

1 The Interactive Group Communication System

The interactive group communication system (IGCS) is designed to support distributed applications particularly applications for cooperative work. Such applications require the exchange of data within a group of participants. It is often necessary to ensure that all group members receive the data in the same order and no data is lost. Furthermore the data should be transported to the group members with low delay in order to enable interactivity between group members. Since cooperative work is most important once long distances separate a group of users, it should be possible to use the IGCS in WAN as well as in LAN. This means the IGCS must be very flexible to be used in varying network environments.

The IGCS provides powerful but easy to use services for group communication. The IGCS offers the functionality to dynamically define groups, group members and channels for data exchange. Data transport is realized by using TCP, if this is possible or by HTTP, if a firewall denies TCP usage. Users have the possibility to choose to use secure data transport which is realized by using SSL. Further there are mechanisms to concatenate IP multicast domains. This enables the integration of multicast based applications for example the Mbone video and audio tools. Users of IGCS do not need to know any details about protocols or the specific network environment to use these functionalities. The IGCS can adapt to fast as well as to slow network environments. Firewalls which often separate LAN from WAN are also taken into account. Before using IGCS for the first time some configuration information must be provided. This task is supported by some tools including a configuration wizard.

2 IGCS Services

IGCS provides services for group communication. This requires some management functionalities to organize groups and group members additionally to the data transport services. The following sub chapters describe the management, the regular data transport services and the IP multicast transport services separately.

2.1 Management

The management of IGCS is performed by a specific service which is called IGCS Management Service (IGCSMS). This service is a process running on a well known host. Each IGCSMS defines its own IGCS domain, this means they are responsible for a disjunctive set of groups. IGCS users of different IGCS domains can not join one common group and communicate with each other via the IGCS.

Among other things the IGCSMS offers the functionalities to create and delete groups and to become a member of a group and to leave a group, which means to change the status of one IGCS domain. In general not every arbitrary user in the Internet should be allowed to do this. Therefore the IGCS provides rudimentary access control mechanisms. Before a user is allowed to change the status of a IGCS domain he has to authenticate himself with the IGCSMS. Therefore the user must provide a login and a password. This will be compared against an access list by the IGCSMS. The current implementation does not classify users resulting that each user has the same rights. Later versions should be able to assign different rights to different users.

To create a new group a IGCS user has to provide a group name only. Then the IGCSMS will launch a new group server and will assign an unique group identifier to the new group. Advanced users may request a specific group identifier in order to define well-known groups. The group server is responsible for managing the members of a group. Nevertheless the group server will also inform the IGCSMS once a user joins or leaves his group. So the IGCSMS is able to provide all status information of a IGCS domain. The main task of a group server is the distribution of data within a group. Therefore it might be important on which host the group server is running. A user may

specify that a group server is running on the same host than the IGCSMS or on the host of the calling client. Advanced users may also launch the group server on any arbitrary host and provide this information when creating the new group.

When a group has been created successfully users may join the group and become group members. Each group member must have a name and might have an additional description. The group server will assign an unique identifier to each group member. Advanced users may request a specific group member identifier in order to define well-known group members.

A IGCS user may request information of exiting groups and group members like names, descriptions and identifiers. Information of group members may be retrieved without joining a group. It is also possible to request to receive a callback from the IGCS whenever the system status has changed.

For administrators there are a few more management services like resetting the IGCSMS or modifying the access list of the IGCS domain. There is also a command line tool to create an (initial) access list.

2.2 Data Transport Services

Data will be transported between group members only. Therefore two IGCS users must join the same group before they are able to exchange data. A IGCS user may send any arbitrary data¹ to all members of a group (including himself), to all other members of a group (excluding himself) or to one specific user of a group.

IGCS supports a channel concept for data transmission. Channels are used to easily distinguish different data flows within a group. The creation of channels is not managed by the IGCS so IGCS users must define which channel number is used for the different traffic flows.

Each IGCS user has several options to control the data transfer on a channel:

- Enable or disable encryption. If encryption is enabled, the IGCS will use SSL for data transport². Switching encryption on or off does not result in any network activity, so it can be used arbitrarily at runtime.
- Enable or disable sending data. If sending data is disabled, all calls to “send” will result in no action. This may help to manage bandwidth consumption of an application without changing core functionalities of the application.
- Enable or Disable receiving data. When a IGCS users disables to receive data, the group server will not forward data of this channel to that IGCS user. The user can enable and disable receiving data at runtime arbitrarily. This may help to reduce the amount of data which is sent to a host. For example a IGCS user may choose to disable receiving data on a channel which contains video data for a video conference in order to save bandwidth for audio transmission.
- IGCS provides a blocking request response mechanism which implies sending of an object to (usually one) receiver and block until a response message arrives. This is a handy mechanism to retrieve an object from another group member. Note: it is still possible to send and receive data on a channel while the request response mechanism is used but another request will block until the prior request is processed or has received a timeout.
- IGCS users may choose to register a listener for incoming messages or to call a blocking receive. If no listener is registered the user may specify the number of messages that are buffered before new messages will overwrite an old one³.

¹ The current IGCS implementation is written in Java. There is a specific support to send byte arrays or to send any serializable java object.

² The IGCS can be configured to use non secure transmission or to return error messages when SSL is not available.

2.3 IP Multicast Support

The IGCS intends to be a communication platform for applications for cooperative work. The multicast capability of the internet protocol (IP) already provides a mechanism to distribute data within a group. Several applications exist which are utilizing the IP multicast capability for cooperative work scenarios. But today it is not common that multicasts can be used in WAN. Often also in LAN routers are configured not to forward IP multicasts to another local network segment.

IGCS supports the use of IP multicasts by concatenating IP multicast domains. A IGCS user may choose to distribute the traffic of an IP multicast group⁴ via a IGCS channel to all other group members. IGCS ensures that different IP multicast domains are concatenated only by one channel to other groups. This reduces the traffic that is tunneled by the IGCS and prevents traffic loops. Figure 1 shows the principle of tunneling IP multicasts with the IGCS. Assume that the IGCS users A to D are in the same IGCS group and LAN1 and LAN2 are IP multicast domains, i.e. users A and B receive messages from each other by using IP multicasts and also C and D receive messages from each other. But A and B are not able to receive multicasts from C and D and vice versa. All four users request the IGCS to forward the multicast traffic. The IGCS automatically selects one group member of each multicast domain to forward the data. In Figure 1 user A and C will forward the data. This selection is transparent for the IGCS user. If user C leaves the group, he will also stop forwarding the multicast traffic. This will be detected by all other IGCS users in the same multicast domain and another user will start to forward the multicast traffic. In the example shown in Figure 1 the IGCS user D will start forwarding multicast traffic, when user C leaves the group⁵. The selection of a multicast forwarder within a multicast domain is based on an election process which also uses multicast messages. The multicast address that is used for the election process can be configured.

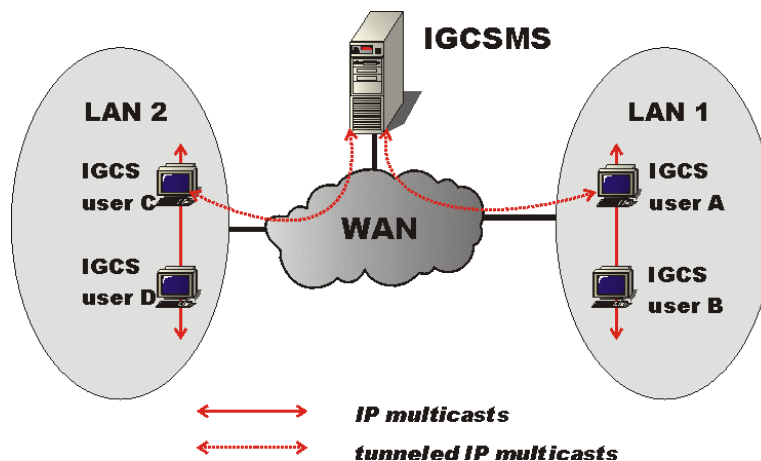


Figure 1: tunneling of IP multicasts

3 IGCS Realization

The IGCS is realized in three layers (Figure 2). The Connection Management layer performs all connection handling and controls the different supported protocols. This layer mainly offers the service to exchange messages between processes independent from the used protocols. The Communication Manager (CM) layer realizes the distribution of messages between IGCS components. The component identifiers are used to address other components. There is only one CM

³ It is assumed that a user will process incoming data when a channel has been created. To ensure that no data will be lost even when messages arrives with a high frequency a user should register a listener.

⁴ IP multicast address and a port number.

⁵ It may take up to two seconds until host D starts forwarding when host C left the group. Programmers may reduce this time, but this will cause more management data to be sent.

in each process that uses the IGCS. The highest layer consist of several services: the IGCS Management Service, the IGCS Group Server and the IGCS Group Members. The group member offers the API to the IGCS users. The following chapters describe the IGCS layers in detail.

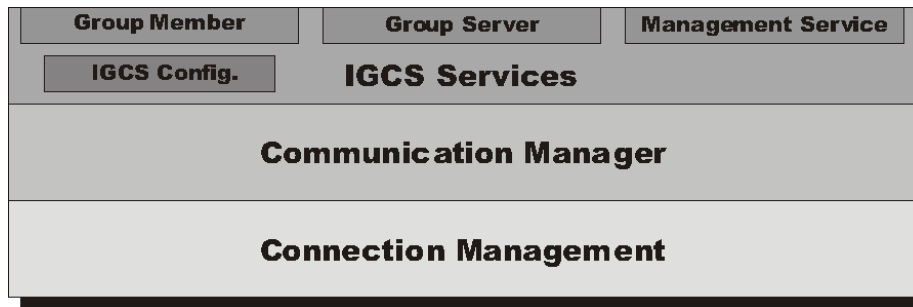


Figure 2: IGCS Layers

3.1 Service Components

The IGCS Managements Service (IGCSMS) and the IGCS group server are processes that provide essential services of the IGCS. The IGCSMS will be started by an administrator and the group servers will usually be started by the IGCS. The IGCS group member is the component that is created by an IGCS user. This component offers the communication service to the users.

3.1.1 IGCS Managements Service

As mentioned earlier the IGCSMS stores and manages the status of a whole IGCS domain. Each user process will be connected to the IGCSMS when using the IGCS the first time. Therefore it is essential that a user is able to establish a connection to the IGCSMS. For example if the IGCSMS is running on a host which is behind a firewall, it is impossible for users outside the firewall to join that IGCS domain. When at least two users reside behind different firewalls, it is necessary to run the IGCSMS on a host that is not protected by a firewall⁶.

The IGCSMS is implemented as a java class that should be started in a separate VM since the IGCSMS will usually run for a long time compared with user applications. But it is possible to run the IGCSMS within the context of another java application. When IGCSMS is started some configuration parameters must be provided via system properties, see chapter 4.1.3 for more details.

3.1.2 IGCS Group Server

The group server is responsible for managing the group members and for the distribution of data within a group. Usually the group server will be started by the IGCS automatically, when a new group is created. The user may choose to run the group server within his own process or within the process of the IGCSMS. Advanced users may also start the group server explicitly on any arbitrary host before creating a new group. When the group server is not running on the same host as the IGCSMS, it must be ensured that all possible group members are able to connect to the group server, e.g. a group server running behind a firewall might be unreachable for clients in front of the firewall.

A group server also communicates with the IGCSMS. First the group server will receive a group identifier from the IGCSMS, this may be any unused identifier number or a number which was requested by the creator of the group. Whenever a group member successfully joins or leaves the group the group server will inform the IGCSMS about the event. When a group is deleted, the IGCSMS will send a termination event to the group server. The IGCSMS assumes that a group server has terminated, when the connection between the group server and the IGCSMS is closed.

IGCS users which want to join a group only need to know the group identifier to do so. The IGCS will transparently connect the user to the group server. A group member will receive a group member identifier from the group server, this can be any unused identifier or an identifier which was

⁶ Or the firewall must be configured to allow access to the IGCSMS.

requested by the IGCS user. When the connection between the group member and the group server is closed the group server will unregister the group member.

The main task of the group server is to distribute data within a group. All data that is sent to all, or all other group members is first sent to the group server. Then the group server will send this data to all group members. This way all group members receive the messages in the same order. A group member may choose to inform the group server that he does not want to receive data of some specific channels. The group server keeps a list of locked channels per group member in order to decide which data must not be forwarded to a specific group member⁷. Further it is possible to send data from one group member to one other group member. In this case the sender must use the identifier of the addressed group member. The IGCS will sent such a message to the group server only if there is no direct connection between both group members⁸.

3.1.3 IGCS Group Member and Channels

A IGCS group member is created by joining a group. Within the user application a group member is represented by an java object. This object offers methods to send and receive data. Usually there are different data flows within a group. A user may choose to use IGCS channels to easily distinguish data flows. Each channel is also represented by a separate java object. This object also offers a more user friendly communication interface than the group member (see chapter 2.2).

3.1.4 Component identifier

As mentioned before the IGCS assigns an identifier to each group (GID = group identifier) and another identifier to each group member (GMID = group member identifier). These identifiers are integer numbers, where the current IGCS implementation requires that these numbers are within the range from 0 to 255. For the unique identification of a IGCS component both identifiers are required: GID, GMID for example 2, 5 is the identifier for group member 5 within group 2. The identifier 0 is reserved for specific components. This is shown in Table 1. Temporary identifiers are for internal use only.

		Group ID	
		0	1 - 255
Group Member ID	0	<i>IGCSMS</i>	<i>Group Servers</i>
	1-255	<i>Temporary Identifiers</i>	<i>Group Members</i>

Table 1: IGCS Identifiers

3.2 Communication Manager

The task of the Communication Manager is to distribute messages between the IGCS components based on the component identifiers. In a process using the IGCS there is only one instance of the CM available. Each IGCS component will be registered with the local CM. Additionally each connection to another process will register the components of the remote process automatically. This way the CM keeps track of all IGCS components that are accessible from the local process.

⁷ To be more precise: the group server handles messages for locking unlocking channels only. The list of locked channels is managed within the Communication Manager that is used by the group server.

⁸ Group members are connected to each other only if they are running in the same Java VM. In general the IGCS does not establish network connections between group members. But advanced users may do this by using services of the IGCS Layers 1 and 2.

When a group server is started, the local CM will be extended by some functions to support the special requirements of data distribution of the group server, e.g. send data to all group members in consideration of locked channels.

The CM is also used to establish new connections and to create new server ports. IGCS components may request to establish a connection to another component with a specified ID and with a destination address. The CM is able to decide, if that new component is already accessible or not. If the component is already accessible no new connection will be established. This reduces the number of connections. There will always be only one (logical) connection between processes.

The CM is also responsible for assigning identifiers to new groups and group members. First a local component must request a new identifier, e.g. the group server requests a new GMID for a new group member. This identifier will be reserved. When the new component has registered successfully or after a timeout the reservation will be removed. Further the CM is able to assign temporary (or one time) identifiers to a component that does not yet have an identifier. The one time identifiers are established by sending a request and are removed when a respond arrives or after a timeout.

3.3 Connection Management

This layer hides the handling of the different connection types from the rest of the IGCS. The Communication Manager may forward a message to a logical connection in exact the same way as to a local IGCS component. The main aspects of handling connections will be described in the next subchapters.

3.3.1 Logical Connections and Transports Protocols

The Connection Management layer distinguishes between logical connections which manage the status of a connection and the real connection which controls the access of specific transport protocols. A logical connection is aware of all remote IGCS components. When a logical connection is established both sides exchange lists of local components and whenever a new local component is created or terminated all logical connections send this information to their remote sides. A logical connection automatically detects when there is no more component that uses this connection. In this case the connection will be terminated. A real connection sends and receives data from a socket. A separate thread is started to wait for incoming data. The real connection always sends a small header (see 3.3.2) followed by the user data. Two types of data can be exchanged: java byte arrays and any other serializable java object. Byte arrays are transmitted as thee and java objects are serialized before transmission.

For each logical connection there must be at least one real connection. The current IGCS implementation supports two types of real connections per logical connection. There must be a real connection for standard data transfer (usually using TCP or HTTP) and another optional one for secure data transmission (usually using SSL). Once a logical connection is established, it is possible to specify which type of real connection should be used. When creating a new logical connection IGCS first tries to establish a TCP connection, if this fails, IGCS tries to establish a HTTP connection, if this fails too, the logical connection fails also, otherwise the logical connection is ready to be used. Then the logical connection automatically tries to establish a SSL connection. This is done by a separate thread because setting up an SSL connection may take some remarkable amount of time i.e. several seconds. Each logical connection has its own unique connection identifier (CID). The CID is used to assign real connections to logical connections. The first step after creating a real connection is to send the corresponding CID to the remote side in order to enable the receiver to assign the real connection to an possibly already existing logical connection or to create a new logical connection. This method enables the IGCS to keep a logical connection to be active while the real network connection is reestablished or replaced by another one⁹.

⁹ Note replacing exiting real connections is yet not implemented, it is only used to assign a real SSL connection to the proper already existing logical connection

3.3.2 Message Format

IGCS uses a simple protocol for the transmission of messages (see Figure 3). The type field describes the content of the messages. The upper four bit of the type field contain flags which can be combined with each other to describe how the messages must be handled. These flags are used:

- Is object: signals that the data field is a serialized java object.
- Is encrypted: signals that the message will be or has been sent over an encrypted channel.
- Is management: signals that the message contains management data (IGCS internal data).
- Is one time message: signals that this message is forwarded based on temporary identifiers.

If a message contains user data, then the lower four bits specify how the message must be delivered. A message can be sent to one group member, to all group members, to all other group members or a message can be a request or a respond to one group member.

If a message contains management information, the lower four bits specify the management message. Currently there are only messages concerning the exchange of components identifiers, termination of a connection, error indication and performance measurements.



Figure 3: IGCS Message Format

The channel field contains the channel number used for the message. Channel number 0 is the default channel. The next fields contain the identifiers of destination and source component. The last field contains a 16 bit size information. This field is used, if a byte array is transmitted only. For java objects this field is 0 since the serialized form of an object contains its length also. Note: users may send byte arrays that are larger than 2¹⁶ byte. In this case the IGCS encapsulates the array into a java object and transmits this object¹⁰.

3.3.3 Firewall tunneling

If some members of a group do not reside in the same LAN, it is usually required to communicate through a firewall. It is common practice to protect a LAN from unknown traffic from the Internet. Therefore firewalls parse all incoming and often also all outgoing traffic. Incoming traffic is forwarded to the LAN only, if the traffic type is well known and/or is a respond to a prior request by a client within the LAN. In this scenario it can not be assumed that a proprietary protocol on top of TCP will be accepted by a firewall. Therefore the IGCS is able to use the HTTP protocol instead of TCP. HTTP specifies that a client sends one request to a server and the server answers with one respond. It is not possible to send two requests at a time on the same connection and a server is not allowed to send two respond message for one request. This way a firewall is able to associate all incoming HTTP data to a prior request of a client.

Using HTTP instead of TCP has negative impacts on the performance. The necessary request response mechanisms drastically reduce the number of messages that can be exchanged within a time range. Therefore the IGCS is able to establish multiple HTTP connections in order to increase the performance. A IGCS client always sends a request to a server whenever data is available and a HTTP channel becomes available, i.e. the respond of the server is received. If a client has no data to send, the client sends an empty request to the server in order to enable the server to send data to the client. But this is done on few connections only, i.e. the server has a higher priority to send data on

¹⁰ It is assumed that for byte arrays of this size the additional overhead of serializing an object does not matter.

these connections. Advanced users might configure how many HTTP connections are established from a client to a server and how many connections are used to send empty requests.

Figure 4 shows a scenario in which three IGCS users of two LANs have connected to the same IGCSMS so it's possible for them to be in the same group. Each user has established exactly one logical connection to the IGCSMS. The IGCS of user C has established a TCP and a SSL connection for data transport. Users A and B are behind a firewall, so they are using multiple HTTP¹¹ connections. In this scenario the firewall behaves like a HTTP proxy. This means the HTTP connections from A and B are terminated by the proxy and the proxy establishes new HTTP connections to the server. How many connections are established from the proxy to the IGCSMS server depends on the firewall configuration. The IGCS tries to force the proxy to establish at least as many connections to the server as one client has established to the proxy. Nevertheless it is possible that the two clients in LAN1 have to share the same connections from the proxy to the server. Additionally a firewall can choose to close a connection, when a respond from the server is received. The IGCS requests not to close any connections and takes care that HTTP connections are not idle for several seconds to avoid the time consuming establishment of new connections.

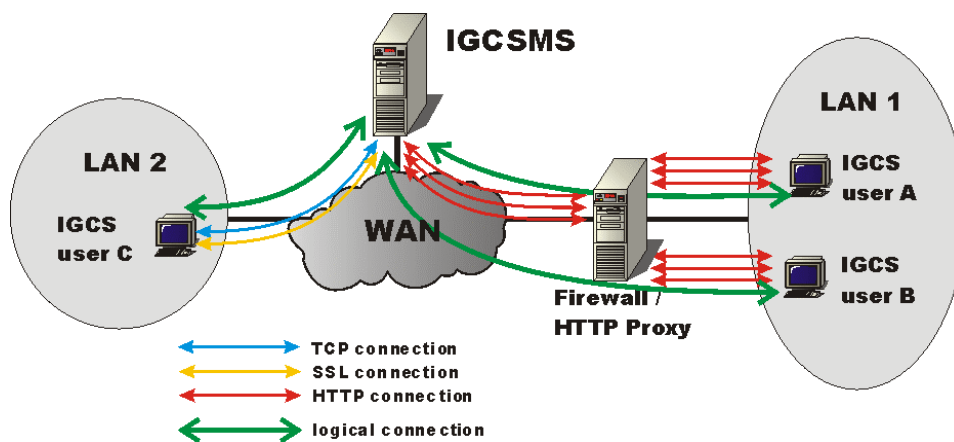


Figure 4: Example of firewall tunneling scenario

For HTTP connections the IGCS uses an additional proprietary protocol in order to implement the following functionalities:

- Reordering of messages. Since parallel HTTP connections are used it is possible that the messages arrive in a wrong order. Sequence numbers are used to reorder the messages.
- Heuristic to detect, if a client has closed a HTTP connection. The server is not able to detect, if a HTTP connection has been closed by a client. It is possible that a proxy closes a connection even though a client still wants to use it. Also a proxy may keep all connections to a server even though a IGCS client has terminated all connections to the proxy. For example assume user A (in Figure 4) will close all his connections to the proxy then the proxy may keep all connections to the server since user B still exists. Therefore the IGCS uses a heuristic at the server side to detect if a HTTP connection has been closed by a client. Note: this mechanism comes into operation, only if a client does not close a connection properly e.g. when an application crashes.
- Flow control mechanisms. The IGCS also implements its own simple flow control mechanism for HTTP connections. The server side of a HTTP connection may request the client to send no more data but to send empty requests only on all connections. The server side of a HTTP connection will do this, when buffers for outgoing data exceed a defined limit. This mechanism reduces the probability that a server – especially the group server – is blocked because of a slow client.

¹¹ It depends on the configuration of their firewall if they are allowed to establish SSL connections also.

3.3.4 Server Ports

A server in the IGCS can open an arbitrary number of server ports. Currently there is support for TCP, HTTP and SSL server ports. The handling of incoming connections is transparent to the IGCS components. When a server port is created, a new thread will accept any incoming connection. Then the client side will send a message containing a connection identifier which specifies the logical connection this real connection belongs to. The server port thread decides, if a new logical connection must be created or if it belongs to an already existing logical connection. A new logical connection will automatically exchange all identifiers of local IGCS components. If a logical connection does not provide access to a new IGCS component, the logical connection will be terminated immediately. This way also the termination of connections is transparent to IGCS components.

4 IGCS Usage

IGCS is designed to support applications for cooperative work. The user of IGCS is a software application that implements a tool for a technical application. In order to use the IGCS it is necessary to provide some configuration information first. The software application has to use the API of the IGCS for communication tasks.

4.1 Configuration

This chapter explains all configuration parameters for the IGCS and describes the test methods that are available to check the configuration parameters. Further there are two tools which will help a user or an administrator to perform the configuration.

4.1.1 IGCS Parameters

Name	Type	Parameter Description
<i>IGCSMS address</i>		
IGCSMS_ADDRESS	String	The IP number or DNS name of the IGCSMS host
TCP_PORT_NUMBER	Integer	The IGCSMS port number for TCP connections. Default is 23230
HTTP_PORT_NUMBER	Integer	The IGCSMS port number for HTTP connections. Default is 80
SSL_PORT_NUMBER	Integer	The IGCSMS port number for SSL connections. Default is 443
<i>Firewall / Proxy parameters</i>		
USE_PROXY	Boolean	When set to true the IGCS to uses HTTP for all connections, without testing if TCP connections are possible. Default is false
HTTP_VERSION	String	The HTTP Version "1.0" or "1.1". Default is "1.1". Maybe some older proxies do not allow version "1.1"
PROXY_IS_TRANSPARENT	Boolean	Indicates a transparent HTTP proxy i.e. the destination of a HTTP connection is a server and not the proxy. The values PROXY_ADDRESS and PROXY_PORT are required only if this value is false. Default is true.
PROXY_ADDRESS	String	The IP number or DNS name of the HTTP proxy
PROXY_PORT	Integer	The port number of the HTTP proxy
PROXY_AUTHENTICATION_REQUIRED	Boolean	Indicates that the HTTP proxy requires authentication. Currently the IGCS supports HTTP authentication only, other mechanisms exist but these are not supported. The

		values PROXY_LOGIN and PROXY_PASSWORD are required, only if this value is true. Default is false.
PROXY_LOGIN	String	The login name for the HTTP proxy
PROXY_PASSWORD	String	The login password for the HTTP proxy
PROXY_SUPPORTS_CONNECT	Boolean	Indicates that the proxy supports the optional HTTP V1.1 CONNECT command. This enables to omit all HTTP headers for data transport. Default is false.
PARALLEL_TUNNELING_CONNECTIONS	Integer	The number of HTTP connections established from a client to a server. Default is 2
INCOMING_TUNNELING_CONNECTIONS	Integer	The minimal number of HTTP connections on which the server has priority to send data. Default is 1
SSL support		
KEYSTORE_FILE	String	The filename of the Java keystore file containing certificates for SSL.
KEYSTORE_PASSWORD	String	The password for the Java keystore file.
Service Classification		
SERVICE_LIMIT_HIGH	Integer	The response time limit (milliseconds) for the indication of a HIGH service quality. Default is 30.
SERVICE_LIMIT_MEDIUM	Integer	The response time limit (milliseconds) for the indication of a MEDIUM service quality. Default is 80
SERVICE_LIMIT_LOW	Integer	The response time limit (milliseconds) for the indication of a LOW service quality. Default is 1500
Multicast Support		
ELECTION_MC_IPNAME	String	The multicast IP number used for the election process. Default is 224.1.1.1
ELECTION_MC_PORT_NUMBER	Integer	The multicast port number used for the election process. Default is 42024

Table 2: IGCS parameters

Table 2 shows all IGCS parameters with their name, type and description. Chapters 4.1.2 and 4.1.3 describe some tools for the configuration of the IGCS. But there are some additional configuration parameters that are used only by the IGCSMS. These parameters must be defined as java system properties in the startup script (see Table 3).

Name	Type	Parameter Description
Access Control List		
ACLFile	String	The filename of the initial access control list.
Port Numbers		
OPTIONAL_TCP_PORTS	CSV of Integer	A server may open several TCP ports because of the requirements of different clients. This option specifies a list of TCP port numbers that should be supported by the IGCS also. Note: the CSV must not contain whitespaces.
Firewall / Proxy parameters		
de.dfki.aitvepop.igcs.net.serverHttpVersion	String	The HTTP Version "1.0" or "1.1". Default is "1.1". Maybe some older proxies do not allow version "1.1"
SSL support		

javax.net.ssl.keyStore	String	The filename of the Java keystore file containing the server keys.
javax.net.ssl.keyStorePasswor	String	The password for the Java keystore file.
IGCS_CERTIFICATION	String	The filename of the IGCS certification containing the public key of the server. The content of this file may be transmitted on request.
Status Output		
PRINT_MAP_CHANGES	Boolean	If set to true the IGCSMS will print its complete status after each change.
PRINT_CONNECTION_CHANGES	Boolean	If set to true the IGCSMS will print its total number of logical connections and the number and type of connections after each change.

Table 3: Additional IGCSMS parameters

4.1.2 Configuration Wizard

The configuration wizard guides a user or an administrator to define the most important IGCS parameters. Some configuration steps require that the IGCSMS is up and running. First of all the wizard checks, if a configuration file is already available and offers to show the current configuration. The wizard performs the configuration in six steps:

1. Check and modify the address of the IGCS Management Service
 The user can modify the address of the IGCSMS.
2. Test and configure HTTP connections for proxy/firewall tunneling
 First the user is asked, if the proxy requires authentication, if so the user is prompted to provide the login and password. Then the wizard tests, if the proxy is transparent. If the proxy does not seem to be transparent, the user is asked to provide the address of the proxy and the port number. When the IGCS is able to contact the IGCSMS, the performance will be tested with different numbers of parallel HTTP connections in order to find out how many HTTP connections should be established.
3. Test and configure TCP connections
 The IGCS tries to connect to the IGCSMS using several ports. If the wizard is able to establish a TCP connection, the performance of the connection will be tested.

Note: if the IGCS was not able to establish a HTTP or TCP connection, the wizard will stop here.

4. Search for open TCP ports of your firewall
 If step 3 did not find a usable TCP port, this step may well search for TCP ports that are supported by the firewall of the client. The user can specify a range of TCP ports that will be tested. Note: this step can be performed, only if the establishment of an HTTP connection was successful.
5. Test and configure SSL connections
 The wizard checks, if a certification for the IGCSMS is available. If it is available, it will be presented to the user. Then the user is able to choose to request the current IGCS certification from the IGCSMS. The IGCS will also import the certification into the specified keystore file. Note: the handling of the certification can be performed, only if the wizard is able to call the standard java command “keytool” i.e. the location of this tool must be in the execution path. Finally the performance of the SSL connection will be tested.
6. Save configuration data
 The configuration data will be saved. Note: If the wizard is interrupted before this step, no configuration data will be saved.

4.1.3 IGCSMS Configuration and Test Methods

The IGCS configuration wizard is useful when the IGCS configuration must be setup the first time. There is another configuration tool (BSCConfigTool) that enables a user to modify each parameter and call each test method separately. This tool loads the current configuration and could modify the configuration or perform tests. After that it saves the new configuration. The help text of the IGCSConfigTool describes how to use this tool in detail.

4.2 API

The application programmer interface defines how a user (an application) may interact with the IGCS. A detailed description of all methods is available in the standard java document format. This description is not part of this document. The following subchapters describe the API concepts and give some examples of how to use the most important methods

4.2.1 Concept

The API of the IGCS is object oriented. Each abstract unit of the IGCS is represented by a separate object (see Figure 5). The whole IGCS system is represented by an object called IGCS. This object provides methods to obtain information about the system status. A user also has to authenticate himself with the IGCS object in order to get access to a IGCS group. The IGCSGroup object represents a group within the IGCS. A user may create a new group or may get access to an already existing group. The group object enables an user to become a member of that group by creating a new IGCSGroupMember object. An user must become a group member before he can participate in group communication. Although the IGCSGroupMember object enables the user to send and receive data, most users will also create IGCSChannels which enable more sophisticated communication services. IGCSChannels represent a traffic flow within a group.

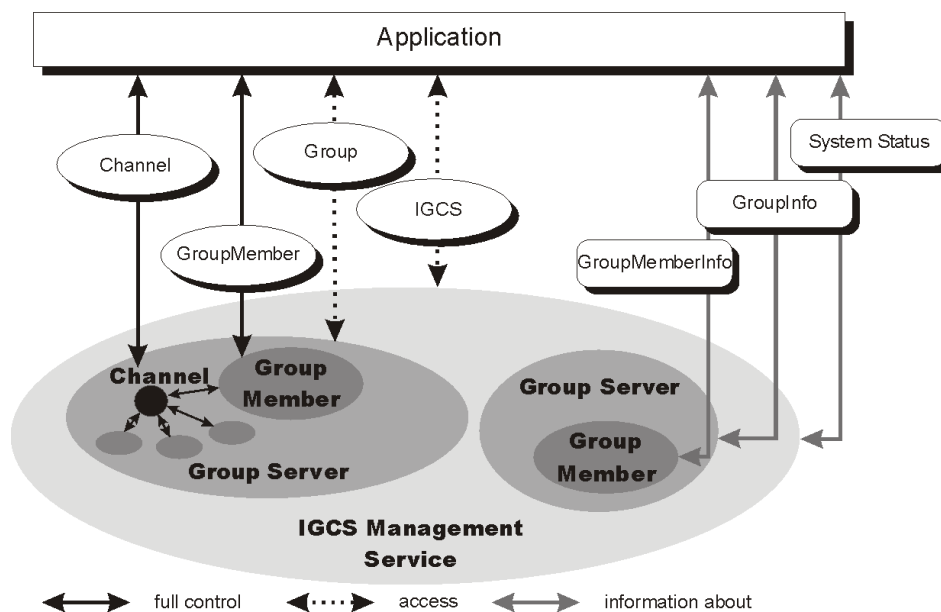


Figure 5: Object oriented API of the IGCS

Further there are objects that contain a description of a group (GroupInfo) or a group member (GroupMemberInfo). These objects are used to describe existing objects as well as to define a new object when the object is created. The user may also request to receive the whole system status which contains GroupInfo and GroupMemberInfo objects for all groups and all group members. Figure 6 shows the relationship between the most important API objects.

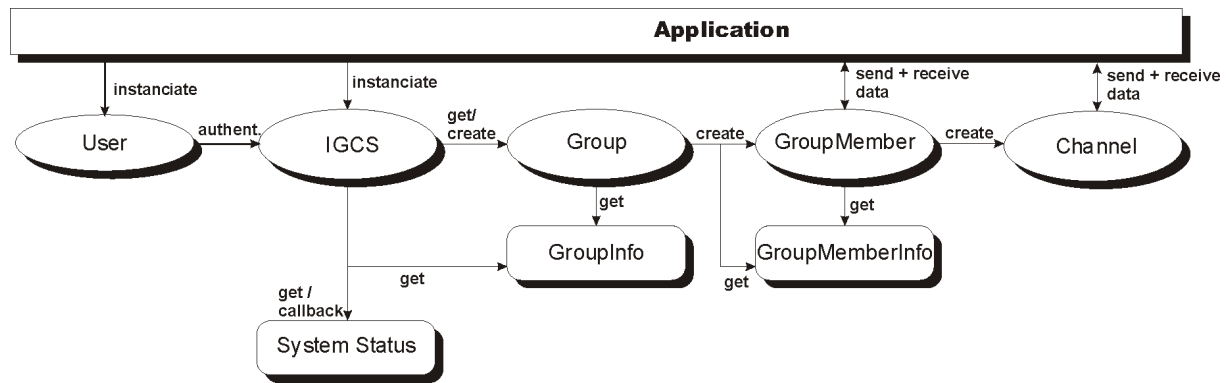


Figure 6: API object relationship

4.2.2 Examples

Example 1: get access to the IGCS and register a listener for system status changes.

```

// create a user object
IGCSUser user = new IGCSUser ("user name", "password");

// create the igcs object and authenticate the user
IGCS igcs = new IGCS (user);

// register listener
igcs.addChangeListener (IGCSSystemChangeListener obj);

```

Example 2: become a member of an existing group, create a channel object and send two objects

```

// get access to the group with the ID 'gid'
IGCSGroup group = igcs.getGroup (gid);
if (group == null) { // error: group does not exist }

// create a description of the new member and create the new member
IGCSGroupMemberInfo gmInfo = new IGCSGroupMemberInfo ("test member",
    "this is a test member only");
IGCSGroupMember member = group.createGroupMember (gmInfo);
if (member == null) { // error: can't create group member exist }

// create channel #5
IGCSChannel channel = member.createChannel (5);

// send obj1 unencrypted (this is default) to all other group
members
channel.send (-1, obj1);

// send obj2 encrypted to group member #2 and reset channel to be
// unencrypted
channel.setEncryptionEnabled (true);
channel.send (2, obj2);
channel.setEncryptionEnabled (false);

```

Example 3: create a group and a group member with predefined identifiers

```
// create a description of the new group including the group
// identifier 13
IGCSGroupInfo gInfo = new IGCSGroupInfo ("test group",
                                         "no description");
gInfo.setGroupID (13);
IGCSGroup group = igcs.createGroup (gInfo);
if (group == null) { // error: can not create group }
// one should check if the group has the requests ID!

// create a description of the new member and create the new member
// with ID 42
IGCSGroupMemberInfo gmInfo = new IGCSGroupMemberInfo ("test member",
                                                       "no description");
gmInfo.setGroupMemberID (42);
IGCSGroupMember member = group.createGroupMember (gmInfo);
if (member == null) { // error: can't create group member exist }
```

Example 4: create a multicast tunnel for a typical Mbone tool

```
// assume 'member' is a valid member of a group
// and the Mbone tool uses two multicast groups 224.2.17.14:32000
// for RTP and 224.2.17.14:32001 for RTCP

// create a forwarder using channel #240 for RTP and 241 for RTCP
MboneForwarder vicRtp = new MboneForwarder (member, 240,
                                             "224.2.17.14", 32000);
MboneForwarder vicRtcp = new MboneForwarder (member, 241,
                                              "224.2.17.14", 32001);
```