SIEMENS

SIMIT - Remote Control Interface (RCI)

Function Manual

06/2018
A5E44876253-AA

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Division Process Industries and Drives
Postfach 48 48
90026 NÜRNBERG
GERMANY

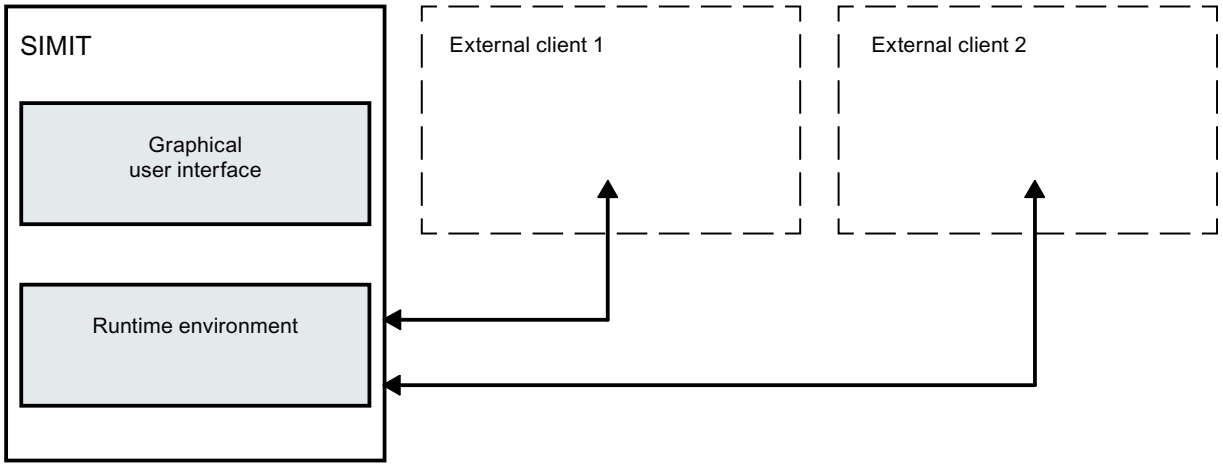
A5E44876253-AA
℗ 07/2018 Subject to change

Table of contents

Introduction

1

SIMIT features an interface (Remote Control Interface - RCI) that informs external applications (external clients) about the simulation state and allows them to assume control of the simulation.



This enables other simulators to be coupled to SIMIT. The operation of the simulation can then be performed by SIMIT or the external simulator.

Setup firewall

2

Introduction

Set the firewall to ensure that SIMIT SP can establish a connection to the following systems:

- Further SIMIT SP instances in a distributed simulation
- Distributed Virtual Controllers
- Third-party systems which establish a connection to SIMIT SP using a simulation controller (RCI)

NOTICE
Read the security notes included in the foreword
Changes to the firewall may influence the security of your system!
Contact, if necessary, your system administrator before performing the steps described in the following.

Requirement

- SIMIT has been installed.
- All nodes are located within the same network.

Procedure

Proceed as follows to set up the firewall:

1. Open the "Windows Defender Firewall with extended safety", for example, by entering "wf.msc" in the system prompt and confirming with "Enter".
2. Click on “Incoming rules”.
3. Double-click "SIMIT-CS Manager".
   The “Properties of SIMIT CS Manager" windows opens.
4. Select the “Area” tab.
5. Under “Remote IP address”, select “Arbitrary IP address”. Alternatively, add the IP addresses of the systems involved to the list of approved remote IP addresses.
6. Confirm with "OK".

Result

The firewall has been set such that SIMIT SP can set up a connection to other systems.

SIMIT operating modes

3

To understand the RCI interface, you need to become familiar with the two strategies SIMIT uses to process the model, the couplings and any logged-on clients.

Asynchronous operating mode

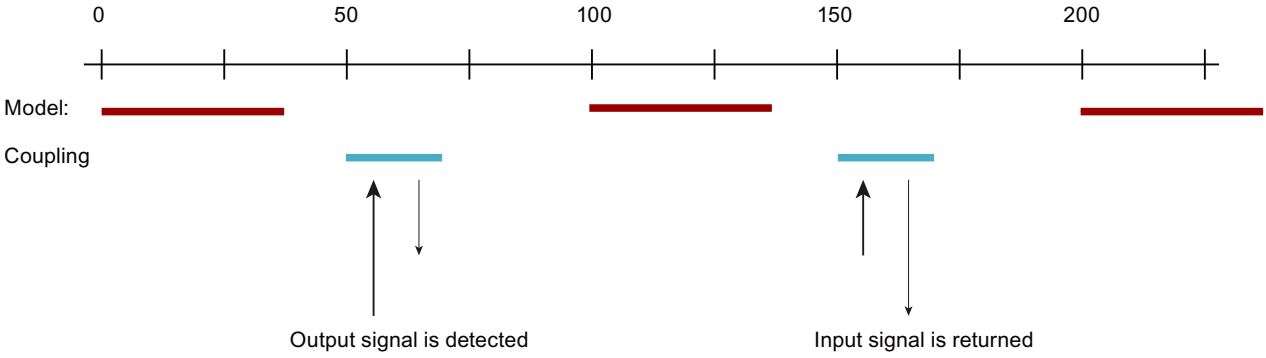
The model calculation of the individual time slices and processing of the couplings in SIMIT is performed time-controlled.

Logged-on clients can only query as to whether SIMIT is working in real-time, faster or slower. All logged-on clients must take care of the time management (scheduling) themselves. If SIMIT or one of the clients cannot comply with the scheduled time, this has no effect on the other logged-on clients.

If SIMIT does not complete the simulation model of a time slice within the scheduled time, one or more processing cycles are omitted and the other time slices are not calculated. The calculation of the simulation model only continues when all the time slices can calculate once again.

If SIMIT cannot calculate a coupling in the scheduled time, once again one or more processing cycles are omitted, but the calculation of the couplings in other time slices and the calculation of the simulation model are not affected. This means that, if couplings are blocked by a communication problem, this does not affect the overall processing.

In order to achieve the fastest possible response times, couplings are scheduled with a time offset compared to the model calculation:



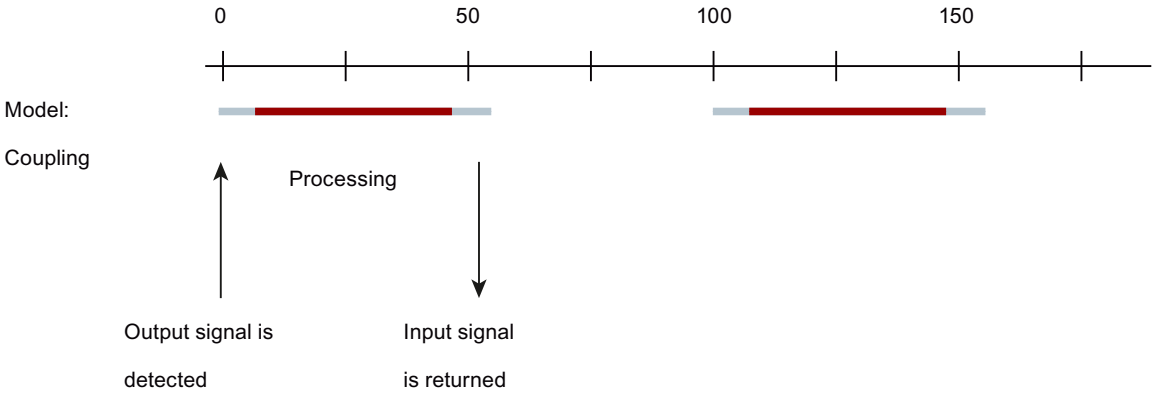
If the model calculation or the data communication of couplings takes longer than half of the cycle time, this time offset does not result in optimization.

Synchronous operating mode

With synchronous mode, submodels, couplings and any logged-on clients are calculated in a precisely specified sequence. The next action does not start until the previous action has finished.

A PROCEED command explicitly triggers each calculation step and determines the amount by which to advance the simulation time.

In addition, the coupling is separated into the acquisition of output signals and the writing of input signals:



This results in better response times with short cycle times. However, each submodel, each coupling and each logged-on client can block the entire simulation execution.

3.1 Determining the operating mode (synchronous/asynchronous)

The client that starts the simulation determines the operating mode. In synchronous mode, the client indicates whether it is actively taking over synchronization. A client can either support (response: yes) or not support (response: no) the operating mode. It cannot request another operating mode.

In SIMIT, you can select the operating mode to be started when the user interface opens.

In synchronous operating mode, SIMIT always starts "passive", i.e. logged-on clients have priority with regard to synchronization. SIMIT allocates control of the timing according to the following rules:

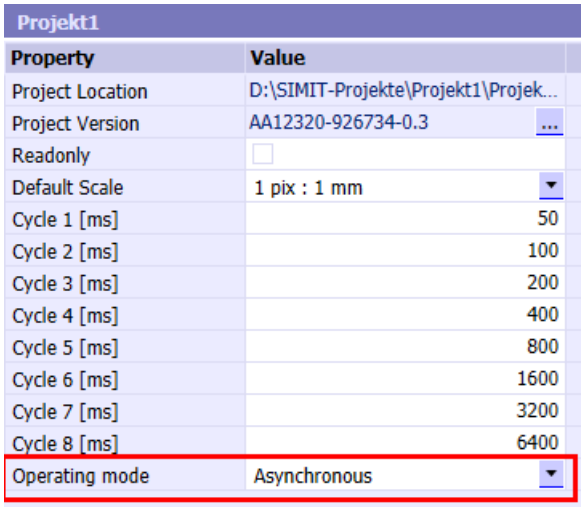
Precisely one client wants to take over the synchronization: Simulation can start.

No client wants to take over the synchronization: SIMIT takes over the synchronization.

Several clients want to take over the synchronization: SIMIT reports an error.

Clients that log on later cannot take over the synchronization. However, if the simulation was started in synchronous operating mode, they get PROCEED calls.

The operating mode depends on the project. It is set in the Project manager.



3.2 Determining the PROCEED time

When simulation starts, each client can specify its desired PROCEED time for synchronous operating mode.

If SIMIT takes over the synchronization, the PROCEED time actually used is determined as follows:

- Lowest cycle time that SIMIT uses
  If other clients provide no information about the PROCEED time or only request values that are greater than the lowest cycle time that SIMIT uses.
- Lowest PROCEED time that a client requests
  If a client requests a PROCEED time that is shorter than the lowest cycle time that SIMIT uses.

This PROCEED time that is actually used is communicated to the clients with SimCommandNotifyExecuted. If another client takes over the synchronization, it must process this information accordingly. SIMIT does not check whether this client is using "reasonable" values for the PROCEED time in subsequent PROCEED calls. This only becomes apparent if other clients report an error, for example, because the time is too high.

First the clients are triggered, then SIMIT calculates the PROCEED step. If the PROCEED step is greater than the lowest cycle time in the SIMIT model, all submodels are calculated in sequence and possibly multiple times until this step is "fully executed".

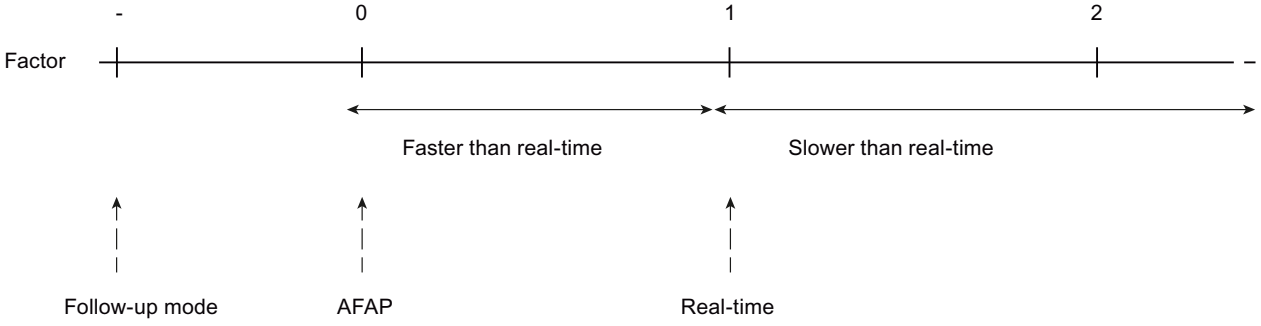
With very long PROCEED steps, the result is that SIMIT processes the submodels in a rhythm that differs from the asynchronous operating mode.

Example:

- SIMIT is active, the lowest cycle time (= PROCEED time) = 100 ms
  The entire calculation (clients and SIMIT including couplings) takes only 20 ms.
  When the step is fully executed, there is a wait time of 80 ms in order to reach "real-time".
  In asynchronous real-time mode, in contrast, the coupling is calculated according to t=50 ms.
- SIMIT is passive, the lowest cycle time = 100 ms, PROCEED time = 1000 ms.
  The calculation takes only 150 ms (client and SIMIT including couplings; the submodel with 100 ms is calculated 10 times, for example). The PROCEED call of the active client returns after 150 ms. It is the responsibility of this client to wait 850 ms until the next PROCEED call. As soon as another client has control of the simulation, the active client is also responsible for complying with the real-time/fast/slow conditions, although real-time is specified. The set speed is of no importance to SIMIT in this case.

3.3 Acceleration and delay of the simulation time

In order to slow the course of the simulation (Slow mode) or accelerate it (Fast mode), the relation of simulation time to time actually elapsed can be changed:



- Factor -1 = FOLLOW-UP mode, selected arbitrarily. The simulation time is managed by a logged-on client
- Factor 1.0 = Real-time
- Factor 0.0 No reference to the actual time, calculation performed as fast as possible

You can set the factor in the SIMIT toolbar:

Note

When the maximum speed is set in asynchronous operating mode, the simulation is paused internally and resumed in synchronous operating mode.

The following factors are offered for selection (you can also enter intermediate values):

Display text	Internal factor
Real-time (100%)	1
AsFastAsPossible	0
Fast mode (150%)	0.667
Fast mode (200%)	0.5
Fast mode (300%)	0.333
Fast mode (500%)	0.2
Slow mode (75%)	1.333
Slow mode (50%)	2
Slow mode (25%)	4

In follow-up mode (synchronous operating mode and SIMIT is passive), the selection box is disabled. You cannot change the factor in this case.

In all other situations (synchronous operating mode and SIMIT is active or asynchronous operating mode), you can change the factor during ongoing simulation. The clients are informed of the change. This is a requirement particularly in asynchronous operating mode.

Maximum acceleration is limited in such a way that the fastest cycle time of the project is at least 10 ms taking into account the acceleration. This means that, depending on the cycle time, acceleration might not be possible. SIMIT only offers appropriate values and does not allow faster values. If the values are too high, the clients get WrongState as a return value.

The following combinations of asynchronous/synchronous operating mode and speeds are possible:

	SIMIT	Client 1	Client 2
Asynchronous	Real-time/fast/slow	Real-time/fast/slow	Real-time/fast/slow
Synchronous, control with SIMIT	Active: Real-time/fast/slow/ AFAP*	Passive: Follow-up mode	Passive: Follow-up mode
Synchronous, Control by client	Passive: Follow-up mode	Active: Real-time/fast/ slow/AFAP*	Passive: Follow-up mode

* AFAP = Maximum speed ("as fast as possible"), only possible with synchronous operating mode.

Maximum Proceed time

Like all other service calls, the PROCEED command is a WCF command which can only need a limited computing time. Therefore, the maximum PROCEED time is always limited to 4 seconds based on real-time, regardless of cycle times used.

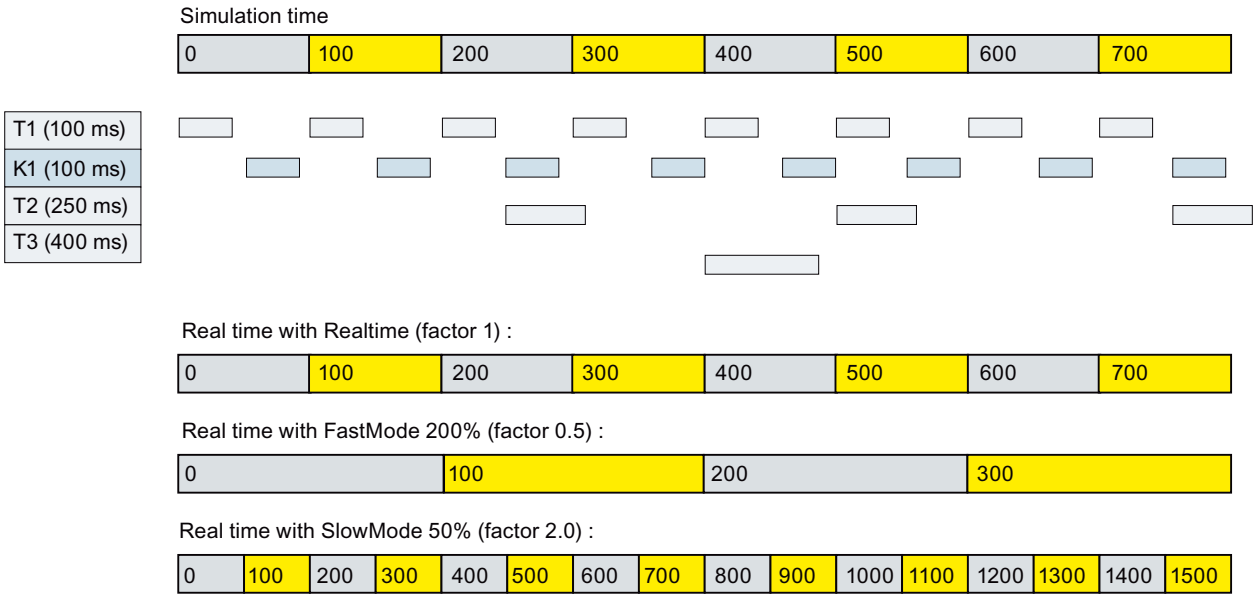
With a simulation speed with 10% or the factor 10.0, the maximum PROCEED time is 400 ms. The return value ProceedTimeTooLong is generated with longer times.

Execution sequence and timing

4

4.1 Asynchronous (real-time/fast/slow)

In asynchronous slow mode or fast mode, the time intervals at which the submodels are called are multiplied by the delay or acceleration factor. The simulation time is still counted with the configured cycle time.



T1, T2, T3: Submodels in time slice 1, 2, or 3

C1: Coupling in time slice 1

Clients are unimportant in asynchronous operating mode. They run with their own time base. No coordination takes place. The clients and SIMIT do not interfere with one another. Submodels and couplings also do not interfere with one another.

4.2 Synchronous (control with SIMIT)

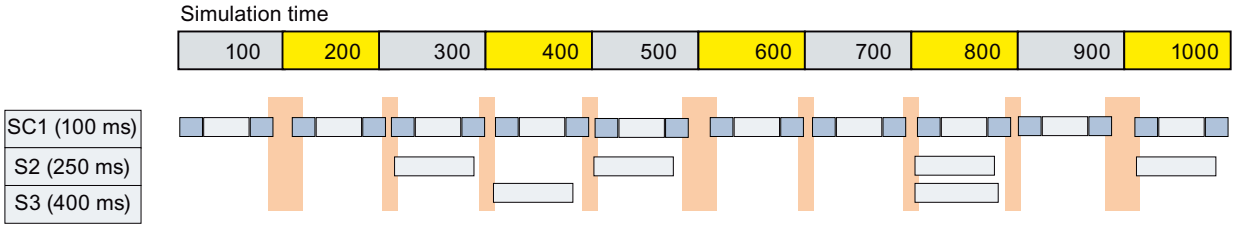
Maximum speed (AFAP)

At maximum speed, there is no reference to real-time and the model is calculated as fast as possible. The following conditions apply:

- The computer is not utilized 100%; otherwise, no operation or communication would be possible.
- The submodels are handled in such a way that they can be computed in parallel on multi-processor PCs.
- The processing of the couplings is divided into reading the outputs and writing the inputs (synchronous operating mode). The calculation of the submodels is "embedded" in the couplings which belong to the submodel. Submodels and couplings can slow one another down, unlike in asynchronous operating mode.
- Data exchange between the submodels takes place in a calculation step. The outputs of Submodel A are copied to the inputs of Submodel B before Submodel B is calculated.

The simulation time is incremented by the cycle time of the fastest submodel.

Each submodel is calculated for the duration of its cycle time plus the increase by its own cycle time not above the simulation time:



SC1:Submodel and coupling in time slice 1
S2, S3: Submodel in time slice 2 or 3

The CPU is released after each step (vertical bars), so that SIMIT does not completely utilize the computer.

Real-time/fast/slow

Here, the waiting time is selected in such a way that there is a reference to real-time. Apart from that, this sequence does not differ from the sequence outlined above.

Examples:

Real-time, PROCEED time 100 ms, required computing time 80 ms: Waiting time is 20 ms

Slow 50% and factor 2.0, PROCEED time 100 ms, required computing time 80 ms: Waiting time is 120 ms

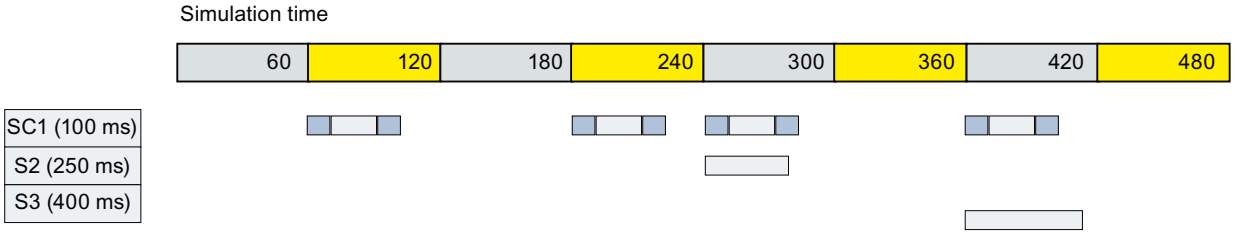
Real-time, PROCEED time 100 ms, required computing time 120 ms. No waiting time. There is no attempt to "make up" for lost time. 12 real minutes are needed to calculate 10 minutes of simulation time. No calculations are lost.

4.3 Synchronous (control by a client)

The simulation time is increased by the time transmitted by the current PROCEED call. This time is not always the same.

Each submodel is calculated for the duration of its cycle time plus the increase by its own cycle time not above the simulation time:

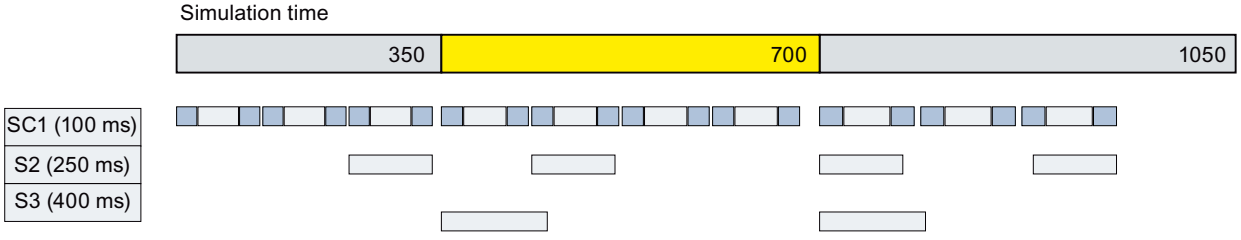
Example (PROCEED time = 60 ms):



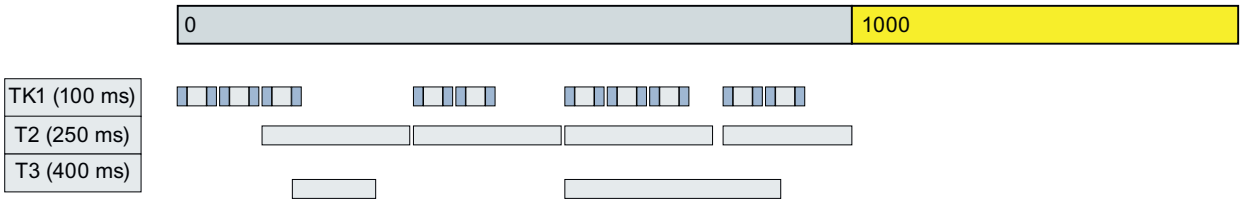
SC1:Submodel and coupling in time slice 1
S2, S3: Submodel in time slice 2 or 3

With PROCEED times that are greater than the lowest cycle time, the submodels are calculated in parallel in order to take advantage of multiple CPUs. However, the submodels must be synchronized so that there is no overlap of the simulation time.

Example (PROCEED time = 350 ms):



For example, with PROCEED time = 1000 ms:



SC1:Submodel and coupling in time slice 1
S2, S3: Submodel in time slice 2 or 3

Architecture of the RCI interface

5

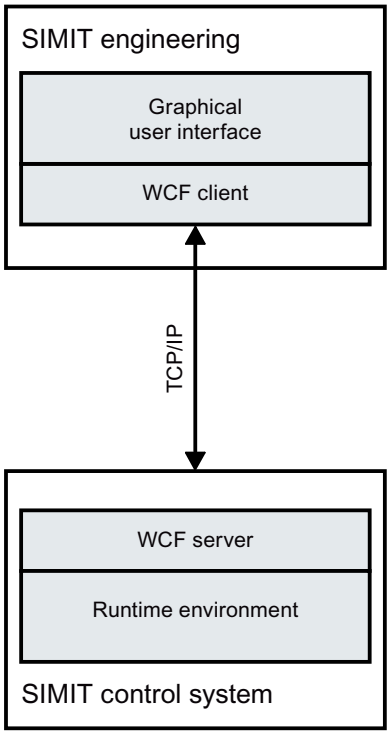
SIMIT consists of two separate processes: The engineering and the control system. Both run in the .NET environment, whereby the control system has higher priority.

Engineering is used to:

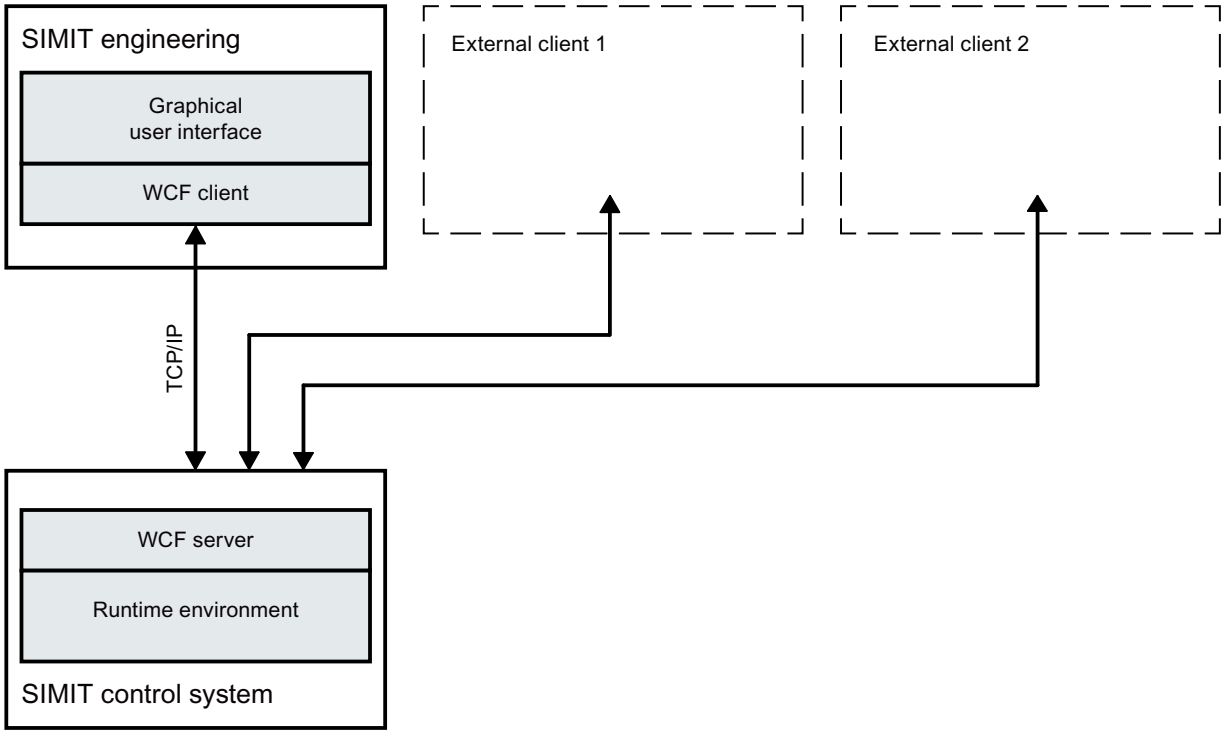
- Generate and compile the simulation model
- Correct errors
- Monitor and control the ongoing simulation

The control system is a runtime environment in which the simulation runs in real-time. It has no graphical user interface.

The communication between the engineering and control system runs via service calls based on the WCF (Windows Communication Foundation) interface.



If additional applications want to communicate with SIMIT, they can also connect to the WCF server:


SIMIT engineering
Graphical user interface
WCF client
External client 1
External client 2
TCP/IP
WCF server
Runtime environment
SIMIT control system

Implementation 6

All interface classes are included in the assembly Siemens.Simit.CS.CSAPI.dll in the namespace Siemens.Simit.CS.CSAPI. After installing SIMIT, this assembly is located in the "Global Assembly Cache" (GAC) under "%SystemRoot%Microsoft.NET\assembly\GAC_MSIL \Siemens.Simit.CS.CSAPI\... \Siemens.Simit.CS.CSAPI.dll". Insert this DLL as reference in your project.

Note

Note the following:

- Reference this DLL direct in the Global Assembly Cache.
- Do not copy this DLL into your own project.
- Do not supply this DLL with your coupling.

The WCF functionality is located in the System.ServiceModel DLL of Microsoft.

Each client that wants to communicate with SIMIT must implement the IControlSystemCallback interface. The client calls methods of the IControlSystem SIMIT interface.

A TCP/IP connection is established for communication. The address string is in the registry. The key is:

HKEY_LOCAL_MACHINE\Software\Wow6432Node\Siemens\Simit\8.0\uri

6.1 Undocumented service calls

The IControlSystem interface provides more service calls than the ones described in this document. Undocumented calls are reserved for internal use and should not be used.

6.2 Common parameters

6.2.1 Client

Each client is assigned a unique identifier (integer) when connecting to SIMIT. This is used to identify the client in all subsequent service calls.

6.2.2 ControlSystemService

The ControlSystemService enumeration includes all the services of the WCF interface, even if they are only used internally. Only the following services are relevant for external clients:

Value	Description
Internal	Internal action of the control system (e.g. in BroadcastMessage)
Connect	Connects a client
Disconnect	Disconnects a client
OpenProject	Opens a project
CloseProject	Closes the open project
RenameProject	Renames the project
Terminate	Stops the control system (cannot be initiated by the client)
Open	Opens the simulation
Close	Closes the simulation
Init	Initializes the simulation
Reset	Resets the simulation
Step	Performs a calculation step
Proceed	Single step for externally controlled operating mode
Run	Starts the simulation
Stop	Stops the simulation
SetSpeed	Sets the speed
CreateSnapshot	Creates a snapshot
CreateSnapshotFolder	Creates a snapshot folder
LoadSnapshot	Loads a snapshot
DeleteSnapshot	Deletes a snapshot
DeleteSnapshotFolder	Deletes a snapshot folder
RenameSnapshot	Renames a snapshot
RenameSnapshotFolder	Renames a snapshot folder
CopySnapshot	Copies a snapshot
CopySnapshotFolder	Copies a snapshot folder
GetInfo	Returns information about the current simulation
GetLanguage	Indicates the current language setting of SIMIT

6.2.3 ControlSystemServiceParams

The ControlSystemServiceParams structure contains the parameters of the original service call, with the exception of the default parameters client, result and errormessage.

Name	Type	Description
Service	ControlSystemService	The service to which these parameters belong.
Caller	int	The ID of the client that called the service.
StringVal1	string	First string parameter (if available)
StringVal2	string	Second string parameter (if available)

Name	Type	Description
StringVal3	string	Third string parameter (if available)
IntVal1	int	First Int parameter (if available)
LongVal1	long	First Long parameter (if available)
BoolVal1	bool	First Bool parameter (if available)
SyncMode	SyncMode	Synchronous or asynchronous operating mode with or without control of the caller.
RequireSyncControl	bool	Is control requested?
RequireSyncTime	int	Required Proceed time [ms]
Speed	double	Speed factor

6.2.4 SimInfo

The information about the current simulation has the following values:

Name	Type	Description
state	SimState	State of the simulation (NoSimulation, Loaded, Running or Stopped)
time	long	Current simulation time in milliseconds
speed	double	Current speed: • 0.0: maximum speed, • >0.0 .. <1.0: faster than real-time, e.g., 0.5 = double speed • 1.0: Real-time, • > 1.0: slower than real-time, e.g., 2.0 = half speed
syncMode	SyncMode	The current synchronization mode: • Async: The simulation runs asynchronously • SyncActive: The simulation runs synchronously, control by SIMIT • SyncPassive: The simulation runs synchronously, control by another client.
load	double	The current simulation load
minCycle	int	The lowest cycle time used in the current simulation model.

6.2.5 ControlSystemResult

The ControlSystemResult enumeration contains the following values:

Value	Description
Ok	No error.
Rejected	The service call was rejected by another client.
Forbidden	The service call was rejected by the SIMIT control system, because it can only be used by SIMIT itself.
Error	General error.

Value	Description
Warning	General warning.
Busy	The SIMIT control system is still working on another call. Note: SIMIT does not allow service calls to accumulate. Instead, a call returns with this result and it must be repeated.
NoLicense	The SIMIT control system has not (yet) found a valid license.
NoLicenseMultipleSimits	Only used internally.
NoLicenseExternalClients	There is no license for connecting to external clients.
WrongSimState	The service call cannot be completed in the current state of the simulation.
WrongProject	SIMIT has already opened another project.
UnknownClient	The specified client is not known to SIMIT.
ScriptError	The script cannot be executed.
FileAccessError	The specified file cannot be accessed.
FileNotFound	The specified file was not found.
ProceedTimeTooLong	The specified time span of a PROCEED call is too long
OutOfMemory	There is not enough memory available.

6.2.6 Error message

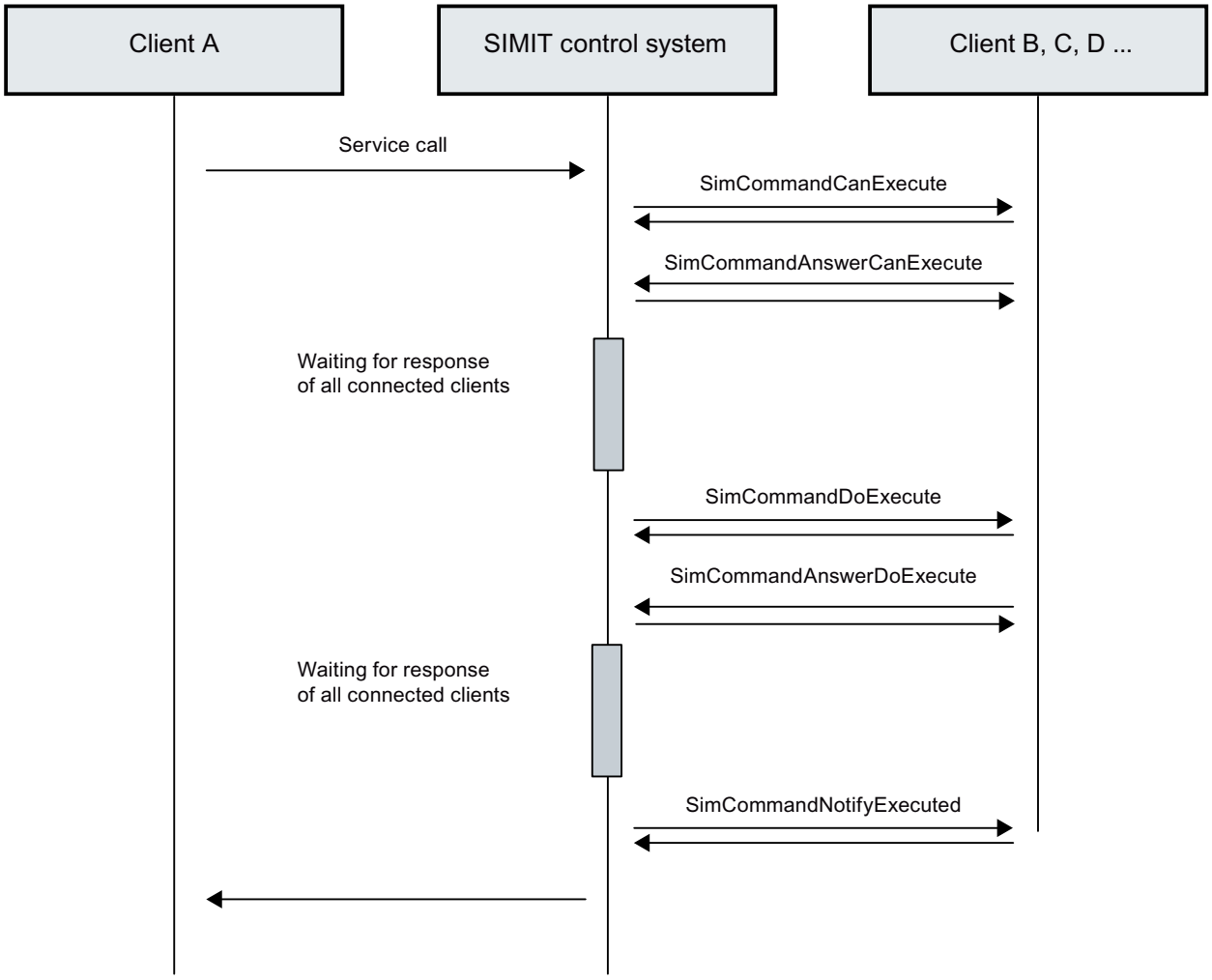
If a client does not return "OK", you can provide additional information as a string in the error message.

Handshake protocol

7

In order to keep the system in a consistent state, all clients need to know the calls issued by other clients. For this reason, relevant service calls are sent to all logged-on clients. In addition, SIMIT asks beforehand whether the service call can be executed by all clients. This ensures that the clients are in the same state.

The following call structure exists if no error has occurred:



7.1 Query of execution capability

When a client makes a service call, other clients are usually asked beforehand whether they are actually capable of executing the call. The SimCommandCanExecute method is called for this. It must be implemented by each client:

```
void SimCommandCanExecute(
  ControlSystemServiceParams parameters);
```

7.2 Confirmation of execution capability

In order to keep the entire system in a consistent state, a client must always send a confirmation.

```
void SimCommandAnswerCanExecute(
  int client,
  ControlSystemResult result,
  string errormessage,
  ControlSystemServiceParams parameters);
```

This service call is not passed to other clients.

7.3 Initiation of service command

After all clients have confirmed that the call can be executed, the call is actually initiated. The SimCommandDoExecute method is called for this. It must be implemented by each client:

```
void SimCommandDoExecute(
  ControlSystemServiceParams parameters);
```

7.4 Confirmation of execution

In order to keep the entire system in a consistent state, a client must always send a confirmation.

```
void SimCommandAnswerDoExecute(
  int client,
  ControlSystemResult result,
  string errormessage,
  ControlSystemService service);
```

This service call is not passed to other clients.

7.5 Information on the completion of the service command

After SIMIT and all logged-on clients have processed the call, they are informed of the completion of the service call. The SimCommandNotifyExecuted method is called for this. It must be implemented by each client:

```
void SimCommandNotifyExecuted(
  ControlSystemServiceParams parameters,
  ControlSystemResult result,
  string errormessage,
  SimInfo info);
```

This call requires no further acknowledgment.

7.6 General error messages

To send error messages to the clients, SIMIT calls the SimBroadcastMessage method, which must be implemented by each client (may be empty):

```
void SimBroadcastMessage(
  long time,
  ControlSystemResult result,
  string category,
  string source,
  string message,
  bool come);
```

This call requires no acknowledgment.

Service calls

8

8.1 Connection establishment and termination

8.1.1 Connection establishment

Each client that wants to communicate with SIMIT first needs to establish a connection.

```
int SimConnect(
  out ControlSystemResult result,
  out string errormessage,
  out string connectedProjectPath,
  out SimInfo info,
  byte[] id,
  string name);
```

Parameter	Description
connectedProjectPath	The path in which the current SIMIT project is located or null if there is no project open.
info	Information about the current simulation; for details, see SimGetInfo.
id	null should always be passed for this parameter.
name	Descriptive name of the client.
Return value	>0: The unique ID of the client with which it identifies itself during future service calls. <0: The service call failed; for details, see result and errormessage.

All logged-on clients receive this service call.

8.1.2 Connection termination

When a client no longer wants to communicate with SIMIT, it must log off.

```
void SimDisconnect(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.1.3 Information about the current simulation

A logged-on client can request information about the current simulation at any time:

```
SimInfo SimGetInfo(
  int client,
  out ControlSystemResult result,
```

```
  out string errormessage);
```

All logged-on clients receive this service call.

For additional information on this, refer to the section: SimInfo (Page 23).

8.1.4 Lifebeat

If a client has not communicated for 24 hours, the connection to SIMIT is terminated. Therefore, a client should at least make regular SimGetInfo calls.

8.1.5 Information about the language setting

A logged-on client can check which language is set for SIMIT at any time.

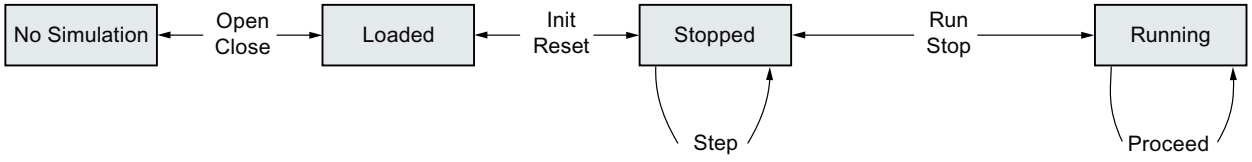
```
string SimGetLanguage(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

Returns "de-de" if the language setting is German, otherwise "en-us".

8.2 Simulation control

To run a simulation, the corresponding project must be opened. Only one project can be open at a given time.

The simulation is in one of the following four states:



8.2.1 Open project

To open a project, the client must specify the SIMIT file ".simit" with an absolute path in the file system. This is only possible when no project is open.

```
void SimOpenProject(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string simitFile);
```

Parameter	Description
simitFile	The SIMIT file with the extension ".simit" or null for opening the most recently used project.

All logged-on clients receive this service call.

8.2.2 Closing a project

Each client can close the current project, as long as a project is open and the simulation is in the NoSimulation state.

```
void SimCloseProject(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.2.3 Renaming a project

Each client can rename the current project, as long as a project is open and the simulation is in the NoSimulation state.

```
void SimRenameProject(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.2.4 Open simulation

Each client can open the simulation, as long as a project is open and the simulation is in the NoSimulation state.

```
void SimOpen(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.2.5 Close simulation

Each client can close the simulation, as long as the simulation is in the Loaded state.

```
void SimClose(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.2.6 Initialize simulation

Each client can initialize the simulation, as long as the simulation is in the Loaded state.

```
void SimInit(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.2.7 Reset simulation

Each client can reset the simulation, as long as the simulation is in the Stopped state.

```
void SimReset(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.2.8 Perform single step

Each logged-on client can perform single steps in the simulation, as long as the simulation is in the Stopped state.

```
void SimStep(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  int time);
```

Parameter	Description
time	The time span in milliseconds by which the simulation time should progress. If the value is <= 0, the lowest cycle time of the current project is used.

All logged-on clients receive this service call.

8.2.9 Advance simulation time

A client with control over the simulation time can use this call to advance the simulation time, as long as the simulation is in the Running state and the synchronization mode is SyncPassive.

```
void SimProceed(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  int time);
```

Parameter	Description
time	The time span in milliseconds by which the simulation time should progress. If the value is <= 0, the lowest cycle time of the project is used.

Note

All logged-on clients receive this service call. The clients are not asked beforehand with SimCommandCanExecute and SimCommandAnswerCanExecute whether they can perform the call. A client that has accepted SimRun with parameter SyncActive or SyncPassive must be able to execute SimProceed. The client has thus implicitly confirmed SimCommandCanExecute in advance.

8.2.10 Start simulation

Each logged-on client can start the simulation, as long as the simulation is in the Stopped state.

```
void SimRun(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  SyncMode syncMode,
  double speed,
  int proceedTime);
```

Parameter	Description
syncMode	Specifies the operating mode in which the simulation should run: • Async: The simulation runs asynchronously • SyncActive: The simulation runs synchronously, the caller assumes control over the simulation time. • SyncPassive: The simulation runs synchronously, the caller does not assume control over the simulation time. If no other client takes control, SIMIT controls the simulation time.
speed	The speed to be used. For values <0, the speed last set is used; the default value is real-time.
proceedTime	The time that is to be used for PROCEED calls when SIMIT assumes control. SIMIT uses the lower out of this value, other values set by other clients and the minimum cycle time of the project.

Other clients can provide the following information via SimCommandAnswerCanExecute by setting the corresponding values in the control system parameters:

RequireSyncTime = Value
Requests a maximum time for Proceed calls.

RequireSyncControl = true
This client wants to take over time control

All logged-on clients receive this service call.

8.2.11 Set speed

Each logged-on client can set the speed of the simulation.

```
void SimSetSpeed (
  int client,
  out ControlSystemResult result,
  out string errormessage,
  double speed);
```

Parameter	Description
speed	Speed; see SimInfo for meaning of the values

All logged-on clients receive this service call, as long as the simulation is in the Running state. Otherwise, the value is evaluated when SimRun is called.

8.2.12 Stop simulation

Each client can stop the simulation, as long as the simulation is in the Running or Stopped state.

```
void SimStop(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

All logged-on clients receive this service call.

8.3 Snapshots

You can save a snapshot of the simulation as a file. Snapshots are stored in the "snap" folder or a subfolder. All calls involved in snapshots refer to this folder.

To keep the simulation state consistent, all logged-on clients must create snapshots of their state and manage them in a similar way to SIMIT.

8.3.1 Create snapshot

This call is only allowed if the simulation is in the Running or Stopped state.

```
void SimCreateSnapshot(
  int client,
```

```
  out ControlSystemResult result,
  out string errormessage,
  string name);
```

Parameter	Description
name	Name of the snapshot. The snapshot is stored in the "snap" folder of the project. You cannot specify a subfolder in this case.

All logged-on clients receive this service call.

8.3.2 Create snapshot folder

```
void SimCreateSnapshotFolder(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string name);
```

Parameter	Description
name	The name of the new folder relative to the "snap" folder.

All logged-on clients receive this service call.

8.3.3 Load snapshot

This call is only allowed if the simulation is in the Loaded state.

```
void SimLoadSnapshot(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string name);
```

Parameter	Description
name	Name of the snapshot. If there are subfolders present, these must be preceded by a slash "/".

All logged-on clients receive this service call.

8.3.4 Delete snapshot

```
void SimDeleteSnapshot(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string name);
```

Parameter	Description
name	Name of the snapshot. A slash "/" must precede the subfolders.

All logged-on clients receive this service call.

8.3.5 Delete snapshot folder

```
void SimDeleteSnapshotFolder(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string name);
```

Parameter	Description
name	The name of the snapshot folder relative to the "snap" snapshot folder. A slash "/" must precede the subfolders.

Note

All subfolders are deleted.

All logged-on clients receive this service call.

8.3.6 Rename snapshot

```
void SimRenameSnapshot(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string oldName,
  string newName);
```

Parameter	Description
oldName	Old name of the snapshot. A slash "/" must precede the subfolders.
newName	New name of the snapshot. A slash "/" must precede the subfolders.

All logged-on clients receive this service call.

8.3.7 Rename snapshot folder

```
void SimRenameSnapshotFolder(
  int client,
  out ControlSystemResult result,
  out string errormessage,
```

```
  string oldName,
  string newName);
```

Parameter	Description
oldName	Old name of the snapshot folder. A slash "/" must precede the subfolders.
newName	New name of the snapshot folder. A slash "/" must precede the subfolders.

All logged-on clients receive this service call.

8.3.8 Copy snapshot

```
void SimCopySnapshot(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string oldName,
  string newName);
```

Parameter	Description
oldName	Name of the existing snapshot. A slash "/" must precede the subfolders.
newName	Name of the copied snapshot. A slash "/" must precede the subfolders.

All logged-on clients receive this service call.

8.3.9 Copy snapshot folder

```
void SimCopySnapshotFolder(
  int client,
  out ControlSystemResult result,
  out string errormessage,
  string oldName,
  string newName);
```

Parameter	Description
oldName	Name of the existing snapshot folder. A slash "/" must precede the subfolders.
newName	Name of the new snapshot folder. A slash "/" must precede the subfolders.

All logged-on clients receive this service call.

Minimum implementation of a client

9

The following code is a syntactically complete minimum implementation of a client without its own functionality.

```
using System;
using Siemens.Simit.CS.CSAPI;
using System.ServiceModel;

namespace WCFClient
{
  public class Program
  {
    static ControlSystemResult result;
    static string errormessage;
    static string connectedProjectPath;
    static SimInfo info;
    public static int clientID;

    static void Main(string[] args)
    {
      Console.WriteLine("Connecting ...");
      Client client = new Client();
      EndpointAddress ep = new EndpointAddress(
        "net.tcp://localhost:50800/ControlSystemServer");
      NetTcpBinding binding =
              new NetTcpBinding(SecurityMode.None, true);
      Client.proxy =
              DuplexChannelFactory<IControlSystem>.CreateChannel(
        new InstanceContext(client), binding, ep);
      clientID = Client.proxy.SimConnect(
        out result,
        out errormessage,
        out connectedProjectPath,
        out info,
        null,
        "WCFDemoClient");

      Console.WriteLine("Press any key to exit!");
      Console.ReadLine();

      Console.WriteLine("Disconnecting ...");
      Client.proxy.SimDisconnect(
            clientID, out result, out errormessage);
      Console.WriteLine("Exit");
    }
  }

  public class Client : IControlSystemCallback
  {
```

```
public static IControlSystem proxy;

ControlSystemResult result = ControlSystemResult.Ok;
string errormessage = "";

public void SimBroadcastMessage(
  long time,
  ControlSystemResult result,
  string category,
  string source,
  string message,
  bool come)
{
  Console.WriteLine("SimBroadcastMessage received:");
  Console.WriteLine("\ttime=" + time);
  Console.WriteLine("\tresult=" + result);
  Console.WriteLine("\tcategory=" + category);
  Console.WriteLine("\tsource=" + source);
  Console.WriteLine("\tmessage=" + message);
  Console.WriteLine("\tcome=" + come);
}

public void SimCommandCanExecute(
     ControlSystemServiceParams parameters)
{
  Console.WriteLine(
     "SimCommandCanExecute received, Service=" +
     parameters.Service);
  proxy.SimCommandAnswerCanExecute(
    Program.clientID,
    result,
    errormessage,
    parameters);
}

public void SimCommandDoExecute(
     ControlSystemServiceParams parameters)
{
  Console.WriteLine(
    "SimCommandDoExecute received, Service=" +
    parameters.Service);
  proxy.SimCommandAnswerDoExecute(
    Program.clientID,
    result,
    errormessage,
    parameters.Service);
}

public void SimCommandNotifyExecuted(
  ControlSystemServiceParams parameters,
  ControlSystemResult result,
  string errormessage,
```

```
        SimInfo info)
      {
        Console.WriteLine(
          "SimCommandNotifyExecuted received, Service=" +
          parameters.Service);
      }
    }
  }
```

Implementation of a passive, synchronized client 10

For the synchronous operating mode, a client must connect with SIMIT via WCF interface and then process PROCEED calls to calculate its own simulation cycles.

The following code is a syntactically complete minimum implementation of a client that performs a separate calculation based on PROCEED commands.

```
using System;
using Siemens.Simit.CS.CSAPI;
using System.ServiceModel;
using System.Threading;

namespace WCFClient
{
  public class Program
  {
    static ControlSystemResult result;
    static string errormessage;
    static string connectedProjectPath;
    static SimInfo info;

    static void Main(string[] args)
    {
      Console.WriteLine("Connecting ...");
      Client client = new Client();
      EndpointAddress ep = new
        EndpointAddress(
           "net.tcp://localhost:50800/ControlSystemServer");
      NetTcpBinding binding =
           new NetTcpBinding(SecurityMode.None, true);
      Client.proxy =
           DuplexChannelFactory<IControlSystem>.CreateChannel(
             new InstanceContext(client), binding, ep);

      try
      {
        Client.clientID = Client.proxy.SimConnect(
          out result,
          out errormessage,
          out connectedProjectPath,
          out info,
          null,
          "ConsoleDemoClient");

        if (result == ControlSystemResult.Ok)
        {
          Client.isConnected = true;
          Console.WriteLine("Connected.");
        }
```

```
      else
      {
        Console.WriteLine("NOT Connected!!!");
        Console.WriteLine("Result       : " + result);
        Console.WriteLine("Errormessage : " + errormessage);
        Client.isConnected = false;
      }
      Console.WriteLine("Press any key to exit!");
      Console.ReadLine();

      Console.WriteLine("Disconnecting ...");
      if (Client.isConnected)
      {
        try
        {
          Client.proxy.SimDisconnect(
            Client.clientID,
            out result,
            out errormessage);
        }
        catch (Exception e)
        {
          Console.WriteLine(e.ToString());
        }
      }
      Console.WriteLine("Exit");
    }
    catch (Exception ex)
    {
      Console.WriteLine(ex.ToString());
    }
  }
}

public class Client : IControlSystemCallback
{
  public static IControlSystem proxy;
  public static int clientID;
  public static ControlSystemServiceParams
      commandReceivedParameters;
  public static bool isConnected = false;

  ControlSystemResult result = ControlSystemResult.Ok;
  string errormessage = "";

  public void SimBroadcastMessage(
    long time,
    ControlSystemResult result,
    string category,
    string source,
    string message,
    bool come)
```

```
{
  Console.WriteLine("SimBroadcastMessage received:");
  Console.WriteLine("\ttime=" + time);
  Console.WriteLine("\tresult=" + result);
  Console.WriteLine("\tcategory=" + category);
  Console.WriteLine("\tsource=" + source);
  Console.WriteLine("\tmessage=" + message);
  Console.WriteLine("\tcome=" + come);
}

public void SimCommandCanExecute(
     ControlSystemServiceParams parameters)
{
  Console.WriteLine(
     "SimCommandCanExecute received, Service=" +
      parameters.Service);
  proxy.SimCommandAnswerCanExecute(
     clientID, result, errormessage, parameters);
}

public void SimCommandDoExecute(
     ControlSystemServiceParams parameters)
{
  Console.WriteLine(
     "SimCommandDoExecute received, Service=" +
      parameters.Service);
  switch (parameters.Service)
  {
    case ControlSystemService.Proceed:
      Console.WriteLine(
        "--> Proceed virtual time for " +
         parameters.IntVal1 + " ms.");
      commandReceivedParameters = parameters;
      new Thread(new ThreadStart(doCommand)).Start();
      // SimCommandAnswerDoExecute is done in doCommand
      break;
    case ControlSystemService.Terminate:
      isConnected = false;
      break;
    default:
      proxy.SimCommandAnswerDoExecute(
        clientID,
        result,
        errormessage,
        parameters.Service);
      break;
  }

}

public void SimCommandNotifyExecuted(
  ControlSystemServiceParams parameters,
```

```
      ControlSystemResult result,
      string errormessage,
      SimInfo info)
    {
      Console.WriteLine(
          "SimCommandNotifyExecuted received, Service=" +
           parameters.Service);
      if (parameters.Service == ControlSystemService.Run)
      {
        Console.WriteLine(
           "--> Run in " + parameters.SyncMode + " mode.");
      }
    }

    private static void doCommand()
    {
      Console.Write("\tWorking ... ");

      // The Sleep simulates the time consumed to do
      // whatever is necessary to proceed the virtual time
      Thread.Sleep(1000);

      Console.WriteLine("finished.");

      proxy.SimCommandAnswerDoExecute(
        clientID,
        ControlSystemResult.Ok,
        null,
        commandReceivedParameters.Service);
    }
  }
}
```