nodes located below it in the connection hierarchy. In the case of the Top GCC Provider, the instance numbers apply to the Conference and Application Rosters which are broadcast to all nodes and reported in the GCC-Application-Roster-Report primitive. The conference information also includes a flag indicating whether nodes have been added and/or removed since the last instance. Similarly, for each Application Protocol Session, a flag indicates whether nodes have been added and/or removed since the last instance of the information for that set of Application Protocol Entities. In this case, nodes being added or removed may indicate that an Application Protocol Entity has been enrolled or un-enrolled at a node, respectively, and not necessarily that the entire node has been added or removed from the conference.

Once the Conference and/or Application Roster information has been recomputed, a GCC Provider, which is not the Top GCC Provider, shall generate a RosterUpdateIndication PDU that shall then be sent to the GCC Provider directly above it in the connection hierarchy. The format of the information

contained in this PDU may be either a full refresh, a refresh of some sets of Peer Application Protocol Entities and/or the conference information, or individual updates. The format selection for sending the update information is left to the GCC Provider. It is preferred that the GCC Provider select a format that minimizes the size of the PDU and, therefore, minimizes transmission time used.

8.3.3.6 Distribution of the Conference and Application Rosters

When the Top GCC Provider has received a RosterUpdateIndication that includes either Conventional or Counted node roster information, it shall modify the Conference and Application Roster information which it maintains (which, in this case, are the full Conference and Application Rosters rather than a subset) in a manner identical to that described for an intermediate MCU updating its subset of the roster. Once the complete roster has been updated, the Top GCC Provider shall broadcast the updated roster information back to either every node in the conference or to the Conventional nodes participating in the conference. The choice depends on the roster information that was affected due to the received RosterUpdateIndication. Any changes to either the list of Conventional Node Records, the Application Roster due to an APE associated with a Conventional node, or to a Capability list, are transmitted to every node in the conference. This is done by issuing an MCS-Send-Data request specifying the GCC-Conventional-Broadcast-Channel as the Channel-ID, specifying High data priority, and including the PDU in the Data field. Changes to either the list of Counted Node Records or to an Application Roster due to an APE associated with a Counted node are transmitted only to Conventional nodes. This is done by issuing an MCS-Send-Data request specifying the GCC-Counted-Broadcast-Channel as the Channel-ID, specifying High data priority, and including the PDU in the Data field.

The RosterUpdateIndication broadcasted from the Top GCC Provider includes only the specific roster information that changed due to the original RosterUpdateIndication received. This could include any or all of the following: one or more Node Records, one or more Application Records, and any list of Application Capabilities that may have been altered due to processing the received RosterUpdateIndication. This should not contain information that has already been broadcasted to the rest of the conference and this should not be a full refresh.

On receiving a RosterUpdateIndication on either the GCC-Conventional-Broadcast-Channel or the GCC-Counted-Broadcast-Channel, each GCC Provider shall generate a GCC-Conference-Roster-Report indication and issue it to the Control GCCSAP if the received PDU indicated any change to the Conference Roster. If the received PDU had indicated a change to some or all of the Application Roster, the GCC Provider shall generate a series of GCC-Application-Roster-Report indications. The GCC Provider shall then issue them to the GCCSAP associated with each enrolled Application Protocol Entity that corresponds to an Application Protocol Session for which an Application Roster update has been received in the PDU. The GCC Provider may also issue GCC-Application-Roster-Report indications to other GCCSAPs, although the need to do so is considered a local matter beyond the scope of this Recommendation. Only that portion of the Conference Application Roster associated with the Session Key for that Application Protocol Entity must be included in the corresponding primitive. The GCC Provider may choose to include portions of the roster corresponding to other Session Keys, although the need to do this is considered a local matter beyond the scope of this Recommendation. In the case of an Application Protocol Entity enrolled inactively with no Session ID, the GCC Provider shall issue a GCC-Application-Roster-Report indication to the corresponding GCCSAP for portions of the roster corresponding to any Application Protocol Session with the same base Application Protocol as the Application Protocol Entity. The GCC Provider shall also issue a GCC-Application-Roster-Report indication to the Control GCCSAP, including Application Roster updates for all sets of Peer Application Protocol Entities which have been indicated as changed by the received PDU.

8.3.3.7 Nodes leaving a conference

On receiving an MCS-Detach-User indication, MCU nodes that have nodes below them in the connection hierarchy shall check the User ID indicated and determine if it corresponds to the Node ID of a node directly below it in the connection hierarchy. If so, the MCU node shall remove all entries corresponding to this node, as well as any nodes known to be connected below that node, from its subset of the Conference and Application Rosters. It shall then recompute the Application Capabilities Lists for all sets of Peer Application Protocol Entities in the manner described in 8.3.4. Once the roster subset has been fully updated to reflect the leaving node, that GCC Provider shall follow the procedure described in 8.3.3.6 to propagate this update to the other nodes in the conference.

At a node which has disconnected from a conference (via either disconnection, termination of the conference, or ejection from the conference), the GCC Provider shall generate a GCC-Permission-To-Enroll indication which revokes permission to enroll in the corresponding conference. The GCC Provider shall issue this to all non-Control GCCSAPs.

Node or Application records associated with Node IDs of Anonymous nodes should not appear in any node's Conference or Application Roster and can therefore be ignored.

8.3.3.8 Acquiring a Full Roster Refresh from a Parent Node

At any time during a conference, a node may request a Full Roster Refresh from its Parent node by making a RosterRefreshRequest to its Parent Node. A node does this by issuing an MCS-Send-Data request specifying the Node ID of the Parent node as the Channel ID, specifying High data priority, and including the RosterRefreshRequest PDU in the Data field.

Typically, a RosterRefreshRequest would be made under the following conditions: either the node has just joined the conference and needs a Full Roster Refresh to synch up its Conference Application Roster to other nodes in the conference, or a Roster Inquire was received by a GCC Provider at an Anonymous node that is not maintaining a local copy of the Conference Application Roster. In either case, after the RosterRefreshRequest PDU is received by the Parent node, it must either process the request (if it has access to the requested rosters), or it must forward the request up to the next node in the connection hierarchy using another RosterRefreshRequest.

When a node is reached that contains the requested information, that node directs a RosterUpdateIndication containing a Full Roster Refresh directly to the requesting node. A node does this by issuing an MCS-Send-Data request that specifies the Node ID of the originating node (contained in the RosterRefreshRequest) as the Channel ID, specifying High data priority, and including the RosterUpdateIndication PDU in the Data field.

As mentioned above, the RosterUpdateIndication PDU issued back to the node that made the original RosterRefreshRequest will look different depending on the Node Category to which the originating node belongs. For instance, if a Conventional node initiated the request, the roster refresh will contain records for both Conventional and Counted nodes. Therefore, a Conventional Parent node must be reached before the request can be processed (because only Conventional nodes maintain a list of Counted nodes). If the requesting node is either Anonymous or Counted, the request can be handled by the first node that contains the Conventional node roster. It is the responsibility of the node processing the RosterRefreshRequest to separate out records that should not be received by certain Node Categories.

The RosterRefreshRequest includes the following: a Node ID field which is filled in with the Node ID of the node initiating the request; a Node Category field which is filled in with Category of the node initiating the request; a Boolean value specifying that a Full Roster refresh should be delivered; a Boolean value specifying that the Conference Roster should be sent; a Session List that includes a

list of specific session rosters being requested; and an Application List that specifies a list of Application Protocols that a node wants to obtain information on. If the Full Refresh Boolean is set to TRUE, then the sendConferenceRoster field, the sessionList field and the applicationList field are ignored. If the Full Refresh Boolean is set to FALSE, then these three fields define what information should be delivered to the node that initiated the request. Therefore, a RosterRefreshRequest can be issued that requests one or more specific Application Rosters, or for just the Conference Roster, as well as a Full Roster Refresh. The process of handling a partial request is identical to the process outlined above, except that only the requested information is returned.

There are timing considerations that need to be addressed when a node attempts to join and synchronize its Conference Application Roster with other nodes in the conference. It is possible for a node to receive RosterUpdateIndications before it has received its initial Full Roster refresh from a Parent node. If these updates include a roster instance number that precedes the roster instance number associated with the Full Roster Refresh, the update can be thrown away. If these instance numbers are higher than the instance number associated with the Full Refresh, they need to be processed. Therefore, it is necessary for a joining node to buffer any incoming RosterUpdateIndications received from the Top GCC Provider prior to receiving the response to its initial RosterRefreshRequest. All RosterUpdateIndications include an instance number that is sequentially incremented every time the Top Provider detects that a roster changed. This roster instance number is used to determine the order that a RosterUpdateIndication should be processed at a joining node. As discussed above, this can be especially complicated for Conventional nodes that are participating in a conference that can include both Conventional and Counted nodes. This is because RosterUpdateIndications will be received on both the GCC-Conventional-Broadcast-Channel and the GCC-Counted-Broadcast-Channel. It is the responsibility of the Conventional node to process the RosterUpdateIndications in the correct order based on prior knowledge that the roster instance numbers are incremented sequentially.

8.3.3.9 An example of a roster update

Figure 8-6 shows the roster-related protocol that occurs when a new Conventional Node joins a conference. A single enrolling Client Application is shown running on the joining node. After the GCC Provider receives a GCC-Announce-Presence-Request and a GCC-Application-Enroll request, an update of the Conference Application Roster is issued to the Next Higher Node. In this example, the joining node is in the third layer of the connection hierarchy. It issues the update indication to the next higher node which then formats and sends an update indication to the next higher node, which, in this example, is the top node in the hierarchy. The top node assembles the full Application Roster and broadcasts the portions associated with the update to all nodes in the conference. This results in a GCC-Conference-Report and a GCC-Application-Report primitive being issued at all nodes in the conference, except the joining Conventional node. Instead, the joining node must acquire the full roster from its Parent node by issuing a RosterRefreshRequest. After the response to this request is received, a GCC-Conference-Report and a GCC-Application-Report primitive can be issued at the joining node.

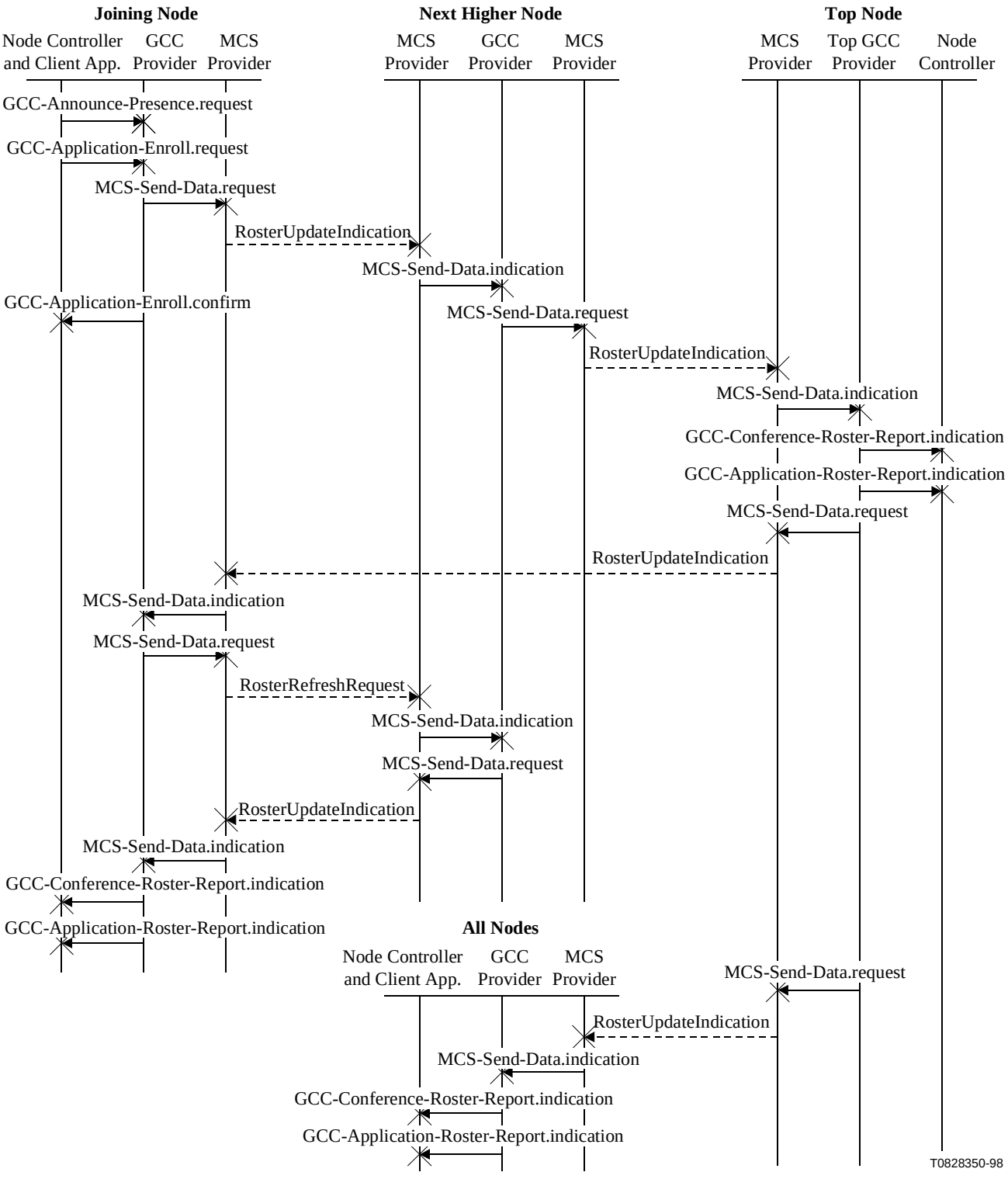

Figure 8-6/T.124 – Another example of updating the Application Roster

8.3.4 Collapsing Application Capabilities Lists

Each GCC Provider is responsible for forming, for each Application Protocol Session, the collapsed Application Capability List corresponding to the Conventional nodes below it in the connection hierarchy, as well as any Peer Application Protocol Entities enrolled at the local node (if it is Conventional). Since the nodes directly below in the connection hierarchy take care of collapsing their own capabilities, the local GCC Provider need only be aware of the collapsed Application Capability Lists from those nodes, rather than the Application Capability Lists from all lower nodes.

Given as input, these collapsed Application Capabilities Lists as well as the Application Capabilities Lists from Peer Application Protocol Entities at the local node from the Local Application Roster, the GCC Provider shall produce a collapsed Application Capabilities List by the following procedure:

- For each capability item in each list, determine the class of the capability.
- For any class, the value of the count parameter for the entry in the collapsed Application Capabilities List shall be set to the sum of the counts indicated in the corresponding entry of each of the Application Capabilities Lists in the input set.
- For the Unsigned-minimum and Unsigned-maximum classes, the new entry should also include the minimum or maximum value, respectively, of the values given for the corresponding entry of each of the Application Capabilities Lists in the input set.

8.3.5 Application and conference roster inquiry

On receipt of a GCC-Application-Roster-Inquire request primitive, a GCC Provider which supports this primitive may respond by generating a GCC-Application-Roster-Inquire confirm primitive and issuing it to the GCCSAP of the requester. The content of the confirm primitive is generated from the locally-maintained Conference Application Roster database and shall include only those entries in the roster for which the Session Key specified in the request exactly matches the Session Key of the entry over the length of the key given in the request.

On receipt of a GCC-Conference-Roster-Inquire request primitive, a GCC Provider which supports this primitive shall respond by generating a GCC-Conference-Roster-Inquire confirm primitive and issuing it to the GCCSAP of the requester. The content of the confirm primitive is generated from the local Conference Roster database accounting for all nodes known to be in the conference.

A GCC Provider at an Anonymous node that receives a roster inquiry may not be maintaining a local Conference Application Roster database since this is not an Anonymous node requirement. These type of nodes can respond to the roster inquiry either by sending back a failed confirm to the requesting GCCSAP or by sending a RosterRefreshRequest up to its Parent node to get the requested roster information. The RosterRefreshRequest will eventually result in a RosterReportIndication that includes the requested information. At this point, the GCC Provider responds by generating a GCC-Application-Roster-Inquire confirm primitive and issues it to the GCCSAP of the requester. Note that Conventional and Counted nodes are required to maintain a local Conference Application Roster database, so roster inquiries at these nodes can be processed immediately.

8.3.6 Remotely invoking an Application Protocol Entity

On receipt of a GCC-Application-Invoke request primitive via either the Control GCCSAP or an ordinary GCCSAP, a GCC Provider which supports this primitive shall broadcast an ApplicationInvokeIndication PDU to all nodes in the specified conference. This is done by issuing an MCS-Send-Data request or MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. Alternatively, the GCC Provider may issue one or more MCS-Send-Data request specifying the Node ID of one of the listed destination nodes as the Channel ID, also specifying High data priority, and including the PDU in the Data field. In this case, it may leave the Destination Node List parameter in the PDU empty. The content of the ApplicationInvokeIndication PDU is shown if Table 8-36. The GCC Provider shall then generate a GCC-Application-Invoke confirm primitive indicating if the requested operation was successful and issue it to the GCCSAP of the requester.

Table 8-36/T.124 – ApplicationInvokeIndication GCCPDU

Content	Source	Sink
Application Protocol Entity List	Request	Indication
Destination Node List or NULL	Request	Destination GCC Providers

On receipt of an ApplicationInvokeIndication PDU, a GCC Provider which supports the GCC-Application-Invoke primitive shall first determine if local node is on the list of destination nodes. If the list is either NULL, or if the local Node ID is present in the list, the GCC Provider may generate a GCC-Application-Invoke indication primitive and issue it to the Control GCCSAP. The Invoking Node parameter is filled in from the Sender User ID field in the received MCS-Send-Data indication or MCS-Uniform-Send-Data indication. If the local node is not on the list of destination nodes (if listed explicitly), no further action is taken.

8.4 The Application Registry

The Application Registry is an active database which resides at the Top GCC Provider for a conference. Any MCU node shall support the full set of registry services which operate on this database, while a terminal node may choose to support only those registry services required by Application Protocols to be supported within that terminal.

When a conference is created, the registry database, located at the Top GCC Provider, shall be initialized to a state in which all registry entries are empty and no registry entries are designated to be monitored. This shall be done prior to the time that the Top GCC Provider joins its Node ID Channel.

During operation of the registry within a conference, each non-empty registry entry contains the following information:

- The Registry Key which identifies the entry.
- The type of information contained – either a Channel ID, a Token ID, or a Parameter.
- The monitoring state – either monitoring enabled, or monitoring disabled.
- The value of the entry – the actual Channel ID, Token ID, or Parameter.
- The owner of the entry – either none, or the Node ID and Entity ID corresponding to the owning Application Protocol Entity.
- For Parameter type entries only, the modification rights of the entry – either Owner, Session, or Public.

Processing by the Top GCC Provider of multiple registry requests from a single node shall be performed strictly in the order that the requests were received. The sequence of response PDUs to a requesting node shall preserve the order of the received requests from that node.

At the requesting node, order shall also be preserved between the request primitives and the corresponding transmitted PDUs as well as between the received PDUs and the corresponding confirm primitives.

Only Conventional nodes are allowed to add, delete, or change an entry in the Registry database. The Registry is read-only for all other Node Categories. Therefore, Anonymous and Counted nodes that do not appear to be enrolled in the conference may still access the Registry database.

8.4.1 Registering a channel

On receipt of a GCC-Registry-Register-Channel request primitive, a GCC Provider at a Conventional node shall send a RegistryRegisterChannelRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryRegisterChannelRequest PDU is shown in Table 8-37. A GCC-Registry-Register-Channel request made by either a Counted or Anonymous node will be ignored by the GCC Provider.

Table 8-37/T.124 – RegistryRegisterChannelRequest GCCPDU

Content	Source	Sink
Entity ID	Source GCC Provider	Top GCC Provider
Key	Request	Top GCC Provider
Channel ID	Request	Top GCC Provider

On receipt of the RegistryRegisterChannelRequest PDU, the Top GCC Provider shall first verify that the Sender User ID in the received MCS-Send-Data indication and the Entity ID included in the PDU correspond exactly to the Node ID and Entity ID of an Application Protocol Entity at a Conventional node currently in the Application Roster. If so, it shall attempt to register the channel by creating an appropriate registry entry. First, it checks whether the registry entry corresponding to the specified Key already exists in the registry database. If the registry entry does not exist, the Top GCC Provider creates an entry for this Key and includes in the database for this entry the Node ID as indicated by the Sender User ID in the received MCS-Send-Data indication along with the Entity ID included in the PDU as the entry owner, the entry type being a Channel ID, and the value being the Channel ID specified in the PDU.

The Top GCC Provider then indicates that the channel has been properly registered by sending a RegistryResponse PDU to the requesting node by issuing an MCS-Send-Data request specifying the Node ID of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryResponse PDU is shown in Table 8-38. In the case of a successful action, the Result parameter is specified as successful. The Modification Rights field is not filled in (this field is for Parameter type entries only).

If the registry entry already existed, if the requester was invalid due to not appearing in the Application Roster, if the requesting node was not a Conventional node, or if the registry entry could not be created due to a limitation in the available resources, the registry is not modified and a RegistryResponse PDU is returned as above, but with a negative Result indicating the reason for the failure. In this case, the value of the entry prior to attempting to modify it is returned as the Registry Item in the RegistryResponse PDU.

Table 8-38/T.124 – RegistryResponse GCCPDU

Content	Source	Sink
Entity ID	Top GCC Provider	Destination GCC Provider
Primitive type (register-channel, assign-token, set-parameter, retrieve-entry, delete-entry, monitor-entry)	Top GCC Provider	Destination GCC Provider
Key	Top GCC Provider	Confirm
Registry Item	Top GCC Provider	Confirm
Owner	Top GCC Provider	Confirm
Modification Rights (optional)	Top GCC Provider	Confirm
Result	Top GCC Provider	Confirm

In the case of a successful modification of an existing registry entry, the Top GCC Provider checks to determine if the registry entry had been set to be monitored. If not, no further action is taken by the Top GCC Provider. If the entry had been set to be monitored, the Top GCC Provider sends a RegistryMonitorEntryIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryMonitorEntryIndication PDU is shown in Table 8-44.

On receipt of a RegistryResponse PDU of this type, a GCC Provider shall generate a GCC-Registry-Register-Channel confirm primitive indicating whether or not the request was successful as indicated in the Result parameter of the RegistryResponse PDU and issue it on the GCCSAP of the Application Protocol Entity indicated by the Entity ID. If the GCC Provider knows of no currently enrolled Application Protocol Entity with the corresponding Entity ID, the indication is ignored and no further action is taken.

8.4.2 Assigning a Token

On receipt of a GCC-Registry-Assign-Token request primitive, a GCC Provider at a Conventional node shall send a RegistryAssignTokenRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryAssignTokenRequest PDU is shown in Table 8-39. A GCC-Registry-Assign-Token request made by either a Counted or Anonymous node will be ignored by the GCC Provider.

Table 8-39/T.124 – RegistryAssignTokenRequest GCCPDU

Content	Source	Sink
Entity ID	Source GCC Provider	Top GCC Provider
Key	Request	Top GCC Provider

On receipt of the RegistryAssignTokenRequest PDU, the Top GCC Provider shall first verify that the Sender User ID in the received MCS-Send-Data indication and the Entity ID included in the PDU correspond exactly to the Node ID and Entity ID of an Application Protocol Entity at a Conventional node currently in the Application Roster. If so, it shall attempt to assign a token by allocating a Token ID and creating an appropriate registry entry. First, it checks whether the registry entry corresponding to the specified Key already exists in the registry database. If the registry entry does not exist, the Top GCC Provider first allocates a new Token ID from the space of dynamic

Token IDs (16 384 through 65 535). The Top GCC Provider then creates an entry for this Key and includes in the database for this entry the Node ID as indicated by the Sender User ID in the received MCS-Send-Data indication along with the Entity ID included in the PDU as the entry owner, the entry type being a Token ID, and the value being the allocated Token ID.

The Top GCC Provider then indicates that the token has been properly allocated, and returns the value of the Token ID to the requester, by sending a RegistryResponse PDU to the requesting node by issuing an MCS-Send-Data request specifying the Node ID of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryResponse PDU is shown in Table 8-38. In the case of a successful action, the Result parameter is specified as successful. The Modification Rights field is not filled in (this field is for Parameter type entries only).

If the registry entry already existed, if the requester was invalid due to not appearing in the Application Roster, if the requesting node was not a Conventional node, if the registry entry could not be created due to a limitation in the available resources, or if there are were no more available dynamic Token IDs, the registry is not modified and a RegistryResponse PDU is returned as above, but with a negative Result indicating the reason for the failure. In this case, the value of the entry prior to attempting to modify it is returned as the Registry Item in the RegistryResponse PDU.

In the case of a successful modification of an existing registry entry, the Top GCC Provider checks to determine if the registry entry had been set to be monitored. If not, no further action is taken by the Top GCC Provider. If the entry had been set to be monitored, the Top GCC Provider sends a RegistryMonitorEntryIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryMonitorEntryIndication PDU is shown in Table 8-44.

On receipt of a RegistryResponse PDU of this type, a GCC Provider shall generate a GCC-Registry-Assign-Token confirm primitive indicating whether or not the request was successful as indicated in the Result parameter of the RegistryResponse PDU, and if successful, the value of the allocated Token ID and issue it on the GCCSAP of the Application Protocol Entity indicated by the Entity ID. If the GCC Provider knows of no currently enrolled Application Protocol Entity with the corresponding Entity ID, the indication is ignored and no further action is taken.

8.4.3 Setting a Parameter

On receipt of a GCC-Registry-Set-Parameter request primitive, a GCC Provider at a Conventional node shall send a RegistrySetParameterRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistrySetParameterRequest PDU is shown in Table 8-40. A GCC-Registry-Set-Parameter request made by either a Counted or Anonymous node will be ignored by the GCC Provider.

Table 8-40/T.124 – RegistrySetParameterRequest GCCPDU

Content	Source	Sink
Entity ID	Source GCC Provider	Top GCC Provider
Key	Request	Top GCC Provider
Parameter	Request	Top GCC Provider
Modification Rights (optional)	Request	Top GCC Provider

On receipt of the RegistrySetParameterRequest PDU, the Top GCC Provider shall first verify that the Sender User ID in the received MCS-Send-Data indication and the Entity ID included in the PDU correspond exactly to the Node ID and Entity ID of an Application Protocol Entity at a Conventional node currently in the Application Roster. If so, it shall attempt to set a parameter in an existing registry entry or create a new entry with the specified parameter. First, it checks whether the registry entry corresponding to the specified Key already exists in the registry database, and if so, determines the type of the entry, the owner, and the current modification rights. If the registry entry does not exist, the Top GCC Provider then creates an entry for this Key and includes in the database for this entry the Node ID as indicated by the Sender User ID in the received MCS-Send-Data indication along with the Entity ID included in the PDU as the entry owner, the entry type being a Parameter, and the value being the Parameter value specified in the PDU. If the registry entry already existed, and if the entry was of the Parameter type, the GCC Provider shall first check if the requester has the right to modify this entry. If the owner has modification rights, the value of the registry entry is modified to reflect the type as a Parameter and the value as the Parameter value specified in the PDU. If the entry was not owned prior to being set, the entry is modified to indicate the new owner.

The determination of whether the owner has modification rights is by the following rules. If the current Modification Rights attribute of the entry is set to Owner, the requester, as indicated by the Entity ID and the Node ID from the Sender User ID field of the received MCS-Send-Data indication, must be the owner or the entry must currently be un-owned for the request to succeed. If the current Modification Rights attribute of the entry is set to Session, the requester must be part of the same Application Protocol Session for this request to succeed. This is determined by finding the entry in the current Application Roster for the requester and examining the Session Key. The Session Key must be identical to that of the owner. Again, if the entry is not owned, this restriction does not apply. If the current Modification Rights attribute of the entry is set to Public, there are no restrictions on the requester for the request to succeed other than the node being a Conventional node.

When a Parameter entry is first created, the state of Modification Rights for the entry is determined. If the Modification Rights parameter was included in the request PDU which resulted in the creation of the entry, the Modification Type is set to the indicated value. If not, the value Public is assumed for Modification Rights. On requests to set the parameter, if the Modification Rights entry is not present, the Modification Rights state is not changed. If it is present, the GCC Provider shall change the Modification Rights state to the state indicated in the PDU only if the requester is the current owner of the entry, or if the entry is currently un-owned. Otherwise, the Modification Rights state is not changed. In the latter case, the set-parameter operation shall still proceed as normal, and the result shall not be effected.

In either of the above cases, the Top GCC Provider then indicates that the parameter has been properly set by sending a RegistryResponse PDU to the requesting node by issuing an MCS-Send-Data request specifying the Node ID of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryResponse PDU is shown in Table 8-38. In the case of a successful action, the Result parameter is specified as successful. The Modification Rights parameter shall be included in the response PDU.

If the registry entry could not be created due to a limitation in the available resources, if the requesting node was not a Conventional node, if the entry already existed but the requester did not have modification rights, if the requester was invalid due to not appearing in the Application Roster, or if the entry was of the Parameter type, the registry is not modified and a RegistryResponse PDU is returned as above, but with a negative Result indicating the reason for the failure. In this case, the value of the entry prior to attempting to modify it is returned as the Registry Item in the RegistryResponse PDU. In the case of insufficient modification rights, the result parameter shall be: belongs to other.

In the case of a successful modification of an existing registry entry, the Top GCC Provider checks to determine if the registry entry had been set to be monitored. If not, no further action is taken by the Top GCC Provider. If the entry had been set to be monitored, the Top GCC Provider sends a RegistryMonitorEntryIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryMonitorEntryIndication PDU is shown in Table 8-44.

On receipt of a RegistryResponse PDU of this type, a GCC Provider shall generate a GCC-Registry-Set-Parameter confirm primitive indicating whether or not the request was successful as indicated in the Result parameter of the RegistryResponse PDU and issue it on the GCCSAP of the Application Protocol Entity indicated by the Entity ID. If the GCC Provider knows of no currently enrolled Application Protocol Entity with the corresponding Entity ID, the indication is ignored and no further action is taken.

8.4.4 Retrieving an Entry

On receipt of a GCC-Registry-Retrieve-Entry request primitive, a GCC Provider shall send a RegistryRetrieveEntryRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryRetrieveEntryRequest PDU is shown in Table 8-41. A RegistryRetrieveEntryRequest can be issued by any node regardless of Node Category.

Table 8-41/T.124 – RegistryRetrieveEntryRequest GCCPDU

Content	Source	Sink
Entity ID	Source GCC Provider	Top GCC Provider
Key	Request	Top GCC Provider

On receipt of the RegistryRetrieveEntryRequest PDU, the Top GCC Provider shall examine the contents of the registry entry specified by the Key. The registry entry may be in one of four possible states: empty, containing a Channel ID, a Token ID, or a Parameter. For any of these cases, the Top GCC Provider returns the state of the entry to the requester by sending a RegistryResponse PDU to the requesting node by issuing an MCS-Send-Data request specifying the Node ID of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryResponse PDU is shown in Table 8-38. The Registry Item parameter contains the state of the entry, and in the case of a non-empty state, the value that the entry currently contains. The Result parameter is specified as successful if the entry is non-empty, and entry-not-found if the entry is empty. If the entry is indicated to be of the Parameter type, the Modification Rights parameter shall be included in the response PDU.

On receipt of a RegistryResponse PDU of this type, a GCC Provider shall generate a GCC-Registry-Retrieve-Entry confirm primitive indicating the registry item included in the response PDU and issue it on the GCCSAP of the Application Protocol Entity indicated by the Entity ID. If the GCC Provider knows of no currently enrolled Application Protocol Entity with the corresponding Entity ID, the indication is ignored and no further action is taken.

8.4.5 Deleting an Entry

On receipt of a GCC-Registry-Delete-Entry request primitive, a GCC Provider at a Conventional node shall send a RegistryDeleteEntryRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel

ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryDeleteEntryRequest PDU is shown in Table 8-42. A GCC-Registry-Delete-Entry request made by either a Counted or Anonymous node will be ignored by the GCC Provider.

Table 8-42/T.124 – RegistryDeleteEntryRequest GCCPDU

Content	Source	Sink
Entity ID	Source GCC Provider	Top GCC Provider
Key	Request	Top GCC Provider

On receipt of the RegistryDeleteEntryRequest PDU, the Top GCC Provider shall attempt to delete the designated registry entry. First, it checks whether the registry entry corresponding to the specified Key exists in the registry database, and if so, determines the owner. If the registry entry exists and is owned by the requester (both the Node ID and Entity ID of the owner and those of the requester are identical) or if the entry is not currently owned and the requesting node is a Conventional node, the Top GCC Provider deletes the contents of the entry setting the state to empty and not monitored.

If the registry entry is successfully deleted, or if the entry already did not exist, the Top GCC Provider then indicates that the entry has been properly deleted by sending a RegistryResponse PDU to the requesting node by issuing an MCS-Send-Data request specifying the Node ID of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryResponse PDU is shown in Table 8-38. In the case of a successful action, the Result parameter is specified as successful, and the registry item is specified to be empty.

If the registry entry already existed but was owned by another node, or if the requesting node is not a Conventional node, the registry is not modified and a RegistryResponse PDU is returned as above, but with a negative Result indicating the reason for the failure. In this case, the registry item is set to the value of the entry prior to the deletion attempt.

In the case of a successful deletion of an existing registry entry, the Top GCC Provider checks to determine if the registry entry had been set to be monitored (prior to deletion). If not, no further action is taken by the Top GCC Provider. If the entry had been set to be monitored, the Top GCC Provider sends a RegistryMonitorEntryIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryMonitorEntryIndication PDU is shown in Table 8-44.

In the case of a successful deletion of a Token type registry entry, the Top GCC Provider may de-allocate the corresponding token. That is, the Top GCC Provider may later re-use the Token ID which had been included in the deleted entry in response to a RegistryAssignTokenRequest.

On receipt of a RegistryResponse PDU of this type, a GCC Provider shall generate a GCC-Registry-Delete-Entry confirm primitive indicating whether or not the request was successful as indicated in the Result parameter of the RegistryResponse PDU and issue it on the GCCSAP of the Application Protocol Entity indicated by the Entity ID. If the GCC Provider knows of no currently enrolled Application Protocol Entity with the corresponding Entity ID, the indication is ignored and no further action is taken.

8.4.6 Monitoring an Entry

On receipt of a GCC-Registry-Monitor request primitive with the Enable/Disable flag set to Enable, a GCC Provider at a Conventional node shall send a RegistryMonitorEntryRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data

field. The content of the RegistryMonitorEntryRequest PDU is shown in Table 8-43. A local record that the requesting Application Protocol Entity had enabled monitoring this entry is made by the GCC Provider to determine whether to generate GCC-Registry-Monitor indications on later receipt of RegistryMonitorEntryIndication PDU corresponding to this entry. A GCC-Registry-Monitor request made by either a Counted or Anonymous node will be ignored by the GCC Provider.

On receipt of a GCC-Registry-Monitor request primitive with the Enable/Disable flag set to Disable, a local record that the requesting Application Protocol Entity had disabled monitoring this entry is made by the GCC Provider to determine whether to generate GCC-Registry-Monitor indications on later receipt of RegistryMonitorEntryIndication PDU corresponding to this entry.

Table 8-43/T.124 – RegistryMonitorEntryRequest GCCPDU

Content	Source	Sink
Entity ID	Source GCC Provider	Top GCC Provider
Key	Request	Top GCC Provider

On receipt of the RegistryMonitorEntryRequest PDU, the Top GCC Provider shall attempt to change the monitoring state of the registry entry if the requesting node is a Conventional node. First, it checks whether the registry entry corresponding to the specified Key already exists in the registry database. If the registry entry exists, the monitoring state for the specified entry is set to enabled.

NOTE 1 – Once monitoring is enabled, monitor indications will continue to be broadcast whenever changes are made to the entry for the duration of the conference, or until the entry is deleted.

Once the appropriate action has been taken, the Top GCC Provider sends a RegistryResponse PDU to the requesting node by issuing an MCS-Send-Data request specifying the Node ID of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryResponse PDU is shown in Table 8-38. In the case of a successful action, the Result parameter is specified as successful. If the requested entry did not exist, the Result parameter is specified as entry-not-found. If the entry is indicated to be of the Parameter type, the Modification Rights parameter shall be included in the response PDU.

If the registry entry did not exist, the registry is not modified and a RegistryResponse PDU is returned as above, but with a negative Result indicating the reason for the failure.

On receipt of a RegistryResponse PDU of this type, a GCC Provider shall generate a GCC-Registry-Monitor confirm primitive indicating whether or not the request was successful as indicated in the Result parameter of the RegistryResponse PDU and issue it on the GCCSAP of the Application Protocol Entity indicated by the Entity ID. If the GCC Provider knows of no currently enrolled Application Protocol Entity with the corresponding Entity ID, the indication is ignored and no further action is taken.

While any registry entry is set to be monitored, any change in the content of the registry entry, including setting a parameter, deletion of the entry, change in ownership, or change in the Modification Rights state, cause the Top GCC Provider to send a RegistryMonitorEntryIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryMonitorEntryIndication PDU is shown in Table 8-44.

Table 8-44/T.124 – RegistryMonitorEntryIndication GCCPDU

Content	Source	Sink
Key	Top GCC Provider	Indication
Registry Item	Top GCC Provider	Indication
Owner	Top GCC Provider	Confirm
Modification Rights (optional)	Top GCC Provider	Confirm

On receipt of a RegistryMonitorEntryIndication PDU, a GCC Provider for a node which supports the registry monitoring function shall generate a GCC-Registry-Monitor indication primitive and send this primitive to any GCCSAP which had previously issued a GCC-Registry-Monitor request specifying the same Key as is indicated in the indication PDU with the Enable/Disable flag set to Enable (without having more recently received one with this flag set to Disable). If there have not been any requesters for the particular Key at this node, the indication PDU is ignored by the GCC Provider. It is a function of each local GCC Provider to keep a local database of requesting GCCSAPs which is updated whenever a GCC-Registry-Monitor request is received.

NOTE 2 – As a local matter, a particular GCC Provider implementation may choose not to keep track of which Application Protocol Entities have enabled or disabled monitoring for each entry and instead issue GCC-Registry-Monitor indications to the GCCSAPs corresponding to all enrolled Application Protocol Entities when a RegistryMonitorEntryIndication PDU is received.

8.4.7 Allocation of Unique Handles

On receipt of a GCC-Registry-Allocate-Handle request primitive, a GCC Provider shall send a RegistryAllocateHandleRequest PDU to the Top GCC Provider by issuing an MCS-Send-Data request specifying the Node ID Channel of the Top GCC Provider as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryAllocateHandleRequest PDU is shown in Table 8-45.

Table 8-45/T.124 – RegistryAllocateHandleRequest GCCPDU

Content	Source	Sink
Entity ID	Source GCC Provider	Top GCC Provider
Number of Handles	Request	Top GCC Provider

On receipt of the RegistryAllocateHandleRequest PDU, the Top GCC Provider shall generate a unique set of handles of the number requested. This shall be done by incrementing a 32-bit state variable after each allocation modulo 2^{32}, using the incremented value for allocation of the next requested handle. If the full set of possible handles has been allocated, the request is rejected indicating that no handles are available. If a sufficient number of handles are available, the handles are returned to the requester by specifying the first allocated handle along with the number of allocated handles. The Top GCC Provider returns the result by sending a RegistryAllocateHandleResponse PDU to the requester. This is done by issuing an MCS-Send-Data request specifying the Node ID of the requester as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the RegistryAllocateHandleResponse PDU is shown in Table 8-46. If the result was successful, the registry item parameter contains the list of handles.

Table 8-46/T.124 – RegistryAllocateHandleResponse GCCPDU

Content	Source	Sink
Entity ID	Top GCC Provider	Destination GCC Provider
Number of Handles	Top GCC Provider	Confirm
First Handle	Top GCC Provider	Confirm
Result	Top GCC Provider	Confirm

On receipt of a RegistryAllocateHandleResponse indication, a GCC Provider shall generate a GCC-Registry-Allocate-Handle confirm primitive and issue it to the GCCSAP of the Application Protocol Entity indicated by the Entity ID. If the GCC Provider knows of no currently enrolled Application Protocol Entity with the corresponding Entity ID, the indication is ignored and no further action is taken.

8.4.8 Changes in ownership and Registry Clean-up

Whenever the Application Roster is updated, the Top GCC Provider shall determine if any Application Protocol Entities were removed from the roster. If so, it shall examine the entire registry database to determine if the disconnecting Application Protocol Entities were the owner of any registry entries. If so, these entries are modified to indicate that there is no current owner.

If the result of a change in the Application Roster indicates that all Application Protocol Entities in a session have become un-enrolled, the Top GCC Provider shall examine all registry entries to determine if the Session Key portion of the Registry Key matches the Session Key of the former session. For entries that do correspond to the removed session, the Top GCC Provider shall delete the registry entry. If that registry entry corresponded to a Token type entry, the Top GCC Provider may de-allocate the Token to allow for its later reuse.

8.5 Conference conductorship

Conference conductorship is controlled through the use of the Conference-Conductorship-Token. When this token is free the conference is in Non-Conducted Mode. When this token is grabbed the conference is in Conducted Mode, and the grabber of the token is the Conference Conductor. The token can only be grabbed by Conventional nodes. Anonymous and Counted nodes can never act as the conductor of a conference.

Upon acquiring the Conference-Conductorship-Token, the Conference Conductor (unless it is the Top GCC Provider itself) shall demonstrate possession of the token to the Top GCC Provider by offering to give it away, invoking MCS-Token-Give request and specifying the Top GCC Provider as recipient. The Top GCC Provider shall respond negatively to the resulting MCS-Token-Give indication, leaving the token in possession of the Conference Conductor. If the token is offered again later by the current Conference Conductor, the Top GCC Provider shall interpret it as part of a GCC-Conductor-Give operation and need not refuse automatically.

8.5.1 Grabbing conductorship

On receipt of a GCC-Conductor-Assign request primitive, a GCC Provider at a Conventional node shall first examine the Conference Profile to determine if the conference is conductible or not. If not, it shall generate a GCC-Conductor-Assign confirm primitive with an unsuccessful result indicating that the conference is non-conductible. If the conference is conductible, the GCC Provider shall then examine its local database to determine if it is currently the conference conductor. If so, it shall generate a GCC-Conductor-Assign confirm primitive with a successful result and issue it to the Control GCCSAP. If not, it shall attempt to grab the conductor token by issuing the MCS primitive

MCS-Token-Grab request, specifying the Conference-Conductorship-Token as the token ID to be grabbed. A GCC-Conductor-Assign request made by either a Counted or Anonymous node will be ignored by the GCC Provider.

On receipt of the MCS-Token-Grab confirm primitive, the GCC Provider shall examine the Result parameter. If the Result is successful, the GCC Provider shall then issue the MCS primitive MCS-Token-Give request, specifying the Conference-Conductorship-Token as the token ID to be offered and the Node ID of the Top GCC Provider as the User ID to receive the token. On receipt of the MCS-Token-Give confirm primitive, the GCC Provider shall examine the Result parameter. If the Result is unsuccessful, the GCC Provider shall locally generate the GCC-Conductor-Assign confirm primitive indicating a successful result and issue it to the Control GCCSAP.

On receipt of an MCS-Token-Give indication specifying the Conference-Conductorship-Token as the token ID offered, from a donor User ID that the Top GCC Provider does not recognize as the current Conference Conductor, the Top GCC Provider shall update its records to recognize the donor as Conference Conductor. It shall also send a ConductorAssignIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The content of the ConductorAssignIndication PDU is shown in Table 8-47.

If the Result parameter of the MCS-Token-Grab confirm indicates an unsuccessful result or the Result parameter of the MCS-Token-Give confirm indicates a successful result, the GCC Provider shall generate a GCC-Conductor-Assign confirm primitive indicating a negative result which reflects the unsatisfactory result and issue it to the Control GCCSAP.

Table 8-47/T.124 – ConductorAssignIndication GCCPDU

Content	Source	Sink
Conducting Node	Top GCC Provider	Destination GCC Provider

8.5.2 Releasing conductorship

On receipt of a GCC-Conductor-Release request primitive, a GCC Provider shall first examine its local database to determine if it is currently the conference conductor. If not, it shall generate a GCC-Conductor-Release confirm primitive indicating a negative result and issue it to the Control GCCSAP. If it is currently the conductor, it shall first send a ConductorReleaseIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The content of the ConductorReleaseIndication PDU is shown in Table 8-48 (there are no parameters in this PDU). It shall then release the conductor token by issuing the MCS primitive MCS-Token-Release request, specifying the Conference-Conductorship-Token as the token ID to be released.

On receipt of the MCS-Token-Release confirm primitive, the GCC Provider shall locally generate the GCC-Conductor-Release confirm primitive indicating a successful result and issue it to the Control GCCSAP.

Table 8-48/T.124 – ConductorReleaseIndication GCCPDU

Content	Source	Sink
-- No parameters --		

8.5.3 Conductor Assignment and Release Indications

A node which supports Application Protocols that are specified to behave differently in Conducted mode and Non-conducted mode shall respond to receipt of ConductorAssignIndication and ConductorReleaseIndication PDUs. A node which does not support such Application Protocols may choose to ignore these indications.

On receipt of a ConductorAssignIndication PDU as part of an MCS-Uniform-Send-Data indication, a GCC Provider shall examine the Conference Profile to determine if the conference is conductible or not. If not, it shall ignore this PDU. If the conference is conductible, it may generate a GCC-Conductor-Assign indication primitive and issue it to the Control GCCSAP as well as the GCCSAP of all enrolled Application Protocol Entities.

For Conventional nodes, the GCC Provider shall first determine if the Sender User ID field of the MCS-Uniform-Send-Data indication indicates that the ConductorAssignIndication PDU was transmitted by the Top GCC Provider. If not, the PDU shall be ignored, and no indication primitive shall be generated. If the PDU was transmitted by the Top GCC Provider, the Node ID parameter of the primitive is set to the value indicated by the Conducting Node parameter in the received PDU. For any node that supports conductorship primitives, the GCC Provider shall also store the fact that the conference is now in conducted mode, as well as the Node ID of the conductor, in its local database.

For Counted and Anonymous nodes, a ConductorAssignIndication can be received from either the Parent node or the Top GCC Provider. If the PDU is received from a node other than one of these, it shall be ignored, and no indication of the primitive shall be generated. If the PDU was transmitted by either the Parent node or the Top GCC Provider, the Node ID parameter of the primitive is set to the value indicated by the Conducting Node parameter in the received PDU. A ConductorAssignIndication received from the Top GCC Provider always overrides one received from a Parent node. Therefore, once a ConductorAssignIndication is received from the Top GCC Provider, any ConductorAssignIndication received from a Parent node shall be ignored. For a node which supports conductorship primitives and receives a valid ConductorAssignIndication, the GCC Provider shall store in its local database the fact that the conference is now in conducted mode, as well as the Node ID of the conductor.

On receipt of a ConductorReleaseIndication PDU as part of an MCS-Uniform-Send-Data indication, a GCC Provider may generate a GCC-Conductor-Release indication primitive and issue it to the Control GCCSAP as well as to the GCCSAP of all enrolled Application Protocol Entities.

For Conventional nodes, the GCC Provider shall first determine if the Sender User ID field of the MCS-Uniform-Send-Data indication indicates that the ConductorReleaseIndication PDU was transmitted by the Top GCC Provider or by the node that it currently records as the conductor. If not, the PDU shall be ignored, and no indication primitive shall be generated. For a node that supports conductorship primitives, the GCC Provider shall also store in its local database the fact that the conference is no longer in conducted mode. The GCC Provider shall also set its locally-stored permission flag to indicate that no conducted-mode permissions have been granted.

For Counted and Anonymous nodes, a ConductorReleaseIndication can be received from either the Parent node, the Top GCC Provider, or the current Conductor. If the PDU is received from a node other than one of these, it shall be ignored, and no indication of the primitive shall be generated. If a Counted or Anonymous node receives a ConductorReleaseIndication on the GCC-Conventional-Broadcast-Channel and has yet to determine the current state of Conductorship within the conference (because it has not received an indication from its Parent), the node must assume that the ConductorReleaseIndication came from a valid node and should store in its local database the fact that the conference is not in conducted mode. Any ConductorReleaseIndication received on the

GCC-Conventional-Broadcast-Channel shall override any ConductorReleaseIndication received from the Parent node. For a node that supports conductorship primitives and receives a valid ConductorReleaseIndication, the GCC Provider shall store in its local database the fact that the conference is not in conducted mode.

A GCC Provider shall preserve the order of received ConductorAssignIndication and ConductorReleaseIndication PDUs received on either the GCC-Broadcast-Channel or the GCC-Conventional-Broadcast-Channel in generation of the corresponding indication primitives.

The GCC Provider in a node which supports the conductorship primitives shall issue a GCC-Conductor-Assign or GCC-Conductor-Release indication to the GCCSAP associated with an Application Protocol Entity which newly enrolls after a conference has already been established. The current conductorship state as known by the GCC Provider determines which of these two primitives to issue.

8.5.4 Asking to be given conductorship

On receipt of a GCC-Conductor-Please request primitive, a GCC Provider at a Conventional node shall first examine its local database to determine if the conference is in conducted mode. If not, it shall generate a GCC-Conductor-Please confirm with a negative result indicating that the conference is not in conducted mode and issue it to the Control GCCSAP. If in conducted mode, the GCC Provider issues an MCS-Token-Please request specifying the Conference-Conductorship-Token as the token requested. It also shall generate a GCC-Conductor-Please confirm indicating a successful result and issue it to the Control GCCSAP.

On receipt of an MCS-Token-Please indication specifying the Conference-Conductorship-Token as the Token ID, a GCC Provider shall generate a GCC-Conductor-Please indication primitive indicating the Node ID of the requesting node obtained from the User ID parameter received in the MCS-Token-Please indication and issue it to the Control GCCSAP. A GCC-Conductor-Please request made by either a Counted or Anonymous node will be ignored by the GCC Provider.

8.5.5 Giving conductorship

On receipt of a GCC-Conductor-Give request primitive, a GCC Provider shall first examine its local database to determine if it is the current conference conductor. If not, it shall generate a GCC-Conductor-Give confirm with a negative result indicating that the node is not currently the conductor and issue it to the Control GCCSAP. Otherwise, the GCC Provider shall attempt to give conductorship to the specified node by issuing an MCS-Token-Give request specifying the Conference-Conductorship-Token as the token ID, and the recipient node as specified in the GCC-Conductor-Give request. The node that is being given Conductorship should always be a Conventional node.

On receipt of the MCS-Token-Give confirm primitive, the GCC Provider shall examine the Result parameter. If the Result is successful, the GCC Provider shall locally generate the GCC-Conductor-Give confirm primitive indicating a successful result and issue it to the Control GCCSAP. If the Result parameter of the MCS-Token-Give confirm indicates an unsuccessful result, the GCC Provider shall generate a GCC-Conductor-Give confirm primitive indicating a negative result which reflects the result indicated by the MCS-Token-Grab confirm and issue it to the Control GCCSAP.

On receipt of the MCS-Token-Give indication primitive specifying the Conference-Conductorship-Token, a GCC Provider other than the Top GCC Provider shall generate a GCC-Conductor-Give indication primitive and issue it to the Control GCCSAP. On receipt of the GCC-Conductor-Give response, the GCC Provider shall examine the result parameter. If the result is successful, the GCC Provider shall issue the MCS-Token-Give response indicating that the token was accepted.

If the GCC-Conductor-Give response indicates that the token is not to be accepted, the GCC Provider shall issue the MCS-Token-Give response indicating that the token was user-rejected.

If a node which does not support conductorship primitives receives an MCS-Token-Give indication, the GCC Provider shall respond with an MCS-Token-Give response indicating a result of user-rejected. This includes Anonymous and Counted nodes.

On receipt of an MCS-Token-Give indication primitive specifying the Conference-Conductorship-Token from a donor User ID that the Top GCC Provider does not record as the current conductor, the Top GCC Provider shall respond as specified in 8.5.1, automatically rejecting the token. If it already recognizes the donor as conductor, however, the Top GCC Provider shall generate a GCC-Conductor-Give indication primitive as specified above, like any other GCC Provider.

8.5.6 Getting conductorship status

On receipt of a GCC-Conductor-Inquire request, a GCC Provider shall first examine its own local database to determine if the conference is currently in conducted mode. If not, the GCC Provider shall generate a GCC-Conductor-Inquire confirm which indicates that the conference is not in conducted mode and issue it to the Control GCCSAP. Otherwise, it shall generate a confirm indicating that the conference is in conducted mode, and shall specify the Node ID of the conducting node as stored in its local database as the current conductor. In this case, if a node has just entered a conference and has not yet received a ConductorAssignIndication PDU from the conductor (and does not yet have the conductor's Node ID in its local database), the GCC Provider shall wait until this PDU is received before generating the GCC-Conductor-Inquire confirm.

If the conference is in conducted mode, the GCC Provider shall determine the setting of the Permission Flag to be included in the GCC-Conductor-Inquire confirm by determining whether the local node is the current conductor (in which case permission is assumed to be granted), or, if not, whether the Node ID of the local node had been listed in the Permission List of most recent ConductorPermissionGrantIndication PDU which had been received since the last transition from non-conducted to conducted mode. If the local node does appear in this list, this flag is set to TRUE. If not, or if there had not been a ConductorPermissionGrantIndication PDU received since the most recent transition from non-conducted to conducted mode, then this flag is set to FALSE.

8.5.7 Conductorship announcement when new nodes enter a conference

If the Top GCC Provider receives an update to the Conference Roster with the flag set indicating that new Conventional nodes are present in the conference and the conference is currently in conducted mode, the Top GCC Provider shall send a ConductorAssignIndication PDU to all nodes in the conference by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The content of the ConductorAssignIndication PDU is shown in Table 8-47. The Conducting Node parameter indicates the Node ID of the current conductor.

If the Top GCC Provider receives an update to the Conference Roster with the flag set indicating that new Conventional nodes are present in the conference and the conference is not currently in conducted mode, the Top GCC Provider shall send a ConductorReleaseIndication PDU to all nodes by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying Top data priority, and including the PDU in the Data field. The content of the ConductorReleaseIndication PDU is shown in Table 8-48.

If the Top GCC Provider receives a ConductorReleaseIndication from another node, which signals a change from the current conductorship state, prior to receiving its own transmitted

ConductorAssignIndication, the Top GCC Provider shall re-issue a ConductorReleaseIndication to reflect the new state.

If the Top GCC Provider receives a RosterUpdateIndication PDU indicating that new Counted or Anonymous nodes are present in the conference, the Top GCC Provider does not broadcast an initial conductorship announcement by sending either a ConductorAssignIndication or a ConductorReleaseIndication. Instead, the initial conductorship announcement is left up to the joining node's Parent node. This initial announcement is generated as follows: when a Parent node receives the Node ID of a newly joining Anonymous or Counted node, it must send the new node either a ConductorAssignIndication or a ConductorReleaseIndication to inform it of the current state of conductorship within the conference. If the conference is currently in conducted mode, the Parent node shall send a ConductorAssignIndication PDU directly to the new Counted or Anonymous node by issuing an MCS-Send-Data request specifying the Node ID of the new node as the Channel ID, specifying High data priority, and including the PDU in the Data field. If the conference is not currently in conducted mode, the Parent node shall send a ConductorReleaseIndication PDU directly to the new Counted or Anonymous node by issuing an MCS-Send-Data request specifying the Node ID of the new node as the Channel ID, specifying High data priority, and including the PDU in the Data field.

A node which has just become joined to a conference shall delay processing of any of the conductor related request primitives until receiving either a ConductorAssignIndication or a ConductorReleaseIndication. This is because all conductor related request primitives require that the local node check its local database of conductorship status which is not in harmony with the actual status of the conference until one of these PDUs is received.

8.5.8 Unexpected disconnection of the conductor

If the conductor of a conference unexpectedly disconnects from a conference, it may not have an opportunity to send a ConductorReleaseIndication PDU to all nodes indicating that the conference is now in non-conducted mode. The GCC Provider at each node (if it supports the conductorship primitives) shall monitor received MCS-Detach-User indications and compare the User ID in the indication to the Node ID of the node that it currently believes to be the conductor. If they match, the GCC Provider shall store in its local database that the conference is no longer in conducted mode, and the GCC Provider shall generate a GCC-Conductor-Release indication and issue it to the Control GCCSAP as well as to the GCCSAP of all enrolled Application Protocol Entities. The GCC Provider shall also set its locally-stored permission flag to indicate that no conducted-mode permissions have been granted.

8.5.9 Asking to be given conducted-mode permission

On receipt of a GCC-Conductor-Permission-Ask request primitive via the Control GCCSAP, a GCC Provider which supports this primitive shall first determine if the specified conference is in conducted mode. If not, it shall generate a GCC-Conductor-Permission-Ask confirm primitive and issue it to the Control GCCSAP indicating not-in-conducted-mode as the result. If in conducted mode, it shall send a ConductorPermissionAskIndication PDU to the conductor of the specified conference by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the data field. The content of the ConductorAskIndication PDU is shown in Table 8-49. When successfully completed, the GCC Provider shall generate a GCC-Conductor-Permission-Ask confirm primitive and issue it to the Control GCCSAP indicating success as the result.

Table 8-49/T.124 – ConductorPermissionAskIndication GCCPDU

Content	Source	Sink
Grant/Release Flag	Request	Indication

On receipt of a ConductorPermissionAskIndication PDU, a GCC Provider shall determine if it is currently the conference conductor. If not, it shall ignore the PDU and take no further action. If so, if the GCC Provider supports the corresponding primitive, it shall generate a GCC-Conductor-Permission-Ask indication primitive and issue it to the Control GCCSAP. The Node ID of the Requester parameter shall be filled in with the Sender User ID parameter from the received MCS-Uniform-Send-Data indication. A GCC Provider shall preserve the order of received ConductorPermissionAskIndication PDUs in generating the corresponding indication primitives.

8.5.10 Granting conducted-mode permission

On receipt of a GCC-Conductor-Permission-Grant request via the Control GCCSAP, a GCC Provider which supports this primitive shall first determine if it is currently the conductor of the specified conference. If not, it shall generate a GCC-Conductor-Permission-Grant confirm indicating not-conductor as the reason for rejection, and issue it to the Control GCCSAP. It shall then take no further action. If the node is the current conference conductor, the GCC Provider shall broadcast a ConductorPermissionGrantIndication PDU to all nodes in the conference. It shall do this by issuing an MCS-Uniform-Send-Data request specifying the GCC-Broadcast-Channel as the Channel ID, specifying Top data priority, and including the PDU in the data field. The content of the ConductorPermissionGrantIndication PDU is shown in Table 8-50. The permission and waiting list parameters are filled in with the list of Node IDs provided in the corresponding parameters of the request primitive. The order of both lists is preserved. When successfully completed, the GCC Provider shall generate a GCC-Conductor-Permission-Grant confirm primitive and issue it to the Control GCCSAP indicating success as the result.

Table 8-50/T.124 – ConductorPermissionGrantIndication GCCPDU

Content	Source	Sink
Permission List	Request	Indication
Waiting List (optional)	Request	Indication

On receipt of a ConductorPermissionGrantIndication PDU, a GCC Provider which supports the corresponding primitive shall first determine if the conference is currently in conducted mode. If not, it shall ignore the received PDU and take no further action. If the conference is in conducted mode, it shall determine if this PDU was received from the node which is the current conference conductor. If not, it shall also ignore this PDU and take no further action. If the source is the current conductor, it shall generate a GCC-Conductor-Permission-Grant indication primitive and issue it to the Control GCCSAP as well as to the GCCSAPs of all enrolled Application Protocol Entities. The Permission and Waiting Lists in the indication primitive are filled in directly from the corresponding parameters of the PDU, preserving the order of these lists. The GCC Provider shall examine the Permission List parameter and determine if the local node is included in this list. If so, it shall set the Permission Flag parameter of the indication primitive to TRUE. Otherwise, it shall set this flag to FALSE. A GCC Provider shall preserve the order of received ConductorPermissionGrantIndication PDUs in generating the corresponding indication primitives.

8.6 Miscellaneous functions

8.6.1 Timed conferences

On receipt of a GCC-Conference-Time-Remaining request primitive, a GCC Provider shall send a ConferenceTimeRemainingIndication PDU to all nodes in the specified conference by issuing an MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTimeRemainingIndication PDU is shown in Table 8-51. The value of the Time Remaining parameter is obtained from the contents of the request primitive. The GCC Provider shall also generate a GCC-Conference-Time-Remaining confirm primitive indicating a successful result, and issue it to the Control GCCSAP.

Table 8-51/T.124 – ConferenceTimeRemainingIndication GCCPDU

Content	Source	Sink
Time Remaining	Request	Indication
Node Identifier (optional)	Request	Indication

On receipt of a ConferenceTimeRemainingIndication PDU, a GCC Provider may optionally generate a GCC-Conference-Time-Remaining indication primitive and issue it to the Control GCCSAP. The Time Remaining parameter is obtained from the contents of the PDU. The Source Node parameter is filled in from the Sender User ID in the received MCS-Uniform-Send-Data indication.

On receipt of a GCC-Conference-Time-Inquire request primitive, a GCC Provider shall send a ConferenceTimeInquireIndication PDU to the Conference Convener by issuing an MCS-Send-Data request specifying the GCC-Convener-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTimeInquireIndication PDU is shown in Table 8-52. The GCC Provider shall then generate a GCC-Conference-Time-Inquire confirm primitive and issue it to the Control GCCSAP.

Table 8-52/T.124 – ConferenceTimeInquireIndication GCCPDU

Content	Source	Sink
Node-Specific Time Flag	Request	Indication

On receipt of a ConferenceTimeInquireIndication PDU, the GCC Provider (of the Conference Convener) may optionally generate a GCC-Conference-Time-Inquire indication primitive and issue it to the Control GCCSAP. The Node ID of the requester, as indicated in the Source Node parameter of the MCS-Send-Data indication, is used as the Node ID of the Requesting Node parameter in the GCC-Conference-Time-Inquire indication primitive.

On receipt of a GCC-Conference-Extend request primitive, a GCC Provider shall send a ConferenceTimeExtendIndication PDU to the Conference Convener by issuing an MCS-Send-Data request specifying the GCC-Convener-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceTimeExtendIndication PDU is shown in Table 8-53. The GCC Provider shall then generate a GCC-Conference-Extend confirm primitive and issue it to the Control GCCSAP.

Table 8-53/T.124 – ConferenceTimeExtendIndication GCCPDU

Content	Source	Sink
Time to Extend	Request	Indication
Node-Specific Time Flag	Request	Indication

On receipt of a ConferenceTimeExtendIndication PDU, the GCC Provider (of the Conference Convener) may optionally generate a GCC-Conference-Extend indication primitive and issue it to the Control GCCSAP. The Node ID of the requester, as indicated in the Source Node parameter of the MCS-Send-Data indication, is used as the Node ID of the Requesting Node parameter in the GCC-Conference-Extend indication primitive.

8.6.2 Requesting conference assistance

On receipt of a GCC-Conference-Assistance request primitive, a GCC Provider shall send a ConferenceAssistanceIndication PDU to all the nodes in the specified conference by issuing an MCS-Send-Data or MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the ConferenceAssistanceIndication PDU is shown in Table 8-54. The GCC Provider shall then generate a GCC-Conference-Assistance confirm primitive indicating whether the operation was successful.

Table 8-54/T124 – ConferenceAssistanceIndication GCCPDU

Content	Source	Sink
User Data (optional)	Request	Indication

On receipt of the ConferenceAssistanceIndication PDU, a node which supports this function may generate a GCC-Conference-Assistance indication primitive and issue it to the Control GCCSAP. The Source Node parameter is obtained from the Sender User ID in the received MCS-Uniform-Send-Data indication.

8.6.3 Broadcasting a text message

On receipt of a GCC-Text-Message request primitive, if no Destination Node parameter was included in the request, a GCC Provider shall send a TextMessageIndication PDU to the all nodes in the specified conference by issuing either an MCS-Send-Data request or MCS-Uniform-Send-Data request specifying both the GCC-Broadcast-Channel (to support older protocol nodes) and the GCC-Conventional-Broadcast-Channel as the Channel ID, specifying High data priority, and including the PDU in the Data field. If a Destination Node was indicated, the GCC Provider shall send the same PDU to that node by issuing an MCU-Send-Data request specifying the Node ID of the requested node as the Channel ID, specifying High data priority, and including the PDU in the Data field. The content of the TextMessageIndication PDU is shown in Table 8-55. The GCC Provider shall then generate a GCC-Text-Message confirm primitive indicating whether the operation was successful.

Table 8-55/T.124 – TextMessageIndication GCCPDU

Content	Source	Sink
Message	Request	Indication

On receipt of the TextMessageIndication PDU, a node which supports this function may generate a GCC-Text-Message indication primitive and issue it to the Control GCCSAP. The Source Node parameter is obtained from the Sender User ID in the received MCS-Uniform-Send-Data indication.

8.7 GCCPDU definitions

The structure of GCCPDUs is specified as follows using the notation ASN.1 of Recommendation X.680. All GCCPDUs shall be encoded for transmission by applying the Packed Encoding Rules of Recommendation X.691 using the Basic Aligned variant.

NOTE – The use of Automatic Tags in the GCC protocol definition implies that the order of SEQUENCE and CHOICE structures contained within this definition effects to the actual encoded values.

```
GCC-PROTOCOL DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

-- Export all symbols

-- ================================================================================
-- Part 1:  Elements of which messages are composed
-- ================================================================================

ChannelID               ::= INTEGER (1..65535)
StaticChannelID         ::= INTEGER (1..1000)           -- Those assigned by specifications
DynamicChannelID        ::= INTEGER (1001..65535)       -- Those created and deleted by MCS
UserID                  ::= DynamicChannelID

TokenID                 ::= INTEGER (1..65535)
StaticTokenID           ::= INTEGER (1..16383)          -- Those assigned by specifications
DynamicTokenID          ::= INTEGER (16384..65535)      -- Those assigned by the registry

Time                    ::= INTEGER (-2147483648..2147483647)  -- Time in seconds
Handle                  ::= INTEGER (0..4294967295)             -- 32-bit value

H221NonStandardIdentifier ::= OCTET STRING (SIZE (4..255))
                                                        -- First four octets shall be country code and
                                                        -- Manufacturer code, assigned as specified in
                                                        -- Annex A/H.221 for NS-cap and NS-comm

Key ::= CHOICE                                          -- Identifier of a standard or non-standard object
{
        object                  OBJECT IDENTIFIER,
        h221NonStandard         H221NonStandardIdentifier
}

NonStandardParameter ::= SEQUENCE
{
        key             Key,
        data            OCTET STRING
}

TextString ::= BMPString (SIZE (0..255))        -- Basic Multilingual Plane of ISO/IEC 10646-1 (Unicode)

simpleTextFirstCharacter UniversalString ::= {0, 0, 0, 0}

simpleTextLastCharacter UniversalString ::= {0, 0, 0, 255}

SimpleTextString ::= BMPString (SIZE (0..255)) (FROM (simpleTextFirstCharacter..simpleTextLastCharacter))
```

```
SimpleNumericString ::= NumericString (SIZE (1..255)) (FROM ("0123456789"))

DiallingString ::= NumericString (SIZE (1..16)) (FROM ("0123456789"))

SubAddressString ::= NumericString (SIZE (1..40)) (FROM ("0123456789"))

ExtraDiallingString ::= TextString (SIZE (1..255)) (FROM ("0123456789#*,"))

UserData ::= SET OF SEQUENCE
{
        key                     Key,
        value                   OCTET STRING OPTIONAL
}

Password ::= SEQUENCE
{
        numeric                 SimpleNumericString,
        text                    SimpleTextString OPTIONAL,
        ...
}

PasswordSelector ::= CHOICE
{
        numeric                 SimpleNumericString,
        text                    SimpleTextString,
        ...
}

ChallengeResponseItem ::= CHOICE
{
        passwordString          PasswordSelector,
        responseData            UserData,
        ...
}

ChallengeResponseAlgorithm ::= CHOICE
{
        passwordInTheClear      NULL,
        nonStandardAlgorithm    NonStandardParameter,
        ...
}

ChallengeItem ::= SEQUENCE
{
        responseAlgorithm       ChallengeResponseAlgorithm,
        challengeData           UserData,
        ...
}

ChallengeRequest ::= SEQUENCE
{
        challengeTag                    INTEGER,
        challengeSet                    SET OF ChallengeItem,  -- Set of algorithms offered for response
        ...
}
```

```
ChallengeResponse ::= SEQUENCE
{
        challengeTag                    INTEGER,
        responseAlgorithm               ChallengeResponseAlgorithm,
                                                -- Specific algorithm selected from the set of
                                                -- items presented in the ChallengeRequest
        responseItem                    ChallengeResponseItem,
        ...
}

PasswordChallengeRequestResponse ::= CHOICE
{
        passwordInTheClear              PasswordSelector,
      challengeRequestResponse          SEQUENCE
        {
                challengeRequest        ChallengeRequest OPTIONAL,
                challengeResponse       ChallengeResponse OPTIONAL,
                ...
        },
        ...
}

ConferenceName ::= SEQUENCE
{
        numeric         SimpleNumericString,
        text            SimpleTextString OPTIONAL,
        ...
}

ConferenceNameSelector ::= CHOICE
{
        numeric         SimpleNumericString,
        text            SimpleTextString,
        ...
}

ConferenceNameModifier ::= SimpleNumericString

Privilege       ::= ENUMERATED
{
        terminate       (0),
        ejectUser       (1),
        add             (2),
        lockUnlock      (3),
        transfer        (4),
        ...
}

TerminationMethod ::= ENUMERATED
{
        automatic       (0),
        manual          (1),
        ...
}

ConferencePriorityScheme ::= CHOICE
{
        nonStandardScheme               nonStandardParameter,
        …
}
```

```
ConferencePriority ::= SEQUENCE
{
        priority                INTEGER (0..65535),
        scheme                  ConferencePriorityScheme,
        …
}

NodeCategory ::= CHOICE
{
        conventional            NULL,
        counted                 NULL,
        anonymous               NULL,
        nonStandardCategory     nonStandardParameter,
        …
}

ConferenceMode ::= CHOICE
{
        conventional-only       NULL,
        counted-only            NULL,
        anonymous-only          NULL,
        conventional-control    NULL,
        unrestricted-mode       NULL,
        non-standard-mode       nonStandardParameter,
        …
}


NetworkAddress  ::=  SEQUENCE (SIZE (1..64)) OF CHOICE      -- Listed in order of use
{
        aggregatedChannel       SEQUENCE
        {
                transferModes   SEQUENCE                    -- One or more
                {
                        speech          BOOLEAN,
                        voice-band      BOOLEAN,
                        digital-56k     BOOLEAN,
                        digital-64k     BOOLEAN,
                        digital-128k    BOOLEAN,
                        digital-192k    BOOLEAN,
                        digital-256k    BOOLEAN,
                        digital-320k    BOOLEAN,
                        digital-384k    BOOLEAN,
                        digital-512k    BOOLEAN,
                        digital-768k    BOOLEAN,
                        digital-1152k   BOOLEAN,
                        digital-1472k   BOOLEAN,
                        digital-1536k   BOOLEAN,
                        digital-1920k   BOOLEAN,
                        packet-mode     BOOLEAN,
                        frame-mode      BOOLEAN,
                        atm             BOOLEAN,
                        ...
                },

                internationalNumber     DiallingString,
                subAddress              SubAddressString OPTIONAL,
                extraDialling           ExtraDiallingString OPTIONAL,
```

```
            highLayerCompatibility      SEQUENCE
            {
                telephony3kHz           BOOLEAN,
                telephony7kHz           BOOLEAN,
                videotelephony          BOOLEAN,
                videoconference         BOOLEAN,
                audiographic            BOOLEAN,
                audiovisual             BOOLEAN,
                multimedia              BOOLEAN,
                ...
            } OPTIONAL,
            ...
        },

        transportConnection             SEQUENCE
        {
            nsapAddress                 OCTET STRING (SIZE (1..20)),
            transportSelector           OCTET STRING OPTIONAL
        },

        nonStandard                     NonStandardParameter,
        ...
}

MediaList :: = SEQUENCE {
        audio  BOOLEAN,
        video  BOOLEAN,
        data        BOOLEAN,
        ...
}


ChannelAggregationMethod ::= CHOICE {
        h221            NULL,
        h244            NULL,
        iso-iec-13871 NULL,     -- The actual mode of bonding is dynamically selected according
                                -- to the procedures described in ISO/IEC 13871.
        nonStandard     NonStandardParameter,
        ...
}


Profile ::= CHOICE {
        simpleProfile               CHOICE {
            -- Basic transfer modes:
            speech                  NULL,       -- Simple telephony
            telephony-3kHz          NULL,       -- Rec. G.711
            telephony-7kHz          NULL,       -- Rec. G.722
            voice-band              NULL,       -- Modems
            frameRelay,             NULL,
            -- T.120-only data profiles (Rec. T.123):
            t123-pstn-basic         NULL,
            t123-psdn-basic         NULL,
            t123-b-isdn-basic       NULL,
        },
        multimediaProfile           SEQUENCE {
            profile                 CHOICE {
                h310                NULL,
                h320                NULL,
                h321                NULL,
                h322                NULL,
```

```
                h323                NULL,
                h324                NULL,
                h324m               NULL,
                asvd                NULL,
                dsvd                NULL,
            },
            t120Data                BOOLEAN
        },
        dsmccDownloadProfile        NULL,
        nonStandard                 NonStandardParameter,
        ...
}


ExtendedE164NetworkAddress ::= SEQUENCE {
        internationalNumber         DiallingString,
        subAddress                  SubAddressString OPTIONAL,
        extraDialling               ExtraDiallingString OPTIONAL,
        ...
}

TransportAddress ::= SEQUENCE {
        nsapAddress                 OCTET STRING (SIZE (1..20)),
        transportSelector           OCTET STRING OPTIONAL
},


GSTNConnection ::= SEQUENCE {
        networkAddress    ExtendedE164NetworkAddress,
        ...
}


ISDNConnection ::= SEQUENCE {
        circuitTypes        SET OF CHOICE {
            digital-64k             NULL,
            digital-2x64k           NULL,
            digital-384k            NULL,
            digital-1536            NULL,
            digital-1920k           NULL,
            multirate-base-64k INTEGER (1..30)   -- See Note 1
        },
        networkAddress    ExtendedE164NetworkAddress,
        highLayerCompatibility      SEQUENCE {    -- Those are supported code points for IE HLC of the D
                                                  -- protocol (Rec. Q.931).
            telephony3kHz           BOOLEAN,
            telephony7kHz           BOOLEAN,
            videotelephony          BOOLEAN,
            videoconference         BOOLEAN,
            audiographic            BOOLEAN,
            audiovisual             BOOLEAN,
            multimedia              BOOLEAN,
            ...
        } OPTIONAL,
        ...
}

-- Note 1:    digital-2x64k differs from multirate-base-64k with a multiplier value of 2; in the first case the
--            network is requested an 8 kHz integrity with Restricted Differential Time Delay (RDTD); in the
--            second case the network is requested a Time Slot Sequence integrity (see 4.5.5/Q.931)
```

```
CSDNConnection ::= SEQUENCE {
        circuitTypes        SET OF CHOICE {
                digital-56k     NULL,
                digital-64k     NULL
        },
        networkAddress      ExtendedE164NetworkAddress,
        ...
}


PSDNConnection ::= SEQUENCE {
        networkAddress      CHOICE {
                extendedE164NetworkAddress      ExtendedE164NetworkAddress,
                transportAddress                TransportAddress,
                nonStandard                     NonStandardParameter
        },
        ...
}

ATMConnection ::= SEQUENCE {
        networkAddress          CHOICE {
                extendedE164    ExtendedE164NetworkAddress,
                nsapAddress     TransportAddress,       -- this case is reserved for NSAPs only: the
                                                        -- optional transport selector shall never be used
                nonStandard     NonStandardParameter
        },
        maxTransferRate         INTEGER (0..MAX) OPTIONAL,      -- in cells per seconds
        ...
}


NetworkConnection ::= CHOICE {
        gstnConnection                  GSTNConnection,
        isdnConnection                  ISDNConnection,
        csdnConnection                  CSDNConnection,
        psdnConnection                  PSDNConnection,
        atmConnection                   ATMConnection,
        extendedE164NetworkAddress      ExtendedE164NetworkAddress,  -- Note: LAN connections and leased
        transportAddress                TransportAddress,            -- lines (Rec. G.703/G.704) may be
        nonStandard                     NonStandardParameter,        -- covered by one of these
        ...
}

NetworkAddressV2 ::= SET OF SEQUENCE {
        networkConnectionCHOICE {
                singleConnection                NetworkConnection,
                aggregatedConnections           SEQUENCE {
                        connectionList                  SET (SIZE(1..30)) OF CHOICE {
                                isdnConnection          ISDNConnection,
                                csdnConnection          CSDNConnection,
                                ...
                        },
                        aggregationMethods              SET OF ChannelAggregationMethod OPTIONAL,
                        ...
                }
        }
        profiles                SET OF Profile OPTIONAL,
        mediaConcerned          MediaList OPTIONAL,
        ...
}
```

```
NodeType ::= ENUMERATED
{
        terminal                (0),
        multiportTerminal       (1),
        mcu                     (2),
        ...
}

NodeProperties ::= SEQUENCE
{
        managementDevice        BOOLEAN,        -- Is the node a device such as a reservation system
        peripheralDevice        BOOLEAN,        -- Is the node a peripheral to a primary node
        ...
}

AsymmetryIndicator ::= CHOICE
{
        callingNode     NULL,
        calledNode      NULL,
        unknown         INTEGER (0..4294967295)     -- Uniformly distributed 32-bit random number
}

AlternativeNodeID ::= CHOICE
{
        h243NodeID              OCTET STRING (SIZE (2)),
        ...
}

ConferenceDescriptor    ::= SEQUENCE
{
        conferenceName                  ConferenceName,
        conferenceNameModifier          ConferenceNameModifier OPTIONAL,
        conferenceDescription           TextString OPTIONAL,
        lockedConference                BOOLEAN,
        passwordInTheClearRequired      BOOLEAN,
        networkAddress                  NetworkAddress OPTIONAL,
        ...,
        defaultConferenceFlag           BOOLEAN,
        conferenceMode                  ConferenceMode
}

NodeRecord ::= SEQUENCE
{
        superiorNode            UserID OPTIONAL,        -- Not present only for the Top GCC Provider
        nodeType                NodeType,
        nodeProperties          NodeProperties,
        nodeName                TextString OPTIONAL,
        participantsList        SEQUENCE OF TextString OPTIONAL,
        siteInformation         TextString OPTIONAL,
        networkAddress          NetworkAddress OPTIONAL,
        alternativeNodeID       AlternativeNodeID OPTIONAL,
        userData                UserData OPTIONAL,
        ...,
        nodeCategory            NodeCategory OPTIONAL,
        networkAddressV2        NetworkAddressV2 OPTIONAL}
```

```
SessionKey ::= SEQUENCE
{
        applicationProtocolKey          Key,
        sessionID                       ChannelID OPTIONAL
}

ChannelType ::= ENUMERATED
{
        static                  (0),
        dynamicMulticast        (1),
        dynamicPrivate          (2),
        dynamicUserId           (3)
}

ApplicationRecord ::= SEQUENCE
{
        applicationActive               BOOLEAN,        -- Active/Inactive flag
        conductingOperationCapable      BOOLEAN,        -- Maximum one per node per session
        startupChannel                  ChannelType OPTIONAL,
        applicationUserID               UserID OPTIONAL, -- User ID assigned to the Application Protocol Entity
        nonCollapsingCapabilities       SET OF SEQUENCE
        {
                capabilityID            CapabilityID,
                applicationData         OCTET STRING OPTIONAL
        } OPTIONAL,
        ...
}

CapabilityID ::= CHOICE
{
        standard                        INTEGER (0..65535),     -- Assigned by Application Protocol specifications
        nonStandard                     Key
}

CapabilityClass ::= CHOICE
{
        logical                         NULL,
        unsignedMin                     INTEGER (0..MAX),       -- Capability value
        unsignedMax                     INTEGER (0..MAX),       -- Capability value
        ...
}

EntityID ::= INTEGER (0..65535)

ApplicationInvokeSpecifier ::= SEQUENCE
{
        sessionKey                      SessionKey,
        expectedCapabilitySet           SET OF SEQUENCE
                                        {
                                                capabilityID            CapabilityID,
                                                capabilityClass         CapabilityClass,
                                                ...
                                        } OPTIONAL,
        startupChannel                  ChannelType OPTIONAL,
        mandatoryFlag                   BOOLEAN,        -- TRUE indicates required Application Protocol Entity
        ...
}
```

```
RegistryKey ::= SEQUENCE
{
        sessionKey          SessionKey,
        resourceID          OCTET STRING (SIZE (0..64))
}

RegistryItem ::= CHOICE
{
        channelID           DynamicChannelID,
        tokenID             DynamicTokenID,
        parameter           OCTET STRING (SIZE (0..64)),
        vacant              NULL,
        ...
}

RegistryEntryOwner ::= CHOICE
{
        owned               SEQUENCE
        {
                nodeID      UserID,                 -- Node ID of the owning node
                entityID    EntityID                -- Entity ID of the owning
        },                                          -- Appliction Protocol Entity
        notOwned            NULL                    -- There is no current owner
}

RegistryModificationRights ::= ENUMERATED
{
        owner       (0),
        session     (1),
        public      (2)
}

-- ================================================================================
-- Part 2:  PDU Messages
-- ================================================================================

UserIDIndication ::= SEQUENCE
{
        tag                                 INTEGER,
        ...
}

ConferenceCreateRequest         ::= SEQUENCE
{                                                   -- MCS-Connect-Provider request user data
        conferenceName                  ConferenceName,
        convenerPassword                Password OPTIONAL,
        password                        Password OPTIONAL,
        lockedConference                BOOLEAN,
        listedConference                BOOLEAN,
        conductibleConference           BOOLEAN,
        terminationMethod               TerminationMethod,
        conductorPrivileges             SET OF Privilege OPTIONAL,
        conductedPrivileges             SET OF Privilege OPTIONAL,
        nonConductedPrivileges          SET OF Privilege OPTIONAL,
        conferenceDescription           TextString OPTIONAL,
        callerIdentifier                TextString OPTIONAL,
        userData                        UserData OPTIONAL,
        ...,
        conferencePriority              ConferencePriority OPTIONAL,
        conferenceMode                  ConferenceMode OPTIONAL
}
```

```
ConferenceCreateResponse ::= SEQUENCE
{                                                       -- MCS-Connect-Provider response user data
        nodeID                          UserID,         -- Node ID of the sending node
        tag                             INTEGER,
        result                          ENUMERATED
        {
                success                                 (0),
                userRejected                            (1),
                resourcesNotAvailable                   (2),
                rejectedForSymmetryBreaking             (3),
                lockedConferenceNotSupported            (4),
                ...
        },
        userData                        UserData OPTIONAL,
        ...
}

ConferenceQueryRequest ::= SEQUENCE
{                                                       -- MCS-Connect-Provider request user data
        nodeType                        NodeType,
        asymmetryIndicator              AsymmetryIndicator OPTIONAL,
        userData                        UserData OPTIONAL,
        ...
}

ConferenceQueryResponse ::= SEQUENCE
{                                                       -- MCS-Connect-Provider response user data
        nodeType                        NodeType,
        asymmetryIndicator              AsymmetryIndicator OPTIONAL,
        conferenceList                  SET OF ConferenceDescriptor,
        result                          ENUMERATED
        {
                success                 (0),
                userRejected            (1),
                ...
        },
        userData                        UserData OPTIONAL,
        ...,
        waitForInvitationFlag           BOOLEAN OPTIONAL,
        noUnlistedConferenceFlag        BOOLEAN OPTIONAL
}

ConferenceJoinRequest ::= SEQUENCE
{                                                       -- MCS-Connect-Provider request user data as well as
                                                        -- MCS-Send-Data on Node ID Channel of Top GCC sent
                                                        -- by the receiver of the MCS-Connect-Provider
        conferenceName                  ConferenceNameSelector OPTIONAL,
                                                        -- Required when part of MCS-Connect-Provider
        conferenceNameModifier          ConferenceNameModifier OPTIONAL,
        tag                             INTEGER OPTIONAL,   -- Filled in when sent on Node ID Channel
                                                            -- of Top GCC
        password                        PasswordChallengeRequestResponse OPTIONAL,
        convenerPassword                PasswordSelector OPTIONAL,
        callerIdentifier                TextString OPTIONAL,
        userData                        UserData OPTIONAL,
        ...,
        nodeCategory                    NodeCategory OPTIONAL
}
```

```
ConferenceJoinResponse  ::= SEQUENCE
{                                                   -- MCS-Connect-Provider response user data as well as
                                                    -- MCS-Send-Data on Node ID Channel of
                                                    -- the receiver of the MCS-Connect-Provider
    nodeID                      UserID OPTIONAL,    -- Node ID of directly connected node only
    topNodeID                   UserID,             -- Node ID of Top GCC Provider
    tag                         INTEGER,
    conferenceNameAlias         ConferenceNameSelector OPTIONAL,
    passwordInTheClearRequired  BOOLEAN,
    lockedConference            BOOLEAN,
    listedConference            BOOLEAN,
    conductibleConference       BOOLEAN,
    terminationMethod           TerminationMethod,
    conductorPrivileges         SET OF Privilege OPTIONAL,  -- No privilege shall be listed more than
                                                            -- once
    conductedPrivileges         SET OF Privilege OPTIONAL,  -- No privilege shall be listed more than
                                                            -- once
    nonConductedPrivileges      SET OF Privilege OPTIONAL,  -- No privilege shall be listed more than
                                                            -- once
    conferenceDescription       TextString OPTIONAL,
    password                    PasswordChallengeRequestResponse OPTIONAL,
    result                      ENUMERATED
    {
        success                     (0),
        userRejected                (1),
        invalidConference           (2),
        invalidPassword             (3),
        invalidConvenerPassword     (4),
        challengeResponseRequired   (5),
        invalidChallengeResponse    (6),
        ...
    },
    userData                    UserData OPTIONAL,
    ...,
    nodeCategory                NodeCategory OPTIONAL,
    conferenceMode              ConferenceMode OPTIONAL
}

ConferenceInviteRequest  ::= SEQUENCE
{                                                   -- MCS-Connect-Provider request user data
    conferenceName              ConferenceName,
    nodeID                      UserID,             -- Node ID of the sending node
    topNodeID                   UserID,             -- Node ID of Top GCC Provider
    tag                         INTEGER,
    passwordInTheClearRequired  BOOLEAN,
    lockedConference            BOOLEAN,
    listedConference            BOOLEAN,
    conductibleConference       BOOLEAN,
    terminationMethod           TerminationMethod,
    conductorPrivileges         SET OF Privilege OPTIONAL,  -- No privilege shall be listed more than
                                                            -- once
    conductedPrivileges         SET OF Privilege OPTIONAL,  -- No privilege shall be listed more than
                                                            -- once
    nonConductedPrivileges      SET OF Privilege OPTIONAL,  -- No privilege shall be listed more than
                                                            -- once
    conferenceDescription       TextString OPTIONAL,
    callerIdentifier            TextString OPTIONAL,
    userData                    UserData OPTIONAL,
    ...,
    conferencePriority          ConferencePriority OPTIONAL,
    nodeCategory                NodeCategory OPTIONAL,
```

```
        conferenceMode              ConferenceMode OPTIONAL
}

ConferenceInviteResponse ::= SEQUENCE
{                                                   -- MCS-Connect-Provider response user data
        result                      ENUMERATED
        {
                success             (0),
                userRejected        (1),
                ...
        },
        userData                    UserData OPTIONAL,
        ...
}

ConferenceAddRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC or
                                                    -- Node ID Channel of Adding MCU if specified
        networkAddress      NetworkAddress,
        requestingNode      UserID,
        tag                 INTEGER,
        addingMCU           UserID OPTIONAL,
        userData            UserData OPTIONAL,
        ...,
        nodeCategory        NodeCategory OPTIONAL,
        networkAddressV2    NetworkAddressV2
}

ConferenceAddResponse  ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of requester

        tag                 INTEGER,
        result              ENUMERATED
        {
                success                 (0),
                invalidRequester        (1),
                invalidNetworkType      (2),
                invalidNetworkAddress   (3),
                addedNodeBusy           (4),
                networkBusy             (5),
                noPortsAvailable        (6),
                connectionUnsuccessful  (7),
                ...
        },
        userData            UserData OPTIONAL,
        ...
}

ConferenceLockRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
        -- No parameters
        ...
}

ConferenceLockResponse ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of requester
        result      ENUMERATED
        {
                success             (0),
                invalidRequester    (1),
```

```
                alreadyLocked           (2),
                ...
        },
        ...
}

ConferenceLockIndication        ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
                                                    -- or MCS-Send-Data on Node ID Channel
        -- No parameters
        ...
}

ConferenceUnlockRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
        -- No parameters
        ...
}

ConferenceUnlockResponse ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of requester
        result                  ENUMERATED
        {
                success                 (0),
                invalidRequester        (1),
                alreadyUnlocked         (2),
                ...
        },
        ...
}

ConferenceUnlockIndication ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
                                                    -- or MCS-Send-Data on Node ID Channel
        -- No parameters
        ...
}

ConferenceTerminateRequest      ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
        reason                  ENUMERATED
        {
                userInitiated                   (0),
                timedConferenceTermination      (1),
                ...
        },
        ...
}

ConferenceTerminateResponse ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of requester
        result                  ENUMERATED
        {
                success                 (0),
                invalidRequester        (1),
                ...
        },
        ...
}
```

```
ConferenceTerminateIndication ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
        reason                  ENUMERATED
        {
                userInitiated                   (0),
                timedConferenceTermination      (1),
                ...
        },
        ...
}

ConferenceEjectUserRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
        nodeToEject             UserID,             -- Node ID of the node to eject
        reason                  ENUMERATED
        {
                userInitiated           (0),
                ...
        },
        ...
}

ConferenceEjectUserResponse    ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of requester
        nodeToEject             UserID,             -- Node ID of the node to eject
        result                  ENUMERATED
        {
                success                 (0),
                invalidRequester        (1),
                invalidNode             (2),
                ...
        },
        ...
}

ConferenceEjectUserIndication   ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
        nodeToEject             UserID,             -- Node ID of the node to eject
        reason                  ENUMERATED
        {
                userInitiated                   (0),
                higherNodeDisconnected          (1),
                higherNodeEjected               (2),
                ...
        },
        ...
}

ConferenceTransferRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
        conferenceName          ConferenceNameSelector, -- Name of conference to transfer to
        conferenceNameModifier  ConferenceNameModifier OPTIONAL,
        networkAddress          NetworkAddress OPTIONAL,
        transferringNodes       SET (SIZE (1..65536)) OF UserID OPTIONAL,
        password                PasswordSelector OPTIONAL,
        ...,
        networkAddressV2        NetworkAddressV2 OPTIONAL

}
```

```
ConferenceTransferResponse      ::= SEQUENCE
{                                                       -- MCS-Send-Data on Node ID Channel of requester
        conferenceName          ConferenceNameSelector, -- Name of conference to transfer to
        conferenceNameModifier  ConferenceNameModifier OPTIONAL,
        transferringNodes       SET (SIZE (1..65536)) OF UserID OPTIONAL,
        result                  ENUMERATED
        {
                success                 (0),
                invalidRequester        (1),
                ...
        },
        ...
}

ConferenceTransferIndication    ::= SEQUENCE
{                                                       -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
        conferenceName          ConferenceNameSelector, -- Name of conference to transfer to
        conferenceNameModifier  ConferenceNameModifier OPTIONAL,
        networkAddress          NetworkAddress OPTIONAL,
        transferringNodes       SET (SIZE (1..65536)) OF UserID OPTIONAL,
                                                        -- List of Node IDs,
                                                        -- not present if destined for all nodes
        password                PasswordSelector OPTIONAL,
        ...,
        networkAddressV2        NetworkAddressV2 OPTIONAL}

RosterUpdateIndication ::= SEQUENCE
{                                                       -- MCS-Send-Data on Node ID Channel or
                                                        -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
        fullRefresh             BOOLEAN,                -- Conference Roster and all
                                                        -- ApplicationProtocol Sessions refreshed
        nodeInformation         SEQUENCE
        {
                nodeRecordList  CHOICE
                {
                        noChange        NULL,
                        refresh         SET (SIZE (1..65536)) OF SEQUENCE
                                                        -- One for each node in the conference;
                                                        -- no node shall be listed more than once
                        {
                                nodeID          UserID,         -- Node ID of the node
                                nodeRecord      NodeRecord
                        },
                        update          SET (SIZE (1..65536)) OF SEQUENCE
                                                        -- One for each node changing its node record;
                                                        -- no node shall be listed more than once
                        {
                                nodeID          UserID,         -- Node ID of the node
                                nodeUpdate      CHOICE
                                {
                                    addRecord       NodeRecord,
                                    replaceRecord   NodeRecord,
                                    removeRecord    NULL,
                                    ...
                                }
                        },
                        ...
                },
                rosterInstanceNumber            INTEGER (0..65535),
                nodesAdded                      BOOLEAN,  -- Nodes have been added since last instance
```

```
        nodesRemoved                BOOLEAN,  -- Nodes have been removed since last instance
        ...
    } ,
    applicationInformation    SET (SIZE (0..65535)) OF SEQUENCE
                                                -- One for each Application Protocol Session;
                                                -- all Application Protocol Sessions if full refresh;
                                                -- no Application Protocol Session shall be
                                                -- listed more than once
    {
        sessionKey                  SessionKey,
        applicationRecordList       CHOICE
        {
            noChange        NULL,
            refresh         SET (SIZE (0..65535)) OF SEQUENCE
                                                -- One for each node with the
                                                -- Application Protocol Session enrolled;
                                                -- no node shall be listed more than once
            {
                nodeID          UserID,         -- Node ID of node
                entityID        EntityID,       -- ID for this Application Protocol Entity at this node
                applicationRecord ApplicationRecord
            },
            update          SET (SIZE (1..65536)) OF SEQUENCE
                                                -- One for each node modifying its Application Record;
                                                -- no node shall be listed more than once
            {
                nodeID          UserID,         -- Node ID of node
                entityID        EntityID,       -- ID for this Application Protocol Entity at this node
                applicationUpdate CHOICE
                {
                    addRecord       ApplicationRecord,
                    replaceRecord   ApplicationRecord,
                    removeRecord    NULL,
                    ...
                }
            },
            ...
        },
        applicationCapabilitiesList     CHOICE
        {
            noChange                NULL,
            refresh                 SET OF SEQUENCE
                                    {
                                        capabilityID        CapabilityID,
                                        capabilityClass             CapabilityClass,
                                        numberOfEntities            INTEGER (1..65536),
                                                -- Number of Application Protocol Entities
                                                --  which issued the capability
                                        ...
                                    },
            ...
        },
        rosterInstanceNumber    INTEGER (0..65535),
        peerEntitiesAdded       BOOLEAN,        -- Peer Entities have been added since last instance
        peerEntitiesRemoved     BOOLEAN,        -- Peer Entities have been removed since last instance
        ...
    },
    ...
}
```

```
ApplicationInvokeIndication ::= SEQUENCE
{                                                   -- MCS-Send-Data or MCS-Uniform-Send-Data
                                                    -- on GCC-Broadcast-Channel or Node ID Channel
    applicationProtocolEntiyList    SET (SIZE (1..65536)) OF ApplicationInvokeSpecifier,
    destinationNodes                SET (SIZE (1..65536)) OF UserID OPTIONAL,
                                                    -- List of Node IDs,
                                                    -- not present if destined for all nodes
    ...
}

RegistryRegisterChannelRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
    entityID                    EntityID,
    key                         RegistryKey,
    channelID                   DynamicChannelID,
    ...
}

RegistryAssignTokenRequest    ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
    entityID                    EntityID,
    key                         RegistryKey,
    ...
}

RegistrySetParameterRequest   ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
    entityID                    EntityID,
    key                         RegistryKey,
    parameter                   OCTET STRING (SIZE (0..64)),
    modificationRights          RegistryModificationRights OPTIONAL,
    ...
}

RegistryRetrieveEntryRequest  ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
    entityID                    EntityID,
    key                         RegistryKey,
    ...
}

RegistryDeleteEntryRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
    entityID                    EntityID,
    key                         RegistryKey,
    ...
}

RegistryMonitorEntryRequest ::= SEQUENCE
{                                                   -- MCS-Send-Data on Node ID Channel of Top GCC
    entityID                    EntityID,
    key                         RegistryKey,
    ...
}

RegistryMonitorEntryIndication ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
    key                         RegistryKey,
    item                        RegistryItem,       -- Contents: channel, token, parameter, or empty
    owner                       RegistryEntryOwner,
```

```
        modificationRights      RegistryModificationRights OPTIONAL,
        ...
}

RegistryAllocateHandleRequest ::= SEQUENCE
{                                                       -- MCS-Send-Data on Node ID Channel of Top GCC
        entityID                EntityID,
        numberOfHandles         INTEGER (1..1024),
        ...
}

RegistryAllocateHandleResponse ::= SEQUENCE
{                                                       -- MCS-Send-Data on Node ID Channel of requester
        entityID                EntityID,
        numberOfHandles         INTEGER (1..1024),
        firstHandle             Handle,
        result                  ENUMERATED
        {
                successful              (0),
                noHandlesAvailable      (1),
                ...
        },
        ...
}

RegistryResponse ::= SEQUENCE
{                                                       -- MCS-Send-Data on Node ID Channel of requester
        entityID                EntityID,               -- Entity ID of the requesting Application Protocol Entity
        primitiveType           ENUMERATED
        {
                registerChannel         (0),
                assignToken             (1),
                setParameter            (2),
                retrieveEntry           (3),
                deleteEntry             (4),
                monitorEntry            (5),
                ...
        },
        key                     RegistryKey,            -- Database index
        item                    RegistryItem,           -- Contents: channel, token, parameter, or vacant
        owner                   RegistryEntryOwner,
        modificationRights      RegistryModificationRights OPTIONAL,
        result                  ENUMERATED
        {
                successful              (0),
                belongsToOther          (1),
                tooManyEntries          (2),
                inconsistentType        (3),
                entryNotFound           (4),
                entryAlreadyExists      (5),
                invalidRequester        (6),
                ...
        },
        ...
}

ConductorAssignIndication ::= SEQUENCE
{                                                       -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
        conductingNode          UserID,
        ...
}
```

```
ConductorReleaseIndication ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
    -- No parameters
    ...
}

ConductorPermissionAskIndication ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
    grantFlag               BOOLEAN,                -- TRUE to request permission grant, FALSE to release
    ...
}

ConductorPermissionGrantIndication ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
    permissionList          SEQUENCE (SIZE (0..65535)) OF UserID,
                                                    -- Node ID of nodes granted permission
    waitingList             SEQUENCE (SIZE (1..65536)) OF UserID OPTIONAL,
                                                    -- Node ID of nodes waiting form permission
    ...
}

ConferenceTimeRemainingIndication   ::= SEQUENCE
{                                                   -- MCS-Send-Data on GCC-Broadcast-Channel
    timeRemaining           Time,
    nodeID                  UserID OPTIONAL,
    ...
}

ConferenceTimeInquireIndication ::= SEQUENCE
{                                                   -- MCS-Send-Data on GCC-Convener-Channel
    nodeSpecificTimeFlag    BOOLEAN,                -- FALSE for conference-wide, TRUE for node-specific
    ...
}

ConferenceTimeExtendIndication ::= SEQUENCE
{                                                   -- MCS-Send-Data on GCC-Convener-Channel
    timeToExtend            Time,
    nodeSpecificTimeFlag    BOOLEAN,                -- FALSE for conference-wide, TRUE for node-specific
    ...
}

ConferenceAssistanceIndication ::= SEQUENCE
{                                                   -- MCS-Uniform-Send-Data on GCC-Broadcast-Channel
    userData            UserData OPTIONAL,
    ...
}

TextMessageIndication ::= SEQUENCE
{                                                   -- MCS-Send-Data or MCS-Uniform-Send-Data
    message             TextString,                 -- on GCC-Broadcast-Channel or Node ID Channel
    ...
}

RosterRefreshRequest ::= SEQUENCE
{
    nodeID                      UserID,
    nodeCategory                NodeCategory,
    fullRefresh                 BOOLEAN,
    sendConferenceRoster        BOOLEAN   OPTIONAL,
```

```
        applicationList                 SEQUENCE
        {
                applicationKeyList      SET OF SEQUENCE
                {
                        applicationProtocolKey  Key,
                        nonStandardParameter    NonStandardParameter OPTIONAL,
                        …
                },
                nonStandardParameter            NonStandardParameter OPTIONAL,
                …
        }       OPTIONAL,
        sessionList             SEQUENCE
        {
                sessionKeyList          SET OF SEQUENCE
                {
                        sessionKey              SessionKey,
                        nonStandardParameter    NonStandardParameter OPTIONAL,
                        …
                },
                nonStandardParameter            NonStandardParameter OPTIONAL,
                …
        }       OPTIONAL,
        nonStandardParameter            NonStandardParameter OPTIONAL,
        …
}

FunctionNotSupportedResponse ::= SEQUENCE
{
        request         RequestPDU
}

NonStandardPDU ::= SEQUENCE
{
        data            NonStandardParameter,
        ...
}


-- ========================================================================
-- Part 3:  Messages sent as MCS-Connect-Provider user data
-- ========================================================================

ConnectData ::= SEQUENCE
{
        t124Identifier  Key,        -- This shall be set to the value {itu-t recommendation t 124 version(0) 1}
        connectPDU      OCTET STRING
}

ConnectGCCPDU   ::= CHOICE
{
        conferenceCreateRequest         ConferenceCreateRequest,
        conferenceCreateResponse        ConferenceCreateResponse,
        conferenceQueryRequest          ConferenceQueryRequest,
        conferenceQueryResponse         ConferenceQueryResponse,
        conferenceJoinRequest           ConferenceJoinRequest,
        conferenceJoinResponse          ConferenceJoinResponse,
        conferenceInviteRequest         ConferenceInviteRequest,
        conferenceInviteResponse        ConferenceInviteResponse,
        ...
}
```

```
-- ================================================================================
-- Part 4:  Messages sent using MCS-Send-Data or MCS-Uniform-Send-Data
-- ================================================================================

GCCPDU ::= CHOICE
{
        request                                 RequestPDU,
        response                                ResponsePDU,
        indication                              IndicationPDU
}

RequestPDU ::= CHOICE
{
        conferenceJoinRequest                   ConferenceJoinRequest,
        conferenceAddRequest                    ConferenceAddRequest,
        conferenceLockRequest                   ConferenceLockRequest,
        conferenceUnlockRequest                 ConferenceUnlockRequest,
        conferenceTerminateRequest              ConferenceTerminateRequest,
        conferenceEjectUserRequest              ConferenceEjectUserRequest,
        conferenceTransferRequest               ConferenceTransferRequest,
        registryRegisterChannelRequest          RegistryRegisterChannelRequest,
        registryAssignTokenRequest              RegistryAssignTokenRequest,
        registrySetParameterRequest             RegistrySetParameterRequest,
        registryRetrieveEntryRequest            RegistryRetrieveEntryRequest,
        registryDeleteEntryRequest              RegistryDeleteEntryRequest,
        registryMonitorEntryRequest             RegistryMonitorEntryRequest,
        registryAllocateHandleRequest           RegistryAllocateHandleRequest,
        nonStandardRequest                      NonStandardPDU,
        ...
}

ResponsePDU ::= CHOICE
{
        conferenceJoinResponse                  ConferenceJoinResponse,
        conferenceAddResponse                   ConferenceAddResponse,
        conferenceLockResponse                  ConferenceLockResponse,
        conferenceUnlockResponse                ConferenceUnlockResponse,
        conferenceTerminateResponse             ConferenceTerminateResponse,
        conferenceEjectUserResponse             ConferenceEjectUserResponse,
        conferenceTransferResponse              ConferenceTransferResponse,
        registryResponse                        RegistryResponse,
        registryAllocateHandleResponse          RegistryAllocateHandleResponse,
        functionNotSupportedResponse            FunctionNotSupportedResponse,
        nonStandardResponse                     NonStandardPDU,
        ...
}

IndicationPDU ::= CHOICE
{
        userIDIndication                        UserIDIndication,
        conferenceLockIndication                ConferenceLockIndication,
        conferenceUnlockIndication              ConferenceUnlockIndication,
        conferenceTerminateIndication           ConferenceTerminateIndication,
        conferenceEjectUserIndication           ConferenceEjectUserIndication,
        conferenceTransferIndication            ConferenceTransferIndication,
        rosterUpdateIndication                  RosterUpdateIndication,
        applicationInvokeIndication             ApplicationInvokeIndication,
        registryMonitorEntryIndication          RegistryMonitorEntryIndication,
        conductorAssignIndication               ConductorAssignIndication,
        conductorReleaseIndication              ConductorReleaseIndication,
```

```
        conductorPermissionAskIndication            ConductorPermissionAskIndication,
        conductorPermissionGrantIndication          ConductorPermissionGrantIndication,
        conferenceTimeRemainingIndication           ConferenceTimeRemainingIndication,
        conferenceTimeInquireIndication             ConferenceTimeInquireIndication,
        conferenceTimeExtendIndication              ConferenceTimeExtendIndication,
        conferenceAssistanceIndication              ConferenceAssistanceIndication,
        textMessageIndication                       TextMessageIndication,
        nonStandardIndication                       NonStandardPDU,
        ...
}

END
```

9 Use of the Multipoint Communication Service

All GCC communication shall be through the Multipoint Communication Service (MCS) as specified in Recommendation T.122. This clause details how GCC makes use of MCS services, channel allocation, token allocation and data priorities.

9.1 MCS services

GCC assumes the MCS services indicated in Table 9-1. All primitives and parameters marked with an "M" are used by mandatory components of GCC. Items marked with an "O" are used only by optional portions of GCC.

Table 9-1/T.124 – MCS services used by GCC

Primitives	Use	Parameters	Use
MCS-Connect-Provider request	M	Calling Address	O
MCS-Connect-Provider indication	M	Calling Domain Selector	M
MCS-Connect-Provider response	M	Called Address	O
MCS-Connect-Provider confirm	M	Called Domain Selector	–
		Upward/Downward Flag	M
		Domain Parameters	M
		Quality of Service	M
		Result	M
		User Data	M
MCS-Disconnect-Provider request	M	Reason	M
MCS-Disconnect-Provider indication	M		
MCS-Attach-User request	M	Domain Selector	M
MCS-Attach-User confirm	M	Result	M
		User ID	M
MCS-Detach-User request	M	Reason	M
MCS-Detach-User indication	M	User ID	M
MCS-Channel-Join request	M	Channel to Join	M
MCS-Channel-Join confirm	M	Result	M
MCS-Channel-Leave request	–	Channel to Leave	–
MCS-Channel-Leave indication	–	Reason	–
MCS-Channel-Convene request	–	Result	–
MCS-Channel-Convene confirm	–	Channel	–

Table 9-1/T.124 – MCS services used by GCC (*concluded*)

Primitives	Use	Parameters	Use
MCS-Channel-Disband request MCS-Channel-Disband indication	– –	Channel Reason	– –
MCS-Channel-Admit request MCS-Channel-Admit indication	– –	Channel Manager User ID List of User IDs	– – –
MCS-Channel-Expel request MCS-Channel-Expel indication	– –	Channel List of User IDs Reason	– – –
MCS-Send-Data request MCS-Send-Data indication	M M	Priority Channel ID Sender User ID Data	M M M M
MCS-Uniform-Send-Data request MCS-Uniform-Send-Data indication	M M	Priority Channel ID Sender User ID Data	M M M M
MCS-Token-Grab request MCS-Token-Grab confirm	O O	Token ID Result	O O
MCS-Token-Inhibit request MCS-Token-Inhibit confirm	O O	Token ID Result	O O
MCS-Token-Give request MCS-Token-Give indication MCS-Token-Give response MCS-Token-Give confirm	O M M O	User ID Giving User ID to Receive Token ID Result	M O M M
MCS-Token-Please request MCS-Token-Please indication	O O	User ID Requesting Token ID	O O
MCS-Token-Release request MCS-Token-Release confirm	O O	Token ID Result	O O
MCS-Token-Test request MCS-Token-Test confirm	O O	Token ID Token Status	O O

9.2 Channel allocation

GCC reserves two static channels for its exclusive use. One channel, GCC-CHANNEL-0 is joined by all GCC Providers in a conference. Another channel, GCC-CHANNEL-1, is joined only by the Conference Convener. Each GCC Provider also joins the Node ID Channel allocated to it by MCS. Table 9-2 shows the channel usage by GCC.

Table 9-2/T.124 – GCC channel usage

Channel ID	Type	Mnemonic	Description
GCC-CHANNEL-0	Static	GCC-Broadcast-Channel	For communication from any GCC Provider in a conference to all GCC Providers.
GCC-CHANNEL-1	Static	GCC-Convener-Channel	For communication from any GCC Provider in a conference to the Conference Convener.
GCC-CHANNEL-2	Static	GCC-Conventional-Broadcast-Channel	For communication from any GCC Provider in a conference to all GCC Providers.
GCC-CHANNEL-3	Static	GCC-Counted-Broadcast-Channel	For communication from any Conventional node GCC Provider in a conference to any other Conventional node GCC Provider in the conference. Used to communicate Counted node activity among Conventional nodes.
–	Dynamic	Node ID Channel	For communication from any GCC Provider in a conference to the GCC Provider at a particular node. Each node is identified by its Node ID.

9.3 Token allocation

GCC reserves a single token for its exclusive use. This token, GCC-TOKEN-0, is used as the Conference Conductorship Token. GCC also reserves tokens 16 384 through 65 535 for use as dynamic tokens which GCC allocates for use by Application Protocol Entities. Table 9-3 shows the token usage by GCC.

Table 9-3/T.124 – GCC token usage

Token ID	Mnemonic	Description
GCC-TOKEN-0	Conference-Conductorship-Token	Grabbed by a GCC Provider to become the Conference Conductor.
16 384 through 65 535	Dynamic Tokens	Allocated by GCC for use by Application Protocol Entities using GCC-Registry-Assign-Token primitive.

9.4 Use of MCS data transmission services

Table 9-4 summarizes the use of MCS data transmission services (MCS-Send-Data and MCS-Uniform-Send-Data) for each GCCPDU. Listed for each GCCPDU are the type of MCS data service used, the Channel ID, and requested data priority.

Table 9-4/T.124 – Use of MCS send data for GCCPDUs

GCCPDU	Channel	Send data type	Priority
UserIDIndication	Node ID Channel of directly connected node	MCS-Send-Data	Top
ConferenceCreateRequest	–	–	–
ConferenceCreateResponse	–	–	–
ConferenceQueryRequest	–	–	–
ConferenceQueryResponse	–	–	–
ConferenceJoinRequest	Node ID Channel of Top GCC	MCS-Send-Data	Top
ConferenceJoinResponse	Node ID Channel of requester	MCS-Send-Data	Top
ConferenceInviteRequest	–	–	–
ConferenceInviteResponse	–	–	–
ConferenceAddRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
	Node ID Channel of adding MCU	MCS-Send-Data	High
ConferenceAddResponse	Node ID Channel of requester	MCS-Send-Data	High
ConferenceLockRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
ConferenceLockResponse	Node ID Channel of requester	MCS-Send-Data	High
ConferenceLockIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High
	Node ID Channel	MCS-Send-Data	High
ConferenceUnlockRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
ConferenceUnlockResponse	Node ID Channel of requester	MCS-Send-Data	High
ConferenceUnlockIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High
	Node ID Channel	MCS-Send-Data	High
ConferenceTerminateRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
ConferenceTerminateResponse	Node ID Channel of requester	MCS-Send-Data	High
ConferenceTerminateIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High
ConferenceEjectUserRequest	Node ID Channel of Top GCC	MCS-Send-Data	Top
ConferenceEjectUserResponse	Node ID Channel of requester	MCS-Send-Data	High
ConferenceEjectUserIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	Top
ConferenceTransferRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
ConferenceTransferResponse	Node ID Channel of requester	MCS-Send-Data	High
ConferenceTransferIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High
RosterUpdateIndication	Node ID Channel of recipient	MCS-Send-Data	High
	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High

Table 9-4/T.124 – Use of MCS send data for GCCPDUs (*concluded*)

GCCPDU	Channel	Send data type	Priority
ApplicationInvokeIndication	GCC-Broadcast-Channel	MCS-Send-Data or MCS-Uniform-Send-Data	High
	Node ID Channel of recipient	MCS-Send-Data	High
RegistryRegisterChannelRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
RegistryAssignTokenRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
RegistrySetParameterRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
RegistryRetrieveEntryRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
RegistryDeleteEntryRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
RegistryMonitorEntryRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
RegistryMonitorEntryIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High
RegistryAllocateHandleRequest	Node ID Channel of Top GCC	MCS-Send-Data	High
RegistryAllocateHandleResponse	Node ID Channel of requester	MCS-Send-Data	High
RegistryResponse	Node ID Channel of requester	MCS-Send-Data	High
ConductorAssignIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	Top
ConductorReleaseIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	Top
ConductorPermissionAskIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High
ConductorPermissionGrantIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	Top
ConferenceTimeRemainingIndication	GCC-Broadcast-Channel	MCS-Uniform-Send-Data	High
ConferenceTimeInquireIndication	GCC-Convener-Channel	MCS-Send-Data	High
ConferenceTimeExtendIndication	GCC-Convener-Channel	MCS-Send-Data	High
ConferenceAssistanceIndication	GCC-Broadcast-Channel	MCS-Send-Data or MCS-Uniform-Send-Data	High
TextMessageIndication	GCC-Broadcast-Channel	MCS-Send-Data or MCS-Uniform-Send-Data	High
	Node ID Channel of recipient	MCS-Send-Data	High
FunctionNotSupportedResponse	Node ID Channel of requester	MCS-Send-Data	Same as request
NonStandardPDU	Not defined	Not defined	Not defined

9.5 Encoding of PDUs in MCS primitives

All PDUs defined in this Recommendation are encoded and placed in the data field of one of several possible MCS primitives. These are either MCS-Connect-Provider, MCS-Send-Data, or MCS-Uniform-Send-Data. In any of these cases, the bit string that results from the ASN.1 encoding is placed in the OCTET STRING used by MCS in the order such that for each octet, the leading bit is placed in the most significant bit position, and the trailing bit is placed in the least significant bit position.

In the case of MCS-Connect-Provider, the PDU itself is not placed in the MCS data parameter directly, but is contained within an enclosing structure defined in 9.6 In this case, the description above applies to the outer structure rather than the PDU itself.

9.6 Format of User Data parameter of MCS-Connect-Provider

The User Data parameter of the MCS-Connect-Provider and MCS-Disconnect-Provider primitives as used by this Recommendation is of a format which provides unique identification of the MCS Controller (as defined in Recommendation T.122). The MCS Controller may either be a standard type (such as one that adheres to this Recommendation), or may be a non-standard type. To distinguish these, the User Data consists of an ASN.1 abstract type which includes an identifier of the type of MCS Controller followed by the PDU itself. The identifier field is of type Key which is a choice of either an Object Identifier, or an H221NonStandardIdentifier. The overall abstract type as well as its components is defined in 8.7. The overall structure is encoded using Packed Encoding Rules (Aligned variant) as defined in Recommendation X.691.

When the MCS Controller adheres to this Recommendation, this header shall be set to the Object Identifier choice with a value {itu-t recommendation t 124 version(0) 1}.

Within this structure, the PDU is not directly encoded but rather is included in a data field which is of type OCTET STRING. The PDU is separately encoded, also using Packed Encoding Rules (Aligned variant) as with all other PDUs defined in this Recommendation. The resulting encoded bit string is placed into the OCTET STRING in the order such that for each octet, the leading bit is placed in the most significant bit position, and the trailing bit is placed in the least significant bit position. The PDU and the enclosing structure are encoded separately to allow for future Recommendations or non-standard MCS Controllers to use different encoding of the PDUs while providing a uniquely encoded identifier of the MCS Controller type (always to be encoded using Packed Encoding Rules).

9.7 Interpretation of the MCS Domain Selector

In using the MCS-Connect-Provider primitive, this Recommendation makes the following assumptions in its use of the Domain Selector.

The Calling Domain Selector always accurately reflects the local Domain Selector which is equal to the Conference ID (which is also used by Application Protocols in making an MCS attachment).

The Called Domain Selector is never used by this Recommendation. Instead it is assumed that the MCS provider passes MCS-Connect-Provider indications to the Control MCSAP regardless of their contents. It is the role of the GCC Provider to determine if the connection should be established, and if so, which domain the connection should be associated with. This is done slightly differently for the four T.124 primitives which generate MCS-Connect-Provider requests:

- GCC-Conference-Create: In this case, there is no domain already in existence prior to reception of the MCS-Connect-Provider indication to which the connection could be attached. On receipt of a valid MCS-Connect-Provider indication, a GCC Provider must establish a new MCS domain and indicate to MCS that this is the domain which the new connection is to be associated with. The Domain Selector is chosen at that time by the GCC Provider as the local Conference ID. The means to perform both of these actions are local matters not specified in Recommendation T.122.
- GCC-Conference-Query: In this case, no connection is established since the MCS-Connect-Provider response indicates that the request was user-rejected. As a result, it is not necessary to associate the request to any MCS domain.
- GCC-Conference-Join: In this case, a domain does already exist at the receiving node. If the Join request is accepted by the GCC Provider, the GCC Provider must indicate to the MCS provider which domain the connection is to be associated with. The Conference Name (and Conference Name Modifier) are used by the GCC Provider to determine which domain to indicate. The Domain Selector is the local Conference ID of that conference. The means to indicate the domain to the MCS provider is a local matter not specified in Recommendation T.122.
- GCC-Conference-Invite: In this case, as in the Create case, the receiving node does not already have a domain established. The GCC Provider, if the Invite is accepted, chooses a Conference ID which is to be used as the Domain Selector. The MCS provider must be told of the establishment of this domain at this node, and that the particular connection is to be associated with this domain. As in the case of the Create, the means to perform both of these actions are local matters not specified in Recommendation T.122.

ANNEX A

Static channel and token ID assignments

A.1 Static channel ID assignments

Table A.1 lists the numerical assignment of static channel IDs for the static channels allocated for use by this Recommendation. The numerical assignment of static channel IDs is intended to be centralized in Recommendation T.120, but is included here until Recommendation T.120 is completed.

Table A.1/T.124 – Static Channel ID assignments

Symbolic name	Channel ID
GCC-CHANNEL-0	1
GCC-CHANNEL-1	2
GCC-CHANNEL-2	3
GCC-CHANNEL-3	4

A.2 Static token ID assignments

Table A.2 lists the numerical assignment of static token IDs for the static tokens allocated for use by this Recommendation. The numerical assignment of static token IDs is intended to be centralized in Recommendation T.120, but is included here until Recommendation T.120 is completed.

Table A.2/T.124 – Static token ID assignments

Symbolic name	Token ID
GCC-TOKEN-0	1

ANNEX B

Object Identifier assignments

Table B.1/T.124

Object Identifier Value	Description
{itu-t recommendation t 124 version(0) 1}	This Object Identifier is used to indicate the version of this Recommendation in use as the MCS Controller. At this time there is a single standardized version defined.

ANNEX C

Network Address Parameter – Description and use

ASN.1 structure NetworkAddressV2 specified in 8.7 (GCCPDU Definitions) reshapes and enhances structure NetworkAddress. By dissociating the concepts of network types, transfer rates and profiles (or transfer modes) which are mixed together into the flat organization used by structure NetworkAddress, it brings more flexibility in describing a network connection toward a conference node, and solves a certain number of pre-signalling issues not addressed so far.

NetworkAddressV2 has been added beside NetworkAddress in order to meet backward compatibility with older T.120 implementations not using it. ITU-T intends to maintain NetworkAddressV2 as T.124 evolves, while letting the use of NetworkAddress diminish until it may be removed from the standard. This Annex constitutes an intermediate step toward this point.

To guarantee interworking between T.120 devices using and not using NetworkAddressV2, the following rules shall be respected by implementations:

1) Any T.120 equipment that conforms to this version of T.124, namely Recommendation T.124 revised – 1998, SHALL make use of NetworkAddressV2 IN CONJUNCTION WITH NetworkAddress.

2) Each individual T.124 provider using NetworkAddressV2 is responsible for supplying a NetworkAddress structure that matches the closest as possible features contained in NetworkAddressV2.

3) Each individual T.124 provider using NetworkAddressV2 which, at some point, must process and relay a GCCPDU containing network address parameters, and which receives the network address information from a node not making use of NetworkAddressV2, is responsible for adding in the relayed PDU a NetworkAddressV2 that matches the closest as possible the description passed through NetworkAddress (see Note).

   NOTE – A typical example is the conference nodes transfer mechanism described in 8.2.11; when the Top GCC provider uses NetworkAddressV2 and receives the ConferenceTransferRequest PDU from a node that uses NetworkAddress only, it shall produce and insert a NetworkAddressV2 structure in the corresponding ConferenceTransferIndication PDU that it will broadcast on the GCC-Broadcast-Channel channel.

4) T.124 providers receiving GCCPDUs containing a NetworkAddressV2 structure and that do not support it will silently ignore it.

APPENDIX I

Relationship of T.120 to H.243 in H.320 conferences

I.1 Introduction

It is the intent of T.120 is to eventually take over the responsibilities of H.243 for multipoint conference establishment in H.320 audio/video/data conferences. In the meantime, H.243 and T.120 will continue to coexist. This Appendix provides recommendations as to how H.243 and T.120 should relate during the long transitional phase from H.243 to T.120 conference control.

NOTE – Clause 15/H.243 places normative requirements on terminals and MCUs in this regard.

I.2 Conference selection and Password protection

Both T.124 and H.243 provide means of establishing a logical connection from a terminal to a particular conference hosted at an MCU. Both Recommendations also provide a password protection mechanism for verifying the authority of a node to join the desired conference. It is envisioned that both T.124 and H.243 will continue to be used for some time. The following subclauses describe means by which T.124 and H.243 should be used during the conference establishment process.

The assumption is made that at most one H.243 conference is associated with each T.124 conference and vice versa. Thus, if a terminal joins either one, the choice of the other conference is then predetermined. If this assumption is not valid, then different procedures may apply.

For the purpose of selecting a conference to join (in the case that the MCU allows such a choice) and for the purpose of password protection, an MCU should choose to use the T.124 or the H.243 mechanisms, but not both. The choice of which mechanism to use may depend on whether the conference is dial-in or dial-out, may be made separately for each MCU port, and may depend on the capabilities of the connecting terminal on each port. In the case of terminals that support T.120, it is preferred that the T.124 mechanisms be used for these purposes rather than the H.243 mechanism.

I.2.1 T.124 conference establishment

The MCU may choose to use the T.124 mechanisms for conference establishment for a given terminal. The MCU may, for example, allow the terminal to choose among a list of conferences to join by providing this list in the GCC-Conference-Query response and allowing selection of a conference using the GCC-Conference-Join primitive. The MCU may instead have a predetermined conference that it wishes the connecting terminal to join. In this case, it should set the Default Conference Flag in the GCC-Conference-Query response's corresponding Conference Descriptor to indicate that the connecting terminal should join the indicated conference. In either case, if the MCU requires password protection, the T.124 password protection mechanisms (Password In The Clear Required or Challenge Response Required) should be used to verify that the connecting terminal is authorized to join the conference.

Alternatively, as may be the case for a prearranged dial-out, if the choice of conference is already determined and password protection is not required, the MCU can decide to take the initiative and to invite the terminal into a T.124 conference by issuing a GCC-Conference-Invite request. In this case, it should set the Wait for Invitation Flag in the GCC-Conference-Query response to indicate that the connecting terminal should not take action to join or create a T.124 conference.

Once the MCU receives a GCC-Conference-Join-Request with a conference name and the corresponding correct password or decides to take the initiative and to invite a terminal into a T.124 conference, the terminal may then start receiving the audio and video associated with the corresponding H.243 conference.

In this case, the MCU should not use the H.243 mechanism to request a password from the connecting terminal. H.243 may still be used for video control and other functions.

I.2.2 H.243 conference establishment

Alternatively, the MCU may choose to use the H.243 mechanism for conference establishment. In this case, if a terminal calls into an MCU, the MCU may use the H.243 password to determine in which conference the terminal belongs. Once a correct H.243 password is supplied, the terminal may start receiving the audio and video associated with the conference.

Since H.243 provides a password mechanism, the additional requirement of a T.124 conference password in this case is superfluous (assuming that only one T.124 conference is permitted in conjunction with the H.243 conference). The MCU may decide to take the initiative and to invite the terminal into the T.124 conference associated with the H.243 conference by issuing a GCC-Conference-Invite request. If queried by the terminal, the MCU should set the Wait for Invitation Flag in the GCC-Conference-Query response to indicate that the terminal need not take action to join or create a T.124 conference. Alternatively the MCU can wait for the terminal to issue a GCC-Conference-Join request. In this case, if queried by the terminal, the MCU should return a single Conference Descriptor in its GCC-Conference-Query response, with the Default Conference Flag set and the Password In The Clear Required Flag reset.

Terminals that support T.120 should allow the H.243 password mechanism to be used for conference establishment. This should be considered an interim approach used by MCUs until they support the T.124 conference establishment procedures described in the previous subclause.

I.3 Alternative Node ID

The Alternative Node ID of T.124 was envisioned for use with the terminal numbering of H.243, allowing nodes in an audio/video/data conference running both T.120 and H.243 to correlate T.124 conference roster information to H.243 site information.

When issuing the GCC-Conference-Announce-Presence request, nodes should include their assigned H.243 terminal number in the Alternative Node ID field. If a node's H.243 terminal number is reassigned during a conference, the node should re-issue a GCC-Conference-Announce-Presence request with the new terminal number contained in the Alternative Node ID field.

NOTE – This requires that each terminal remember the content of the most recently received H.230 C&I code TIA.

The Alternative Node ID is a two-octet field. The first octet should contain the H.243 MCU ID (M), and the second octet should contain the H.243 Terminal ID (T).

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	**Terminals for telematic services**
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communication
Series Y	Global information infrastructure
Series Z	Programming languages