SIEMENS

SIMIT - Remote Control Interface (RCI)

Funktionshandbuch

03/2020
A5E44876264-AB

Rechtliche Hinweise

Warnhinweiskonzept

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.

⚠ GEFAHR
bedeutet, dass Tod oder schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ WARNUNG
bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ VORSICHT
bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

ACHTUNG
bedeutet, dass Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur von für die jeweilige Aufgabenstellung **qualifiziertem Personal** gehandhabt werden unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise. Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkten/Systemen Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

Bestimmungsgemäßer Gebrauch von Siemens-Produkten

Beachten Sie Folgendes:

⚠ WARNUNG
Siemens-Produkte dürfen nur für die im Katalog und in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und -komponenten zum Einsatz kommen, müssen diese von Siemens empfohlen bzw. zugelassen sein. Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

Haftungsausschluss

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten.

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
DEUTSCHLAND

A5E44876264-AB
℗ 05/2020 Änderungen vorbehalten

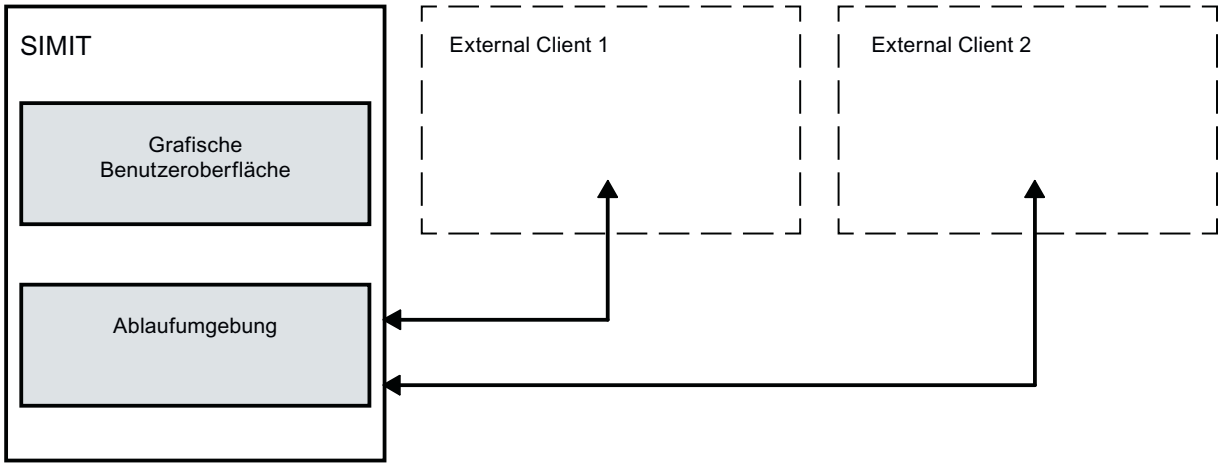
Copyright © Siemens AG 2020.
Alle Rechte vorbehalten

Inhaltsverzeichnis

Einleitung

1

SIMIT verfügt über eine Schnittstelle (Remote Control Interface RCI), die externe Applikationen (External Clients) über den Simulationszustand informiert und es ihnen ermöglicht, die Simulationssteuerung zu übernehmen.



Damit können andere Simulatoren an SIMIT angekoppelt werden. Die Bedienung der Simulation kann dann durch SIMIT oder den fremden Simulator erfolgen.

Firewall einrichten

2

Einleitung

Stellen Sie die Firewall ein, damit SIMIT SP sich mit folgenden Systemen verbinden kann:

- Weitere Instanzen von SIMIT SP in einer verteilten Simulation
- Verteilte Virtual Controller
- Fremdsysteme, die eine Verbindung mit SIMIT SP über die Simulationssteuerung (RCI) herstellen

ACHTUNG
Beachten Sie die Security-Hinweise im Vorwort
Änderungen an der Firewall können die Sicherheit Ihres Systems beeinflussen!
Kontaktieren Sie gegebenenfalls Ihren Systemadministrator, bevor Sie die nachfolgend beschriebenen Schritte durchführen.

Voraussetzung

- SIMIT ist installiert
- Alle Teilnehmer befinden sich im gleichen Netzwerk

Vorgehen

Um die Firewall einzurichten, gehen Sie folgendermaßen vor:

1. Öffnen Sie die "Windows Defender Firewall mit erweiterter Sicherheit", z. B. indem Sie "wf.msc" in der Eingabeaufforderung eingeben und mit "Enter" bestätigen.
2. Klicken Sie auf "Eingehende Regeln".
3. Doppelklicken Sie "SIMIT-CS Manager".
   Das Fenster "Eigenschaften von SIMIT-CS Manager" wird geöffnet.
4. Wählen Sie den Reiter "Bereich".
5. Wählen Sie unter "Remote-IP-Adresse" "Beliebige IP-Adresse". Alternativ fügen Sie die IP-Adressen der beteiligten Systeme in die Liste der zugelassenen Remote-IP-Adressen ein.
6. Bestätigen Sie mit "OK".

Ergebnis

Die Firewall ist so eingestellt, dass sich SIMIT SP mit anderen Systemen verbinden kann.

SIMIT-Betriebsarten

3

Um die RCI-Schnittstelle zu verstehen, müssen Sie die beiden Strategien kennen, nach denen SIMIT das Modell, die Kopplungen und eventuell angemeldete Clients bearbeitet.

Asynchrone Betriebsart

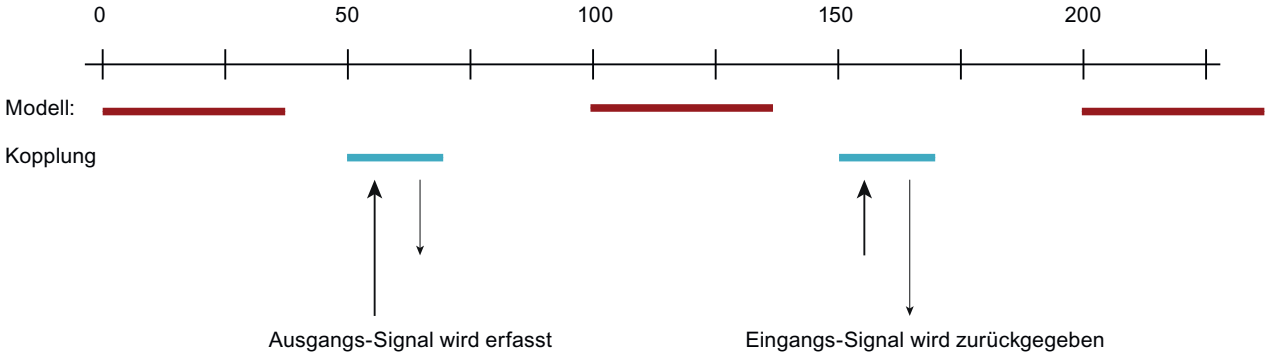
In SIMIT erfolgt die Modellberechnung der einzelnen Zeitscheiben und die Bearbeitung der Kopplungen zeitgesteuert.

Angemeldete Clients können nur abfragen, ob SIMIT in Echtzeit, schneller oder langsamer arbeitet. Alle angemeldeten Clients müssen die Zeitverwaltung (Scheduling) selbst übernehmen. Wenn SIMIT oder einer der Clients die vorhergesehenen Zeiten nicht einhalten kann, dann hat das keine Auswirkungen auf die anderen angemeldeten Clients.

Wenn SIMIT das Simulationsmodell einer Zeitscheibe nicht in der vorgesehenen Zeit abschließt, dann fallen ein oder mehrere Bearbeitungszyklen aus und die anderen Zeitscheiben werden nicht berechnet. Erst wenn alle Zeitscheiben wieder rechnen können, wird die Berechnung des Simulationsmodells fortgesetzt.

Wenn SIMIT eine Kopplung nicht in der vorgesehenen Zeit berechnet, dann fallen zwar auch hier ein oder mehrere Bearbeitungszyklen aus, aber die Berechnung der Kopplungen in anderen Zeitscheiben und die Berechnung des Simulationsmodells behindert das nicht. Dadurch blockieren Kopplungen durch eventuelle Kommunikationsprobleme nicht die Gesamtbearbeitung.

Um möglichst schnelle Reaktionszeiten zu erzielen, werden Kopplungen zeitversetzt gegenüber der Modellberechnung eingeplant:



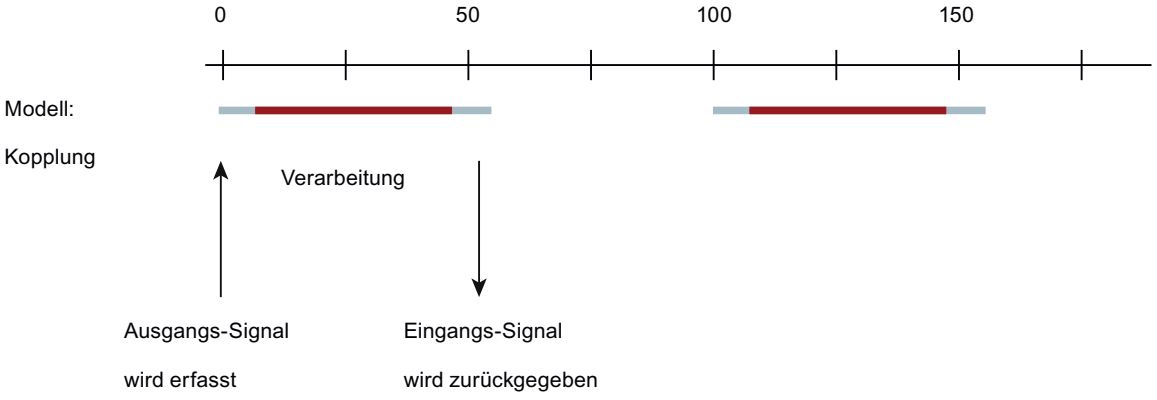
Wenn die Modellberechnung oder der Datenaustausch der Kopplungen länger als die Hälfte der Zykluszeit dauert, dann führt diese Zeitversetzung nicht zur Optimierung.

Synchrone Betriebsart

Bei synchroner Betriebsart werden Teilmodelle, Kopplungen und eventuell angemeldete Clients in einer genau vorgegebenen Reihenfolge berechnet. Die nächste Aktion startet erst, wenn die vorherige Aktion fertig ist.

Ein PROCEED-Befehl löst jeden Berechnungsschritt explizit aus und legt fest, um welchen Betrag die Simulationszeit voranschreiten soll.

Außerdem wird die Kopplung aufgetrennt in das Erfassen von Ausgangssignalen und das Schreiben von Eingangssignalen:



Das führt bei kleinen Zykluszeiten zu besseren Reaktionszeiten. Allerdings kann jedes Teilmodell, jede Kopplung und jeder eventuell angemeldete Client die gesamte Simulationsausführung blockieren.

3.1 Festlegung der Betriebsart (Synchron/Asynchron)

Der Client, der die Simulation startet, legt die Betriebsart fest. Bei synchroner Betriebsart gibt er an, ob er aktiv die Synchronisierung übernimmt. Ein Client kann die Betriebsart entweder unterstützen (Antwort: ja) oder nicht unterstützen (Antwort: nein). Er kann keine andere Betriebsart anfordern.

In SIMIT können Sie einstellen, welche Betriebsart beim Start über die Benutzeroberfläche gestartet werden soll.

Bei synchroner Betriebsart startet SIMIT immer "passiv", d. h. angemeldete Clients haben bzgl. der Synchronisierung Vorrang. SIMIT verteilt die Kontrolle über die Zeitsteuerung nach folgender Regel:

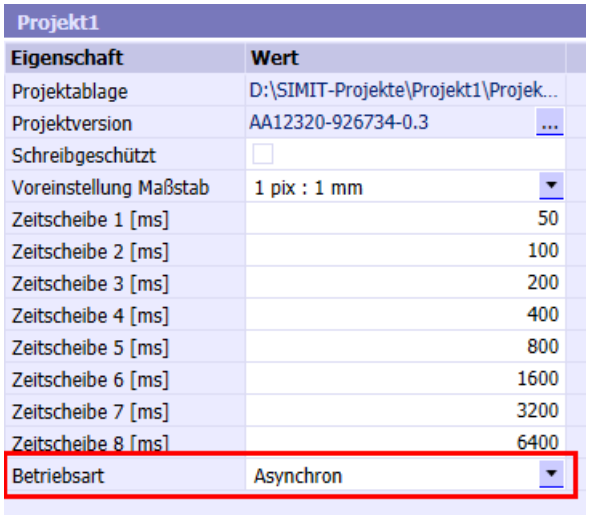
Genau ein Client will die Synchronisation übernehmen: Die Simulation kann starten.

Kein Client will die Synchronisation übernehmen: SIMIT übernimmt die Synchronisation.

Mehrere Clients wollen die Synchronisation übernehmen: SIMIT meldet einen Fehler.

Clients, die sich nachträglich anmelden, können die Synchronisation nicht übernehmen. Wenn die Simulation in synchroner Betriebsart gestartet wurde, erhalten sie aber PROCEED-Aufrufe.

Die Betriebsart hängt vom Projekt ab. Sie wird im Projektmanager eingestellt.



3.2 Festlegung der PROCEED-Zeit

Während die Simulation startet, kann jeder Client seine gewünschte PROCEED-Zeit für die synchrone Betriebsart angeben.

Falls SIMIT die Synchronisation übernimmt, dann ergibt sich die tatsächlich verwendete PROCEED-Zeit wie folgt:

- Kleinste Zykluszeit, die SIMIT nutzt
  Wenn andere Clients keine Angaben zur PROCEED-Zeit machen oder nur Werte fordern, die größer sind als die kleinste Zykluszeit, die SIMIT nutzt.
- Kleinste PROCEED-Zeit, die ein Client anfordert
  Falls ein Client eine PROCEED-Zeit fordert, die kürzer ist als die kleinste Zykluszeit, die SIMIT nutzt.

Diese tatsächlich verwendete PROCEED-Zeit wird im SimCommandNotifyExecuted an die Clients kommuniziert. Falls ein anderer Client die Synchronisation übernimmt, dann muss er diese Information entsprechend verarbeiten. SIMIT prüft nicht, ob dieser Client bei späteren PROCEED-Aufrufen "vernünftige" Werte für die PROCEED-Zeit verwendet. Das erkennt man nur daran, dass andere Clients eventuell einen Fehler liefern, wenn z.B. die Zeit zu groß ist.

Es werden zuerst die Clients angestoßen, danach rechnet SIMIT den PROCEED-Schritt. Wenn der PROCEED-Schritt größer ist als die kleinste Zykluszeit im SIMIT-Modell, dann werden alle Teilmodelle hintereinander und eventuell mehrfach berechnet, bis dieser Schritt "abgearbeitet" ist.

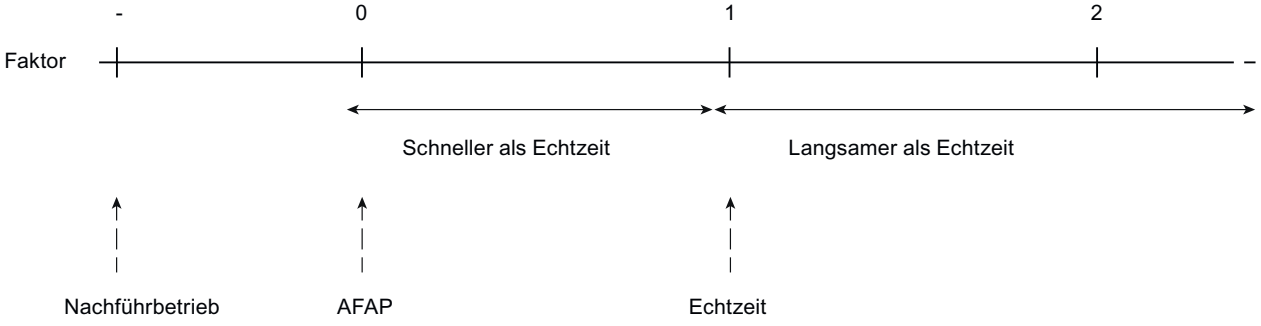
Bei sehr langen PROCEED-Schritten führt das dazu, dass SIMIT die Teilmodelle im Vergleich zur asynchronen Betriebsart in einem anderen Rhythmus abarbeitet.

Beispiel:

- SIMIT ist aktiv, kleinste Zykluszeit (= PROCEED-Zeit) = 100ms
  Die gesamte Berechnung (Clients und SIMIT inklusive Kopplungen) dauert nur 20ms. Wenn der Schritt komplett abgearbeitet ist, wird 80ms gewartet, um "Echtzeit" zu erreichen. Im asynchronen Realtime-Modus hingegen wird die Kopplung erst nach t=50ms berechnet.
- SIMIT ist passiv, kleinste Zykluszeit = 100ms, PROCEED-Zeit = 1000ms.
  Die Berechnung dauert nur 150ms (Client und SIMIT inkl. Kopplungen; das Teilmodell mit 100ms wird z. B. 10mal berechnet). Der PROCEED-Aufruf des aktiven Clients kehrt nach 150ms zurück. Es ist Aufgabe dieses Clients, bis zum nächsten PROCEED-Aufruf 850ms zu warten. Sobald ein anderer Client die Simulationskontrolle hat, ist der aktive Client auch dafür zuständig, die Echtzeit/schnell/langsam-Bedingungen einzuhalten, obwohl Echtzeit angegeben ist. Die eingestellte Geschwindigkeit hat für SIMIT hier keine Bedeutung.

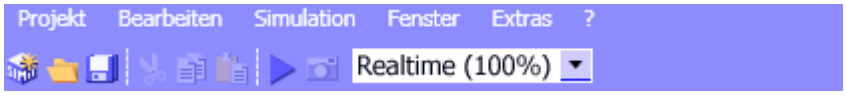
3.3 Beschleunigung und Verzögerung der Simulationszeit

Um den Ablauf der Simulation zu verlangsamen (Slow Mode) oder zu beschleunigen (Fast Mode), kann das Verhältnis von Simulationszeit zu tatsächlich verstrichenen Zeit verändert werden:



- Faktor -1 = Nachführbetrieb (FOLLOW-UP), willkürlich gewählt. Die Simulationszeit wird von einem angemeldeten Client verwaltet
- Faktor 1.0 = Echtzeit
- Faktor 0.0 kein Bezug zur tatsächlichen Zeit, Berechnung so schnell wie möglich

Sie können den Faktor in der SIMIT-Toolbar einstellen:



Hinweis

Wenn bei asynchroner Betriebsart auf maximale Geschwindigkeit umgeschaltet wird, dann wird intern die Simulation angehalten und in synchroner Betriebsart fortgesetzt.

Folgende Faktoren werden als Auswahl angeboten (Sie können auch Zwischenwerte eingeben):

Anzeigetext	Interner Faktor
Realtime (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

Im Nachführbetrieb (synchrone Betriebsart und SIMIT ist passiv) ist die Auswahlbox inaktiv. Sie können den Faktor dann nicht mehr verändern.

In allen anderen Fällen (synchrone Betriebsart und SIMIT ist aktiv oder asynchroner Modus) können Sie den Faktor bei laufender Simulation ändern. Die Clients werden über die Änderung informiert. Das ist besonders bei asynchroner Betriebsart erforderlich.

Die maximale Beschleunigung ist so begrenzt, dass die schnellste Zykluszeit des Projekts unter Berücksichtigung der Beschleunigung mindestens 10ms beträgt. Je nach Zykluszeit ist also keine Beschleunigung möglich. SIMIT bietet nur passende Werte an und lässt keine schnelleren Werte zu. Bei zu großen Werten erhalten Clients WrongState als Rückgabewert.

Folgende Kombinationen aus asynchroner/synchroner Betriebsart und Geschwindigkeiten sind möglich:

	SIMIT	Client 1	Client 2
Asynchron	Echtzeit / schnell / langsam	Echtzeit / schnell / langsam	Echtzeit / schnell / langsam
Synchron, Kontrolle bei SIMIT	Aktiv: Echtzeit / schnell / langsam / AFAP*	Passiv: Nachführbetrieb	Passiv: Nachführbetrieb
Synchron, Kontrolle beim Client	Passiv: Nachführbetrieb	Aktiv: Echtzeit / schnell / langsam / AFAP*	Passiv : Nachführbetrieb

* AFAP = Maximale Geschwindigkeit ("as fast as possible") ist nur bei synchroner Betriebsart möglich.

Maximale Proceed-Zeit

Der PROCEED-Befehl ist wie alle anderen Service-Aufrufe ein WCF-Kommando, das nur eine begrenzte Rechenzeit benötigen darf. Deswegen ist die maximale PROCEED-Zeit unabhängig von genutzten Zykluszeiten immer auf 4 Sekunden begrenzt, bezogen auf Echtzeit.

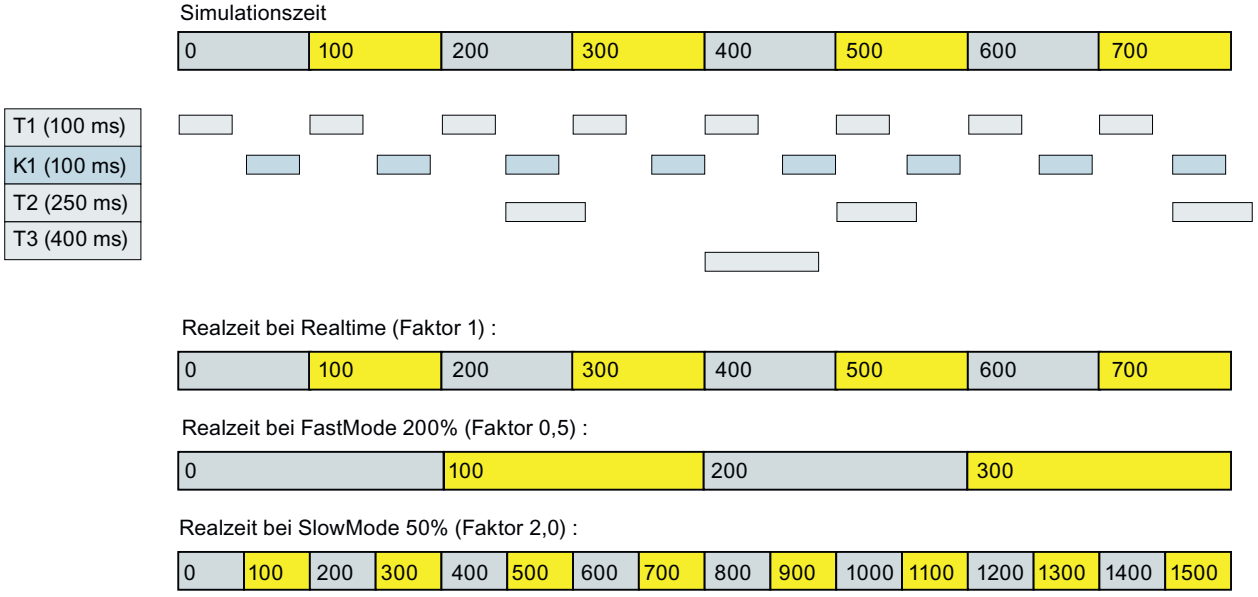
Bei einem Simulationslauf mit 10% bzw. Faktor 10.0 beträgt die maximale PROCEED-Zeit 400ms. Bei längeren Zeiten wird der Rückgabewert ProceedTimeTooLong geliefert.

Bearbeitungsreihenfolge und Zeitabläufe

4

4.1 Asynchron (Echtzeit/schnell/langsam)

Im asynchronen Slow Mode bzw. Fast Mode werden die Zeitabstände, in denen die Teilmodelle aufgerufen werden, mit dem Verzögerungsfaktor bzw. Beschleunigungsfaktor multipliziert. Die Simulationszeit wird weiterhin mit der projektierten Zykluszeit gezählt.



T1, T2, T3: Teilmodelle in Zeitscheibe 1, 2 bzw. 3

T1, T2, T3: Teilmodelle in Zeitscheibe 1, 2, bzw. 3

K1: Kopplung in Zeitscheibe 1

Clients sind bei asynchroner Betriebsart unwichtig. Sie laufen mit eigener Zeitbasis. Eine Abstimmung findet nicht statt. Die Clients und SIMIT behindern sich nicht gegenseitig. Teilmodelle und Kopplungen behindern sich ebenfalls nicht gegenseitig.

4.2 Synchron (Kontrolle bei SIMIT)

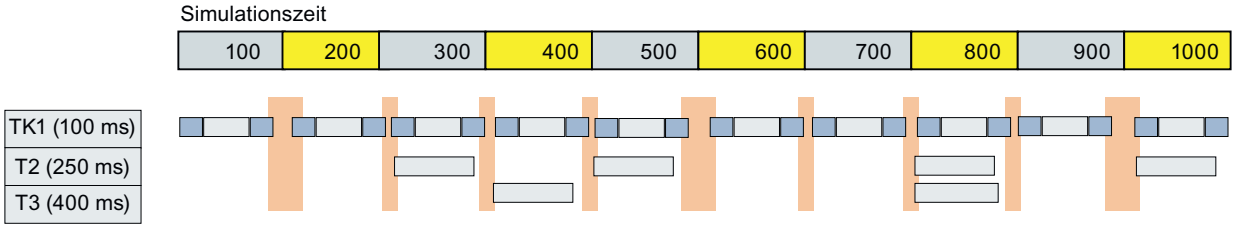
Maximale Geschwindigkeit (AFAP)

Bei maximaler Geschwindigkeit besteht kein Bezug zur Realzeit und das Modell wird so schnell wie möglich berechnet. Dabei gelten folgende Randbedingungen:

- Der Rechner wird nicht zu 100% ausgelastet, da sonst keine Bedienung und Kommunikation mehr möglich ist.
- Die Teilmodelle werden so angesteuert, dass sie auf Mehrprozessor-PCs parallel berechnet werden können.
- Die Bearbeitung der Kopplungen wird aufgeteilt in Lesen der Ausgänge und Schreiben der Eingänge (synchrone Betriebsart). Die Berechnung der Teilmodelle wird in die Kopplungen "eingebettet", die zum Teilmodell gehören. Teilmodelle und Kopplungen bremsen sich eventuell gegenseitig aus, im Gegensatz zur asynchronen Betriebsart.
- Der Datenaustausch zwischen den Teilmodellen findet in einem Berechnungsschritt statt. Die Ausgänge aus Teilmodell A werden auf die Eingänge des Teilmodells B kopiert, bevor Teilmodell B berechnet wird.

Die Simulationszeit wird um die Zykluszeit des schnellsten Teilmodells hochgezählt.

Jedes Teilmodell wird so lange berechnet, wie seine Zykluszeit plus der Erhöhung um die eigene Zykluszeit nicht über der Simulationszeit liegt:



TK1:Teilmodell und Kopplung in Zeitscheibe 1
T2, T3:Teilmodell in Zeitscheibe 2 bzw. 3

Nach jedem Schritt die CPU freigegeben (senkrechte Balken), sodass SIMIT den Rechner nicht völlig auslastet.

Echtzeit / Schnell / Langsam

Hier wird die Wartezeit so gewählt, dass Bezug zur Realzeit besteht. Abgesehen davon, unterscheidet sich dieser Ablauf nicht von dem oben skizzierten Ablauf.

Beispiele:

Echtzeit, PROCEED-Zeit 100ms, benötigte Rechenzeit 80ms: Wartezeit ist 20ms

Langsam 50% bzw. Faktor 2.0, PROCEED-Zeit 100ms, benötigte Rechenzeit 80ms: Wartezeit ist 120ms

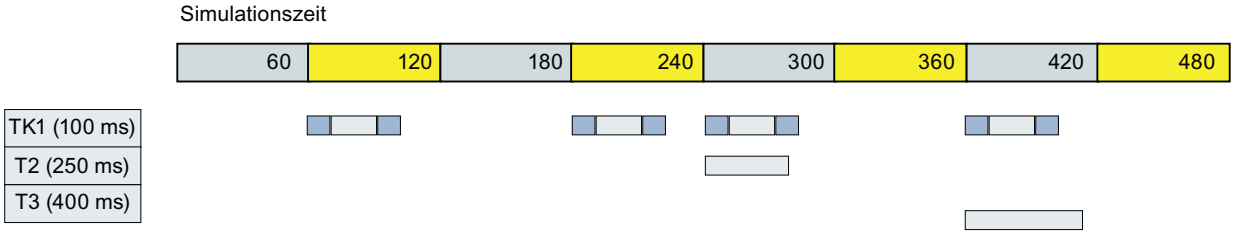
Echtzeit, PROCEED-Zeit 100ms, benötigte Rechenzeit 120ms, Keine Wartezeit. Es wird auch nicht versucht, verlorene Zeit "aufzuholen". Um 10 Minuten Simulationszeit zu berechnen, werden real 12 Minuten benötigt. Keine Berechnungen gehen verloren.

4.3 Synchron (Kontrolle bei einem Client)

Die Simulationszeit wird um die Zeit erhöht, die der aktuelle PROCEED-Aufruf übermittelt. Diese Zeit ist nicht immer gleich.

Jedes Teilmodell wird so lange berechnet, wie seine Zykluszeit plus der Erhöhung um die eigene Zykluszeit nicht über der Simulationszeit liegt:

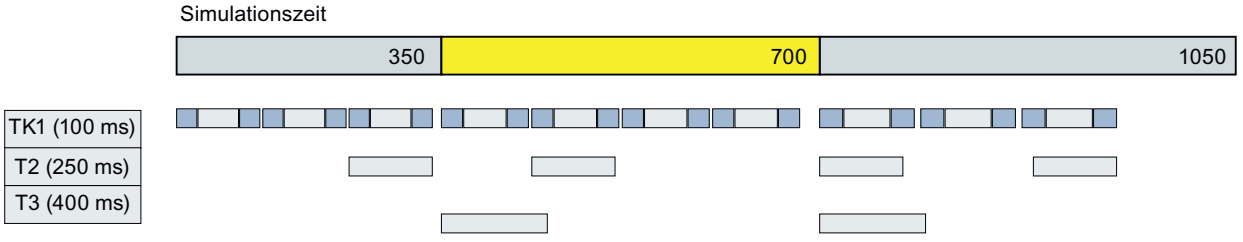
Beispiel (PROCEED-Zeit = 60ms):



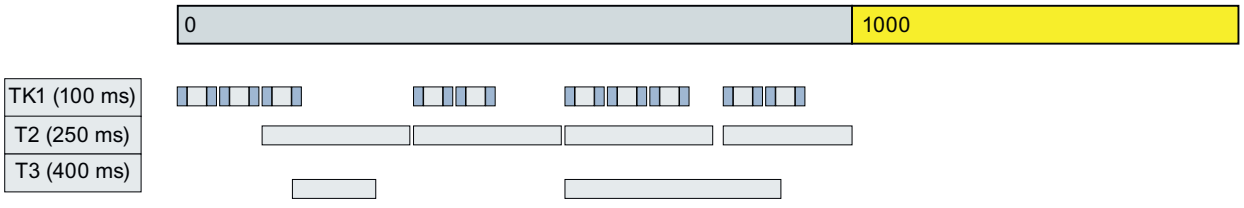
TK1:Teilmodell und Kopplung in Zeitscheibe 1
T2, T3:Teilmodell in Zeitscheibe 2 bzw. 3

Bei PROCEED-Zeiten, die größer als die kürzeste Zykluszeit sind, werden die Teilmodelle parallel berechnet, um mehrere CPUs nutzen zu können. Die Teilmodelle müssen aber synchronisiert werden, damit es keine Überschneidungen bei der Simulationszeit gibt.

Beispiel (PROCEED-Zeit = 350ms):



z.B. bei PROCEED-Zeit = 1000ms:



TK1:Teilmodell und Kopplung in Zeitscheibe 1
T2, T3:Teilmodell in Zeitscheibe 2 bzw. 3

Architektur der RCI-Schnittstelle

5

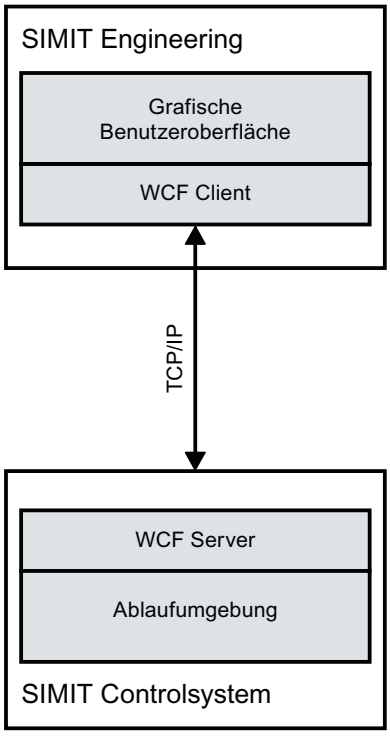
SIMIT besteht aus zwei getrennten Prozessen: Dem Engineering und dem Controlsystem. Beide laufen in der .NET-Umgebung, wobei das Controlsystem eine höhere Priorität hat.

Das Engineering wird verwendet, um

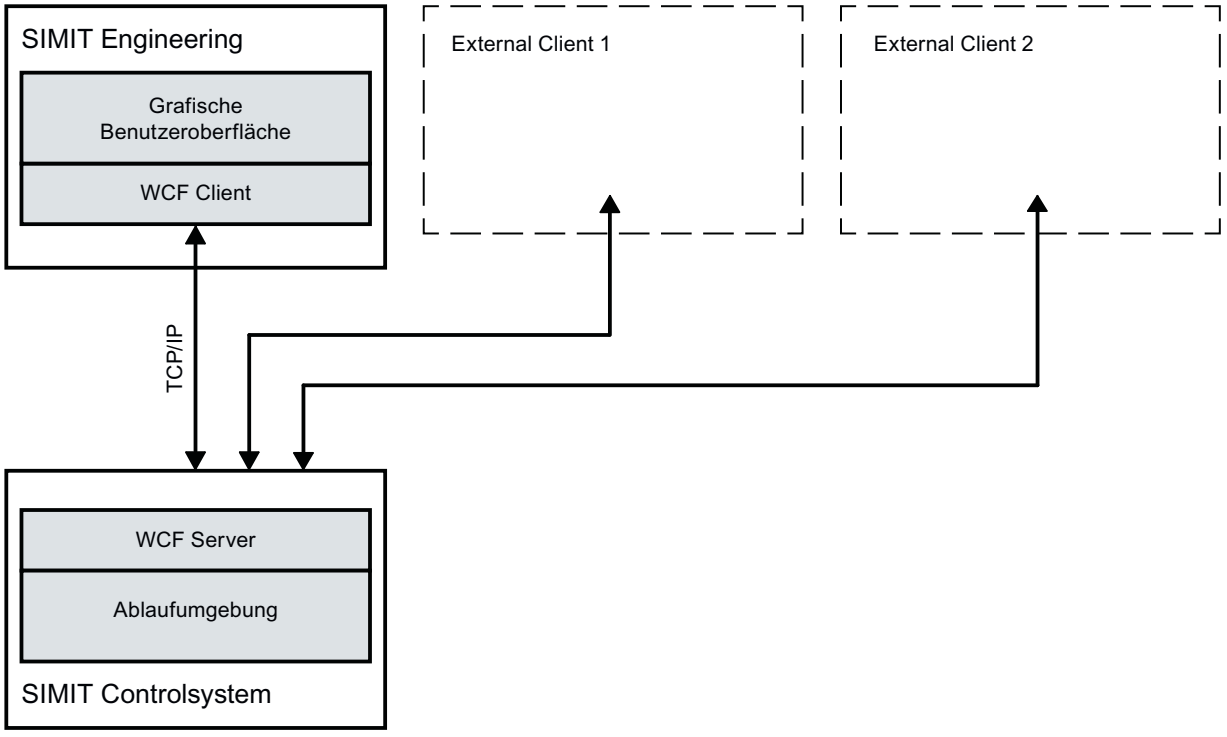
- das Simulationsmodell zu erzeugen und zu übersetzen
- Fehler zu beheben
- die laufende Simulation zu beobachten und zu steuern

Das Controlsystem ist eine Ablaufumgebung, in der die Simulation in Echtzeit läuft. Es hat keine grafische Benutzeroberfläche.

Die Kommunikation zwischen Engineering und Controlsystem läuft über Service-Aufrufe auf Basis der WCF-Schnittstelle (Windows Communication Foundation).



Wenn zusätzliche Anwendungen mit SIMIT kommunizieren möchten, können sich diese ebenfalls mit dem WCF-Server verbinden:


SIMIT Engineering
Grafische Benutzeroberfläche
WCF Client
External Client 1
External Client 2
TCP/IP
WCF Server
Ablaufumgebung
SIMIT Controlsystem

Implementierung 6

Alle Schnittstellenklassen liegen in der Assembly Siemens.Simit.CS.CSAPI.dll im Namespace Siemens.Simit.CS.CSAPI. Diese Assembly liegt nach der Installation von SIMIT im "Global Assembly Cache" (GAC) unter "%SystemRoot%Microsoft.NET\assembly\GAC_MSIL \Siemens.Simit.CS.CSAPI\... \Siemens.Simit.CS.CSAPI.dll". Fügen Sie diese DLL als Referenz in Ihr Projekt ein.

Hinweis

Beachten Sie:

- Referenzieren Sie diese DLL direkt im Global Assembly Cache.
- Kopieren Sie diese DLL nicht in Ihr eigenes Projekt.
- Liefern Sie diese DLL nicht mit Ihrer Kopplung aus.
- Übersetzen Sie Ihre Applikation mit dem Target framework "Net framework 4.6.1".

Die WCF-Funktionalität befindet sich in der DLL System.ServiceModel von Microsoft.

Jeder Client, der mit SIMIT kommunizieren möchte, muss das Interface IControlSystemCallback implementieren. Der Client ruft Methoden des SIMIT-Interfaces IControlSystem auf.

Zur Kommunikation wird eine TCP/IP-Verbindung aufgebaut. Der Adress-String steht in der Registry. Der Schlüssel lautet:

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

6.1 Undokumentierte Service-Aufrufe

Das Interface IControlSystem stellt mehr Service-Aufrufe zur Verfügung als in diesem Dokument beschrieben sind. Undokumentierte Aufrufe sind für die interne Nutzung reserviert und dürfen nicht genutzt werden.

6.2 Gemeinsame Parameter

6.2.1 Client

Jeder Client erhält bei der Verbindung zu SIMIT eine eindeutige Kennung (Ganzzahl). Damit identifiziert sich der Client bei allen folgenden Service-Aufrufen.

6.2.2 ControlSystemService

Die Aufzählung ControlSystemService enthält alle Services der WCF-Schnittstelle, auch wenn sie nur intern genutzt werden. Nur die folgenden Services sind für externe Clients relevant:

Wert	Beschreibung
Internal	Interne Aktion des ControlSystem (z.B. bei BroadcastMessage)
Connect	Verbindet einen Client
Disconnect	Trennt einen Client
OpenProject	Öffnet ein Projekt
CloseProject	Schließt das offene Projekt
RenameProject	Benennt das Projekt um
Terminate	Beendet das ControlSystem (Kann vom Client nicht veranlasst werden)
Open	Öffnet die Simulation
Close	Schließt die Simulation
Init	Initialisiert die Simulation
Reset	Setzt die Simulation zurück
Step	Führt einen Berechnungsschritt durch
Proceed	Einzelschritt für fremdgesteuerten Modus
Run	Startet die Simulation
Stop	Hält die Simulation an
SetSpeed	Setzt die Geschwindigkeit
CreateSnapshot	Erzeuge einen Snapshot
CreateSnapshotFolder	Erzeuge einen Snapshot-Ordner
LoadSnapshot	Lädt einen Snapshot
DeleteSnapshot	Löscht einen Snapshot
DeleteSnapshotFolder	Löscht einen Snapshot-Ordner
RenameSnapshot	Benennt einen Snapshot um
RenameSnapshotFolder	Benennt einen Snapshot-Ordner um
CopySnapshot	Kopiert einen Snapshot
CopySnapshotFolder	Kopiert einen Snapshot-Ordner
GetInfo	Liefert Infos über die aktuelle Simulation
GetLanguage	Liefert die aktuelle Spracheinstellung von SIMT
CreateBacktrack	Erstellt einen Backtrack
LoadBacktrack	Lädt einen Backtrack
ConvertBacktrack	Wandelt einen Backtrack in einen Snapshot um

6.2.3 ControlSystemServiceParams

Die Struktur ControlSystemServiceParams enthält die Parameter des ursprünglichen Service-Aufrufs, mit Ausnahme der Standardparameter client, result und errormessage.

Name	Typ	Beschreibung
Service	ControlSystemService	Der Service, zu dem diese Parameter gehören.
Caller	int	Die ID des Client, der den Service aufgerufen hat.
StringVal1	string	Erster String-Parameter (falls vorhanden)
StringVal2	string	Zweiter String-Parameter (falls vorhanden)
StringVal3	string	Dritter String-Parameter (falls vorhanden)
IntVal1	int	Erster Int-Parameter (falls vorhanden)
LongVal1	long	Erster Long-Parameter (falls vorhanden)
BoolVal1	bool	Erster Bool-Parameter (falls vorhanden)
SyncMode	SyncMode	Synchrone Betriebsart oder asynchrone Betriebsart mit oder ohne Kontrolle des Aufrufers.
RequireSyncControl	bool	Wird die Kontrolle angefordert?
RequireSyncTime	int	Geforderte Proceed-Zeit [ms]
Speed	double	Geschwindigkeitsfaktor
Timeout	int	Timeout in Millisekunden
Timeout in Millisekunden	int	Zweiter Int-Parameter (falls vorhanden)

6.2.4 SimInfo

Die Information über die aktuelle Simulation enthält folgende Werte:

Name	Typ	Beschreibung
state	SimState	Zustand der Simulation (NoSimulation, Loaded, Running oder Stopped)
time	long	Aktuelle Simulationszeit in Millisekunden
speed	double	aktuelle Geschwindigkeit: • 0.0: maximale Geschwindigkeit, • >0.0 .. <1.0: schneller als Echtzeit, z.B. 0.5 = doppelte Geschwindigkeit • 1.0: Echtzeit, • >1.0:langsamer als Echtzeit, z.B. 2.0 = halbe Geschwindigkeit
syncMode	SyncMode	Der aktuelle Synchronisations-Modus: • Async: Die Simulation läuft asynchron • SyncActive: Die Simulation läuft synchron, die Kontrolle liegt bei SIMIT • SyncPassive: Die Simulation läuft synchron, die Kontrolle liegt bei einem anderen Client.

Name	Typ	Beschreibung
load	double	Die aktuelle Simulations-Last
minCycle	int	Die kleinste im aktuellen Simulationsmodell genutzte Zykluszeit.

6.2.5 ControlSystemResult

Die Aufzählung ControlSystemResult enthält folgende Werte:

Wert	Beschreibung
Ok	Kein Fehler.
Rejected	Der Service-Aufruf wurde von einem anderen Client abgelehnt.
Forbidden	Der Service-Aufruf wurde vom SIMIT-ControlSystem abgelehnt, weil er nur von SIMIT selbst zu nutzen ist.
Error	Allgemeiner Fehler.
Warning	Allgemeine Warnung.
Busy	Das SIMIT Controlsystem arbeitet noch einen anderen Aufruf ab. Hinweis: SIMIT lässt nicht zu, dass sich Service-Aufrufe aufstauen. Stattdessen kehrt ein Aufruf mit diesem Ergebnis zurück, er muss also wiederholt werden.
NoLicense	Das SIMIT Controlsystem hat (noch) keine gültige Lizenz gefunden.
NoLicenseMultipleSimits	Nur intern verwendet.
NoLicenseExternalClients	Die Lizenz zur Anbindung externer Clients fehlt.
WrongSimState	Der Service-Aufruf kann im aktuellen Zustand der Simulation nicht durchgeführt werden.
WrongProject	SIMIT hat bereits ein anderes Projekt geöffnet.
UnknownClient	Der angegebene Client ist SIMIT nicht bekannt.
ScriptError	Das Skript kann nicht ausgeführt werden.
FileAccessError	Auf die angegebene Datei kann nicht zugegriffen werden.
FileNotFound	Die angegebene Datei wurde nicht gefunden.
ProceedTimeTooLong	Die angegebene Zeitspanne eines PROCEED-Aufrufs ist zu lang
OutOfMemory	Es steht nicht mehr genug Speicher zur Verfügung.

6.2.6 Errormessage

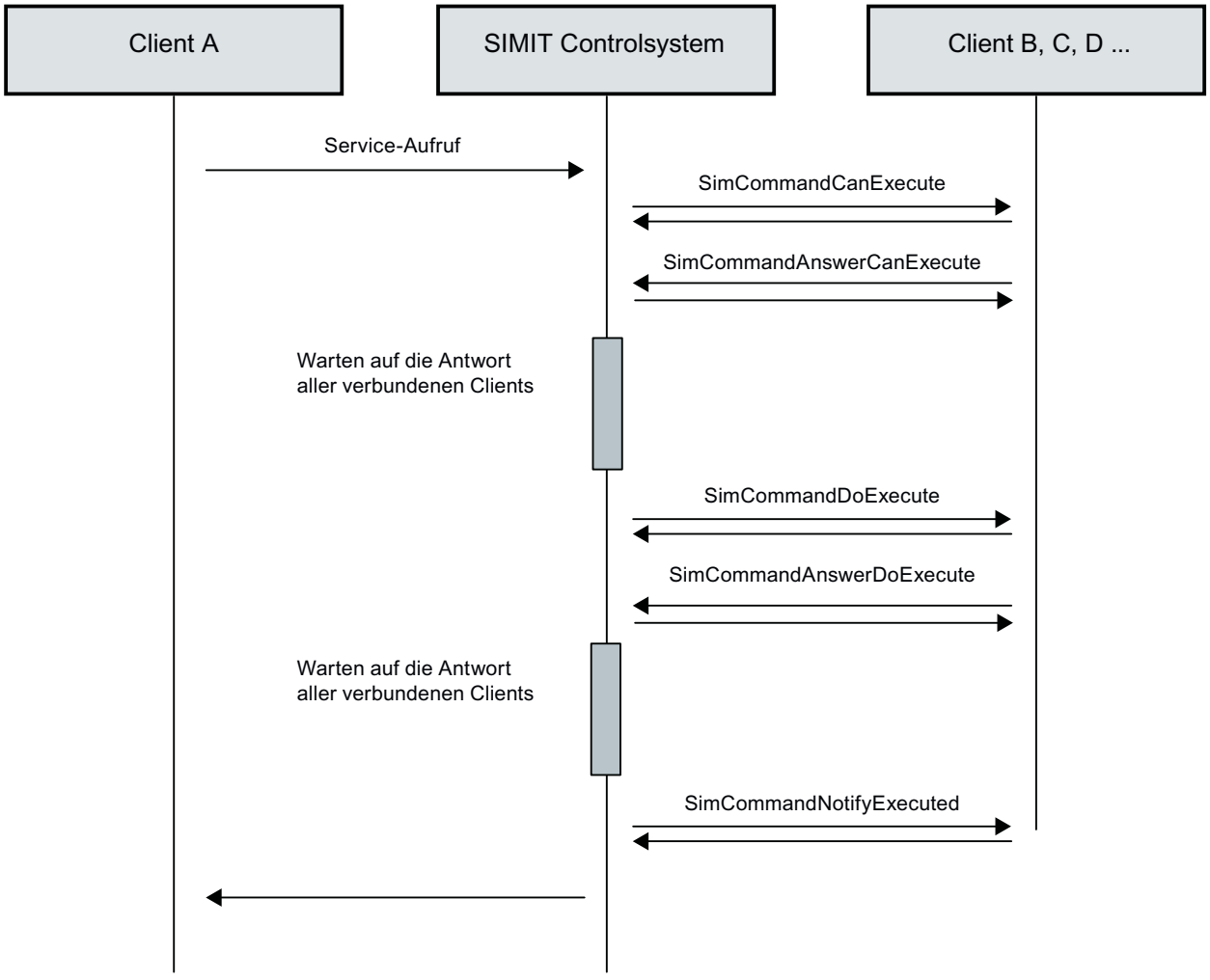
Wenn ein Client nicht "OK" zurückgibt, dann kann man in der Fehlermeldung zusätzliche Informationen als string übergeben.

Handshake-Protokoll

7

Um das System in einem konsistenten Zustand zu halten, müssen alle Clients wissen, welche Aufrufe andere Clients absetzen. Deswegen werden relevante Service-Aufrufe an alle angemeldeten Clients geschickt. Zusätzlich fragt SIMIT vorher nach, ob der Service-Aufruf von allen Clients ausgeführt werden kann. Das stellt sicher, dass sich die Clients im gleichen Zustand befinden.

Wenn kein Fehler auftritt, dann ergibt sich folgende Aufrufstruktur:



7.1 Abfrage der Ausführbarkeit

Wenn ein Client einen Service-Aufruf tätigt, dann werden andere Clients in der Regel vorher gefragt, ob sie überhaupt in der Lage sind, diesen Aufruf auszuführen. Dazu wird die Methode SimCommandCanExecute aufgerufen, die jeder Client implementieren muss:

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

7.2 Bestätigung der Ausführbarkeit

Um das Gesamtsystem in einem konsistenten Zustand zu halten, muss ein Client immer eine Bestätigung schicken.

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

Dieser Service-Aufruf wird nicht an andere Clients weitergereicht.

7.3 Anstoss des Service-Befehls

Nachdem alle Clients bestätigt haben, dass der Aufruf ausgeführt werden kann, wird der Aufruf tatsächlich ausgeführt. Dazu wird die Methode SimCommandDoExecute aufgerufen, die jeder Client implementieren muss:

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

7.4 Bestätigung der Ausführung

Um das Gesamtsystem in einem konsistenten Zustand zu halten, muss ein Client immer eine Bestätigung schicken.

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

Dieser Service-Aufruf wird nicht an andere Clients weitergereicht.

7.5 Information über den Abschluss des Service-Befehls

Nachdem SIMIT und alle angemeldeten Clients den Aufruf abgearbeitet haben, werden sie über die Beendigung des Service-Aufrufs informiert. Dazu wird die Methode SimCommandNotifyExecuted aufgerufen, die jeder Client implementieren muss:

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

Dieser Aufruf muss nicht mehr quittiert werden.

7.6 Allgemeine Fehlermeldungen

Um Fehlermeldungen an die Clients zu verschicken, ruft SIMIT die Methode SimBroadcastMessage auf, die jeder Client implementieren muss (gegebenenfalls leer):

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

Dieser Aufruf muss nicht quittiert werden.

Service-Aufrufe

8

8.1 Auf- und Abbau der Verbindung

8.1.1 Aufbau der Verbindung

Jeder Client, der mit SIMIT kommunizieren möchte, muss sich vorher verbinden.

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

Parameter	Beschreibung
connectedProjectPath	Der Pfad, in dem das aktuelle SIMIT Projekt liegt oder null, wenn noch kein Projekt geöffnet ist.
info	Information über die aktuelle Simulation, für Details siehe SimGetInfo.
id	Für diesen Parameter ist immer null zu übergeben.
name	Sprechender Name des Clients.
Rückgabewert	>0: Die eindeutige ID des Client, mit der er sich bei künftigen Service-Aufrufen identifiziert. <0: Der Service-Aufruf schlug fehl, für Details siehe result und errormessage.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.1.2 Abbau der Verbindung

Wenn ein Client nicht mehr mit SIMIT kommunizieren möchte, dann muss er sich abmelden.

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

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.1.3 Informationen über die aktuelle Simulation

Ein angemeldeter Client kann jederzeit Informationen über die aktuelle Simulation abfragen:

```
SimInfo SimGetInfo(
  int client,
```

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

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

Weitere Informationen hierzu finden Sie im Abschnitt: SimInfo (Seite 23).

8.1.4 Lebenszeichen

Wenn ein Client 24 Stunden lang nicht kommuniziert, wird die Verbindung zu SIMIT getrennt. Deswegen sollte ein Client wenigstens regelmäßige Aufrufe von SimGetInfo absetzen.

8.1.5 Informationen über die Spracheinstellung

Ein angemeldeter Client kann jederzeit abfragen, welche Sprache für SIMIT eingestellt ist.

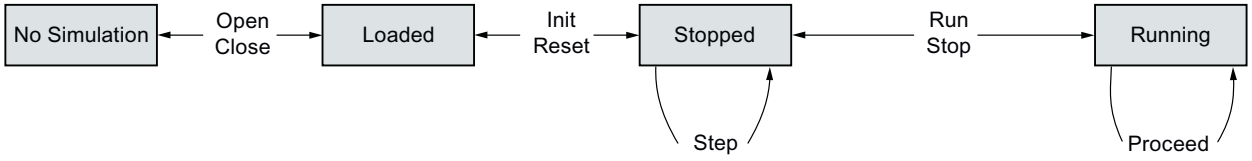
```
string SimGetLanguage(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

Liefert "de-de", wenn die Spracheinstellung deutsch ist, ansonsten "en-us".

8.2 Simulationskontrolle

Um eine Simulation auszuführen, muss das zugehörige Projekt geöffnet werden. Es kann immer nur ein Projekt offen sein.

Die Simulation befindet sich in einem der folgenden vier Zustände:



8.2.1 Projekt öffnen

Um ein Projekt zu öffnen, muss der Client die SIMIT-Datei ".simit" mit absolutem Pfad im Dateisystem angeben. Das geht nur, wenn kein Projekt geöffnet ist.

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

Parameter	Beschreibung
simitFile	Die SIMIT-Datei mit Endung ".simit" oder null, um das zuletzt genutzte Projekt zu öffnen.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.2 Projekt schließen

Jeder Client kann das aktuelle Projekt schließen, sofern ein Projekt offen ist und die Simulation sich im Zustand NoSimulation befindet.

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

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.3 Projekt umbenennen

Jeder Client kann das aktuelle Projekt umbenennen, sofern ein Projekt offen ist und die Simulation sich im Zustand NoSimulation befindet.

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

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.4 Simulation öffnen

- Jeder Client kann die Simulation öffnen, sofern ein Projekt geöffnet ist und die Simulation sich im Zustand NoSimulation befindet.
```
void SimOpen(
int client,
out ControlSystemResult result,
out string errormessage);
```
Alle angemeldeten Clients erhalten diesen Service-Aufruf.
- Jeder Client kann beim Öffnen der Simulation angeben, ob das Backtracking aktiv sein soll. Im bussynchronen Modus ist kein Backtracking möglich.
```
public void SimOpenEx(
   int client, out ControlSystemResult result, out string
errormessage,
   int backtrackingCycle, int maxNbrOfBacktracks)
```

Parameter	Beschreibung
backtrackingCycle	Die Zykluszeit, mit der Backtracks angelegt werden sollen. Die minimale Zykluszeit beträgt 30 Sekunden. Bei Angabe von 0 wird kein Backtracking aktiviert, bei Werten zwischen 1 und 29 wird die Zykluszeit auf 30 gesetzt.
maxNbrOfBacktracks	Die maximale Anzahl Backtracks im Ringpuffer. Sie muss zwischen 10 und 1000 liegen, bzw. wird auf diesen Wertebereich eingeschränkt.

Die Werte für "backtrackingCycle" und "maxNbrOfBacktracks" stehen beim Broadcast des Open-Befehls in der Struktur "ControlSystemServiceParams" in "IntVal1" bzw. "IntVal2".

8.2.5 Simulation schließen

Jeder Client kann die Simulation schließen, sofern die Simulation sich im Zustand Loaded befindet.
```
void SimClose(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.6 Simulation initialisieren

Jeder Client kann die Simulation initialisieren, sofern die Simulation sich im Zustand Loaded befindet.
```
void SimInit(
  int client,
  out ControlSystemResult result,
  out string errormessage);
```

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.7 Simulation zurücksetzen

Jeder Client kann die Simulation zurücksetzen, sofern die Simulation sich im Zustand Stopped befindet.

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

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.8 Einzelschritt durchführen

Jeder angemeldete Client kann einen Einzelschritt in der Simulation durchführen, sofern die Simulation sich im Zustand Stopped befindet.

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

Parameter	Beschreibung
time	Die Zeitspanne in Millisekunden, um die die Simulationszeit fortschreiten soll. Wenn der Wert <=0 ist, dann wird die kleinste Zykluszeit des aktuellen Projekts verwendet.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.9 Simulationszeit weiterschalten

Ein Client mit Kontrolle über die Simulationszeit kann mit diesem Aufruf die Simulationszeit fortschalten, sofern die Simulation sich im Zustand Running befindet und der Synchronisationsmodus SyncPassive ist.

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

Parameter	Beschreibung
time	Die Zeitspanne in Millisekunden, um die die Simulationszeit fortschreiten soll. Wenn der Wert <=0 ist, dann wird die kleinste Zykluszeit des Projekts verwendet.

Hinweis

Alle angemeldeten Clients erhalten diesen Service-Aufruf. Die Clients werden vorher nicht mit SimCommandCanExecute und SimCommandAnswerCanExecute gefragt, ob sie den Aufruf durchführen können. Ein Client, der SimRun mit Parameter SyncActive oder SyncPassive akzeptiert hat, muss SimProceed durchführen können. Der Client hat damit also SimCommandCanExecute implizit im Voraus bestätigt.

8.2.10 Simulation starten

Jeder angemeldete Client kann die Simulation starten, sofern die Simulation sich im Zustand Stopped befindet.

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

Parameter	Beschreibung
syncMode	Gibt an, in welcher Betriebsart die Simulation laufen soll: • Async: Die Simulation läuft asynchron • SyncActive: Die Simulation läuft synchron, der Aufrufer wird die Kontrolle über die Simulationszeit übernehmen. • SyncPassive: Die Simulation läuft synchron, der Aufrufer wird nicht die Kontrolle übernehmen. Wenn kein anderer Client die Kontrolle übernimmt, wird SIMIT die Simulationszeit steuern.
speed	Die zu verwendende Geschwindigkeit. Bei Werten <0 wird die zuletzt gesetzte Geschwindigkeit genutzt, der Standard-Wert ist Echtzeit.
proceedTime	Die Zeit, die für PROCEED-Aufrufe genutzt werden soll, wenn SIMIT die Kontrolle übernimmt. SIMIT wird das Minimum aus diesem Wert, anderen Werten die von anderen Clients gesetzt werden und der minimalen Zykluszeit des Projekts verwenden.

Andere Clients können über SimCommandAnswerCanExecute folgende Informationen geben, indem die entsprechenden Werte in den ControlSystemParameters gesetzt werden:

RequireSyncTime = Wert
Fordert eine maximale Zeit für Proceed-Aufrufe.

RequireSyncControl = true
Dieser Client möchte die Zeitsteuerung übernehmen

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.2.11 Geschwindigkeit setzen

Jeder angemeldete Client kann die Geschwindigkeit der Simulation setzen.

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

Parameter	Beschreibung
speed	Geschwindigkeit, Bedeutung der Werte siehe SimInfo

Alle angemeldeten Clients erhalten diesen Service-Aufruf, sofern die Simulation sich im Zustand Running befindet. Ansonsten wird der Wert beim Aufruf von SimRun ausgewertet.

8.2.12 Simulation anhalten

Jeder Client kann die Simulation anhalten, sofern die Simulation sich im Zustand Running oder Stopped befindet.

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

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3 Snapshots

Sie können den Zustand der Simulation als Snapshot in einer Datei speichern. Snapshots werden im Ordner "snap" oder einem Unterordner abgelegt. Alle Aufrufe, die mit Snapshots zu tun haben, beziehen sich auf diese Ordner.

Um den Simulationszustand konsistent zu halten, müssen alle angemeldeten Clients Snapshots ihres Zustandes erzeugen und ähnlich wie SIMIT verwalten können.

8.3.1 Snapshot erzeugen

Dieser Aufruf ist nur zulässig, wenn sich die Simulation im Zustand Running oder Stopped befindet.

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

Parameter	Beschreibung
name	Name des Snapshots. Der Snapshot wird im Ordner "snap" des Projekts abgelegt. Hier können Sie keinen Unterordner angeben.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.2 Snapshot-Ordner anlegen

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

Parameter	Beschreibung
name	Der Name des neuen Ordners, relativ zum Ordner "snap".

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.3 Snapshot laden

Dieser Aufruf ist nur zulässig, wenn sich die Simulation im Zustand Loaded befindet.

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

Parameter	Beschreibung
name	Name des Snapshots. Wenn Unterordner vorhanden sind, muss vor diesen ein Schrägstrich "/" stehen.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.4 Snapshot löschen

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

Parameter	Beschreibung
name	Name des Snapshots. Vor Unterordnern muss ein Schrägstrich "/" stehen.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.5 Snapshot-Ordner löschen

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

Parameter	Beschreibung
name	Name des Snapshot-Ordners, relativ zum Snapshot-Ordner "snap". Vor Unterordnern muss ein Schrägstrich "/" stehen.

Hinweis

Alle Unterordner werden gelöscht.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.6 Snapshot umbenennen

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

Parameter	Beschreibung
oldName	Alter Name des Snapshots. Vor Unterordnern muss ein Schrägstrich "/" stehen.
newName	Neuer Name des Snapshots. Vor Unterordnern muss ein Schrägstrich "/" stehen.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.7 Snapshot-Ordner umbenennen

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

Parameter	Beschreibung
oldName	Alter Name des Snapshot-Ordners. Vor Unterordnern muss ein Schrägstrich "/" stehen.
newName	Neuer Name des Snapshot-Ordners. Vor Unterordnern muss ein Schrägstrich "/" stehen.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.8 Snapshot kopieren

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

Parameter	Beschreibung
oldName	Name des bestehenden Snapshots. Vor Unterordnern muss ein Schrägstrich "/" stehen.
newName	Name des kopierten Snapshots. Vor Unterordnern muss ein Schrägstrich "/" stehen.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.3.9 Snapshot-Ordner kopieren

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

Parameter	Beschreibung
oldName	Name des bestehenden Snapshot-Ordners. Vor Unterordnern muss ein Schrägstrich "/" stehen.
newName	Name des neuen Snapshot-Ordners. Vor Unterordnern muss ein Schrägstrich "/" stehen.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

8.4 Backtracks

Backtracks enthalten die gleiche Information wie Snapshots, allerdings in einem anderen Format.

Sie können nur geladen oder in Snapshots umgewandelt werden, solange die Simulation angehalten, aber noch nicht beendet ist.

8.4.1 Backtracks erzeugen

Der Befehl zum Erzeugen von Backtracks wird grundsätzlich nur von SIMIT selbst ausgegeben.

Angemeldete Clients werden über diesen Aufruf aufgefordert, ihrerseits Backtracks anzulegen.

```
public void SimCreateBacktrack(
   int client, out ControlSystemResult result, out string
errormessage,
   out int index)
```

Parameter	Beschreibung
index	Der Index des neuen Backtracks im Ringpuffer.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

Dieser Index steht beim Broadcast dieses Befehls in der Struktur "ControlSystemServiceParams" in "IntVal1".

8.4.2 Liste der verfügbaren Backtracks

Die Liste der verfügbaren Backtracks liefert den Inhalt des Backtrack-Ringpuffers.

Das Array hat immer die Größe des aktuellen Werts für die maximale Anzahl der Backtracks.

Die Backtracks sind gekennzeichnet durch die Systemzeit sowie die dazu gehörige Simulationszeit z. B.:

2018-06-22 09:44:50 30100

Die Systemzeit ist, wann die Backtracks erstellt wurden.

Die noch nicht belegte Plätze im Ringpuffer enthalten keine Zeichenfolge, sondern die Null-referenz.

```
public string[] SimGetBacktracks(
  int client, out ControlSystemResult result, out string
errormessage)
```

Dieser Service-Aufruf wird nicht weitergeleitet.

8.4.3 Backtracks laden

Dieser Aufruf ist nur zulässig, wenn sich die Simulation im Zustand "Stopped" befindet.

```
public void SimLoadBacktrack(
   int client, out ControlSystemResult result, out string
```

```
errormessage,
   int index)
```

Parameter	Beschreibung
index	Der Index des Backtracks im Ringpuffer.

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

Dieser Index steht beim Broadcast dieses Befehls in der Struktur "ControlSystemServiceParams" in "IntVal1".

8.4.4 Backtracks in Snapshots wandeln

Dieser Aufruf ist nur zulässig, wenn sich die Simulation im Zustand "Loaded" befindet.

```
public void SimConvertBacktrack(
   int client, out ControlSystemResult result, out string
errormessage,
   int index, string name)
```

Parameter	Beschreibung
index	Der Index des Backtracks im Ringpuffer.
name	Name des zu erstellenden Snapshots

Alle angemeldeten Clients erhalten diesen Service-Aufruf.

Der Index und der Name stehen beim Broadcast dieses Befehls in der Struktur "ControlSystemServiceParams" in "IntVal1" bzw. "StringVal1".

Minimal-Implementierung eines Client

9

Der folgende Code ist eine syntaktisch vollständige Minimalimplementierung eines Clients ohne eigene Funktionalität.

```
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);
      }
    }
  }
```

Implementierung eines passiven, synchronisierten Clients

10

Für die synchrone Betriebsart muss ein Client sich per WCF-Interface mit SIMIT verbinden und dann PROCEED-Aufrufe verarbeiten, um seine eigenen Simulationszyklen zu berechnen.

Der folgende Code ist eine syntaktisch vollständige Minimalimplementierung eines Clients, der auf PROCEED-Befehle hin eine eigene Berechnung durchführt.

```
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);
    }
  }
}
```