SIMPROCESS®
TimeServer

SIMPROCESS TimeServer

Table of Contents

1 Introduction

SIMPROCESS® models now have the ability to let an outside agent manage the advancement of simulation time. That agent is the SIMPROCESS TimeServer. This document describes the TimeServer application and its operational concept, the included TimeServerMonitor application, and the technical information needed by developers to help applications other than SIMPROCESS take advantage of the TimeServer.

2 TimeServer Concept of Operations

The principal idea behind the TimeServer application is to provide an outside agent that can manage simulation time for multiple participants. Only a few basic concepts need be involved in order to handle its responsibilities, thereby providing maximum flexibility and imposing minimal constraints on those using it. Its basic concepts are listed in the following paragraphs.

2.1 Participants are Players

Participants sharing time management facilities are referred to as players. Each will represent a different SIMPROCESS model, or they may represent other applications able to take advantage of the TimeServer's offerings.

2.2 Players are in Groups

To prevent the need for multiple instances of the TimeServer application, it supports groups. Each group will have a unique name among those handled by a single instance of the TimeServer application. Each group will manage its own time independently of all others. They are all under the auspices of a single running instance of the TimeServer application, so that starting or stopping it therefore applies to all of them.

2.3 Time is in Units

Models in SIMPROCESS may have simulation time periods in units of measure ranging from nanoseconds to years. Which one any given model or other player uses isn't relevant to the TimeServer. What it does need, however, is to know what unit of measure each player uses so that it can communicate with a proper understanding. The TimeServer will maintain its current time as a number of nanoseconds from its starting value of 0.0. Each player's time advancement request will be in the player's own unit of measure, which will be translated into nanoseconds when received. When it's time to notify all players of a new time advancement authorization, each will receive the new time in its own time units.

2.4 Time Advances Equally for All Players

At startup, a group will expect a previously known number of players (configuration details are described later). Any player that has joined the group may begin submitting requests for time advancement, but none will be advanced until all members of the group have joined. Once the expected number of players has joined, time advancement will begin only when requests have arrived from all of them. At that point, the smallest requested advance (in the server's own units of nanoseconds) will be selected and all

players will receive notification in their respective time units. After initially joining, some players may drop out or indicate that they have finished their participation. This has the effect of reducing the number of requests that must be on hand before any time advancement notifications can be sent.

2.5 Players Make the Rules

The TimeServer can do nothing to enforce rules concerning time advancement requests except to maintain a current time, reject requests it determines to be invalid, and send out advancement notices only when all of a group's players have submitted requests. But this has the advantage of imposing no unreasonable constraints on how players behave. An individual player can decide whether it's reasonable to drop out when notified that some other player has done so. If the TimeServer's menus are used to reset the group's time back to the beginning, each player must determine whether it can continue or must drop out. What a player does between sending and receiving messages is unknown to the TimeServer, which is only interested in managing time for the group. In short, it's the players who make the rules and are in charge.

3 Installed TimeServer Components

The components of the SIMPROCESS TimeServer application are included with SIMPROCESS in its *timeserver* directory. The individual components are described in the following paragraphs.

3.1 The *timeserver.jar* file

This is the single most critical component, as it contains the Java code which is the heart of the TimeServer. Though it may be copied to other locations to run a TimeServer or TimeServerMonitor application, this file ***must*** remain in the *timeserver* directory since that's where SIMPROCESS will expect to find it when running any simulation that uses a TimeServer.

3.2 The *timesrvconfig.xml* file

This file contains some configuration properties required by the TimeServer. It is required to be in the same directory with the *timeserver.jar* file unless its location and name have been changed using command line options discussed later. If a TimeServer is to be operated from a different location (or on another system), make sure this file is also copied there.

3.3 The *StartTimeServer* scripts

SIMPROCESS is available for multiple platforms, so files are provided for Windows (with the *.bat* extension) and other Unix-like platforms (with the *.sh* extension) that can be used to start the TimeServer. Each will launch the TimeServer using the installed Java Runtime Environment (JRE) in the *jre* directory of SIMPROCESS, and each will launch the program in the background. If it's desirable to run the TimeServer from some other location, or from another system that includes an appropriate version of Java, these files can be copied and modified as needed.

3.4 The *StartMonitor* scripts

Files are also provided for starting the TimeServerMonitor on various platforms. Each script will launch the TimeServerMonitor using the installed JRE and each will launch the program in the background. These files can also be copied and modified as needed to run a monitor from another location or on another system.

3.5 The *doc* directory

This directory contains documentation for portions of the Java source code used by the TimeServer and TimeServerMonitor programs. Its contents are in the popular "Javadoc" format familiar to Java developers. Look inside this directory for the *index.html* file and open it in a web browser. The Java classes documented here will be discussed further in the portion of this document addressing technical details.

4 Operation of a TimeServer

The SIMPROCESS TimeServer is designed to be as simple as possible to configure and operate. When run, it displays a single window that displays occasional messages about things that are occurring (enabling debugging will produce much more text). Its menus will be described in the following paragraphs, along with the properties found in the *timesrvconfig.xml* file and other options for overriding some properties.

4.1 Configuration

The *timesrvconfig.xml* file will typically contain all the configuration properties for a TimeServer, though any property can be overridden at the command line. Refer to the sample file provided at installation time to see the format this file must take. It must be an XML file; its root element name must be *properties*; it may contain a single optional *comment* element; and each of the properties to be used by the TimeServer must appear in an element named *entry* having an attribute named *key*, the value of which provides the name of the configuration property for which a value is being given. For example:

```
<entry key="timesrv.timeout">30</entry>
```

This *entry* element specifies a value of 30 for the property named *timesrv.timeout*.

4.1.1 Supported Configuration Properties

- timesrv.groups – This is the only property that is **absolutely required**. It must provide a unique name for each simulation group the TimeServer is expected to support during any session, and it must identify the number of players expected to join the group. It must, at a minimum, name one group and the group must have at least one participant (e.g., SimGroupA.1). (Group names are case sensitive.) This allows a TimeServer to support any number of simultaneous simulation groups, each operating independently of others. By providing a count of the expected number of participants, each group is able to determine when all have connected so that it can begin evaluating requests for time advancement.

- timesrv.debug – This optional property should have a value of "true" or "false". When "true", additional debugging information will appear in the TimeServer's display window.
- timesrv.timeout – This optional property specifies the length of time in seconds that will be used by sockets created by the TimeServer when initial connections are received. The default is 30 seconds. It is recommended that this property always be used. When a non-zero value is used, new connections will be discarded if no communication occurs within this amount of time, thereby avoiding dead connections and unnecessary network traffic.
- timesrv.logfile – This optional property specifies a file name (without extension) to be used for redirection of standard output and error streams when a TimeServer launches. When this property is not present, default names of TimeServer.log and TimeServer.err will be used. If a value is provided, it will have the extension ".log" appended for standard output redirection and ".err" for standard error.
- timesrv.port – This optional property specifies the network port on which socket communications with the TimeServer will take place. The default port of 26100 was selected from among those not reserved by the Internet Assigned Numbers Authority (IANA). A complete list of "well known" ports, as well as those reserved for specific uses or otherwise registered, can be found on the Internet by visiting http://www.iana.org/assignments/port-numbers. The port selected for use by a TimeServer must be one between 1024 and 65535, inclusive.
- timesrv.config – *This property cannot be used in the configuration file.* Instead, it must be specified on the command line (learn how in the next section). When provided, its value will be used to locate a configuration file that should be used instead of the default *timesrvconfig.xml* file. The value provided for this property must be the complete file name (it must still be in the format described above, whether or not its name uses the *.xml* extension), and may include path information if desired.

4.1.2 Setting Properties via Command Line

All configuration properties that are supported by the TimeServer can also be provided via the command line. When this method is used to provide a property value, it will override any entry for that property in the configuration file. A property value can be provided in this way even if no entry appears in the configuration file.

The command line mechanism for providing a property value is that used by Java. For each property value to be set, a string similar to the following must be added to the command line (such as the one that appears in the script files described earlier):

```
-Dtimesrv.port=26100
```

In this example, a value is being provided for the network port to be used by the TimeServer. The "-D" portion indicates to Java that a property name and value follow. Any property name supported by the TimeServer can be used, followed (without any intervening spaces) by "=" and an appropriate value. When the value contains spaces (as is possible if a path is provided for the timesrv.config or timesrv.logfile properties), it

should be quoted to ensure it's correctly interpreted. **Important Note:** This switch, when used, must appear prior to the "-classpath" entry found on the command line in the provided script files.

Any number of configuration properties can be set in this way, subject to any operating system restrictions on the total length of the command line used to invoke the TimeServer program. Though all properties can be provided in this way, the TimeServer program still requires a properly formed configuration file. If it is unable to locate the default *timesrvconfig.xml* file or one specified in the "timesrv.config" property, the TimeServer will report an error and terminate.

4.2 Using the TimeServer Application

4.2.1 Running the TimeServer Application

The TimeServer application can be started by using one of the StartTimeServer script files provided at installation, or it can be run from another location or even on another system. To run it from another location or system, copy the *timeserver.jar* and *timesrvconfig.xml* files to the location from which it will run. You can also copy one of the provided script files and modify it as appropriate to indicate the location of the "java" or "javaw" commands it needs and to include any optional configuration parameters. **Important:** The TimeServer application uses language features introduced in Java 5.0 (also sometimes called Java 1.5), so this is the minimum version required to run the application.

4.2.2 Using the Menus

When launched, the TimeServer application will read its configuration file and, if no errors occur, display its window indicating the names of each group and their player counts obtained from configuration properties. It will then indicate that it is accepting connections on the designated port. The TimeServer's window provides two menus, described below.

4.2.2.1 File Menu

The `File` menu contains the following menu items.

- `Save` – When this item is selected, a file browser will appear with a default file name of *TimeServer.txt* entered. The location and/or name can be changed as desired. Clicking "Save" will write the text from the window to the specified file. If the file already exists, it will only be overwritten after requesting permission.
- `Clear` – When this item is selected, the contents of the TimeServer's display window will be cleared.
- `Exit` – When this item is selected, the TimeServer program will terminate. If there are any existing connections, they will be shut down without warning. Clicking the window's close decoration will act as if this menu item were selected.

4.2.2.2 Reset Menu

The `Reset` menu includes the following menu items.

- `Reinitialize Group` – This item is designed to be used when multiple groups are in operation and one needs to reinitialize as though the TimeServer program had just been started. Existing participants will have messages sent to inform them that the server is reinitializing and then their connections will be closed. Once this is done, the group will once again wait for the required number of participants to join before honoring requests for time advancement.
- `Reset Group Time` – This item will reset the time for the specified group to zero. All currently connected participants will be sent messages informing them of this change.

5 Operation of a TimeServerMonitor

The TimeServerMonitor program is provided as a convenience and allows monitoring much of the information exchanged between a TimeServer and applications participating in groups (players).

5.1 Starting TimeServerMonitor

Script files are provided for running TimeServerMonitor using the JRE installed with SIMPROCESS. No configuration is required, and the program can run on any system that has the ability to communicate with the desired TimeServer and an appropriate Java version. Copy *timeserver.jar* and an appropriate script file, changing the latter as needed.

5.2 TimeServerMonitor Menus

Like TimeServer, the TimeServerMonitor program displays a window where text will appear to provide its information. This text can be cleared or saved via the `File` menu, as described above for TimeServer. Its other menus are described below.

5.2.1 Monitor

This menu contains two items, only one of which will be enabled at any time.

- `Monitor Group…` – This menu item will be enabled when no group is being monitored. When selected, it will prompt for the information required to connect to a TimeServer, including a host name or Internet Protocol (IP) address, port, the name of the group to monitor, and the time unit to be used for its displays of time related messages. When a successful connection is established, this menu item will become disabled. The TimeServerMonitor program can only monitor one group at any time. Multiple instances can be run if multiple groups need to be monitored.
- `Disconnect Monitor` – This item will be enabled only when a monitor connection has been established. Selecting it will cause that connection to be terminated, after which this item will be disabled and the `Monitor Group…` item will be enabled. In the event that the TimeServer sends a message of type RESET_TIME (as described later), the monitor's connection will be dropped and this item automatically selected.

5.2.2 Monitor Commands

This menu will only be enabled when an active monitor connection has been established. It contains the following:

- `Get Current TimeServer Time…` - When selected, a message will be sent to the TimeServer requesting the current time for the group being monitored. A reply will be expected indicating that the request came from "Monitor" and providing the group's current time. It will be displayed as a number representing the time units used when the monitor connected.

5.3 TimeServerMonitor Message Window

Communication between "players" and the TimeServer will take the form of messages with specific "messageType" values described later in this document. (See that section for more detail on their meanings and the contents of each message.) In general, TimeServerMonitor will receive information about the most common information exchanges that occur. Some examples are:

- `IDENT` – Each player making a connection to a TimeServer will initially be sent a `SEND_IDENT` message. (This is also true for a monitor; see the discussion of protocols later in this document.) The required response is a message of type `IDENT` that also includes the name of the group a player wishes to join, the name by which it will be known (unique within the group; all monitors use the name "Monitor"), and the time units it will use. If a player's `IDENT` message is invalid, no monitors will be advised. A successful player `IDENT` message will be forwarded to all monitors, and a message will appear in the window indicating the essential details.
- `REQUEST_TIME_ADVANCE` – Each player in a group must request permission to advance to the desired time. Requests will be in messages of this type and will be forwarded to all monitors. TimeServerMonitor will display the name of the player and the time, in the player's time units, requested.
- `END_RUN` – A player can send this message indicating that it will not proceed until the TimeServer sends a `GROUP_RESET_TIME` message. This is analogous to a SIMPROCESS model completing a replication when the Reset System setting is used. Each such message will be passed on to all monitors.
- `ADVANCE_TO_TIME` – When requests have been received from all players in a group, the shortest time will be selected and a message of this type sent to all players and monitors. TimeServerMonitor will display the time in its own units (the one specified when establishing the connection to a group).
- `CURRENT_TIME` – Any player or monitor can ask the TimeServer for its group's current time by sending a `GET_CURRENT_TIME` message, and the reply will be this type. If requested by a player, the group will forward the reply to all monitors. When received by TimeServerMonitor, the name and time units of the requester will be displayed.
- `PLAYER_COMPLETION` – When a monitor wishes to stop, it sends a message of this type to the TimeServer and will no longer receive other messages. When a player wishes to indicate that it has completed its operations, it will send this message and all other players and monitors in the group will be notified. When

TimeServerMonitor receives a message of this type, it will display the name and time of the sender.

- `PLAYER_TERMINATION` – When a player wishes to quit, or encounters an error of some kind, its group will see a message of this type. It will be forwarded to all other players and monitors. TimeServerMonitor will display the name indicated in the message.
- `GROUP_REINITIALIZE` – When `Reinitialize Group` is selected by the operator of TimeServer, a message of this type is sent to all the group's players and monitors. TimeServerMonitor will display a message when it receives such a message and break its connection to the TimeServer. The menu states will then be as if the `Disconnect Monitor` menu item had been selected, so that monitoring of the same or another group may begin if desired.
- `GROUP_RESET_TIME` – When `Reset Group Time` is selected by the operator of TimeServer, monitors will also be notified. TimeServerMonitor will display a message indicating that it has received such a message. The TimeServer will also send this message once it has received `END_RUN` messages from all players in a group.

6 Software Interface to TimeServer

This section contains information for software developers that will help them to enable their own programs to communicate with a TimeServer and act as players. It includes discussion of the Java classes that serve as the Application Programming Interface (API). It also addresses the protocol for communication between the server and a player (that is, the specific sequence of messages that must occur when establishing contact and to take other actions that will be recognized by the TimeServer). Because the TimeServer is written entirely in Java, all of this information is presented in terms of that language. The underlying mechanism for communicating – sockets – is not limited to Java, however, and code in C or C++ that takes advantage of the Java Native Interface (JNI) and can readily be devised to communicate with the TimeServer application.

6.1 TimeServer API

All discussion in this section of using the API documented in the Javadoc files installed in the *doc* directory assumes that the player is developed in Java.

6.1.1 TimeUnits

`TimeUnits` is an enumeration which provides a set of constants representing all of the time units supported by the TimeServer application. Each message needing to include time units must use one of these constants in its `timeUnits` field (see the `CommData` class below).

`TimeUnits` also provides static convenience methods for translating time values to or from nanoseconds, and for obtaining any of the constants from a `String`.

Important Note concerning NANOSECOND values: The smallest unit of time supported by the TimeServer application is nanoseconds, and all its internal operations

use this unit of measure. It **does not** allow fractional nanoseconds. All nanosecond values will be rounded up using the *java.lang.Math.ceil(double)* method. See comments in the Javadocs for any further information.

6.1.2 MessageType

`MessageType` is an enumeration which provides a set of constants for use in the `messageType` field of `CommData` objects exchanged between a TimeServer and players. Some of the values there will never be placed into a message by the TimeServer application, especially if their documentation indicates that they are to be used internally.

6.1.3 CommData

The `CommData` class is perhaps the single most important one in the TimeServer API. Every message exchanged between the TimeServer application and players will be a *serialized instance* of this class. Every instance requires a value in its `messageType` field that will be understood at the other end (the protocol section will provide further information in this area). When a connection is received from a potential player, the TimeServer will invoke a handler that will immediately send a message in which this field contains a value of `MessageType.SEND_IDENT`. If the first response received by a player after connecting is not an instance of this class, or if it does not contain `MessageType.SEND_IDENT` in the `messageType` field, the player can be confident that the sender isn't a TimeServer. In reply, a player is expected to send an instance of `CommData` with a `messageType` of `MessageType.IDENT`. The `groupName` field will indicate which group the player wants to join. The `playerName` field will provide a name by which the player will be known (when seen in a monitor, for instance) within the group. And the `timeUnits` field will state the player's choice of time units using one of the values from `TimeUnits` as described above. The TimeServer application can be confident that the connection comes from a potential player when it receives this recognizable reply.

All of the member variables in this class are public and can be set directly in code. The only methods provided are a single constructor and some static convenience methods which simplify creation of the most commonly exchanged types of messages, each taking parameters that will populate the fields required for that particular message type.

6.2 Communication Protocol

In order for any two programs to communicate, there must be an understood and agreed upon "language" for communicating between them. When one side sends a message, it should know what kinds of responses the other might send in reply. This constitutes the protocol to which all participants must adhere. This section describes the protocol for communications between the TimeServer application and any player (or monitor).

6.2.1 First Contact

When the TimeServer application starts and successfully completes its initialization, it listens for connections on the port indicated in its configuration. Any application can

connect to that port and the TimeServer will immediately invoke its handler to initiate communication according to these protocols.

The handler's first actions will be to open input and output streams on the newly created socket, enabling it to read from and write to it. If it is unable to do so, it will terminate the connection and take no further action. The following Java code demonstrates how these streams are initially created:

```
// The output stream must be opened first to
// prevent blocking on the input stream's constructor
oos = new ObjectOutputStream(socket.getOutputStream());
ois = new ObjectInputStream(socket.getInputStream());
```

As the comment in this sample indicates, the constructor of `ObjectInputStream` will block if there's not an open `ObjectOutputStream` on the socket (both of these classes are in the package `java.io`). A client application (whether a player or monitor) needs to execute actions like those above as soon as possible after successfully opening a socket to the TimeServer.

The handler's next step is to send an instance of `CommData` with a `messageType` field containing a value of `MessageType.SEND_IDENT`. For this reason, the player should very quickly execute code that listens for a message of this type. An example follows.

```
CommData data = null;
try {
    data = (CommData)ois.readObject();
    if (data.messageType != MessageType.SEND_IDENT) {
        closeStreams();
        closeSocket();
        return;
    }
}
catch (ClassNotFoundException cnfe) {
    System.err.println("ClassNotFoundException reading SEND_IDENT
request");
    cnfe.printStackTrace();
    closeStreams();
    closeSocket();
    return;
}
catch (IOException ioe) {
    System.err.println("IOException reading SEND_IDENT request");
    ioe.printStackTrace();
    closeStreams();
    closeSocket();
    return;
}
```

This code blocks while waiting for its `ObjectInputStream` to read an instance of `CommData`, handling the `ClassNotFoundException` that occurs if the result is not an instance of that class as well as any possible `IOException` that might occur during

the operation. It also checks to ensure that the received object's `messageType` field contains the appropriate value, since this will always be the first thing sent by a handler.

The TimeServer's handler, after sending the message to be received by player code much like that above, will execute similar code awaiting the player's reply. That reply must be an instance of `CommData` which has a `messageType` field that has a value of `MessageType.IDENT`; it must also contain appropriate values for the group name, player name and the `timeUnits` field. Sample code for sending this key message might look like the following:

```
// Reply with IDENT
try {
    sendMessage(CommData.IDENT(groupName, playerName, timeUnits));
}
catch (IOException ioe) { /* handle the exception here */ }
```

Note the use of the static convenience method on the `CommData` class.

6.2.2 READY…or Not

If the values placed in the `IDENT` message are for some reason not acceptable, the next message received from the handler will indicate the nature of the problem. The reply would then contain one of the following `messageType` values:

- `MessageType.INVALID_REPLY` – indicates that the message sent was not an `IDENT` message or that either of the group or player name fields were `null`.
- `MessageType.UNKNOWN_GROUP` – indicates that the message referred to a group name which is not known by the TimeServer.
- `MessageType.GROUP_FULL` – indicates that the named group already has the expected number of players and will not honor the request (this will never occur when a monitor is connecting, as all will use the name "Monitor" and will not be counted).
- `MessageType.DUPLICATE_PLAYER_NAME` – indicates that the message included a player name which has already been used within the group (this will not occur when a monitor is connecting).
- `MessageType.ERROR` – indicates that some other unexpected condition prevented the group from accepting the player's request to participate. There may be additional details as to the actual source of the problem in the server's display window or in its redirection files.

If any of these situations should occur, the TimeServer's handler will wait briefly to allow time for the reply message to received by the new player or monitor and then close the streams and socket, preventing any further attempts to communicate until a new connection is established.

Otherwise, the reply to a successful `IDENT` message will have a `messageType` field that has a value of `MessageType.READY`. The TimeServer and players (or monitors) are now free to communicate according to the protocols described in the next section.

6.2.3 Protocol Message Sequences

The tables below list the messages that may be sent from players, monitors or from the TimeServer itself after the initial exchange already described. All message type entries in the tables represent constants from the `MessageType` class.

Player Message Types	Description	Other Fields Required	Expected Response
`REQUEST_TIME_ADVANCE`	Ask server to advance time to the value indicated; server will forward to all monitors but will never send messages of this type to players.	Requested time, in player's own time units.	`REQUESTED_TIME_IN_PAST` if time is already past; `RESET_NOT_SENT` if player has previously sent `END_RUN`; otherwise none.
`END_RUN`	Advises the server that the player will wait for receipt of a `GROUP_RESET_TIME` message before proceeding further (equivalent to a SIMPROCESS replication ending when Reset System is specified in a model's Run Settings).	None	None; player should send no further messages until it receives a `GROUP_RESET_TIME` message.
`BAD_DATA_PACKET`, `CONNECTION_CLOSED`, `UNKNOWN_ERROR`	These messages from any player will be treated as error conditions; all monitors and remaining players will receive a `PLAYER_TERMINATION` notice for the sending player.	None	None
`GET_CURRENT_TIME`	Asks the TimeServer to send the current time for the group.	None	`CURRENT_TIME` in player's own time units
`PLAYER_COMPLETION`	Advises the server that this player is completing its participation and disconnecting.	None	None; server will forward to all remaining players and all monitors with the player's name added.
`PLAYER_TERMINATION`	Sent by player to advise the server when dropping out.	None	None; server will forward to all remaining players and all monitors with the player's name added.

Table 1: Message types allowed from Players

Monitor Message Types	Description	Other Fields Required	Expected Response
`GET_CURRENT_TIME`	Asks the TimeServer to send the current time for the group.	None	`CURRENT_TIME` in monitor's own time units
`PLAYER_COMPLETION`	Advises the server that this monitor is disconnecting.	None	None

Table 2: Message types allowed from Monitors

TimeServer Message Types	Description	Recipient	Other Fields Required	Expected Response
`REQUEST_TIME_ADVANCE`	When received from any player and no error reply is warranted, forwarded to all monitors	All monitors	Name of requesting player and time requested, in monitor's time units.	None
`REQUESTED_TIME_IN_PAST`	Sent to any player in reply to a `REQUEST_TIME_ADVANCE` message when the requested time has already passed.	Requesting player only	None	None

TimeServer Message Types	Description	Recipient	Other Fields Required	Expected Response
RESET_NOT_SENT	Sent to any player in reply to a REQUEST_TIME_ADVANCE message when the player previously sent END_RUN.	Requesting player only	None	None
CURRENT_TIME	Sent in response to a GET_CURRENT_TIME message.	Requesting monitor only if requested by a monitor; otherwise to requesting player and all monitors.	Name of requesting player and time requested, in monitor's time units.	None
PLAYER_TERMINATION	Sent to advise that one player has dropped out.	All remaining players and all monitors.	Name of player terminating.	None
PLAYER_COMPLETION	Sent to advise that one player has reported completion.	All players and all monitors.	Name of player terminating.	None
GROUP_RESET_TIME	Sent to advise that the operator of the TimeServer has reset the group's time to zero; will also be sent if all players have sent END_RUN messages.	All players and monitors.	None	Players may respond by terminating or completing if they cannot comply.
GROUP_REINITIALIZE	Sent to advise that the operator of the TimeServer has reinitialized the group.	All players and monitors.	None	All players and monitors must disconnect.
SERVER_STOPPING	Sent to advise that the operator of the TimeServer has instructed it to shut down.	All players and monitors.	None	All players and monitors should disconnect; the TimeServer program is terminating and will break all connections.

Table 3: Message types sent by the TimeServer

If any player or monitor sends a message not described above, it will be noted in the TimeServer window's display window if debugging is enabled but will otherwise be discarded.

6.3 Developing a TimeServer Interface using JNI

The SIMPROCESS TimeServer application is written entirely in Java, which makes it able to run on a variety of platforms (whether or not SIMPROCESS is available for them). However, that does not preclude the possibility of another non-Java application being able to participate in a group's simulation activities. The Java Native Interface (JNI) is a specification from Sun Microsystems that allows Java code to call platform native code written in C or C++, and vice versa. If you are interested in adapting an application to use the features of the TimeServer application, contact us to discuss it.

7 What Messages Mean to Players

The preceding section provides technical details needed to implement a player application. But it's important for those using existing player applications, such as SIMPROCESS, to understand what some messages mean to them.

7.1 Basic TimeServer Philosophy

The TimeServer application's only purpose is to provide a simple time management service. That can best be accomplished by imposing as few rules as possible on what players do. Therefore, the TimeServer doesn't need to understand concepts that player applications might need, such as SIMPROCESS replications or warm-ups. On the other hand, it needs to provide ways that some of those concepts can be embodied in the service that it does provide. The explanations that follow for some of the messages and TimeServer responses are sometimes in terms of a SIMPROCESS concept, but it's important to understand that the TimeServer is deliberately aimed at providing a service that can be utilized by any player application.

7.2 Player Messages

7.2.1 Error Messages from Players

When a player sends a `BAD_DATA_PACKET`, `CONNECTION_CLOSED`, or `UNKNOWN_ERROR` message, it will be viewed by the TimeServer as an exceptional situation that it's unable to handle. The connection to that player will be invalidated, and all remaining players and monitors will be sent a `PLAYER_TERMINATION` message to inform them of the loss of a player. In other words, receipt of any of these message types will be as if the player had sent a `PLAYER_TERMINATION` message (see below).

7.2.2 PLAYER_TERMINATION

Any player can elect to terminate its participation prematurely for any reason it may choose. When it does so, it is expected to send a `PLAYER_TERMINATION` message. When SIMPROCESS terminates a simulation due to an error, it will send such a message. When SIMPROCESS receives a `PLAYER_TERMINATION` message noting that another player has stopped early, its response will be to ask whether to allow the simulation to continue. If allowed, it will continue normally; if not, it will send its own `PLAYER_TERMINATION` message.

7.2.3 REQUEST_TIME_ADVANCE

This is the type of message all players are required to send in order to synchronize their time clocks with the TimeServer. The following are the steps a TimeServer will take each time it receives one of these messages:

1. The identity of the player sending the message will be checked to see if it has previously sent an `END_RUN` message. If it has, the TimeServer will reject the request and send a `RESET_NOT_SENT` message in reply.
2. The requested time (after converting it into Nanoseconds, since the TimeServer tracks its own time in that unit of measure) will be examined to see if it is in the past. If it is, the TimeServer will send a `REQUESTED_TIME_IN_PAST` message in reply and reject the request. Otherwise, the request is accepted.
3. Having been accepted, the TimeServer will first notify any monitors of the request and then place it on a list with any others it has previously received. It will then check to see if it should send all players a new `ADVANCE_TO_TIME` message. The conditions that must be met are:

- All expected players (as indicated in the configuration of the TimeServer) must have joined the group;
- The number of players still participating (those who have not yet sent PLAYER_COMPLETION, error or PLAYER_TERMINATION messages) must be greater than zero;
- All of those remaining players must either have previously submitted their own REQUEST_TIME_ADVANCE messages or be on the list of players who have sent END_RUN messages.

4. When these conditions have been met, a new ADVANCE_TO_TIME message will be built using the smallest requested value found on any of the requests. This message will then be sent to all players except those having sent END_RUN messages, and to all monitors.

7.2.4 PLAYER_COMPLETION

When a player reaches what it considers to be the conclusion of its simulation activities, it should send a PLAYER_COMPLETION message. When the TimeServer receives this message, it first forwards it on to all monitors and all players (including the sender). It then removes the sender from its list of active players. Finally, the TimeServer takes two additional steps to see if the removal of this player warrants further action. First, it checks the list of REQUEST_TIME_ADVANCE messages to see if the reduction in player count satisfies the conditions described above and sends a new ADVANCE_TO_TIME message if appropriate. Then it checks the list of players having sent END_RUN messages to determine if it should send a GROUP_RESET_TIME message (see the description of END_RUN below).

7.2.5 END_RUN

The END_RUN message signifies that a player has reached a point where it requires resetting simulation time to 0.0 in order to continue. In SIMPROCESS models, this occurs when a model has multiple replications and Reset System is set in its Run Settings. (This may or may not have an analog in other players.) After sending its END_RUN message, a player should send no additional messages until it receives a GROUP_RESET_TIME message. In a SIMPROCESS model meeting the above conditions, an END_RUN message will be sent at the end of each replication except the last; that one will send a PLAYER_COMPLETION message instead.

When the TimeServer receives an END_RUN message, it will start by adding the sender to its list of those waiting for time to be reset. Then it will check its conditions to see if it should send a GROUP_RESET_TIME message. The conditions that must be met are:

- All expected players must have joined the group, as indicated in the configuration of the TimeServer;
- The number of players still participating (those who have not yet sent PLAYER_COMPLETION or PLAYER_TERMINATION messages) must be greater than zero;

- The list of players having sent `END_RUN` messages must include all remaining players.

Once these conditions are met, the TimeServer will send a new `GROUP_RESET_TIME` message to all players. It will also reset its own time to 0.0.

7.2.6 GET_CURRENT_TIME

Any player may ask the TimeServer to tell it the current time by sending a message of this type. When received, the reply will be a `CURRENT_TIME` message using the player's own time units and the time as maintained by the TimeServer. SIMPROCESS does not send messages of this type.

7.3 TimeServer Messages

Aside from the responses to player messages just described, there are three additional messages that might come to a player from the TimeServer unsolicited.

7.3.1 GROUP_RESET_TIME

This message may be sent when all players have sent `END_RUN` messages. But it may also be sent in response to actions taken by the operator of the TimeServer application itself. If the TimeServer's menus are used to cause a group to reset its time, a message of this type will be sent to all players and monitors in the group. In SIMPROCESS, receiving a `GROUP_RESET_TIME` message when it isn't expected will be treated as an error condition and the simulation will end.

7.3.2 GROUP_REINITIALIZE

This message can only be sent in response to a menu item on the TimeServer application. When sent, it will indicate that the group is being reinitialized as though the TimeServer had just been launched. No further communication with the group will be possible without first going through the initial steps described in an earlier section. When a model in SIMPROCESS receives this type of message, it will be treated as an error and the simulation terminated.

7.3.3 SERVER_STOPPING

This message can only be sent when the operator of the TimeServer application has told it to shut down by either closing its window or using a menu item. When this type of message is received, the only possible action is to stop all communication. When SIMPROCESS receives such a message, it will terminate the simulation.