INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

T.121
(07/96)

SERIES T: TERMINAL EQUIPMENTS AND PROTOCOLS FOR TELEMATIC SERVICES

Generic application template

ITU-T Recommendation T.121

(Previously "CCITT Recommendation")

ITU-T T-SERIES RECOMMENDATIONS

TERMINAL EQUIPMENTS AND PROTOCOLS FOR TELEMATIC SERVICES

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

FOREWORD

The ITU-T (Telecommunication Standardization Sector) is a permanent organ of the International Telecommunication Union (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 (Helsinki, March 1-12, 1993).

ITU-T Recommendation T.121 was prepared by ITU-T Study Group 8 (1993-1996) and was approved under the WTSC Resolution No. 1 procedure on the 3rd of July 1996.

____

NOTE

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

CONTENTS

SUMMARY

This Recommendation provides guidance for application developers and application protocol developers on the correct and effective use of the T.120 infrastructure. It provides a generic model for an application that communicates using T.120 services and defines a Generic Application Template specifying the use of T.122 (MCS) and T.124 (GCC) services that are commonly required by application protocols making use of T.120 services. This Recommendation provides a common structure to application protocol entities, ensuring that separate application protocol entities can coexist in the same conference. Application protocol entities that do not follow the operations defined in this Recommendation risk interfering with other application protocols.

Recommendation T.121

GENERIC APPLICATION TEMPLATE

(Geneva, 1996)

1 Scope

Whilst user applications themselves are not the subject of standardization, the protocols employed by a user application need to be standardized in order to guarantee interworking. This Recommendation provides a generic model for a T.120 application with a view to defining a common framework (the Generic Application Template) to form the basis of standardized application protocols. It also provides guidance to user application developers on how to exploit the T.120 infrastructure to achieve the desired application behaviour. The templates defined in this Recommendation are derived from terminal based application protocols; application protocols which incorporate an MCU component may require different provisions.

A T.120 user application employs one or more Application Protocol Entities to communicate with user applications of similar functionality at other nodes in a conference. It is important to observe that, whilst it is not necessary to have identical implementations at each node in order to achieve interoperability, groups of communicating user applications must employ the same application protocols for the same general purpose. Differences in capabilities are resolved through a capability exchange mechanism.

Each Application Protocol Entity (APE) consists of two components:

- A generic portion which includes elements common to most or all application protocols, for example initialization and resource management. Such operations need only be specified once (in this Recommendation) and then referenced by each protocol specification.
- A function specific portion which enables user applications of like functionality to interwork. This part may be standardized in which case it is specified in a T.120 application protocol Recommendation, or non-standard.

This Recommendation describes the components identified above and defines a Generic Application Template (GAT) containing those operations that are expected to be common to most T.120 application protocols. The Generic Application Template eases the task of the application protocol developer and provides a common structure for application protocol entities. This modular approach allows the function specific elements of a protocol specification to be clearly identified, and thus enables it to be reused more readily in other protocol specifications. This Recommendation defines a rigorous protocol which is expressed through a sequence of primitive operations. Furthermore, this Recommendation can be made normative by reference from standardized application protocol Recommendations. Use of the Generic Application Template in the definition of T.120 application protocols is encouraged.

2 Normative References

The following 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 T.120 (1996), *Data protocols for multimedia conferencing*.

– ITU-T Recommendation T.122 (1993), *Multipoint communication service for audiographics and audio-visual conferencing service definition*.

– ITU-T Recommendation T.124 (1995), *Generic conference control*.

– ITU-T Recommendation T.125 (1994), *Multipoint communication service protocol specification*.

– ITU-T Recommendation T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information Technology – 7-bit coded character set for information interchange*.

3 Definitions

For the purposes of this Recommendation, the following definitions apply:

3.1 application resource manager: The part of an Application Protocol Entity that provides generic functionality for managing GCC and MCS resources.

3.2 application service element: The part of an Application Protocol Entity that provides Application Protocol specific functionality, e.g. message formats and sequences.

3.3 non-standard base session: An Application Protocol Session that can be located by non-standard APEs without requiring user intervention. As such it is similar to a standard base session used by standard APEs.

3.4 private session: An Application Protocol Session with restricted membership controlled by the creator of that session.

3.5 public session: An Application Protocol Session with unrestricted membership that is typically used if the relevant base session for the protocol is already in use and a user wishes to establish further independent sessions.

3.6 registration session: An Application Protocol Session used by applications to advertise their presence to the conference. A Registration Session makes use of the T.124 default session in which there is no Session ID.

3.7 session creator: The Application Protocol Entity that enrolls in a new Application Protocol Session. The concept of a session creator only applies to public and private sessions.

3.8 session member: An Application Protocol Entity that enrolls in an existing Application Protocol Session. The concept of a session member only applies to public and private sessions.

3.9 standard base session: An Application Protocol Session that can be located by standard APEs without requiring user intervention. Standard base session identifiers are specified in Recommendation T.120.

4 Abbreviations

For the purposes of this Recommendation, the following abbreviations are used:

ARM	Application Resource Manager
APE	Application Protocol Entity
ASE	Application Service Element
GAT	Generic Application Template
GCC	Generic Conference Control
GCC SAP	GCC Service Access Point
MCS	Multipoint Communication Service
MCSAP	MCS Service Access Point
MCU	Multipoint Control Unit
PDU	Protocol Data Unit
SAP	Service Access Point

5 Overview

Figure 1 illustrates the T.120 system model in which user applications employ both standard and non-standard application protocols in order to communicate with their peers at other nodes within the same conference. The scope of a user application is restricted to local functions (e.g. user interface) that do not affect communication between peer user applications. Each user application makes use of one or more Application Protocol Entities (APEs) that implement interworking protocols as shown in Figure 2. In the model described here, an APE can be further subdivided into two elements: an Application Resource Manager (ARM) and an Application Service Element (ASE). The ARM is responsible for the generic management of GCC and MCS resources whilst the ASE provides interworking of specific functionality; for example, an ASE may provide general purpose file transfer functionality as specified in Recommendation T.127, or still image exchange functionality as specified in Recommendation T.126.

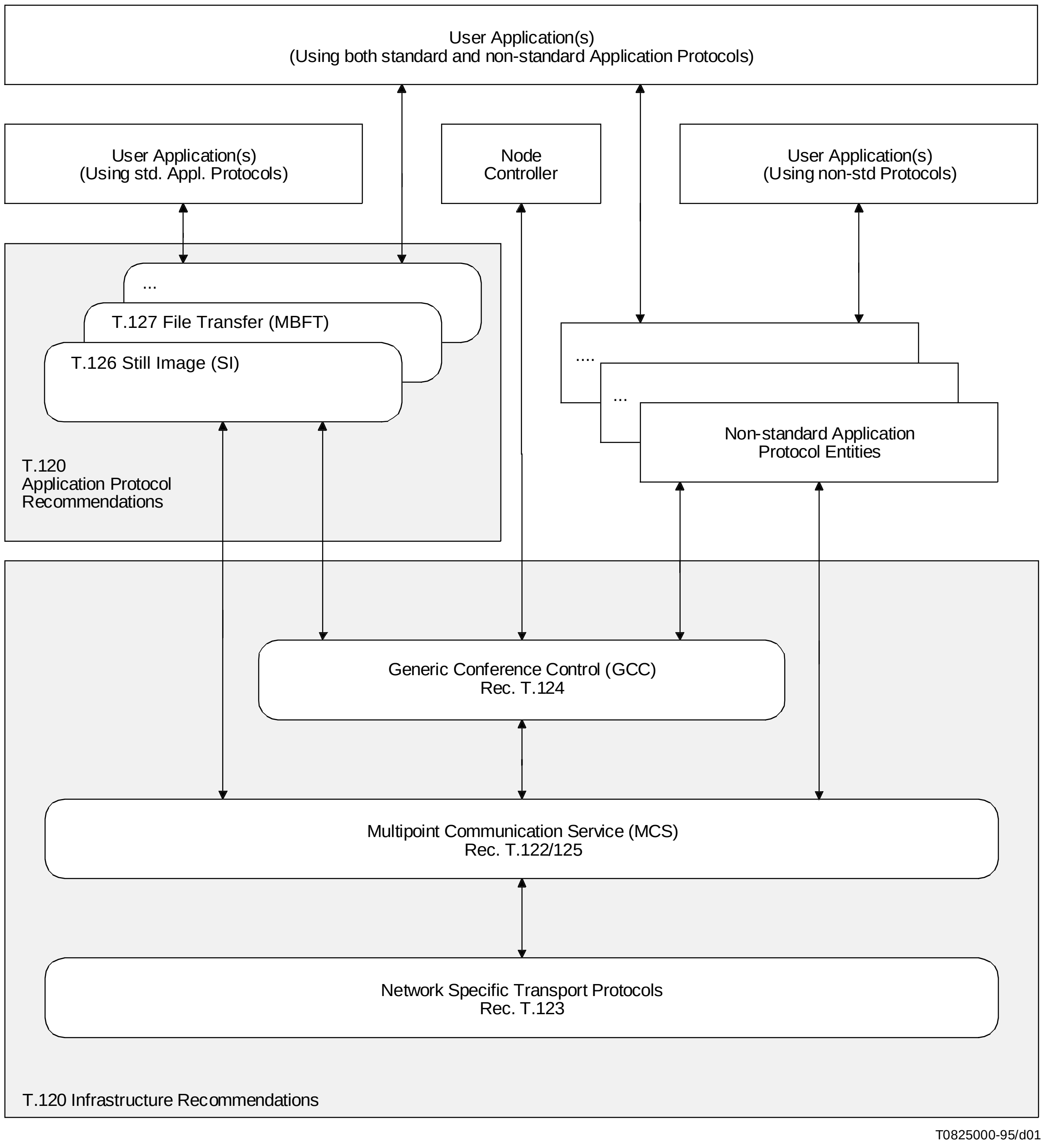

User Application(s)
(Using both standard and non-standard Application Protocols)
User Application(s)
(Using std. Appl. Protocols)
Node
Controller
User Application(s)
(Using non-std Protocols)
...
T.127 File Transfer (MBFT)
T.126 Still Image (SI)
T.120
Application Protocol
Recommendations
....
...
Non-standard Application
Protocol Entities
Generic Conference Control (GCC)
Rec. T.124
Multipoint Communication Service (MCS)
Rec. T.122/125
Network Specific Transport Protocols
Rec. T.123
T.120 Infrastructure Recommendations
T0825000-95/d01

FIGURE 1/T.121

T.120 system model

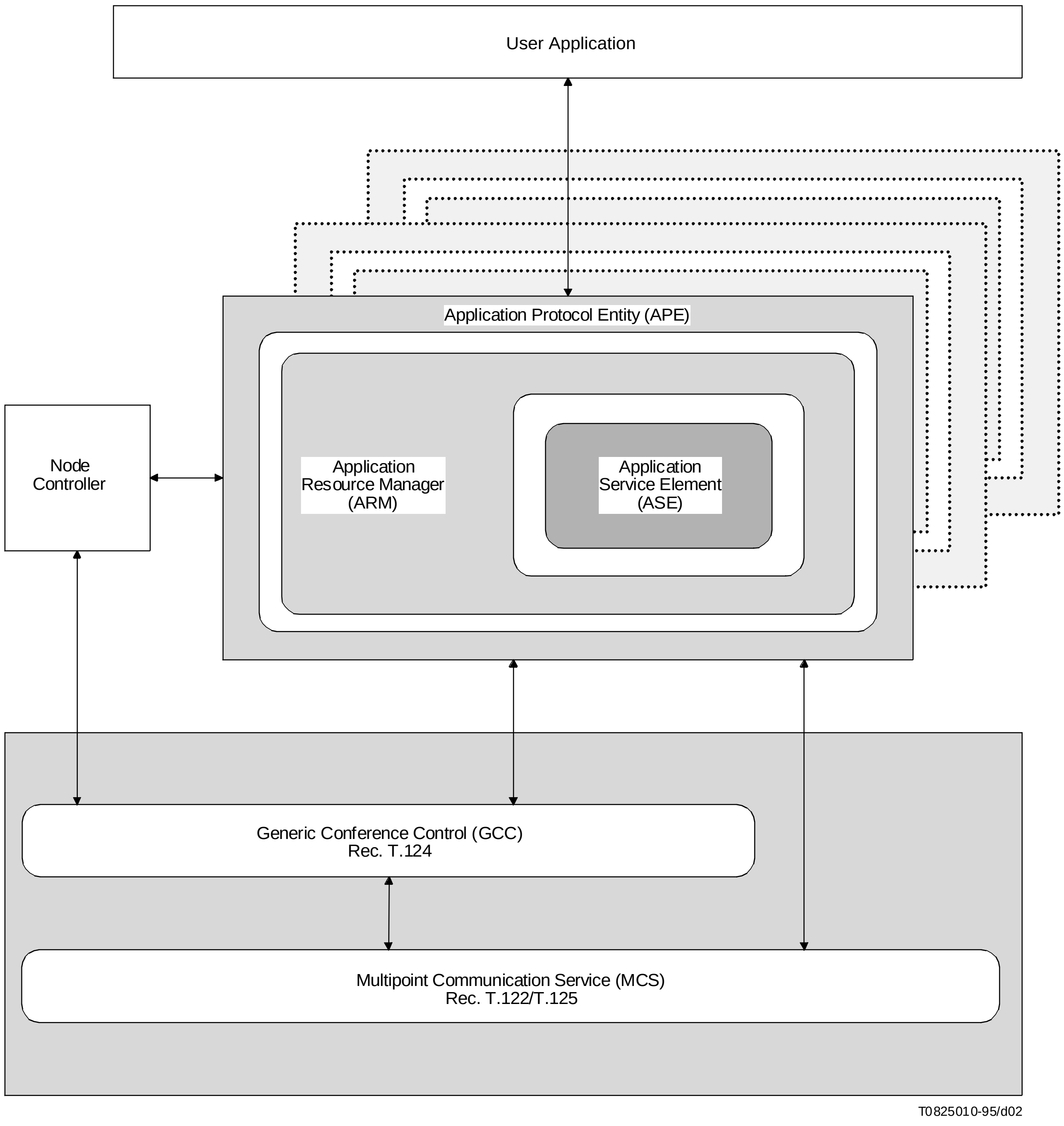

User Application
Application Protocol Entity (APE)
Node Controller
Application Resource Manager (ARM)
Application Service Element (ASE)
Generic Conference Control (GCC) Rec. T.124
Multipoint Communication Service (MCS) Rec. T.122/T.125
T0825010-95/d02

FIGURE 2/T.121

Model of Generic T.120 Application Protocol Entity

Peer Application Protocol Elements communicating with each other are said to be participating in the same application protocol *session*. Note that more than one user application at a given site may participate in the same session. This entails multiple instances of the application protocol, i.e. multiple APEs, one per user application. It is also possible to have multiple concurrent sessions using the same application protocol. Again, each participating user application requires its own APE. An example scenario is illustrated in Figure 3.

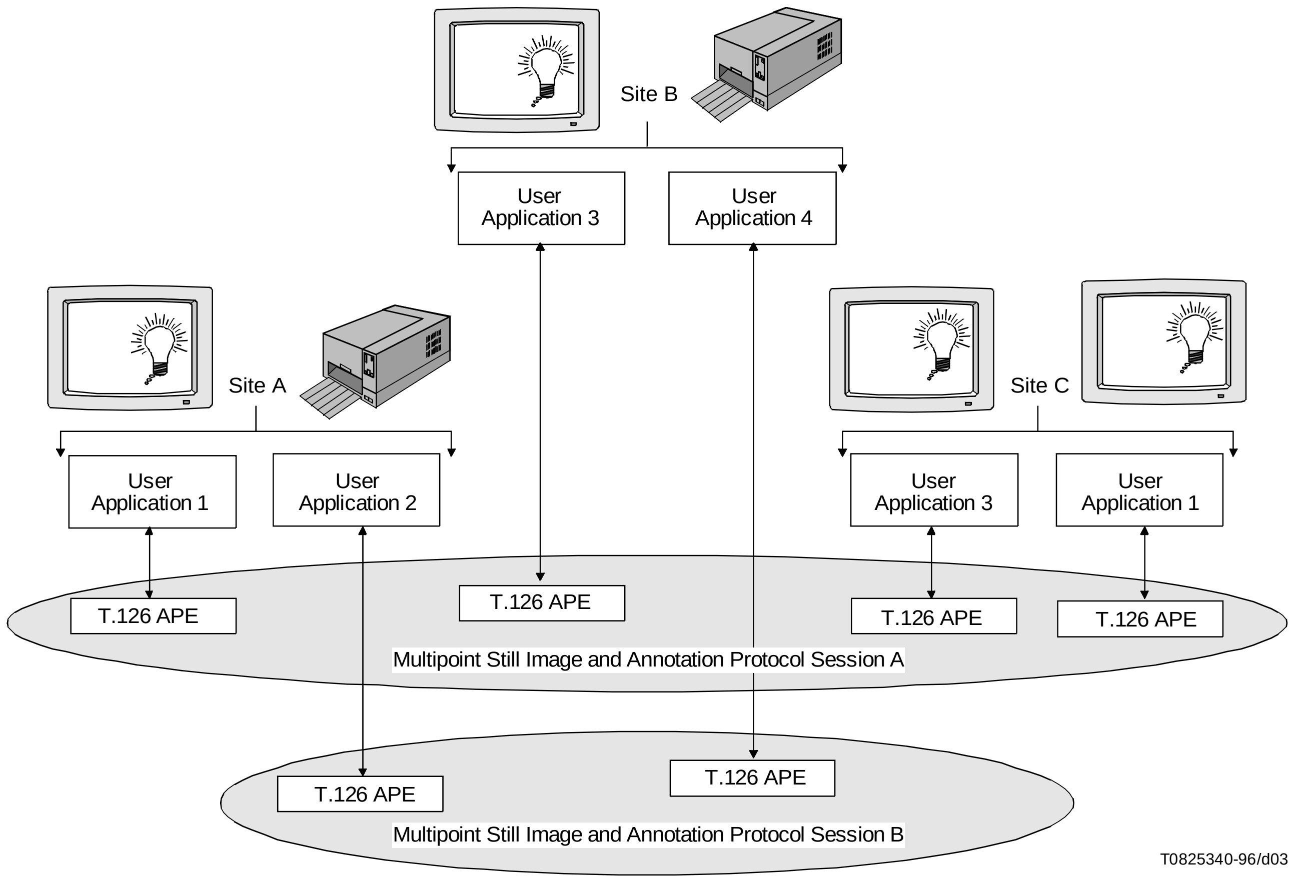


FIGURE 3/T.121

Example of multiple applications at a site using the same protocol

A session comprises a group of peer Application Protocol Entities that employ a common application protocol to communicate with each other. Each Session has a Session Identifier whose value is equal to that of an MCS Channel ID, typically one that is used to transfer data within the group of APEs. In the models presented here, the channel associated with the Session Identifier is the first channel joined, although this is not required to be the case. It is known as the *Session Identifier Channel*. A user application that employs multiple APEs may participate in multiple application protocol sessions. Each session could have a unique Session Identifier, but this is not required. Different types of application protocol entity participate in different sessions, even if they have the same Session ID. An example scenario is illustrated in Figure 4. For each application protocol, it is possible to have a single application protocol session without a Session Identifier; this is referred to as the *registration* session.

The following session types exist:

Registration Session: may be used by both standard and non-standard application protocols. The purpose of a registration session is to enable applications to advertise their presence to the conference, without having to become active. This enables applications to determine which nodes in the conference have similar functionality, whilst at the same time consuming minimal resources. The registration session also allows applications to advertise application protocol specific functionality to the conference as non-collapsing capabilities. Applications would typically remain enrolled in the registration session for the duration of the conference in order to receive notification of any changes in functionality advertised by other nodes. An application enrolled in the registration session for a specific application protocol receives information on all sessions in the conference using that particular application protocol.

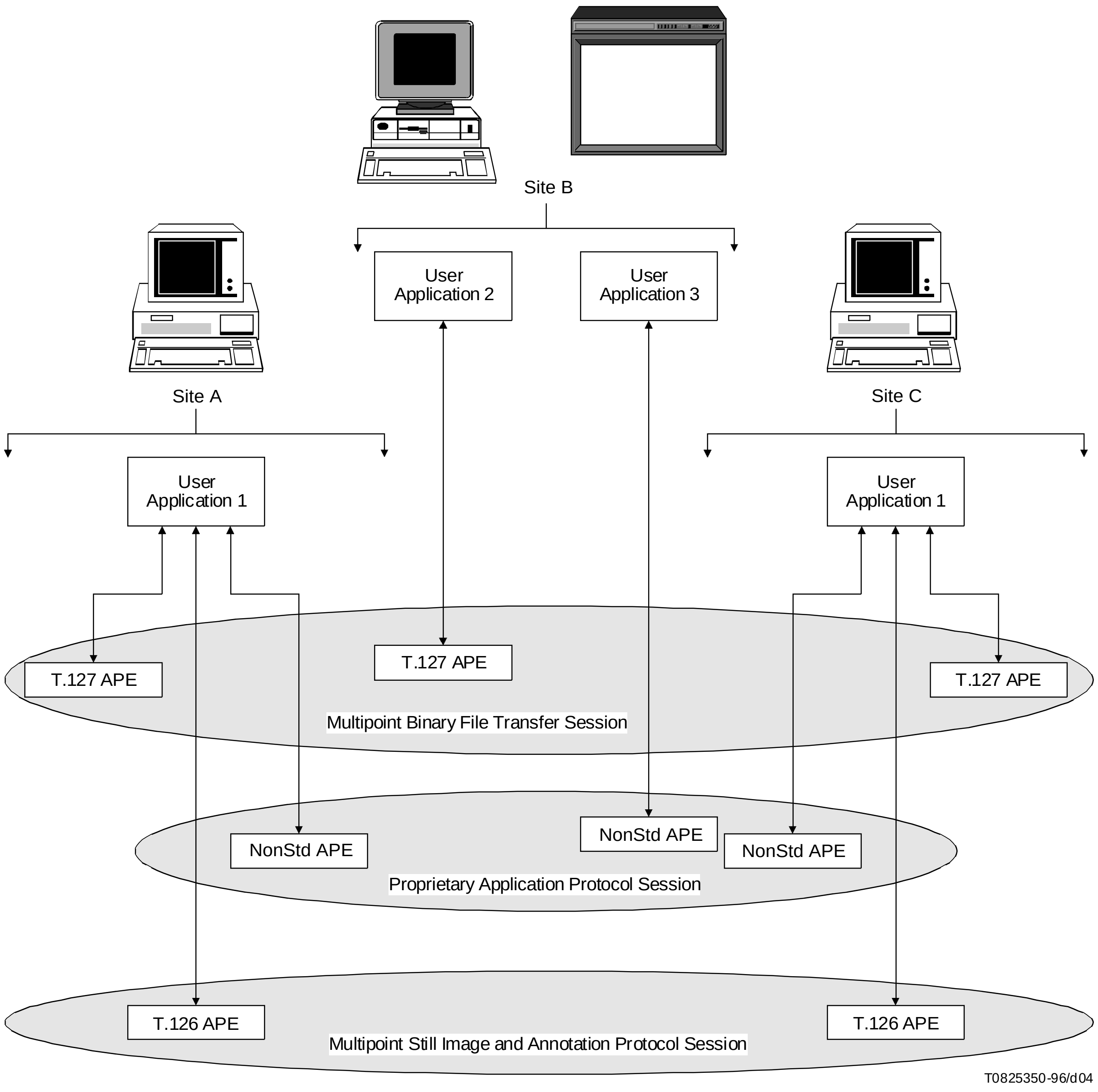


FIGURE 4/T.121

Example of applications using multiple protocols

Use of a registration session for actual application activity is not part of these models and is discouraged. This ensures a clean segregation of active and dormant applications. This segregation is important because:

- Any collapsing capabilities advertised by inactive applications may have an adverse impact on the capability set of the session.
- Defunct registry entries in a registration session resulting from application activity are likely to persist until the end of the conference.

For each application protocol within a conference there can be at most one registration session. It is characterized by having a Session Key comprised of the appropriate Application Protocol Key and no Session ID.

Standard Base Session: reserved for use by standardized application protocols. Access to a standard base session is unrestricted and all participants within the session have equal status, that of session member; they may join and leave the session at will. As the Session Identifier for a standard base session of a standardized application protocol is predefined (in Recommendation T.120), an application may join the session without any user intervention. A standard base session uses a static channel as the Session ID channel. The set of resources that need to be identified during the enrollment process for a standard base session may include further static or dynamic channels and tokens, as specified by the application protocol. Dynamic channels and tokens may be employed if additional resources are required beyond this basic set.

Typically, the standard base session would be the preferred choice for an application using a standard application protocol that wishes to communicate with all peer applications within a conference. For each standardized application protocol within a conference, there is at most one standard base session.

Non-Standard Base Session: reserved for use by non-standard application protocols . The non-standard base session allows applications which employ a common non-standard protocol to independently locate the same session, without requiring user intervention to select a specific session. Typically, the non-standard base session would be the preferred choice for an application using a non-standard application protocol that wishes to communicate with all peer applications within a conference. The non-standard base session uses an assigned (i.e. dynamic multicast) channel as Session ID, the identity of which is discovered using the GCC Application Registry. A consequence of participating in the non-standard base session is that the corresponding registration session is joined. For each non-standard application protocol within a conference there can be not more than one non-standard base session. Once all participants have left a non-standard base session it ceases to exist and must be recreated if required subsequently.

Public Session: may be used by either standard or non-standard application protocols. A public session is typically used if the relevant base session for the protocol is already in use and a user wishes to establish further independent sessions to communicate with all peer applications within a conference. Access to a public session is uncontrolled. A public session has a designated creator. This is the APE which assigns the multicast Session Identifier channel and is responsible for registering the resources for the session; all other participants are members of the session. Any participant may join and leave a public session at will; they may optionally choose to invite other nodes to participate in the session using the GCC-Application-Invoke mechanism. A public session may persist after the creator of the session has left it or even detached. Once all participants have left a public session, no one can rejoin as the session ceases to exist.

Private Session: may be used by either standard or non-standard application protocols. A private session is typically employed when a user wishes to restrict membership of an application protocol session to a selected subset of conference participants. A private session has a designated creator. This is the APE which convenes the private session and is responsible for registering the resources for the session. Participation in the private session is controlled by the session creator who must explicitly invite other applications to become members of the session. A private session ceases to exist after the creator of the session has disbanded it or detached.

5.1 User application

A user application addresses those tasks which have no direct effect on interworking (e.g. user interface) and which may thus be product and platform specific. The influence of the user application is felt at other sites through the application protocols it employs.

A user application relies on the services of one or more Application Protocol Entities (APEs) to communicate with peer user applications at other nodes. It does not communicate with MCS or GCC; this is done by the APE(s). The interface between a user application and its APE(s) is a local matter and is outside the scope of this Recommendation.

For each APE the user application employs, the user application defines the subset of that APE's capabilities to be advertised. It also selects the type of session (standard base, non-standard base, public, private or registration) when creating a new instance of that APE.

The user application communicates with the appropriate APE to initiate an application session specifying the application capabilities and mode of operation for the session. Once a session has been established, all application protocol specific transactions are performed by the APE on behalf of the user application.

5.2 Application Protocol Entity

An Application Protocol Entity (APE) is employed by a user application to enable it to communicate with other user applications of like functionality, i.e. user applications which employ the same application protocol for the same purpose. For example, user applications which require general purpose file transfer functionality must employ an application protocol entity which uses the protocol defined in Recommendation T.127 and the application protocol key specified in Recommendation T.127 to guarantee interoperability. APEs may use either standard or non-standard application protocols.

An APE has two conceptual components as shown in Figure 2: an Application Resource Manager (ARM) and an Application Service Element (ASE). The ARM provides generic functionality, common to many standardized application protocols, whilst the ASE provides functionality specific to its respective application protocol.

In the general model, an Application Protocol Entity is characterized by the following attributes:

- A single Application Resource Manager (ARM).
- A single Application Service Element (ASE).
- A single GCC SAP.
- Interaction with a Node Controller (outside the scope of T.120 Recommendations).
- One or more MCSAPs.
- A single enrolled MCS User ID.
- A capability list (including collapsing and non-collapsing capabilities).
- A single Session ID. (Peer APEs communicating with each other share the same Session ID. The absence of a Session ID indicates that the APE is enrolled in the registration session for the application protocol.)
- An Application Protocol Key.

5.3 Application Resource Manager

The Application Resource Manager (ARM) is responsible for managing GCC and MCS resources on behalf of the ASE within its APE. The ARM provides the following services:

- Responding to indications from GCC (e.g. permission to enroll).
- Enrolling its APE with GCC.
- Obtaining handles from GCC.
- Attaching to an MCS domain to obtain a unique MCS User ID for its APE.
- Joining static channels.
- Identifying and joining assigned channels using the GCC Registry and MCS.
- Convening private channels and admitting peer APEs to such channels.
- Joining any private channels to which its APE has been admitted.
- Identifying and obtaining tokens from the GCC Registry.
- Identifying and assigning parameters using the GCC Registry.
- Deleting any redundant Registry entries for which the APE is responsible.
- Invoking peer APEs at other nodes.
- Processing Application Roster reports to determine the current negotiated Application Capability list and identity of peer nodes.
- Receiving GCC-Conductor-Assign and Release indications.
- Notifying its ASE of changes in conductorship status.

5.4 Application Service Element

The Application Service Element (ASE) provides application protocol specific functionality to the user application with resources obtained by the ARM. For example, in Recommendation T.127 the ASE provides general purpose file transfer functionality. Its operation is independent of the type (i.e. static or dynamic) and identity of tokens and channels passed to it. The ASE obtains the identity of resources to use from its ARM. In the case of transactions on private channels, the user application must specify the GCC Node IDs of those nodes with which communication is required.

The ASE provides the following services:

- Sending and receiving application protocol specific PDUs.
- Grabbing and releasing tokens and determining token status using MCS.
- Joining and leaving channels via the ARM.

Table 1 defines the partitioning of primitives used by an APE between its ARM and ASE components. GCC primitives not referenced in this table are processed by the Node Controller.

TABLE 1/T.121

Primitives used by ARM/ASE

Primitive	ARM	ASE	Node Controller
GCC-Registry-Register-Channel	✓		
GCC-Registry-Assign-Token	✓		
GCC-Registry-Set-Parameter	✓		
GCC-Registry-Retrieve-Entry	✓		
GCC-Registry-Delete-Entry	✓		
GCC-Registry-Monitor	✓		
GCC-Registry-Allocate-Handle	✓		
GCC-Conference-Roster-Inquire	✓		✓
GCC-Conductor-Assign	Indication		✓
GCC-Conductor-Release	Indication		✓
GCC-Conductor-Please			✓
GCC-Conductor-Give			✓
GCC-Conductor-Inquire	✓		✓
GCC-Conductor-Permission-Ask	Indication		✓
GCC-Conductor-Permission-Grant	Indication		✓
GCC-Application-Permission-To-Enroll	✓		
GCC-Application-Enroll	✓		
GCC-Application-Roster-Report	✓		✓
GCC-Application-Roster-Inquire	✓		✓
GCC-Application-Invoke	Request		✓
MCS-Attach-User	✓		
MCS-Detach-User	✓		
MCS-Channel-Join	✓		
MCS-Channel-Leave	✓		
MCS-Channel-Convene	✓		
MCS-Channel-Disband	✓		
MCS-Channel-Admit	✓		
MCS-Channel-Expel	✓		
MCS-Send-Data		✓	
MCS-Uniform-Send-Data		✓	
MCS-Token-Grab		✓	
MCS-Token-Inhibit		✓	

TABLE 1/T.121 *(concluded)*

Primitives used by ARM/ASE

Primitive	ARM	ASE	Node Controller
MCS-Token-Give		✓	
MCS-Token-Please		✓	
MCS-Token-Release		✓	
MCS-Token-Test		✓	

6 Generic Application Template

The Generic Application Template (GAT) is designed to provide generic resource management functionality to both standard and non-standard application protocols. As such, it has an Application Resource Manager (the Generic ARM), but does not have an Application Service Element (ASE) – see Figure 5. Separate application protocol specifications detail the operation of their respective ASEs and identify which services offered by the Generic ARM are required by the ASEs. They do not need to describe the operation of the ARM, but should instead reference this Recommendation. The Generic ARM provides all the services described in 5.3.

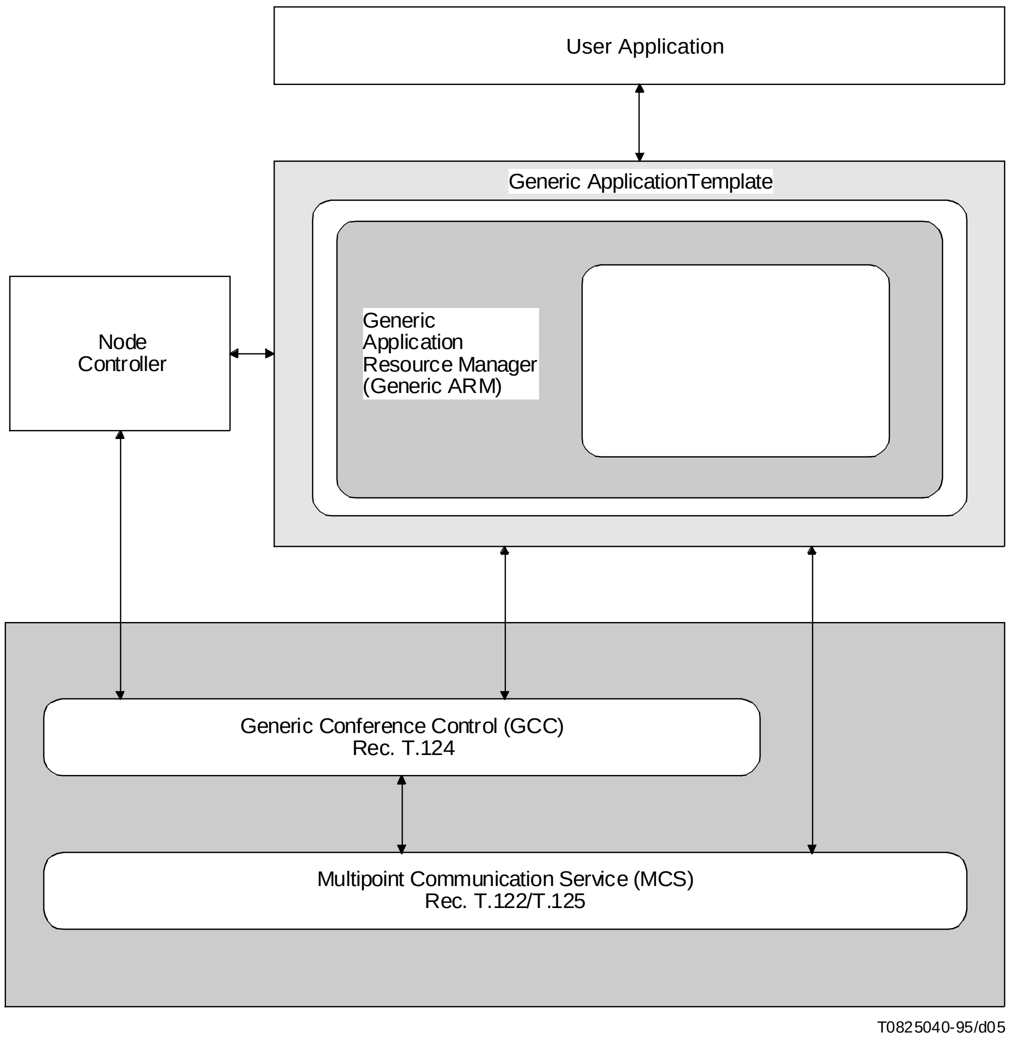

User Application
Generic ApplicationTemplate
Generic Application Resource Manager (Generic ARM)
Node Controller
Generic Conference Control (GCC) Rec. T.124
Multipoint Communication Service (MCS) Rec. T.122/T.125
T0825040-95/d05

FIGURE 5/T.121

Generic Application Template

A description of functionality provided by a Generic ARM is described below.

6.1 Initialization

An ARM is responsible for enrolling its Application Protocol Entity on behalf of the user application, and for obtaining the necessary resources required by the ASE. For each application protocol employed, the user application must specify the following parameters to the corresponding ARM:

- Whether it wishes to enroll or not (no further parameters are required if user application does not wish to enroll).
- Application protocol key.
- Session type (registration, standard base, non-standard base, public or private).
- Session ID (omitted if declaring support of a protocol through the registration session, enrolling in a non-standard base session or creating a new public or private session).
- List of dynamic tokens required.
- List of assigned channels required.
- List of private channels required (and optionally, nodes to be invited to those channels).
- Whether it is able to act as the application session conductor (if there is more than one application at a site participating in the same session, the local GCC provider shall determine which shall become conductor).
- List of application capabilities supported (as defined in the application protocol specification). Collapsing capabilities should be omitted for the registration session, but should be included for all other sessions.

Table 2 specifies the parameters that are passed to the ARM for each application protocol employed by the user application.

6.2 Enrollment

In order to participate in a conference, a user application requires an APE (consisting of an ARM and an ASE) for each protocol it employs. The method of creating APEs is a local matter, outside the scope of this Recommendation. Each APE must then, in turn, establish a GCC SAP to allow it to communicate with the GCC provider at that node; again, how this is achieved is a local matter. Similarly, each APE must establish an MCSAP to allow it to communicate with its local MCS Provider.

When the node joins a conference, or an APE establishes a GCC SAP subsequent to joining one or more conferences, the local GCC provider notifies all APEs at that node by issuing a GCC-Application-Permission-to-Enroll indication with the Grant/Revoke flag set to Grant. For each protocol employed by the user application, the respective ARM shall then issue a GCC-Application-Enroll request, regardless of whether the user application wishes to enroll at that time. If the user application does not wish to enroll an APE, the corresponding ARM shall specify the conference ID and set the Enroll/Un-enroll flag in the GCC-Application-Enroll request to Un-enroll. No other parameters are required. The user application may instruct its ARM(s) to enroll at any time subsequently, unless permission is revoked by receipt of a GCC-Application-Permission-to-Enroll indication with the Grant/Revoke flag set to Revoke.

A user application may wish to receive information on all sessions in progress before deciding whether to participate in an existing session or to create a new session. It may do this by instructing its ARM(s) to enroll Inactive in the registration session as described in 6.2.1.

If the user application wishes to alert peer applications at other nodes to its presence but does not wish to participate in a session immediately, it may instruct its ARM to enroll inactive in the registration session without an MCS User ID. This enables the user application to declare support of one or more protocols without consuming MCS resources.

TABLE 2/T.121

APE parameters

Parameter	Description
Enroll/Un-enroll	This parameter is set to ENROLL if the user application wishes to enroll immediately and to UN-ENROLL if the user application wishes to withdraw enrolment. If set to UN-ENROLL, no further parameters are required.
Application-protocol-key	The Application Protocol Key identifies the Application Protocol used in an Application Protocol Session. Multiple Application Protocol Sessions with the same purpose are 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 T.124 (which uses T.35 country codes like Recommendation H.221).
SessionType	This parameter can have one of the following five values: *registration*: The registration session allows a user application to advertise its presence to the conference and also provides a means of locating the non-standard base session. No sessionID is required *standard base*: This value indicates that the ARM should enroll using a Session Key composed of the application protocol key and the sessionID parameter. It shall use the static predefined channel(s) and static token(s) as defined in the application protocol specification and assigned in Recommendation T.120 as the basic set of resources. This session type is only valid for standardized application protocols *non-standard base:* This value indicates that the ARM should first enroll in the registration session to determine the sessionID assigned to the non-standard base session by consulting the GCC Registry. If a sessionID does not already exist, one is assigned by the ARM. The ARM shall then enroll in the non-standard base session using a Session Key composed of the application protocol key and the sessionID parameter. Members of a non-standard base session may determine Token and Channel IDs via the GCC Registry. This session type is only valid for non-standard application protocols *public*: This value indicates that the ARM should enroll using a Session Key composed of the application protocol key and the sessionID parameter. If the sessionID parameter is omitted, one is assigned by the ARM. All channel and token resources are dynamic and are allocated by the creator of the public session using the MCS-Channel-Join and GCC-Registry-Assign-Token mechanisms respectively. Members of a public session may determine Token and Channel IDs via the GCC Registry *private*: This value indicates that the ARM should enroll using a Session Key composed of the application protocol key and the sessionID parameter. If the sessionID is omitted, one is assigned by the ARM. All channel and token resources are dynamic and are allocated by the Private Convenor ARM using the MCS-Channel-Convene and GCC-Registry-Assign-Token mechanisms respectively. The ARM creating the session may then admit to the channel(s) peer APEs at the nodes whose GCC Node Ids are present in the admitList protocol parameter. A Private Member ARM must wait for its APE to be admitted by the convenor of the private MCS channels before attempting to join them
SessionID	This parameter is used to differentiate the resources used by multiple sessions of the protocol that may be in existence simultaneously within the same MCS domain. The sessionID must be specified if the user application wishes to participate in a standard base session or in an existing public or private session. It is omitted if the user application wishes to create a new public or private session, participate in a non-standard base session, or if the user application wishes to enroll in the registration session without consuming MCS resources. With the exception of the registration session, the MCS Channel ID assigned as a start-up channel is used as the sessionID, since this is guaranteed to be unique within the conference domain.

TABLE 2/T.121 *(concluded)*

APE parameters

Parameter	Description
AdmitList	SessionType = private and sessionID omitted List of GCC NodeIDs corresponding to the Nodes at which peer APEs are to be admitted to the privately convened channels. Else Omitted
ResourceList	This is a list of resources (channels, tokens, registry parameters and handles) required by the user application. SessionType = standard base: Resource List comprises the MCS IDs of static channels to be joined by the APE and static tokens to be used by the APE. It may optionally contain a list of Resource Ids for determining the MCS IDs of dynamic channels and tokens via the GCC Registry. It may also specify the number of Registry handles required and include a list of Resource Ids for Registry parameters and value of those parameters (where appropriate). SessionType = public, private, or non-standard base: Resource List comprises a list of Resource Ids for determining the MCS IDs of dynamic channels and tokens via the GCC Registry. It may also specify the number of Registry handles required and include a list of Resource Ids for Registry parameters and value of those parameters (where appropriate). If the Session ID is omitted (i.e. the APE is the session convenor), the ARM is responsible for assigning and registering all channels and tokens specified in the Resource List. If the Session ID is present (i.e. the APE is a session member), the ARM is responsible for retrieving all channels and tokens. In both cases, the Resource ID and associated MCS Channel ID or Token ID are passed by the ARM to its ASE. SessionType = registration Omitted
Conducting Operation Capable	Set if the APE is capable of becoming the session Conductor in conducted mode
Non-Collapsing Capabilities List	Any non-collapsing capabilities specified by the application protocol and supported by the ASE
Application Capability List	Any collapsing capabilities specified by the application protocol and supported by the ASE
Expected Capability List	SessionType = private and sessionID omitted: Indicates any application capabilities that must be supported by invoked APEs, allowing the creator of a private session to specify a minimum capability set for that session. Else Omitted

When the application decides to become active, each ARM must adopt an MCS User ID. A new MCS User ID is acquired by the ARM issuing an MCS-Attach-User request to the MCS provider, using the Conference ID contained in the GCC-Application-Permission-to-Enroll indication as the Domain Selector. On receipt of a successful MCS-Attach-User confirm in response, an ARM shall join the User ID channel indicated by issuing an MCS-Channel-Join request and supply the User ID to the corresponding ASE.

Subsequenct operations are dependent on the session type. Note that the processes described need to be repeated for each APE employed by a user application. Observe that a common feature of the model templates (with the exception of the registration session template) is that the Session ID channel is the start-up channel, its type determines the session type, and it is always joined during the initiation sequence of an APE.

The procedures defined for session enrollment and joining the start-up channel must be followed in the order specified for each of the session types described. If an application protocol does not require the use of any further resources (channels, tokens or registry parameters), an ARM may omit the inactive enrollment operation. Where an application protocol makes use of tokens, registry parameters or additional dynamic channels, this Recommendation imposes no requirement on the order of assigning or identifying those resources. If an application protocol makes use of GCC handles, these may be allocated during the application enrollment process and at any time subsequently during the lifetime of a session.

To minimize the initialization period it is permissible to identify multiple channels in parallel by interleaving the operations required for each channel. For example, a session creator may choose to convene all required channels and then register all those channels, rather than convening and registering each channel separately. For any given channel, however, the sequence of operations defined in this Recommendation must be followed.

An APE can only be considered to be part of a session when its ARM receives a GCC-Application-Roster-Report indication in which its Entity ID is present. An APE shall not submit data to a session until it receives such a roster report indication. However, APEs should be prepared to buffer all data received for a session between enrolling active in a session and receiving a roster report in which the APE Entity ID is present. This is because another APE may have received a roster report indication containing the new participant and submit data to the session which reaches the new participant before the corresponding roster report. Application protocols may specify how this buffered data is to be processed.

6.2.1 Registration session

If the user application wishes to alert peer applications at other nodes to its presence, but does not wish to participate in a session immediately, it may instruct its ARM to enroll in the registration session without an MCS User ID. This enables the user application to declare support of one or more application protocols without consuming MCS resources. It also receives information on all sessions in progress for each of the application protocols advertised. Each ARM issues a GCC-Application-Enroll request with the Enroll/Un-enroll flag set to Enroll and the Active/Inactive flag set to Inactive, specifying the Session Key of its respective protocol without a Session ID. See Figure 6.

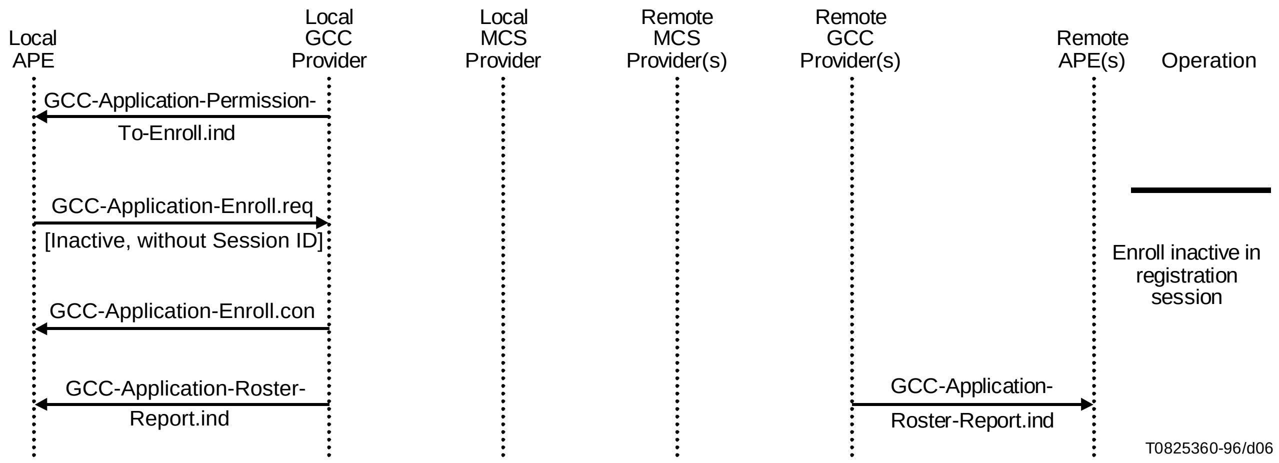

Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
GCC-Application-Enroll.req [Inactive, without Session ID]
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
Enroll inactive in registration session
T0825360-96/d06

FIGURE 6/T.121

Enrolling in the registration session to declare support of a protocol

6.2.2 Standard base session

After joining its User ID Channel, the ARM shall join static channels as specified in the corresponding application protocol specification. Once positive confirmation of joining these channels has been received, the ARM shall enroll the APE Inactive by issuing a GCC-Application-Enroll request to the GCC provider, with the parameters specified in Table 3.

After receiving a GCC-Application-Roster-Report indication with an entry corresponding to its APE, the ARM shall assign any dynamic assigned channels mandated by the protocol specification. The ARM shall issue a GCC-Registry-Retrieve-Entry request primitive for each such channel to determine whether it has already been registered. If the Result parameter returned in the GCC-Registry-Retrieve-Entry confirm is "successful", then the ARM shall attempt to join the channel indicated in the Registry Item parameter by issuing an MCS-Channel-Join request. If the Result parameter returned in the GCC-Registry-Retrieve-Entry is "entry not found", the ARM shall attempt to assign a new channel by issuing an MCS-Channel-Join request with Channel ID = 0. The returned MCS-Channel-Join confirm, if successful, contains the assigned Channel ID which the ARM will attempt to register in the standard base session by issuing a GCC-Registry-Register-Channel request. On receipt of the GCC-Registry-Register-Channel confirm, the ARM shall examine the result parameter. If the result is "successful", the ARM shall take no further action. If the result is "index already exists", then the channel has already been registered by another APE; its Channel ID is provided in the Registry Item parameter. In this case, the ARM shall first leave the channel it had previously assigned by issuing an MCS-Channel-Leave request specifying the Channel ID returned in the MCS-Channel-Join confirm. It shall then join the channel indicated in the Registry Item parameter by issuing an MCS-Channel-Join request.

If the application protocol mandates that an APE must identify any dynamic tokens before interacting with its peers, the ARM may determine their identity via the GCC Registry. For each Token Resource ID specified by the application protocol, the ARM shall issue a GCC-Registry-Assign-Token request to the GCC provider using the parameters specified in Table 7. If the Result parameter returned in the GCC-Registry-Assign-Token confirm is "successful" or "index already exists" the Token ID contained in the Registry Item of the returned confirm primitive is used as the Token ID for the token corresponding to the Resource ID used in the Registry Key. The ARM shall pass the Resource ID and associated Token ID to its ASE.

If the application protocol requires an APE to assign or identify registry parameters before interacting with its peers, the ARM shall issue a GCC-Registry-Set-Parameter request to the GCC Provider.

Once all resources required for the session have been identified, the ARM shall then enroll Active by issuing a GCC-Application-Enroll request. The Active/Inactive flag shall be set to Active, the Session ID shall be specified as part of the Session Key, the Start-Up Channel shall be specified as Static and the capability lists shall be provided. See Figures 7 and 8.

6.2.3 Non-standard base session

The registration session may be used to provide access to a non-standard base session. After joining its User ID Channel, the ARM shall enroll in the registration session by issuing a GCC-Application-Enroll request with the Active/Inactive flag set to Inactive and specifying the Session Key with no Session ID.

After receiving a GCC-Application-Roster-Report indication with an entry corresponding to its APE, the ARM shall attempt to create the non-standard base session by first issuing an MCS-Channel-Join request with Channel ID = 0. The returned MCS-Channel-Join confirm, if successful, contains the assigned Channel ID which the ARM will attempt to register in the registration session as the start-up channel for the non-standard base session. The ARM shall issue a GCC-Registry-Register-Channel request using a Registry Key comprised of the Session Key (with no Session ID) and the Resource ID "BASE" encoded as a four consecutive octets according to Recommendation T.50. On receipt of the GCC-Registry-Register-Channel confirm, the ARM shall examine the result parameter.

If the result is "index already exists", then the non-standard base session has already been registered by another APE; its Session ID is provided in the Registry Item parameter. In this case, the ARM shall first leave the channel it had previously assigned by issuing an MCS-Channel-Leave request specifying the Channel ID returned in the MCS-Channel-Join confirm. The ARM shall then assume the role of session member and enroll in the existing base session by following the process described for a public session member in 6.2.4. This process is illustrated in Figure 9. An attempt to join the start-up channel identified in the GCC-Registry-Register-Channel confirm may fail if the Registry Item parameter points to a defunct non-standard session (i.e. the session's start-up channel no longer exists); in such circumstances the ARM may attempt to reuse this defunct session by issuing a GCC-Registry-Delete-Entry request, using the same Registry Key as before. On receipt of the corresponding GCC-Registry-Delete-Entry confirm, the ARM shall then examine the result parameter and if it is "successful", the ARM may repeat the non-standard base session creation process.

If the GCC-Registry-Register-Channel confirm result parameter is "successful", the ARM shall assume the responsibility of creator for the non-standard base session as illustrated in Figure 10. It shall enroll Inactive in the new base session as specified in Table 3.

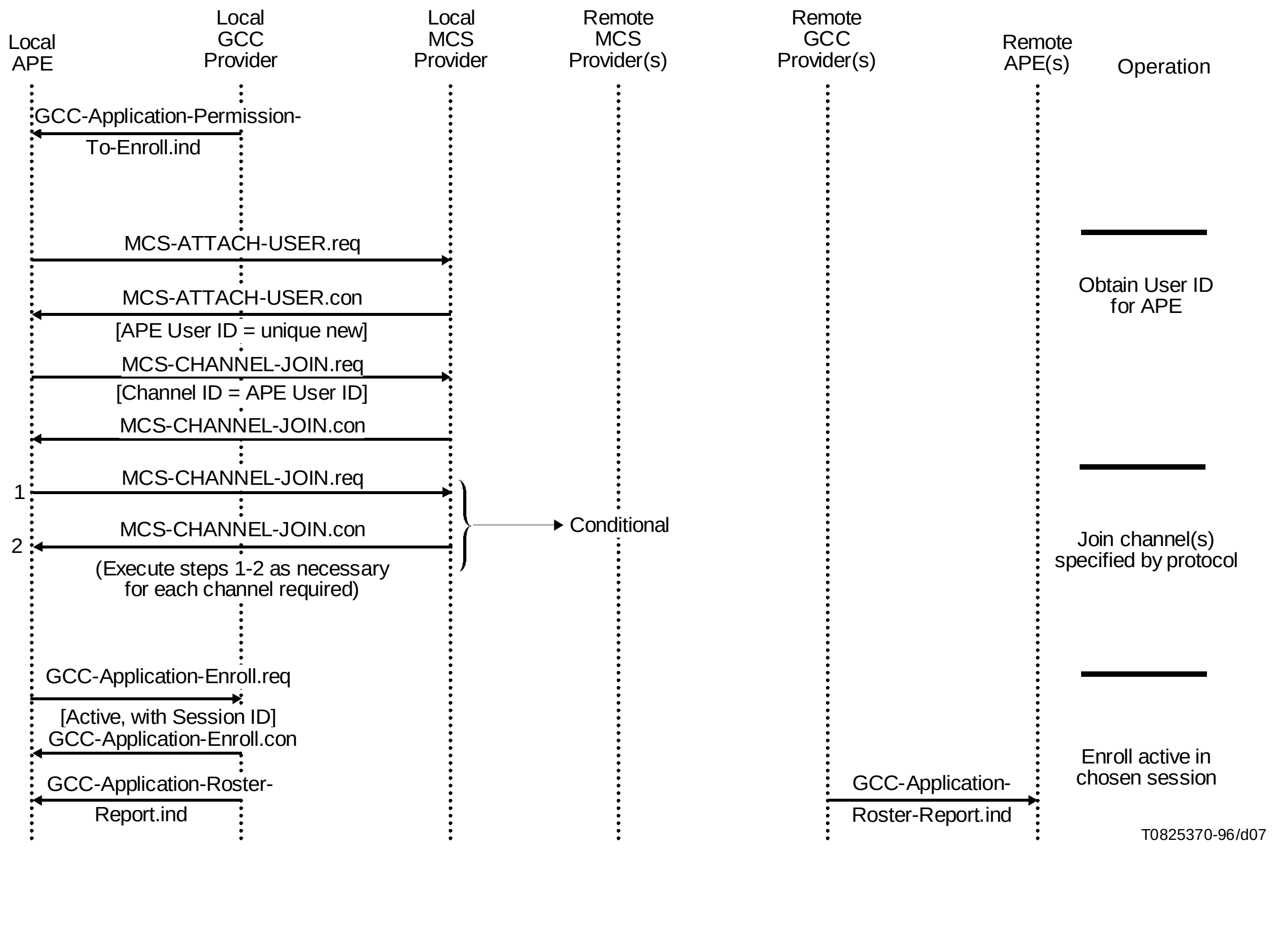
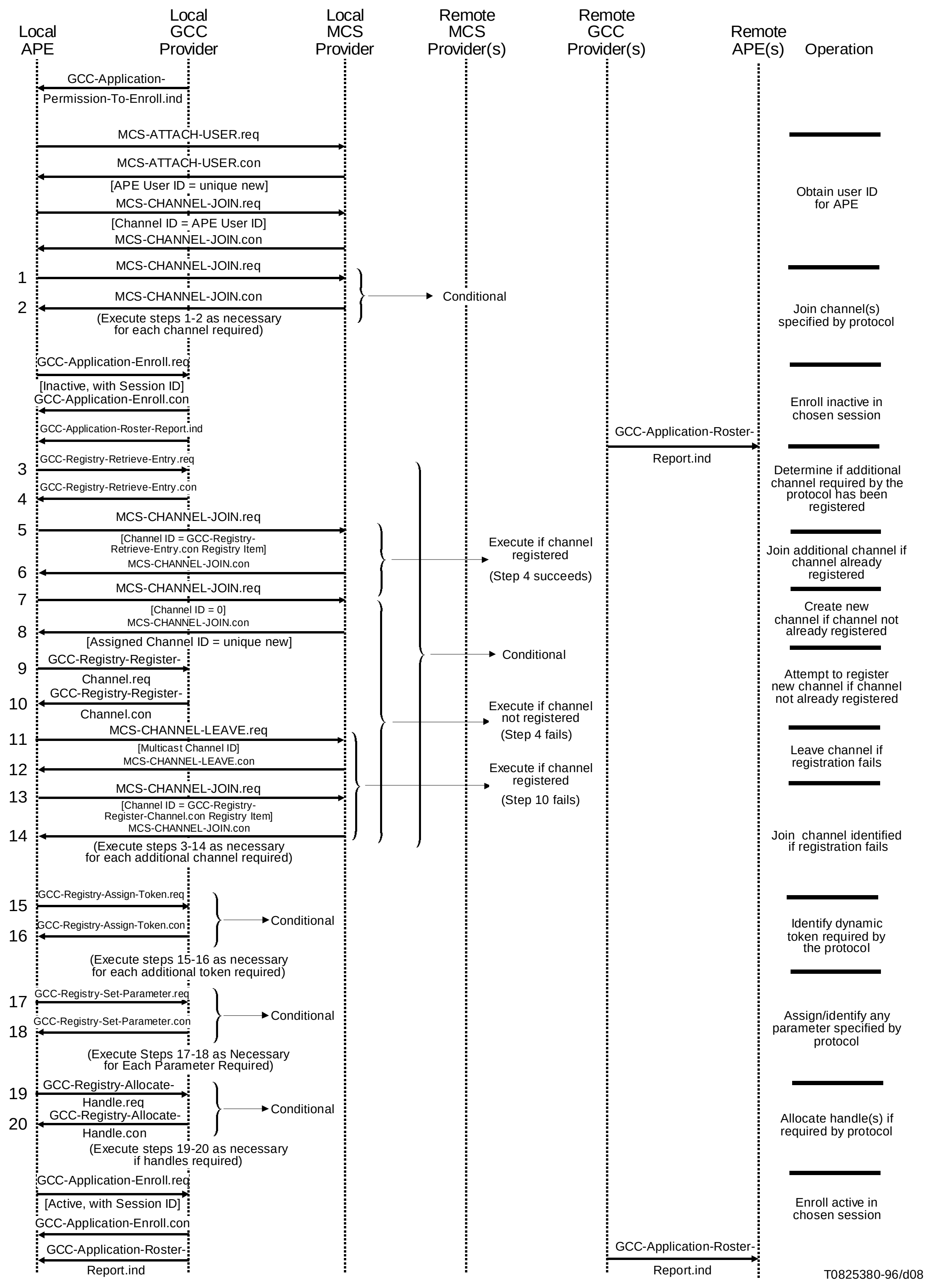

Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
Obtain User ID for APE
1
MCS-CHANNEL-JOIN.req
2
MCS-CHANNEL-JOIN.con
(Execute steps 1-2 as necessary for each channel required)
Conditional
Join channel(s) specified by protocol
GCC-Application-Enroll.req
[Active, with Session ID]
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
Enroll active in chosen session
T0825370-96/d07

FIGURE 7/T.121

Example standard base session protocol initiation sequence case 1: using only static resources



Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
Obtain user ID for APE
1
MCS-CHANNEL-JOIN.req
2
MCS-CHANNEL-JOIN.con
Conditional
(Execute steps 1-2 as necessary for each channel required)
Join channel(s) specified by protocol
GCC-Application-Enroll.req
[Inactive, with Session ID]
GCC-Application-Enroll.con
Enroll inactive in chosen session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
3
GCC-Registry-Retrieve-Entry.req
4
GCC-Registry-Retrieve-Entry.con
Determine if additional channel required by the protocol has been registered
5
MCS-CHANNEL-JOIN.req
[Channel ID = GCC-Registry-Retrieve-Entry.con Registry Item]
6
MCS-CHANNEL-JOIN.con
Execute if channel registered
(Step 4 succeeds)
Join additional channel if channel already registered
7
MCS-CHANNEL-JOIN.req
[Channel ID = 0]
8
MCS-CHANNEL-JOIN.con
[Assigned Channel ID = unique new]
Create new channel if channel not already registered
Conditional
9
GCC-Registry-Register-Channel.req
10
GCC-Registry-Register-Channel.con
Attempt to register new channel if channel not already registered
Execute if channel not registered
(Step 4 fails)
11
MCS-CHANNEL-LEAVE.req
[Multicast Channel ID]
12
MCS-CHANNEL-LEAVE.con
Leave channel if registration fails
Execute if channel registered
(Step 10 fails)
13
MCS-CHANNEL-JOIN.req
[Channel ID = GCC-Registry-Register-Channel.con Registry Item]
14
MCS-CHANNEL-JOIN.con
(Execute steps 3-14 as necessary for each additional channel required)
Join channel identified if registration fails
15
GCC-Registry-Assign-Token.req
16
GCC-Registry-Assign-Token.con
Conditional
(Execute steps 15-16 as necessary for each additional token required)
Identify dynamic token required by the protocol
17
GCC-Registry-Set-Parameter.req
18
GCC-Registry-Set-Parameter.con
Conditional
(Execute Steps 17-18 as Necessary for Each Parameter Required)
Assign/identify any parameter specified by protocol
19
GCC-Registry-Allocate-Handle.req
20
GCC-Registry-Allocate-Handle.con
Conditional
(Execute steps 19-20 as necessary if handles required)
Allocate handle(s) if required by protocol
GCC-Application-Enroll.req
[Active, with Session ID]
GCC-Application-Enroll.con
Enroll active in chosen session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
T0825380-96/d08

FIGURE 8/T.121

Example standard base session protocol initiation sequence case 2: using static and dynamic resources

TABLE 3/T.121

Parameters for GCC-Application-Enroll Request

Parameter	Contents	
Conference ID	Provided by GCC-Application-Permission-To-Enroll indication	
Session Key	Comprised of the Application Protocol key and the Session ID	
Application User ID	Provided by MCS-Attach-User confirm	
Active/Inactive	Active when indicating that the ARM has identified the set of resources that need to be identified as part of the enrollment process and joined all channels required by the protocol. Inactive for the registration session and in the case of standard base, non-standard base, public or private sessions when enrolling prior to identifying the set of resources that need to be identified as part of the enrollment process	
Conducting Operation Capable Flag	Set if the APE is capable of becoming the session Conductor in conducted mode. This flag must not be set if the Active/Inactive flag is set to Inactive	
Start-Up Channel	Optional parameter specifying the type of session as follows:	
	Session Type:	Start-Up Channel:
	Registration	*Omitted*
	Standard Base	*Static*
	Non-Standard Base	*Dynamic Multicast*
	Public	*Dynamic Multicast*
	Private	*Dynamic Private*
Non-Collapsing Capabilities List	Any non-collapsing capabilities specified by the application protocol and supported by the ASE	
Application Capability List	Any collapsing capabilities specified by the application protocol and supported by the ASE. Omitted if Active/Inactive flag is set to Inactive	
Enroll/Un-enroll	ENROLL	

If any additional channels are mandated by the protocol specification, the ARM shall issue a GCC-Registry-Retrieve-Entry request primitive for each such channel to determine whether it has already been registered. The Registry Key used to register these resources shall include the Session ID of the non-standard base session as part of the Session Key. If the Result parameter returned in the GCC-Registry-Retrieve-Entry confirm is "successful", then the ARM shall attempt to join the channel indicated in the Registry Item parameter by issuing an MCS-Channel-Join request. If the Result parameter returned in the GCC-Registry-Retrieve-Entry is "entry not found", the ARM shall attempt to assign a new channel by issuing an MCS-Channel-Join request with Channel ID = 0. The returned MCS-Channel-Join confirm, if successful, contains the assigned Channel ID which the ARM will attempt to register in the non-standard base session by issuing a GCC-Registry-Register-Channel request. On receipt of the GCC-Registry-Register-Channel confirm, the ARM shall examine the result parameter. If the result is "successful", the ARM shall take no further action. If the result is "index already exists", then the channel has already been registered by another APE; its Channel ID is provided in the Registry Item parameter. In this case, the ARM shall first leave the channel it had previously assigned by issuing an MCS-Channel-Leave request specifying the Channel ID returned in the MCS-Channel-Join confirm. It shall then join the channel indicated in the Registry Item parameter by issuing an MCS-Channel-Join request.

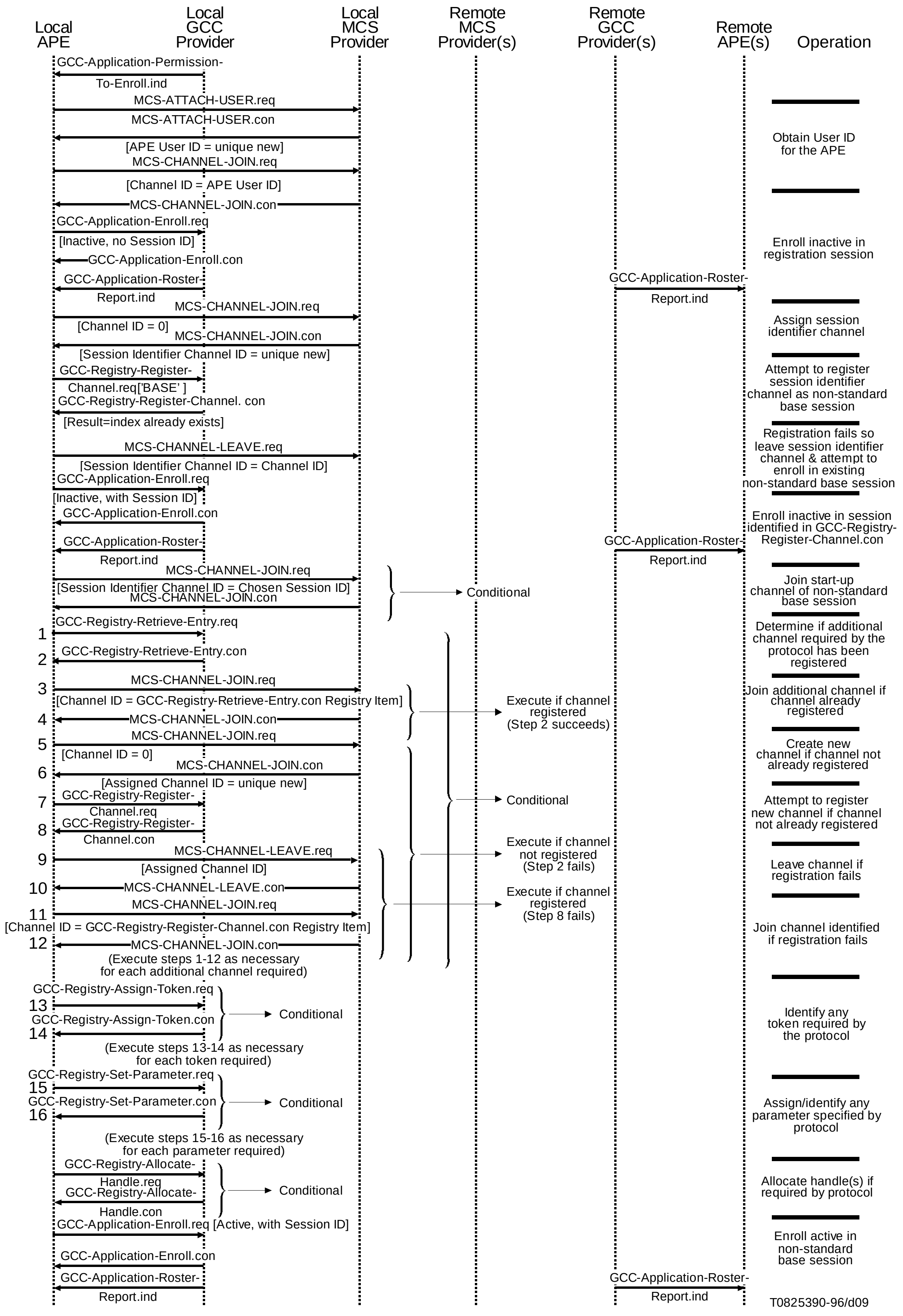

Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
Obtain User ID for the APE
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
GCC-Application-Enroll.req
[Inactive, no Session ID]
Enroll inactive in registration session
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
MCS-CHANNEL-JOIN.req
Assign session identifier channel
[Channel ID = 0]
MCS-CHANNEL-JOIN.con
[Session Identifier Channel ID = unique new]
GCC-Registry-Register-Channel.req['BASE']
Attempt to register session identifier channel as non-standard base session
GCC-Registry-Register-Channel. con
[Result=index already exists]
Registration fails so leave session identifier channel & attempt to enroll in existing non-standard base session
MCS-CHANNEL-LEAVE.req
[Session Identifier Channel ID = Channel ID]
GCC-Application-Enroll.req
[Inactive, with Session ID]
GCC-Application-Enroll.con
Enroll inactive in session identified in GCC-Registry-Register-Channel.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
MCS-CHANNEL-JOIN.req
[Session Identifier Channel ID = Chosen Session ID]
MCS-CHANNEL-JOIN.con
Conditional
Join start-up channel of non-standard base session
1
GCC-Registry-Retrieve-Entry.req
Determine if additional channel required by the protocol has been registered
2
GCC-Registry-Retrieve-Entry.con
3
MCS-CHANNEL-JOIN.req
[Channel ID = GCC-Registry-Retrieve-Entry.con Registry Item]
Execute if channel registered (Step 2 succeeds)
Join additional channel if channel already registered
4
MCS-CHANNEL-JOIN.con
5
MCS-CHANNEL-JOIN.req
[Channel ID = 0]
Create new channel if channel not already registered
6
MCS-CHANNEL-JOIN.con
[Assigned Channel ID = unique new]
7
GCC-Registry-Register-Channel.req
Conditional
Attempt to register new channel if channel not already registered
8
GCC-Registry-Register-Channel.con
9
MCS-CHANNEL-LEAVE.req
[Assigned Channel ID]
Execute if channel not registered (Step 2 fails)
Leave channel if registration fails
10
MCS-CHANNEL-LEAVE.con
11
MCS-CHANNEL-JOIN.req
Execute if channel registered (Step 8 fails)
[Channel ID = GCC-Registry-Register-Channel.con Registry Item]
Join channel identified if registration fails
12
MCS-CHANNEL-JOIN.con
(Execute steps 1-12 as necessary for each additional channel required)
GCC-Registry-Assign-Token.req
13
GCC-Registry-Assign-Token.con
14
Conditional
Identify any token required by the protocol
(Execute steps 13-14 as necessary for each token required)
GCC-Registry-Set-Parameter.req
15
GCC-Registry-Set-Parameter.con
16
Conditional
Assign/identify any parameter specified by protocol
(Execute steps 15-16 as necessary for each parameter required)
GCC-Registry-Allocate-Handle.req
GCC-Registry-Allocate-Handle.con
Conditional
Allocate handle(s) if required by protocol
GCC-Application-Enroll.req [Active, with Session ID]
Enroll active in non-standard base session
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
T0825390-96/d09

FIGURE 9/T.121

Example non-standard base session protocol initiation sequence case 1: base session already registered

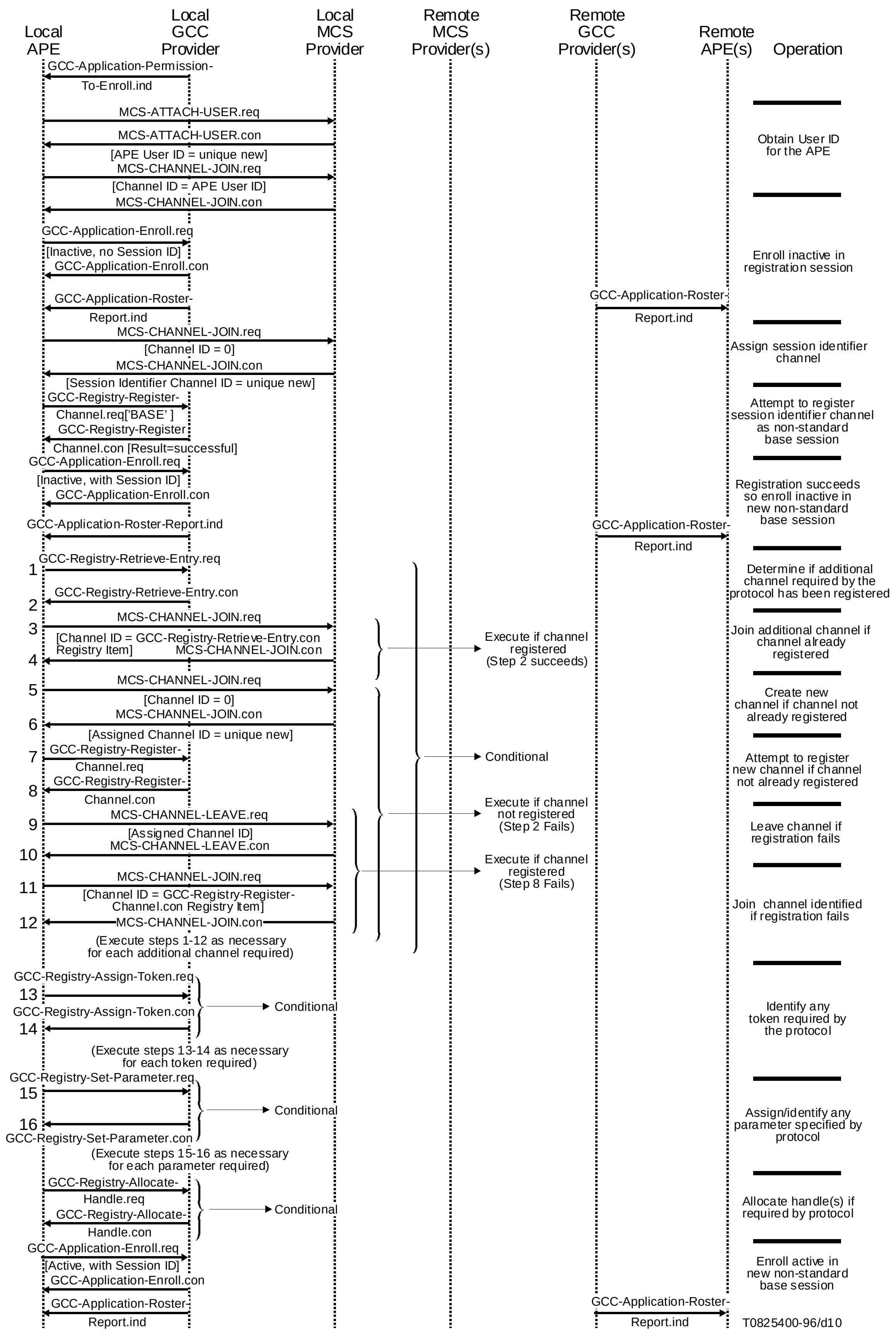

Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
Obtain User ID for the APE
GCC-Application-Enroll.req
[Inactive, no Session ID]
GCC-Application-Enroll.con
Enroll inactive in registration session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
MCS-CHANNEL-JOIN.req
[Channel ID = 0]
MCS-CHANNEL-JOIN.con
[Session Identifier Channel ID = unique new]
Assign session identifier channel
GCC-Registry-Register-Channel.req['BASE']
GCC-Registry-Register Channel.con [Result=successful]
Attempt to register session identifier channel as non-standard base session
GCC-Application-Enroll.req
[Inactive, with Session ID]
GCC-Application-Enroll.con
Registration succeeds so enroll inactive in new non-standard base session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
1 GCC-Registry-Retrieve-Entry.req
2 GCC-Registry-Retrieve-Entry.con
Determine if additional channel required by the protocol has been registered
3 MCS-CHANNEL-JOIN.req
[Channel ID = GCC-Registry-Retrieve-Entry.con Registry Item]
4 MCS-CHANNEL-JOIN.con
Execute if channel registered (Step 2 succeeds)
Join additional channel if channel already registered
5 MCS-CHANNEL-JOIN.req
[Channel ID = 0]
6 MCS-CHANNEL-JOIN.con
[Assigned Channel ID = unique new]
Create new channel if channel not already registered
Conditional
7 GCC-Registry-Register-Channel.req
8 GCC-Registry-Register-Channel.con
Attempt to register new channel if channel not already registered
Execute if channel not registered (Step 2 Fails)
9 MCS-CHANNEL-LEAVE.req
[Assigned Channel ID]
10 MCS-CHANNEL-LEAVE.con
Leave channel if registration fails
Execute if channel registered (Step 8 Fails)
11 MCS-CHANNEL-JOIN.req
[Channel ID = GCC-Registry-Register-Channel.con Registry Item]
12 MCS-CHANNEL-JOIN.con
Join channel identified if registration fails
(Execute steps 1-12 as necessary for each additional channel required)
13 GCC-Registry-Assign-Token.req
14 GCC-Registry-Assign-Token.con
Conditional
(Execute steps 13-14 as necessary for each token required)
Identify any token required by the protocol
15 GCC-Registry-Set-Parameter.req
16 GCC-Registry-Set-Parameter.con
Conditional
(Execute steps 15-16 as necessary for each parameter required)
Assign/identify any parameter specified by protocol
GCC-Registry-Allocate-Handle.req
GCC-Registry-Allocate-Handle.con
Conditional
Allocate handle(s) if required by protocol
GCC-Application-Enroll.req
[Active, with Session ID]
GCC-Application-Enroll.con
Enroll active in new non-standard base session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
T0825400-96/d10

FIGURE 10/T.121

Example non-standard base session protocol initiation sequence case 2: base session not already registered

If the application protocol mandates that an APE must know the identity of tokens before interacting with its peers, the ARM may determine their identity via the GCC Registry. For each Token Resource ID specified by the application protocol, the ARM shall issue a GCC-Registry-Assign-Token request to the GCC provider using the parameters specified in Table 7. If the Result parameter returned in the GCC-Registry-Assign-Token confirm is "successful" or "index already exists" the Token ID contained in the Registry Item of the returned confirm primitive is used as the Token ID for the token corresponding to the Resource ID used in the Registry Key. The ARM shall pass the Resource ID and associated Token ID to its ASE.

If the application protocol requires an APE to assign or identify registry parameters before interacting with its peers, the ARM shall issue a GCC-Registry-Set-Parameter request to the GCC Provider.

Once all resources required for the session have been identified, the ARM shall then enroll Active by issuing a GCC-Application-Enroll request with the parameters specified in Table 3.

6.2.4 Public session

After joining its User ID Channel, the ARM shall examine the session ID parameter to determine, if it is present, to participate in an existing public session (as a public member) or, if it is omitted, to create a new one (as a public creator). See Figures 11 and 12.

If the sessionID parameter is present, the ARM shall enroll in the session indicated by issuing a GCC-Application-Enroll request with the Active/Inactive flag set to Inactive and specifying the Session Key with the requisite Session ID. It shall then issue an MCS-Channel-Join request, specifying the Session ID of the chosen session as the Channel ID parameter.

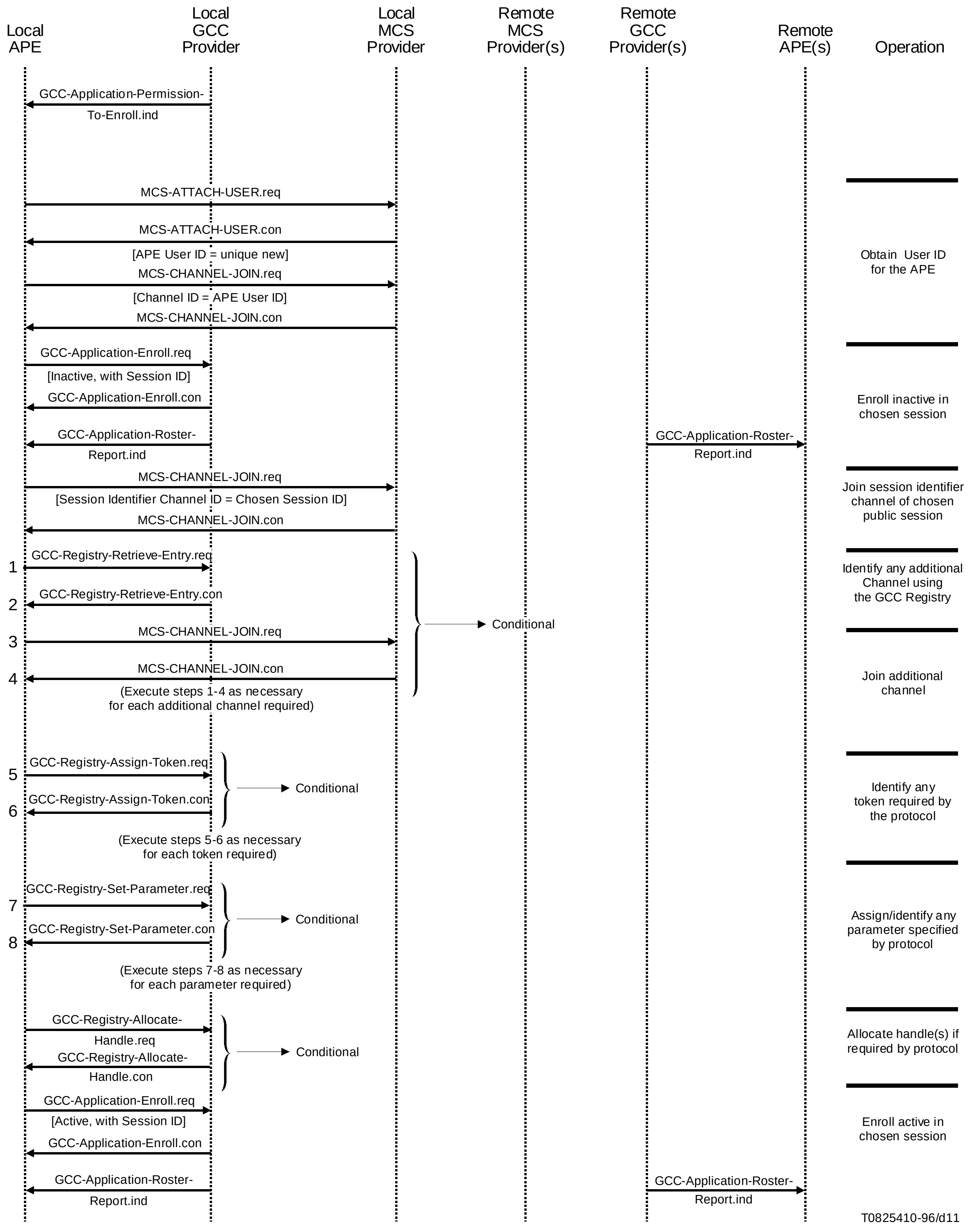
If the application protocol requires multiple channels to be joined before an APE can be considered to be an active participant in a session, then the ARM shall attempt to identify and join these channels using the services of the GCC Registry and MCS respectively. Further channels may be joined as required when the application session is in progress. Likewise, if an application protocol mandates that an APE must know the identity of tokens or registry parameters before interacting with its peers, the ARM may determine their identity via the GCC Registry.

Once all resources required for the session have been identified, the ARM shall then enroll Active in the chosen session by issuing a GCC-Application-Enroll request with the parameters specified in Table 3.

If the sessionID parameter is omitted, the ARM shall enroll in a new public session. The ARM shall first allocate an assigned channel for the session by issuing an MCS-Channel-Join request with Channel ID = 0. The returned MCS-Channel-Join confirm, if successful, contains the assigned Channel ID. This Channel ID is also used as the Session ID for the APE. The ARM shall then enroll Inactive in this session by issuing a GCC-Application-Enroll request with the parameters specified in Table 3.

The onus of acquiring the set of resources that need to be identified as part of the enrollment process for a public session rests with the ARM that created the session. If any additional channels, registry parameters or tokens are mandated by the protocol specification, the ARM should assign these and register them with GCC as necessary. The Registry Key used to register these resources shall include the Session ID as part of the Session Key. Note that the ARM must not attempt to use the GCC-Registry until it has received a GCC-Application-Roster-Report indication containing an entry corresponding to its own APE.

Once all resources have been successfully assigned, the ARM shall enroll Active by issuing a GCC-Application-Enroll request with the parameters specified in Table 3.



Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
Obtain User ID for the APE
GCC-Application-Enroll.req
[Inactive, with Session ID]
GCC-Application-Enroll.con
Enroll inactive in chosen session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
MCS-CHANNEL-JOIN.req
[Session Identifier Channel ID = Chosen Session ID]
MCS-CHANNEL-JOIN.con
Join session identifier channel of chosen public session
1 GCC-Registry-Retrieve-Entry.req
2 GCC-Registry-Retrieve-Entry.con
3 MCS-CHANNEL-JOIN.req
4 MCS-CHANNEL-JOIN.con
Conditional
(Execute steps 1-4 as necessary for each additional channel required)
Identify any additional Channel using the GCC Registry
Join additional channel
5 GCC-Registry-Assign-Token.req
6 GCC-Registry-Assign-Token.con
Conditional
(Execute steps 5-6 as necessary for each token required)
Identify any token required by the protocol
7 GCC-Registry-Set-Parameter.req
8 GCC-Registry-Set-Parameter.con
Conditional
(Execute steps 7-8 as necessary for each parameter required)
Assign/identify any parameter specified by protocol
GCC-Registry-Allocate-Handle.req
GCC-Registry-Allocate-Handle.con
Conditional
Allocate handle(s) if required by protocol
GCC-Application-Enroll.req
[Active, with Session ID]
GCC-Application-Enroll.con
Enroll active in chosen session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
T0825410-96/d11

FIGURE 11/T.121

Example public session protocol initiation sequence case 1: session member

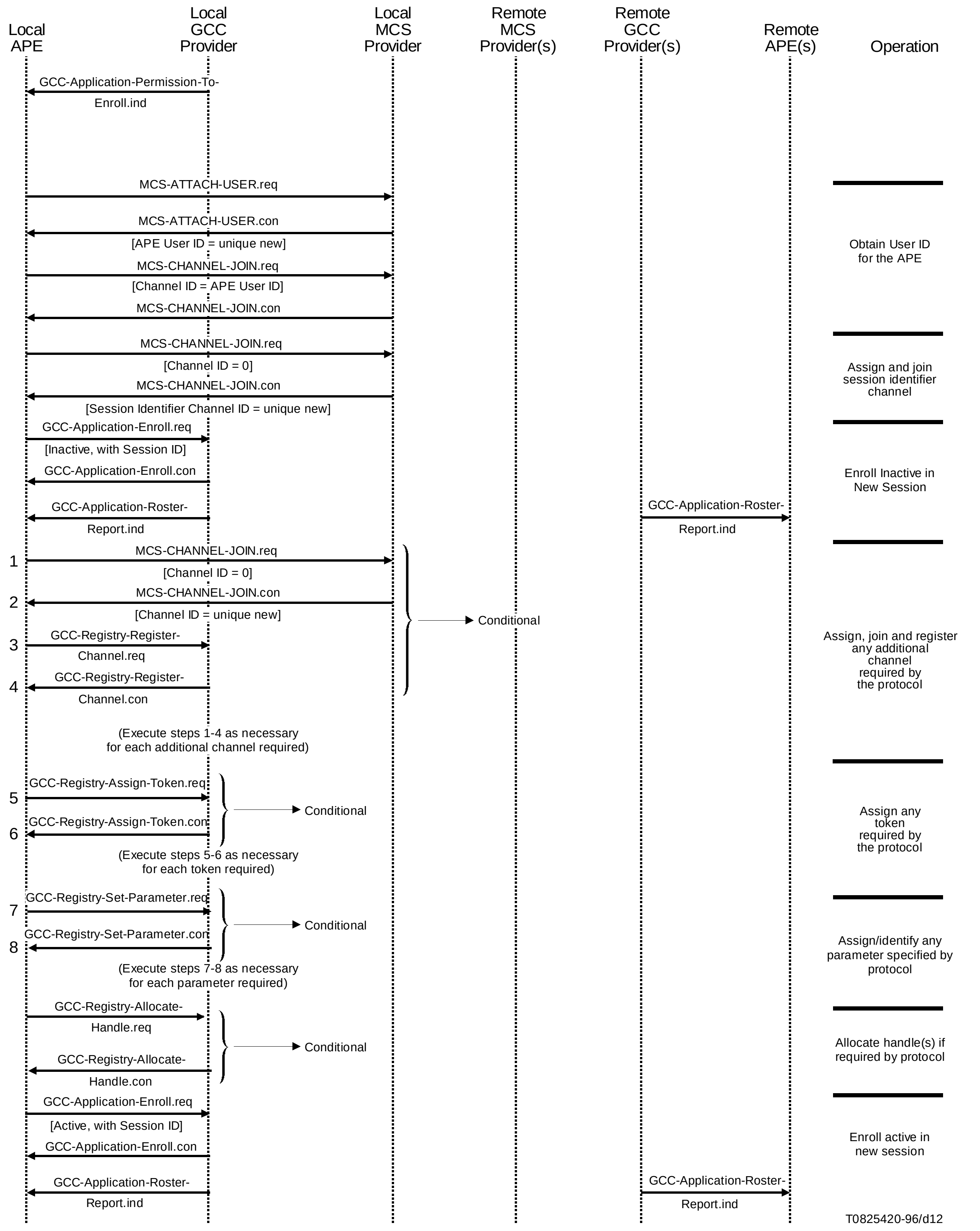

Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
Obtain User ID for the APE
MCS-CHANNEL-JOIN.req
[Channel ID = 0]
MCS-CHANNEL-JOIN.con
[Session Identifier Channel ID = unique new]
Assign and join session identifier channel
GCC-Application-Enroll.req
[Inactive, with Session ID]
GCC-Application-Enroll.con
Enroll Inactive in New Session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
1 MCS-CHANNEL-JOIN.req
[Channel ID = 0]
2 MCS-CHANNEL-JOIN.con
[Channel ID = unique new]
3 GCC-Registry-Register-Channel.req
4 GCC-Registry-Register-Channel.con
Conditional
Assign, join and register any additional channel required by the protocol
(Execute steps 1-4 as necessary for each additional channel required)
5 GCC-Registry-Assign-Token.req
6 GCC-Registry-Assign-Token.con
Conditional
(Execute steps 5-6 as necessary for each token required)
Assign any token required by the protocol
7 GCC-Registry-Set-Parameter.req
8 GCC-Registry-Set-Parameter.con
Conditional
(Execute steps 7-8 as necessary for each parameter required)
Assign/identify any parameter specified by protocol
GCC-Registry-Allocate-Handle.req
GCC-Registry-Allocate-Handle.con
Conditional
Allocate handle(s) if required by protocol
GCC-Application-Enroll.req
[Active, with Session ID]
GCC-Application-Enroll.con
Enroll active in new session
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
T0825420-96/d12

FIGURE 12/T.121

Example public session protocol initiation sequence case 2: session creator

6.2.5 Private session

After joining its User ID Channel, the ARM shall examine the sessionID parameter to determine, if it is present, to participate in an existing private session (as a private member) or, if it is omitted, to create a new session (as a private convenor). See Figure 13.

If the sessionID parameter is present, the ARM shall enroll Inactive in the session indicated by issuing a GCC-Application-Enroll request with the Active/Inactive flag set to Inactive and specifying the Session Key with the requisite Session ID. The ARM shall then wait until it has received an MCS-Channel-Admit indication from the peer ARM at the node convening the private session. It then attempts to join the channel indicated in this primitive by issuing an MCS-Channel-Join request and notifies the corresponding ASE if the channel join succeeded. The Channel ID is used as the session ID.

If the application protocol requires multiple channels to be joined before an APE can be considered to be an active participant in a session, then the ARM shall wait to be admitted to those channels and then join them accordingly. Further channels may be joined as required when the application session is in progress. Likewise, if an application protocol mandates that an APE must know the identity of tokens or registry parameters before interacting with its peers, the ARM may determine their identity via the GCC Registry.

The ARM shall then enroll Active by issuing a GCC-Application-Enroll request with the parameters specified in Table 3.

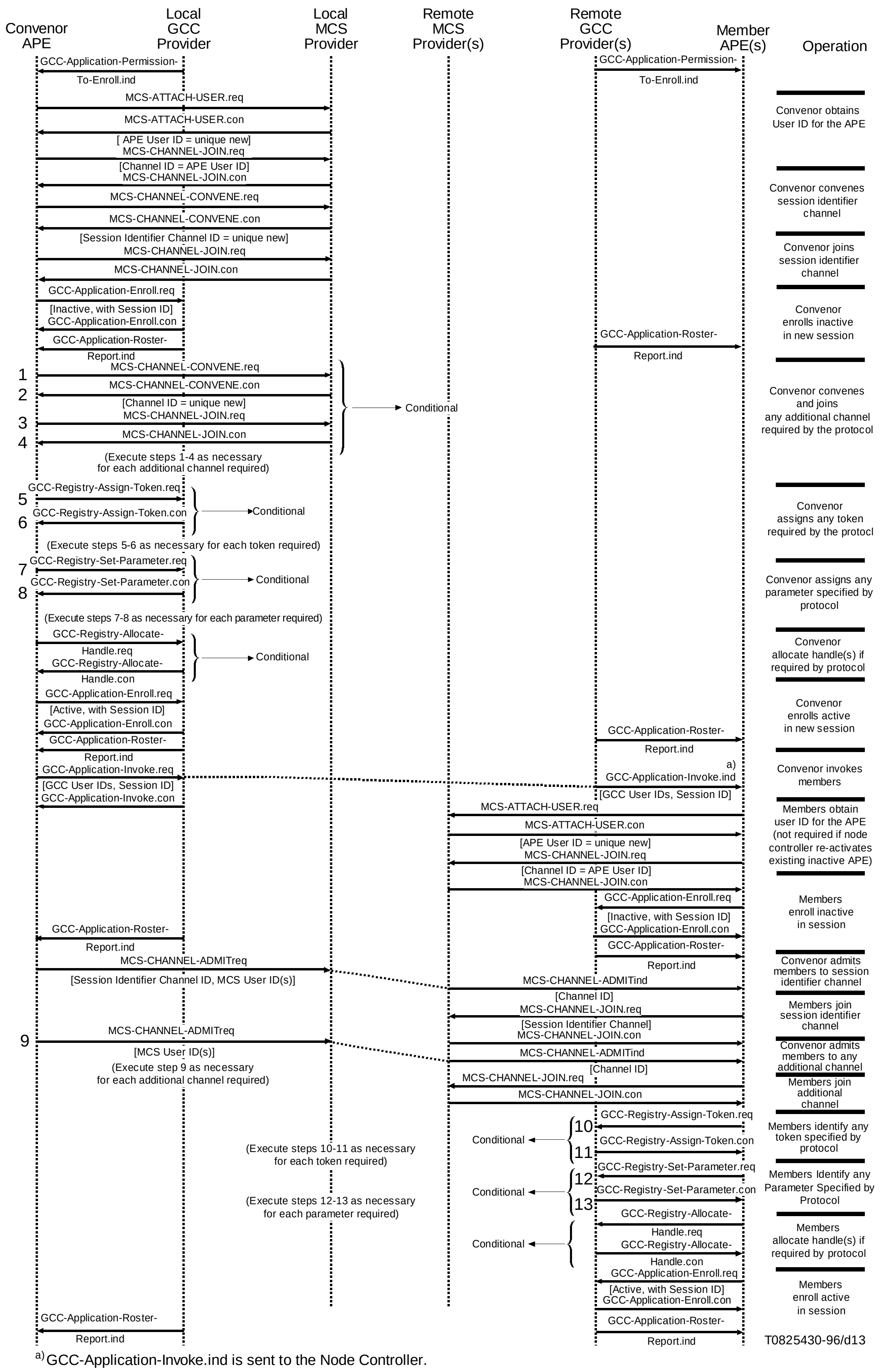
If the sessionID parameter is omitted, the ARM shall attempt to create a new private session. The private convenor ARM shall first convene a private channel by issuing an MCS-Channel-Convene request. If successful, the returned MCS-Channel-Convene confirm contains the ID of the channel allocated. The channel ID also becomes the Session ID. The ARM shall then join this channel by issuing an MCS-Channel-Join request. The ARM shall then enroll Inactive in the session by issuing a GCC-Application-Enroll request with the parameters specified in Table 3.

The onus of acquiring the set of resources that needs to be identified as part of the enrollment process for a private session rests with the ARM that created the session. If any additional channels, registry parameters or tokens are mandated by the protocol specification, the ARM should create these and register them with GCC as necessary. The Registry Key used to register these resources shall include the Session ID as part of the Session Key. Observe that it may not be necessary to register private channels to allow members of a private session to determine the function of each channel. This information may be conveyed more efficiently if an application protocol mandates the order in which the convenor issues MCS-Channel-Admit request primitives to session members. Registering of private channels is not precluded, however.

Once all resources have been successfully assigned, the ARM shall enroll Active in the session by issuing a GCC-Application-Enroll request with the parameters specified in Table 3.

The ARM shall then attempt to invite other nodes to participate in the private session by issuing a GCC-Application-Invoke request, specifying a list of GCC Node IDs to be invited or NULL (indicating that all nodes in the conference are to be invited) as the destination Nodes parameter. It shall also specify Dynamic Private as the Start-Up Channel in the Application Protocol entry.

The convening ARM must then wait until it has received a GCC-Application-Roster-Report indication containing the MCS User IDs of the APEs to be invited to the private session. It should be anticipated that an APE may not be activated at one or more of the invited nodes. The ARM shall then issue an MCS-Channel-Admit request for the start-up channel, specifying the MCS-User IDs of the APEs to be invited as the list of MCS User IDs. The list of invitees is provided by the user application. This process is then repeated for any additional channels specified by the protocol.



Convenor APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Member APE(s)
Operation
GCC-Application-Permission-To-Enroll.ind
GCC-Application-Permission-To-Enroll.ind
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
Convenor obtains User ID for the APE
MCS-CHANNEL-CONVENE.req
MCS-CHANNEL-CONVENE.con
Convenor convenes session identifier channel
[Session Identifier Channel ID = unique new]
MCS-CHANNEL-JOIN.req
MCS-CHANNEL-JOIN.con
Convenor joins session identifier channel
GCC-Application-Enroll.req
[Inactive, with Session ID]
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
Convenor enrolls inactive in new session
1
MCS-CHANNEL-CONVENE.req
2
MCS-CHANNEL-CONVENE.con
[Channel ID = unique new]
3
MCS-CHANNEL-JOIN.req
4
MCS-CHANNEL-JOIN.con
Conditional
(Execute steps 1-4 as necessary for each additional channel required)
Convenor convenes and joins any additional channel required by the protocol
5
GCC-Registry-Assign-Token.req
6
GCC-Registry-Assign-Token.con
Conditional
(Execute steps 5-6 as necessary for each token required)
Convenor assigns any token required by the protocl
7
GCC-Registry-Set-Parameter.req
8
GCC-Registry-Set-Parameter.con
Conditional
(Execute steps 7-8 as necessary for each parameter required)
Convenor assigns any parameter specified by protocol
GCC-Registry-Allocate-Handle.req
GCC-Registry-Allocate-Handle.con
Conditional
Convenor allocate handle(s) if required by protocol
GCC-Application-Enroll.req
[Active, with Session ID]
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
Convenor enrolls active in new session
GCC-Application-Invoke.req
[GCC User IDs, Session ID]
GCC-Application-Invoke.con
a)
GCC-Application-Invoke.ind
[GCC User IDs, Session ID]
Convenor invokes members
MCS-ATTACH-USER.req
MCS-ATTACH-USER.con
[APE User ID = unique new]
MCS-CHANNEL-JOIN.req
[Channel ID = APE User ID]
MCS-CHANNEL-JOIN.con
Members obtain user ID for the APE (not required if node controller re-activates existing inactive APE)
GCC-Application-Enroll.req
[Inactive, with Session ID]
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
Members enroll inactive in session
MCS-CHANNEL-ADMITreq
[Session Identifier Channel ID, MCS User ID(s)]
MCS-CHANNEL-ADMITind
Convenor admits members to session identifier channel
[Channel ID]
MCS-CHANNEL-JOIN.req
[Session Identifier Channel]
MCS-CHANNEL-JOIN.con
Members join session identifier channel
9
MCS-CHANNEL-ADMITreq
[MCS User ID(s)]
(Execute step 9 as necessary for each additional channel required)
MCS-CHANNEL-ADMITind
[Channel ID]
Convenor admits members to any additional channel
MCS-CHANNEL-JOIN.req
MCS-CHANNEL-JOIN.con
Members join additional channel
GCC-Registry-Assign-Token.req
10
GCC-Registry-Assign-Token.con
11
Conditional
(Execute steps 10-11 as necessary for each token required)
Members identify any token specified by protocol
GCC-Registry-Set-Parameter.req
12
GCC-Registry-Set-Parameter.con
13
Conditional
(Execute steps 12-13 as necessary for each parameter required)
Members Identify any Parameter Specified by Protocol
GCC-Registry-Allocate-Handle.req
GCC-Registry-Allocate-Handle.con
Conditional
Members allocate handle(s) if required by protocol
GCC-Application-Enroll.req
[Active, with Session ID]
GCC-Application-Enroll.con
GCC-Application-Roster-Report.ind
GCC-Application-Roster-Report.ind
Members enroll active in session
T0825430-96/d13

[a)] GCC-Application-Invoke.ind is sent to the Node Controller.

FIGURE 13/T.121

Example private session protocol initiation sequence

6.3 Forming registry keys

To determine the identity of a dynamic token or channel via the GCC Registry, an ARM must form a registry key which consists of the Session Key for the current session and a Resource ID. A Resource ID is an ASN.1 OCTET STRING which describes the specific channel or token whose ID is required. The construction of Resource IDs for channels and tokens shall be defined in the application protocol specification.

6.4 Capability negotiation

APEs use the application enrollment mechanism for capabilities negotiation. The Application Capabilities List parameter of the GCC-Application-Enroll request primitive is used to specify the list of capabilities supported by the local APE.

The Application Capabilities List included in the GCC-Application-Enroll request is composed of a list of capabilities to be advertised as supported by the APE. The result of the capabilities negotiation procedure is made apparent to the ARM by the receipt of a GCC-Application-Roster-Report indication from the GCC provider. The application roster report includes the Application Roster for peer APEs within the indicated conference – that is, APEs which have designated the same Session Key. The Application Roster includes a list of nodes for which a peer APE has enrolled. For each node, the list contains the GCC User ID of that node, and the Application User ID of the peer APE at that node. The Application Roster also includes an instance number, a flag indicating whether new nodes have been added since the last instance, a flag indicating whether nodes have been removed since the last instance, a flag indicating whether the Application Capabilities List has been updated since the last instance, and if so, the new Application Capabilities List. In the case of a newly enrolling APE, the Application Capabilities List is always updated since previous instances of the list are not available to this APE.

When first enrolling, an ARM shall ignore received GCC-Application-Roster-Report indications in which the APE is not included (i.e. there is no entry in the Application Roster which has the GCC User ID of the local node and the Application Protocol Entity ID of the local APE as indicated in the GCC-Application-Enroll confirm). Once a roster is received which includes the local APE, that APE is now considered part of the conference and may proceed examining the roster to determine how to proceed.

The Application Capabilities List received as part of the GCC-Application-Roster-Report indication corresponds to the collapsed Application Capabilities Lists of all enrolled peer APEs. That is, the list includes an entry for each capability which has been issued by any peer APE. For each entry, it includes the Capability ID, the number of peer APEs (including the local one) which had advertised this capability as part of their enroll procedure, and, in the case of capabilities of the MIN or MAX class, the minimum or maximum value of the parameter among all peer APEs which advertised this capability. For each capability, the rules used to determine the result of the capability exchange are specified by the application protocol.

At any time whilst an APE is enrolled in a conference, the APE may receive additional GCC-Application-Roster-Report indications from the GCC provider indicating that the contents of the roster have changed. This may be due to new peer APEs enrolling in the conference, peer APEs leaving the conference, or peer APEs having modified their enrollment information.

If, at any time, the local APE desires to indicate a change in its Application Capabilities List, it may re-enroll. This is done by re-issuing a GCC-Application-Enroll request to the GCC provider with the Enroll/Un-enroll flag set to Enroll and the updated Application Capabilities List, as well as the other parameters normally included in the enroll request. The result of this is potentially a change to the Application Roster, resulting in receipt of a GCC-Application-Roster-Report indication by the local APE as well as all other peer APEs in the conference.

6.5 Leaving a session

A user application wishing to leave a session shall signal this to the ARM. Before un-enrolling from the session, the ARM is responsible for deleting any GCC Registry entries it may have created, subject to them being no longer required by the rest of the participants in the session. The method of determining this is specified by the application protocol. The ARM leaves the session by issuing a GCC-Application-Enroll request with the Enroll/Un-enroll flag set to Un-enroll. No other parameters are required.

If, at any time, the ARM receives a GCC-Application-Roster-Report indication in which it is no longer included (i.e. its Entity ID is absent), the ARM shall issue an MCS-Detach-User request immediately to detach from the specified conference. The APE is no longer considered to be enrolled in the conference at this time but may attempt to re-enroll in the conference by repeating the steps of its initialization.

If, at any time, the ARM receives a GCC-Application-Permission-To-Enroll indication with the Grant/Revoke flag set to Revoke, it shall issue an MCS-Detach-User request immediately to detach from the specified conference. The APE is no longer considered to be enrolled in the conference at this time and shall not attempt to re-enroll.

If the ARM is a member of a private session and it receives a GCC-Application-Roster-Report indication for that session in which the session creator is no longer included (i.e. its entity id is absent), the session is considered to have ceased to exist. The ARM shall immediately issue a GCC-Application-Enroll request with the Enroll/Un-enroll flag set to Un-enroll to detach from the specified session. No other parameters are required.

7 Review of MCS resources and GCC services

This clause provides a review of the use of MCS resources (channels and tokens) and the GCC Registry. A fuller treatment of these topics is given in Recommendations T.122, T.125 and T.124. Note that whilst the onus of obtaining the initial set of resources for a session rests with the session creator, any session member may acquire additional resources at any time, if so permitted in the protocol specification.

Application Service Elements (ASEs) should be designed such that, as far as possible, their operation is independent of the type of resource (i.e. static or dynamic) available to the session.

7.1 Channels

MCS channels are used for the distribution of data within a conference. MCS Channels are divided into two types *static* and *dynamic*, the latter type having three variants, viz User ID Channel, Assigned Channel and Private Channel (see 8.4/T.122). An ARM subscribes to and leaves channels required by its user application by using the *channel join* and *channel leave* services respectively.

7.1.1 Static channel

A channel with MCS Channel ID in the range 1-1000. Static channels are reserved for use by standardized application protocols such as T.126 and T.127. Static channels are assigned to their respective protocols in Recommendation T.120; each static channel has a specified purpose as described in the corresponding application protocol Recommendation. A standardized application protocol may use one or more static channels. For each standardized application protocol, one of the channels shall be designated as the *start-up* channel; this is the channel whose MCS Channel ID is used as the Session ID for the standard base session of that protocol.

Static channels may only be used in the standard base session of the corresponding standardized application protocol. They are typically joined by all APEs participating in such a session and are thus used for broadcasting data to the session.

The ARM shall examine the Resource List parameter to determine whether it needs to join any static channels. For each static channel identified in the Resource List the ARM shall issue an MCS-Channel-Join request, specifying the appropriate Channel ID. If this is successful, the ARM shall pass the Channel ID to its ASE.

7.1.2 Dynamic channel

A channel with MCS Channel ID in the range 1001-65535. Dynamic channels form a pool of channel resources available to all sessions within a conference. The function of a dynamic channel is not predefined, but is determined by an APE when it issues a request for a dynamic channel. There are three variants of dynamic channel: User ID Channel, Assigned Channel and Private Channel.

7.1.3 User ID channel

An MCS Channel joined by a single APE. An MCS User ID Channel uniquely identifies an APE and may be used for sending data exclusively to that APE. A User ID is required when an APE undertakes to enroll Active. To acquire a unique User ID, the ARM shall issue an MCS-Attach-User request to the MCS provider, using the Conference ID contained in the GCC-Application-Permission-to-Enroll indication as the Domain Selector. On receipt of a successful MCS-Attach-User confirm in response, the ARM shall join the User ID channel indicated by issuing an MCS-Channel-Join request. See Figure 14.

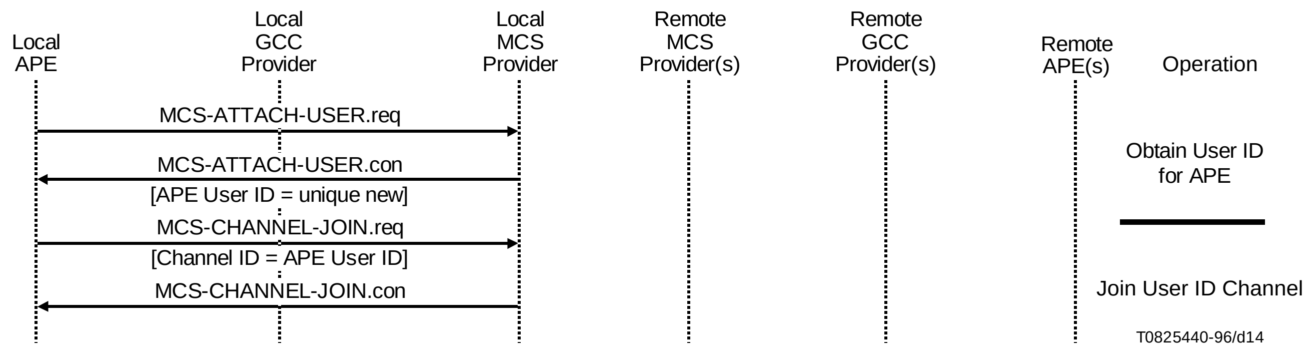


FIGURE 14/T.121

Obtaining an MCS User ID channel

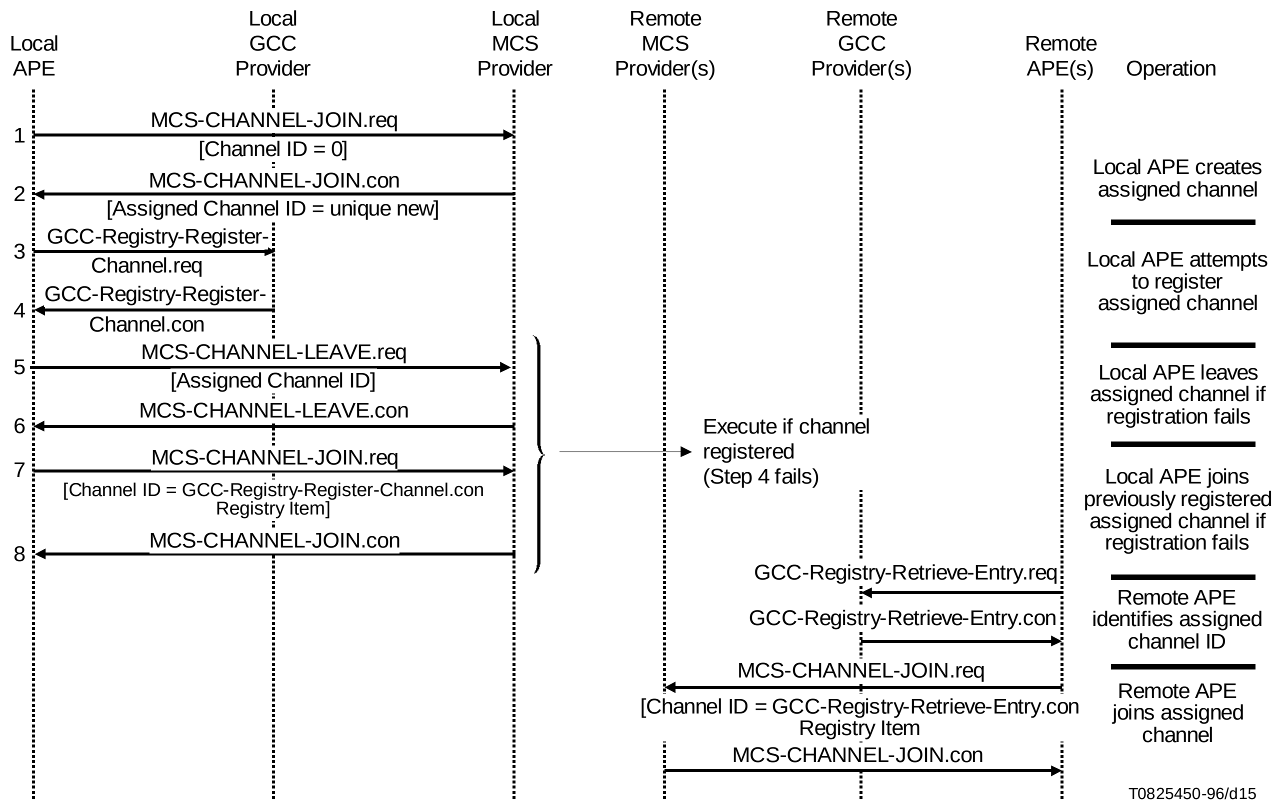
7.1.4 Assigned Channel

An MCS Channel which may be used for broadcasting of data to all participants within a session. Access to these channels is unregulated, i.e. any APE may join or leave an assigned channel at any time. If all APEs leave an assigned channel, it ceases to exist and its Channel ID may be re-assigned to a different use; this is a potential pitfall. See Figure 15.

Assigned channels may be used in either standard base, non-standard base or public sessions. Use in private sessions is not recommended since any APE can join an assigned channel if it is aware of the channel's existence.

The start-up channel of a public or non-standard base session is an assigned channel.

If additional assigned channels are required after a session has been established, then the ARM shall first attempt to create a new assigned channel by issuing an MCS-Channel-Join request primitive with Channel ID = 0. The returned MCS-Channel-Join confirm, if successful, contains the assigned Channel ID which is to be used as the assigned channel. The ARM may then attempt to register assigned this channel by issuing a GCC-Registry-Register-Channel request with the parameters given in Table 4. On receipt of the corresponding GCC-Registry-Register-Channel confirm, the ARM shall examine the Result parameter. If this is "successful" the ARM shall notify its ASE of the Channel ID assigned to the assigned channel. A Result of "index already exists" indicates that another ARM has already registered an assigned channel with the same Registry Key. In this case the ARM shall leave the channel it had previously created by issuing an MCS-Channel-Leave request. It shall then attempt to join the assigned channel registered by the other ARM by issuing an MCS-Channel-Join request, specifying the Channel ID returned in the Registry Item parameter of the GCC-Registry-Register-Channel confirm as Channel ID. If this succeeds, the ARM shall pass the Resource ID and corresponding Channel ID to its ASE. If the attempt fails, and the result is "no such channel", the ARM may re-attempt the channel creation process. In all other cases, the ARM should indicate to the ASE that it was unable to assign a channel. See also Table 5.



Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
1
MCS-CHANNEL-JOIN.req
[Channel ID = 0]
2
MCS-CHANNEL-JOIN.con
[Assigned Channel ID = unique new]
Local APE creates assigned channel
3
GCC-Registry-Register-Channel.req
4
GCC-Registry-Register-Channel.con
Local APE attempts to register assigned channel
5
MCS-CHANNEL-LEAVE.req
[Assigned Channel ID]
6
MCS-CHANNEL-LEAVE.con
Local APE leaves assigned channel if registration fails
7
MCS-CHANNEL-JOIN.req
[Channel ID = GCC-Registry-Register-Channel.con Registry Item]
8
MCS-CHANNEL-JOIN.con
Execute if channel registered (Step 4 fails)
Local APE joins previously registered assigned channel if registration fails
GCC-Registry-Retrieve-Entry.req
GCC-Registry-Retrieve-Entry.con
Remote APE identifies assigned channel ID
MCS-CHANNEL-JOIN.req
[Channel ID = GCC-Registry-Retrieve-Entry.con Registry Item
MCS-CHANNEL-JOIN.con
Remote APE joins assigned channel
T0825450-96/d15

FIGURE 15/T.121

Creating and registering an assigned channel

TABLE 4/T.121

Parameters for GCC-Registry-Register-Channel Request

Parameter	Contents
Conference ID	Provided by GCC-Application-Permission-To-Enroll indication
Registry Key	Registry Key formed as described in 6.3
Channel ID	Channel ID returned in MCS-Channel-Join confirm

TABLE 5/T.121

Parameters for GCC-Registry-Retrieve-Entry Request

Parameter	Contents
Conference ID	Provided by GCC-Application-Permission-To-Enroll indication
Registry Key	Registry Key formed as described in 6.3

7.1.5 Private channel

A private channel is an MCS Channel with an authorized group of users. It may be used for distribution of data to a selected subset of participants within a session. Access to a private channel is controlled by the APE that convened the channel. This APE is termed the Channel Manager and has the following privileges:

- Privilege to admit other APEs to the private channel.
- Privilege to expel other APEs from the private channel.
- Privilege to disband the private channel.

Private channels may be used in standard base, non-standard base, public or private sessions.

The start-up channel of a private session is a private channel.

If the convening user application wishes to remove one or more APEs from membership of the private channel, the Convenor ARM shall issue an MCS-Channel-Expel request, specifying the MCS-User IDs of the ASEs to be expelled as the list of MCS User IDs.

Once the user application at the convening node indicates that it has no further use for the private channel, the Convenor ARM shall disband the channel by issuing an MCS-Channel-Disband request, specifying the private channel as Channel ID. All ARMs currently joined to the private channel then receive an MCS-Channel-Expel indication notifying them that the private channel is no longer available for use.

If additional private channels are required after a session has been established, then the ARM shall follow the procedure described for the convenor or member as appropriate. See Figure 16.

7.1.6 Potential problems with the use of MCS channels

i) MCS does not provide any indication to the members of a channel when an APE joins or leaves a channel. This is a particular problem in the use of private channels, since the private channel manager has no confirmation of whether the invited APEs have joined the private channel. A possible workround (as implemented in Recommendation T.127) is for APEs to confirm that they have joined the private channel by sending a PDU to the Channel Manager’s User ID Channel.

ii) Caution is recommended in the use of assigned channels, since such channels cease to exist when all joined users leave. No indication is provided when this occurs. Since MCS can reuse dynamic channels, an APE should not attempt to use an assigned channel unless it is certain of its existence and current use. The presence of an entry in the GCC Application Registry for an assigned channel does not guarantee that the channel exists or that the use indicated by the Registry is still valid.

iii) Data in transit on a private channel when the channel manager leaves, disbands or expels members from the channel may not reach its intended destinations since MCS may expel channel members before such data is delivered. Use of the MCS token inhibit service allows channel members to signal completion of a transaction to the channel manager, thus enabling the channel manager to avoid inadvertent data loss through premature expulsion of channel members. Each channel member is required to inhibit a designated token before the transaction may begin and must release that token on completion of the transaction. Application protocols must ensure that all channel members have inhibited the relevant token before the transaction starts. Alternatively, application protocols may define a PDU exchange between channel manager and channel members which ensures that all data is delivered before a private channel is disbanded.

iv) Data in transit on any channel may not reach its intended destinations if the sending APE detaches before such data is delivered. The sending APE may avoid this by using the remedy described in iii).

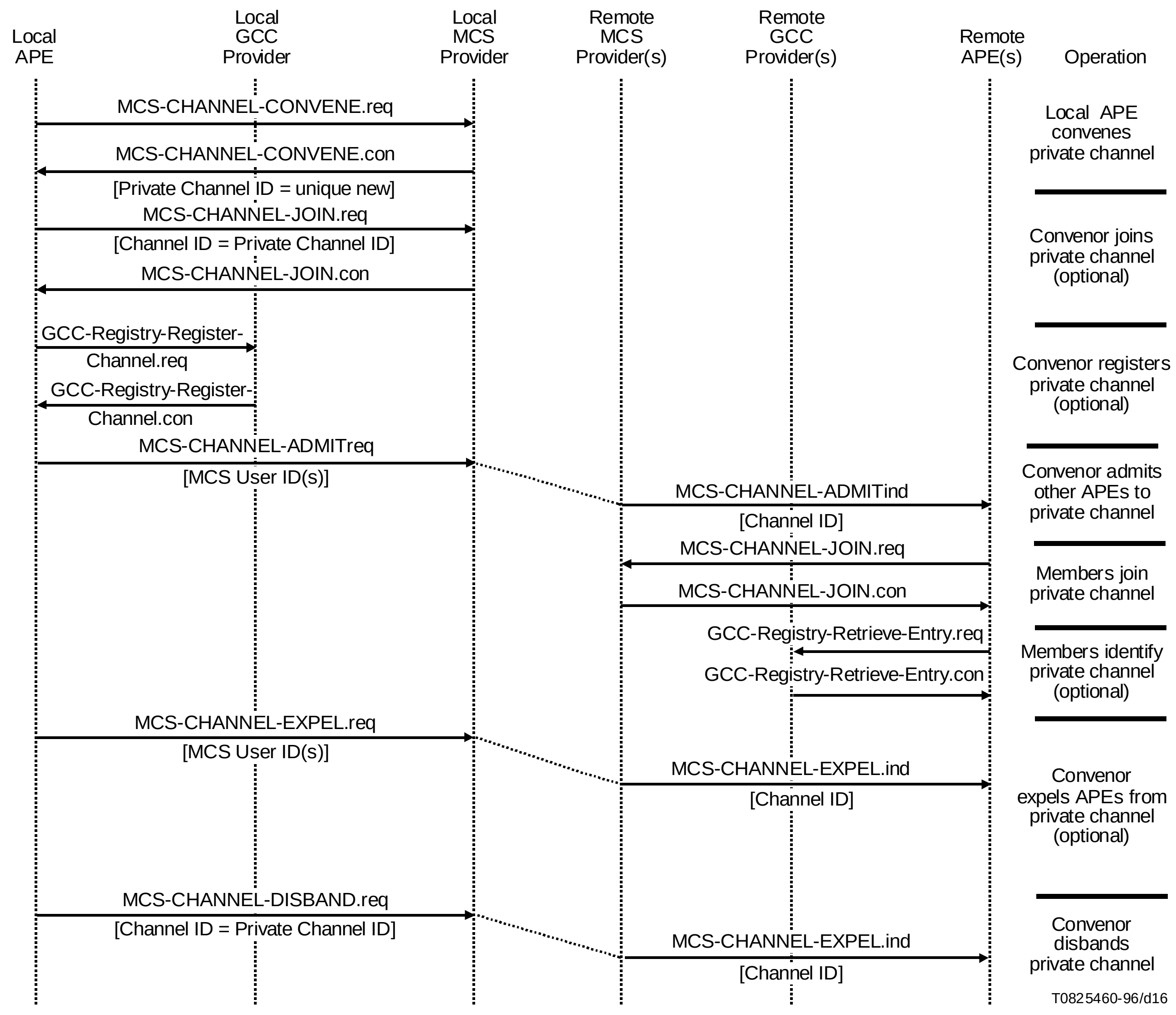

Local APE
Local GCC Provider
Local MCS Provider
Remote MCS Provider(s)
Remote GCC Provider(s)
Remote APE(s)
Operation
MCS-CHANNEL-CONVENE.req
MCS-CHANNEL-CONVENE.con
[Private Channel ID = unique new]
Local APE convenes private channel
MCS-CHANNEL-JOIN.req
[Channel ID = Private Channel ID]
MCS-CHANNEL-JOIN.con
Convenor joins private channel (optional)
GCC-Registry-Register-Channel.req
GCC-Registry-Register-Channel.con
Convenor registers private channel (optional)
MCS-CHANNEL-ADMITreq
[MCS User ID(s)]
MCS-CHANNEL-ADMITind
[Channel ID]
Convenor admits other APEs to private channel
MCS-CHANNEL-JOIN.req
MCS-CHANNEL-JOIN.con
Members join private channel
GCC-Registry-Retrieve-Entry.req
GCC-Registry-Retrieve-Entry.con
Members identify private channel (optional)
MCS-CHANNEL-EXPEL.req
[MCS User ID(s)]
MCS-CHANNEL-EXPEL.ind
[Channel ID]
Convenor expels APEs from private channel (optional)
MCS-CHANNEL-DISBAND.req
[Channel ID = Private Channel ID]
MCS-CHANNEL-EXPEL.ind
[Channel ID]
Convenor disbands private channel
T0825460-96/d16

FIGURE 16/T.121

Management of private channel

7.1.7 Summary

Static channels are used by the standard base session of standardized application protocols for broadcasting data.

A User ID Channel is required by each active APE. It may be used for sending data exclusively to that APE.

Assigned Channels are used within any session for broadcasting of data.

Private Channels are used within a private session for broadcasting of data. They may also be used within any session for selective distribution of data.

The choice of channels available to non-standard application protocols for general use is between assigned and private. This subclause compares the merits of each in order to assist the application protocol developer to make the appropriate choice. See Table 6.

Private

- Known lifetime, determined by channel manager.
- Regulated access.
- Ceases to exist if channel manager leaves session.

Assigned

- Unknown lifetime.
- Unregulated access.
- Can persist after the creator has left the session.

TABLE 6/T.121

Use of MCS channels

Channel type	Session Type			
	Standard base	Non-standard base	Public	Private
Static	✓	7	7	7
User ID	✓	✓	✓	✓
Assigned	✓	✓	✓	💣
Private	✓	✓	✓	✓

✓	Permitted
7	Not Permitted
💣	Not Recommended

7.2 Data transfer

There are two mechanisms available for distributing data, viz *Send Data* and *Uniform Send Data*. Both techniques transfer data to all APEs joined to an MCS channel.

Send Data provides delivery of data to each APE joined to an MCS Channel by the shortest possible route, thus minimizing network traffic and latency. The ordering of data received at each APE may differ and the sender of the data does not receive a copy of the data.

Uniform Send Data provides sequenced delivery of data, i.e. it guarantees that each APE joined to an MCS Channel receives data in the same order. The sender of the data receives a copy of the data if the sender is joined to that channel.

An APE must join an MCS Channel in order to *receive* data on that channel. An APE may *send* data to any MCS channel, irrespective of whether it is joined to that channel, with the exception of Private Channels where an APE must be admitted to the channel before it attempts to send data to the channel.

7.3 Tokens

Tokens provide a means to implement exclusive access to resources within a conference. For example, to prevent conflicting commands being issued to a device (such as a camera) by different users, it is desirable to restrict access to that device to a single user at any given instance. This can be achieved by associating a token with the device and mandating in the corresponding application protocol that only the current token holder is permitted to issue control commands to that device.

Manipulation of tokens is supported by a number of MCS services:

The *token grab* service allows one user to exclusively hold a specified token. Users may use the *token test* service to determine the status of a token at any time and may request the token from the holder with the *token please* service. The token holder may transfer control of a token to another specified user with the *token give* service or return a token to a generally available status with the *token release* service. The token please and token give services allow a token to be transferred between two users without the possibility of it being grabbed by a third user.

Tokens may also be used for event coordination between multiple users via the *token inhibit* service. Users can independently inhibit and release the same token. For example, if it was desired to know when all users have completed reception and processing of a bulk file transfer, all users would inhibit the same token at the start of the operation and each individual user would release the token when it had completed the process. To avoid race conditions, it is necessary to ensure that all users have inhibited the token before testing whether the token has become uninhibited. Any user could test the token at will to determine if the token is free which means all users have completed processing.

Tokens may be either static or dynamic. MCS treats all tokens identically; the distinction between static and dynamic tokens is made within GCC which uses the Registry to assign dynamic tokens. All operations on tokens are performed by the ASE using the MCS token services.

7.3.1 Static tokens

A token with MCS Token ID in the range 1-16383. Static tokens are reserved for use by standardized application protocols such as T.126 and T.127. Static tokens are assigned to their respective protocols in Recommendation T.120; each static token has a specified purpose as described in the corresponding application protocol Recommendation. A standardized application protocol may use zero or more static tokens.

Static tokens may only be used in the standard base session of the corresponding standardized application protocol.

The ARM has no involvement in the use of static tokens.

7.3.2 Dynamic tokens

A token with MCS Token ID in the range 16384-65535. Dynamic tokens form a pool of token resources available to all sessions within a conference. The function of a dynamic token is not predefined, but is determined when it is assigned by GCC. Dynamic tokens may be used in any type of session. See Figure 17.

The ARM shall examine the Resource List parameter to determine whether it needs to identify Token IDs for any dynamic tokens. The same method may be used regardless of whether the ARM is the session convenor or a session member. For each Token Resource ID specified in the Resource List, the ARM shall issue a GCC-Registry-Assign-Token request to the GCC provider using the parameters specified in Table 7. If the Result parameter returned in the GCC-Registry-Assign-Token confirm is "successful" or "index already exists" the Token ID contained in the Registry Item of the returned confirm primitive is used as the Token ID for the token corresponding to the Resource ID used in the Registry Key. The ARM shall pass the Resource ID and associated Token ID to its ASE.

Observe that it is not necessary to use the GCC-Registry-Retrieve-Entry to identify tokens since GCC-Registry-Assign-Token returns the Token ID irrespective of whether the token existed before the request was made.

Additional dynamic tokens required during a session may be assigned by any participant in that session, subject to any constraints defined by the relevant application protocol specification.

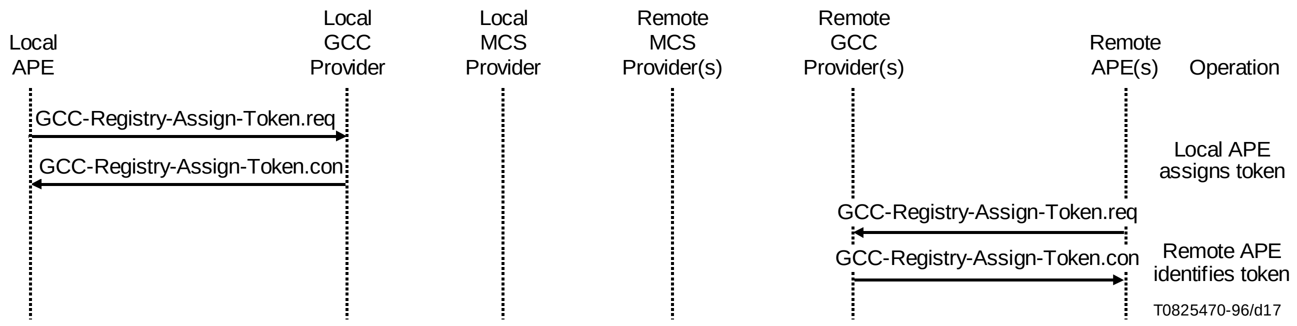


FIGURE 17/T.121

Assigning and identifying dynamic tokens

TABLE 7/T.121

Parameters for GCC-Registry-Assign-Token Request

Parameter	Contents
Conference ID	Provided by GCC-Application-Permission-To-Enroll indication
Registry Key	Registry Key formed as described in 6.3

7.4 Application roster

The ARM is responsible for monitoring GCC-Application-Roster report indications and notifying its ASE of any changes. The Application Roster Entry Report contains that portion of the roster corresponding to the application protocol session in which the APE is enrolled. In the case of an APE that has enrolled inactively without a Session ID, the Application Roster Report will contain that portion of the roster corresponding to all application sessions with the same application protocol key.

At any time while an APE is enrolled in a conference, it may receive additional GCC-Application-Roster-Report indication, notifying it of a change in the roster contents. This may be due to new peer APEs enrolling in the conference, peer APEs leaving the conference, or peer APEs having modified their enrollment information.

A user application may change its Application Capabilities List at any time by instructing the appropriate ARM to re-enroll. The ARM then issues a GCC-Application-Enroll request with the Enroll/Un-enroll flag set to Enroll, including the revised Application Capabilities List, and providing all other parameters as included in the initial active enrollment. This may result in a change to the collapsed capability set, in which case all APEs in the session will receive a GCC-Application-Roster-Report indication.

The effect of any change in the collapsed capability set on any transactions occurring at the time of the change shall be specified by the respective application protocol.

7.5 Application Registry

The Application Registry is a centralized database maintained at the Top GCC Provider that may be used to allow APEs to independently determine the resources (such as dynamic channels and tokens) to be used for a specific application protocol session. The significance of the Registry contents is specified by individual application protocols.

The Application Registry provides mechanisms to:

- Register channels.
- Assign tokens.
- Assign globally unique handles, either individually or as a block of handles.
- Establish common parameters for the session.
- Change and delete registry entries.
- Retrieve registry entries.
- Monitor changes made to registry entries (including the deletion of entries).

Each registry entry is identified by a unique key which identifies the application protocol session with which the entry is associated. The key also comprises a Resource ID, defined by the application protocol, to describe the use of the registry entry within that session.

Each registry entry has a designated owner – the APE that created the entry. If the owner of an entry leaves the conference that entry becomes unowned. Owned entries may only be deleted by their respective owners; unowned entries may be deleted by any APE. Channel and token entries may not be modified. Owned parameter entries may be modified by their respective owners. Unowned parameter entries may be modified by any APE; this APE becomes the new owner of the entry. Registry entries persist for the lifetime of the session and care must be exercised by registry entry owners to ensure that registry entries are deleted when no longer required. This is particularly true in the case of the registration session where redundant entries are likely to persist indefinitely, unless explicitly deleted by the owner. Observe that in some cases it may be inappropriate or impossible for departing APEs to delete the entries they created.

All communication between an APE and the Application Registry is performed by the ARM.

7.6 Conductorship

The ARM is responsible for notifying its ASE and user application when the conference enters or leaves conducted mode, and of any change in the conducting node. Other operations in conducted mode are performed directly by the user application or ASE.

A user application shall issue requests for its local node to become the GCC conductor, transfer conductorship, or relinquish conductorship via its local Node Controller.

The *session conductor* is an APE at the conducting node which has the Conducting Operation Flag set in the Application Roster for that session. The Node Controller at the conducting node proxies as session conductor in the absence of such an APE.

If an APE adopts the role of session conductor, its ASE shall arbitrate requests from peer ASEs to act. Such requests and the associated responses shall be issued using PDUs specified by the application protocol. These may be used to assign different privileges to each ASE in the session.

If a Node Controller becomes session conductor, it may only grant or refuse permission to act on a node by node basis. This gives much courser control of ASEs. An ASE may request permission to act via its local Node Controller, which in turn issues a GCC-Conductor-Permission-Ask. If the conducting Node Controller issues a GCC-Conductor-Permission-Grant which includes the requesting node in the “List of Nodes Granted Permission” then all ASEs at that node may act.

The detailed effect of conductorship is specified by each application protocol specification.

7.7 Remote invoking

A user application wishing to dynamically invoke a peer application at another node and invite it to join the application session shall do this via the ARM. The user application must provide the following information for each application protocol it wishes to use:

- Session Key.
- Start-up channel type (static, dynamic multicast, dynamic private or dynamic userid).
- Minimum capability set.
- List of nodes to be invited to the application session.

The ARM then issues a GCC-Application-Invoke request, specifying a list of GCC Node Ids of the nodes to be invited, the start-up channel type, a list of application protocol sessions to enroll in and, for each application protocol, the expected (i.e. minimum) capability set.

GCC-Application-Invoke indications are delivered to the Node Controller which is responsible for determining which APEs to activate.

8 User application guidelines

This clause provides guidelines on use of the T.120 infrastructure to ensure consistent and predictable application behaviour.

8.1 Identification of user applications

In certain circumstances it may be useful to be able to determine which nodes host a specific user application or the range of user applications available at a remote node. This may be required for both local and remote invocation of user applications. An APE may optionally choose to identify its user application by inclusion of a non-standard non-collapsing capability parameter in the GCC-Application-Enroll request. By consulting the Application Roster via its APE, a user application can thus detect the presence of a particular user application at other nodes. A Node Controller receiving a GCC-Application-Invoke indication may choose to consult the user application capability entries in the Application Roster for the appropriate session(s) to determine which user applications are active at other nodes before activating a user application locally. Note that a Node Controller will only be able to detect the presence of a given user application if it is aware of the non-collapsing capability parameter advertised by that application.

Non-standard application protocol keys may only be used to advertise support of a non-standard application protocol. Enrolling with a non-standard application protocol key to advertise the presence of a specific user application is strongly discouraged as it would prevent interoperability with other user applications that employ the same application protocol for the same purpose.

8.2 Use of collapsing and non-collapsing capabilities

An APE may use the non-collapsing capabilities field in the GCC-Application-Enroll request to identify its user application and advertise its application protocol specific capabilities.

Use of a non-collapsing capability parameter in the GCC-Application-Enroll request is appropriate if there is a need to identify *which particular* peer APEs support a given capability in order to guarantee interoperability. Alternatively, it may be used to advertise the availability of a resource to the conference (e.g. a camera available for remote control). If the APE only needs to determine the *number* of peer APEs that support a specific capability, then it should use a collapsing capability. In order to avoid making the Application Roster unnecessarily large, it is strongly recommended that the number of non-collapsing capabilities be kept to an absolute minimum.

8.3 Session context

If a conference has multiple concurrent sessions of a specific application protocol then there may be a need to distinguish those sessions so that users can select the most appropriate session in which to participate. Typically, a session would be used in a specific context which is determined by the participants of the session. For example, one T.126 whiteboard session may be used to present one topic, another could be used to present a second topic. The different contexts of these sessions allow them to be distinguished.

A GCC Application Registry Parameter may optionally be used to describe the context of a specific application protocol session. A Resource ID (constructed by encoding the 7 characters of the word "CONTEXT" into successive octets according to Recommendation T.50) is reserved for this purpose. The content of the Registry Parameter is a user readable string which shall be encoded as an unrestricted BMPString using Packed Encoding Rules (Basic Aligned variant). 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.

8.4 Selecting a session to participate in

If a user simply wishes to communicate with *any* node that supports *the same application protocol(s) for the same purpose* then the most appropriate session to use is the:

- standard base session for standardized application protocols (see 8.3);
- non-standard base session for non-standardized application protocols.

The base sessions are intended for general purpose communication between all terminals that support a specific protocol (e.g. the T.127 standard base session would be used for general purpose file transfer).

Note that an APE may be configured to enroll in a base session without user intervention, thus allowing user applications based on standard or non-standard protocols to be started automatically upon joining a conference.

If there are multiple active sessions of a given application protocol present in a conference, a user may need to intervene. The user requires sufficient information to choose the most appropriate session(s) in which to participate, or to determine whether it is necessary to establish a new session.

This decision may be based on one or more of the following criteria:

- the session context (e.g. topic of discussion).
- which nodes are participating in the session.
- the negotiated capability set of the session.

8.5 Controlling the use of a session

The creator of a session may wish to impose constraints on a session to ensure that less capable APEs do not inadvertently enroll in the session and potentially degrade the capabilities of the session to a level which is considered unacceptable by the session creator. If a user wishes to impose a *minimum capability set* on a session, then a dynamic private session shall be used. The session creator specifies the requisite capability criteria in the GCC-Application-Invoke request.

If a user wishes to indicate that a session is to be used in a *specific context*, then a dynamic session should be used and a session context (user readable text string) shall be provided in the GCC Application Registry for that session.

8.6 Adding new participants to a session

The GCC-Application-Invoke request primitive may be used by the convenor of a private session to invite additional nodes (e.g. late joiners) to that session at any time. Any participant in a standard base, non-standard base or public session may invite new nodes to that session at any time using GCC-Application-Invoke request. A Node Controller receiving a GCC-Application-Invoke indication may choose to ignore it if there is already an active APE at that node in the session identified.

8.7 Extension, modification and reuse of application protocols

Application protocols are tools which can be extended, modified or tailored to a specific use. In order to determine whether such changes result in a new distinct protocol or are merely an extension of the existing protocol, it is necessary to decide whether APEs conforming to the changed application protocol are intended to interwork with APEs conforming to the original application protocol. If interworking is intended, then the revised protocol can be considered to be an extension of the existing protocol, otherwise it is a new application protocol and must be assigned a unique application protocol key and resources.

If an APE offers proprietary extensions to an application protocol, such *that it is intended to be able to interwork with APEs conforming to the original protocol*, then it shall enroll with the application protocol key of the original protocol and advertise its enhanced functionality using collapsing or non-collapsing capabilities, according to the guidelines specified in 8.2.

Where an application protocol is to be used for a distinct purpose or is modified in such a way that *APEs conforming to the revised protocol are not intended to be able to interwork with APEs conforming to the original protocol*, then the revised protocol is considered to be a new application protocol and it must be assigned a unique application protocol key to differentiate it from the original protocol. Any static channels and tokens used by the original application protocol cannot be used by the revised protocol. These rules also apply if use of an application protocol is constrained in any manner. For example, APEs employing the methods (PDUs and procedures) of Recommendation T.127 exclusively for transfer of documents between word processor applications shall not enroll with the T.127 application key if they are not intended to be able to interwork with a general purpose file transfer APE, nor shall they use the static channels and tokens reserved for Recommendation T.127.

Similarly, if a number of application protocols are required to collaborate to achieve certain functionality (e.g. a business card application protocol reusing the methods of Recommendations T.126 and T.127), then new application key(s) are required if the component APEs are only intended to interwork with other APEs collaborating for the same purpose.

Where application protocols are being combined to achieve a specific function, the preferred approach is to define a single new application protocol which is an aggregation of the component protocols. This, in effect, creates a new APE by the merger of pre-existing APEs. It may be necessary to resolve conflicts in capability IDs and GCC Registry Resource IDs between the application protocols for the new APE.

An alternative approach is to retain the pre-existing APEs as distinct components, each with a new application protocol key to distinguish it from the protocol from which it was derived. Observe, however, that in this case it is necessary to define a means to associate application protocol sessions which are cooperating to achieve a common purpose so that user applications can determine the correct combination of sessions for their APEs to enroll in. To avoid duplication of information, it is suggested that the list of associated sessions is maintained in the Application Registry for one of the sessions, to be termed the *root session*. It is up to the application protocol developer to determine which application protocol should serve as the root session; a suitable choice would be the application protocol whose functionality most closely matches the intended common purpose.