COMPACT TALK

Integrationsbeschreibung

Dokumentenversion 1.2

WELAND SOLUTIONS

Inhalt

1 Einleitung 4

2 Referenzen 4

3 Systemübersicht 4

3.1 Compact Talk Service 5

3.2 CTClient-API 5

3.3 Webservice 5

3.4 Import und Export 5

4 Servicekonzept 5

4.1 Servicetypen 6

4.1.1 BasicService 6

4.1.2 BasicControllableService 6

4.1.3 BasicSharableService 7

4.1.4 ServiceManager 7

4.1.5 ServiceDescriptor 7

5 Basiskonzepte 7

5.1 Auftragsabwicklung 7

5.1.1 Statusfluss quittiert durch Bediener 8

5.1.2 Statusfluss quittiert durch Client 9

5.2 Eventabwicklung 9

5.3 Datenmodell 10

5.3.1 Tray 10

6 Allgemeine Programmierfälle 11

6.1 Externe Quittierung 11

6.2 Bestandsmanagement 11

6.3 Auftragssynchronisierung 11

6.4 Wartung 12

6.4.1 ReturnTrays 12

6.4.2 ClearOrderQueue 13

7 Zubehör und deren Konfiguration 13
7.1 Methodenaufruf 14
7.2 Import von Auftrag 14
7.3 Import von Tray 14
8 CTCLIENT-API 14
8.1 CTConnection 14
8.2 StopCodes 16
8.2.1 Allgemeine Informationen 16
8.2.2 StopCodes 16
8.2 Code-Samples 19
8.2.1 Sample 1 20
8.2.2 Sample 2 20
8.2.3 Ein umfangreicheres Sample, Mini WMS 21
8.2.4 Beispiel für Tray-Zugriff 28
9 Webservice 28
9.1 Konvertierung von Client CT 1.x zu 2.x 29
9.1.1 Neues Event-Handling 29
9.1.2 Methoden-Übersetzungstabelle 29
Interface 10 XML 32
10.1 Übersicht 32
10.2 Befehle 33
10.2.1 Methode AddToQueue 33
10.2.2 ExtAckOrder 38
10.2.3 AddTrayConfig 39
10.2.4 ResetElevator 40
10.3 Rückmeldungen 40
10.3.1 CommandResponse 40
10.3.2 OrderStatusResponse 41
10.3.3 TaskDoneResponse 41

11 Import und Export 42
11.1 Import 42
11.2 Export 42
11.3 Import von Tray-Konfiguration 42
11.3.1 Konfiguration 42
11.3.2 Format 42

1 Einleitung

Dieses Dokument beschreibt externe Schnittstellen zur Integration mit Compact Talk. Das Dokument bietet einen kurzen Überblick über das System und eine Beschreibung der drei Möglichkeiten zur Integration mit Compact Talk. CTClient-API (Application Programming Interface), XML-interface und Import/Export über externe Quellen.
Eine Kommunikation mit Compact Talk kann über die folgenden Formate erfolgen:

- Windows Communication Foundation (WCF)
  Compact Talk bietet die API CTClient zur vereinfachten Integration und für vollen Zugang zu allen Funktionen. CTClient ist die empfohlene Lösung für Systeme mit Unterstützung der .NET-Plattform. Es werden die Protokolle TCP/IP und IPC unterstützt (Named Pipes).
- Webservice
  Zur einfachen Integration über Systeme ohne die Unterstützung der .NET-Plattform verfügt Compact Talk außerdem über einen Webservice auf Basis von HTTP. Da keine Full-Duplex-Kommunikation verfügbar ist, müssen Events manuell abgefragt werden.
  Der Webservice unterstützt SOAP 1.1 und folgt WS-I BP 1.1. Eine Installation von IIS ist nicht erforderlich.
- XML-Interface
  zum Management von Standardbefehlen über XML-Dateien
- Import- und Exportschnittstelle
  - Zum Management des Imports von Auftrag und Export von Auftragsdaten über Flat-Files. Die visuelle Konfigurationsschnittstelle bietet hohe Flexibilität

2 Referenzen

[1] Konfigurationshandbuch Compact Talk V3
[2] API-Referenzhandbuch (CompactTalkAPI.chm)

3 Systemübersicht

Compact Talk ist in zwei zentrale Teile unterteilt: Compact Talk Service und CTClient-API, wie in Abbildung 1 erläutert.

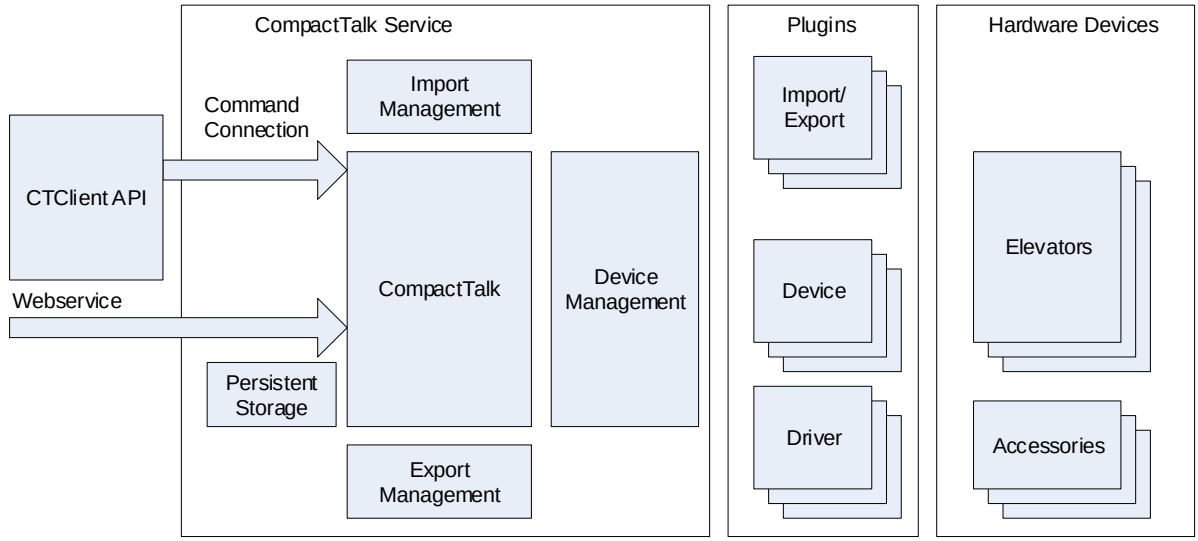


Abbildung 1, Systemübersicht

3.1 Compact Talk Service

Die Serviceanwendung managt die Kommunikation mit Implementierungen, Auftragsmanagement, Import- und Exportmanagement und persistentes Speichermanagement von Dritten. Auf die Befehlsverbindung kann über Windows Communication Foundation (WCF) direkt zugegriffen werden. Dies wird in diesem Dokument allerdings nicht erläutert. Außerdem ist ein Zugriff auf die Befehlsverbindung wie in Abschnitt 9 beschrieben über HTTP möglich.

3.2 CTClient-API

Die Client-API deckt die Bereiche Verbindungsmanagement, Fehlermanagement und synchronisierte Eventabwicklung ab. Diese API wird von diesem Dokument umfassend beschrieben. Diese API macht Nutzen von der Befehlsverbindung.

3.3 Webservice

Der Webservice ist ein standardisierter SOAP-basierter Service mit Zugriff über HTTP.

3.4 Import und Export

Diese Schnittstellen ermöglichen einen Import von Aufträgen und einen Export von gewählten Auftragsinformationen, ausgelöst durch Statusänderungen. Eine grundlegende Implementierung dieser Interfaces über Textdateien wird in diesem Dokument beschrieben.

4 Servicekonzept

Die meisten aktiven und austauschbaren Komponenten von Compact Talk basieren auf der Klasse BasicService oder einer ihrer Unterklassen. Ein Service hat einen Zustand, der überwacht werden kann, sowie einige Methoden zur Kontrolle des Status.

Das Management von Services erfolgt über einen Servicemanager, der alle Servicereferenzen in einer hierarchischen Struktur speichert. Die nachfolgende Abbildung stellt ein Beispiel einer Hierarchie mit zwei Partitionen, drei Elevators und ein Zubehör pro Elevator.

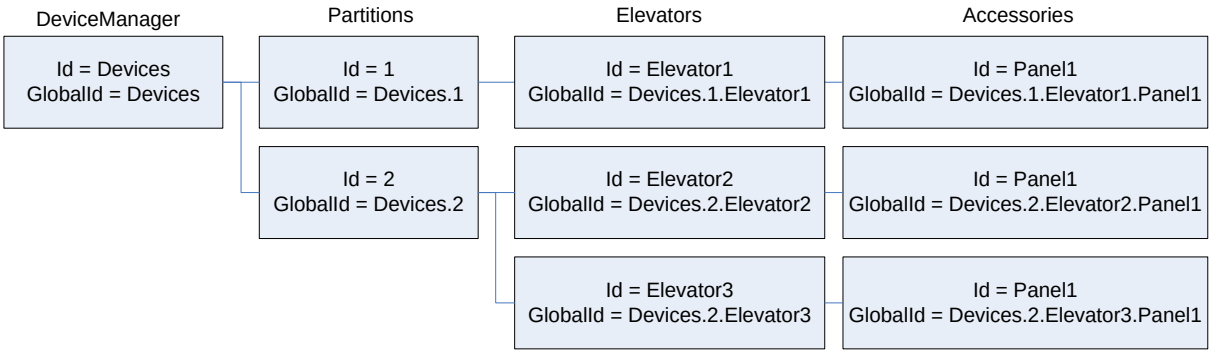


Abbildung 2, Beispiel einer Servicehierarchie

4.1 Servicetypen

Es gibt fünf Klassen und einen in das Servicekonzept involvierten Aufzählungstypen, wie in Abbildung 3 gezeigt.

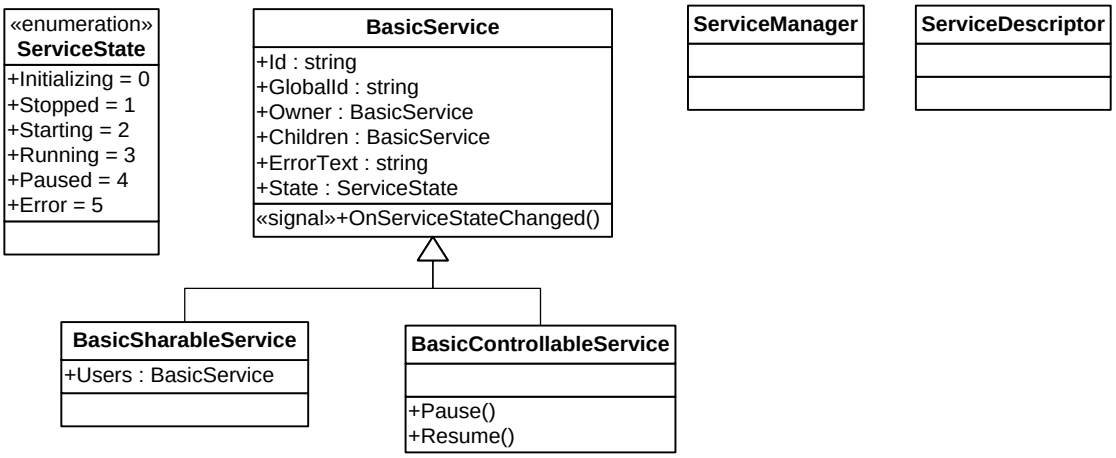


Abbildung 3, Serviceklassenhierarchie

4.1.1 BasicService

Dieser Typ ist die Basisklasse an Servicetypen. Er umfasst Eigenschaften für Identität, Beziehungen und Zustand.
Diese Klasse umfasst beispielsweise Kommunikationstreiber. Außerdem verfügt sie über einen Event-Handler, der immer ein Event auslöst, wenn sich der Zustand ändert.

4.1.2 BasicControllableService

Diese Klasse ist eine Unterklasse von *BasicService* und ergänzt zwei Methoden zur Kontrolle des Zustands eines Services Diese Klasse umfasst beispielsweise Elevator-Gerät.

4.1.3 BasicSharableService

Dieser Typ ist auch eine Unterklasse von *BasicService*. Das Besondere an einer Instanz dieser Klasse ist, dass sie von mehreren Quellen, die sich eine einzige Ressource teilen, referenziert werden kann. Das Bustreiber in der nachfolgenden Abbildung ist ein Beispiel für eine Komponente, die eine Ressource teilt. In diesem Fall wird ein Multidrop-Modem mit einem oder mehreren Elevator-Geräten geteilt.

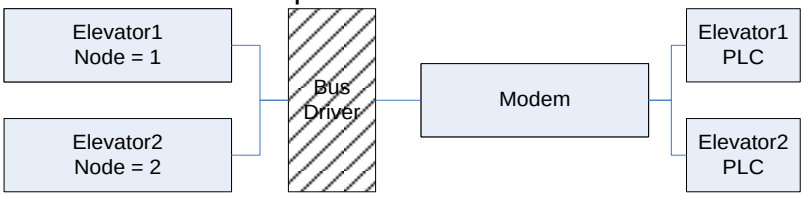


Abbildung 4, Beispiel mit Nutzung von BasicSharableService

4.1.4 ServiceManager

Der Servicemanager ist ein Container für Services und bietet Methoden zum Durchsuchen der Servicehierarchie und dem Abrufen des aktuellen Zustands eines Services. Außerdem verfügt er über einen Event-Handler, der immer auslöst, wenn sich der Zustand eines Service ändert.

4.1.5 ServiceDescriptor

Diese Klasse beschreibt einen geladenen Service im Server. Zum Beispiel Elevator-Geräte, Zubehör und ERP-Plugins.

5 Basiskonzepte

CT nutzt Auslageraufträge zur Beauftragung von Elevators zum Transport von Trays zu Bedienern. Ein Auslagerauftrag umfasst Informationen zu Elevator, Tray und Ziel-Serviceöffnung sowie Informationen, die an Zubehör dargestellt werden. können.

5.1 Auftragsabwicklung

Auslageraufträge werden in eine Warteschleife gegeben, bevor der entsprechende Elevator gesendet wird. Dadurch wird der Auftrag durch den Aufruf *AddToQueue* nicht direkt an den Elevator geleitet, weshalb es wichtig ist, den Status bei Aufträgen zu überwachen.
Aufträge werden nach der Aus- oder Einlagerung entweder durch den Bediener oder extern durch den Client quittiert. Das Argument **noReturnOfTray** im Aufruf *AddToQueue* steuert, ob eine Quittierung des Auftrags am Bedienfeld ermöglicht wird oder nicht. Wird es auf 1 gesetzt, ist ein Aufruf von *ExtAchOrder* vor Rücksendung des Trays erforderlich.
Die verschiedenen Status eines Auftrags sind nachfolgend aufgelistet:

Status	**Beschreibung**
Posted	Der Auftrag ist passiv und erfordert eine Aktivierung zur Versendung.
Selected	Der Auftrag ist aktiv und zur Auswahl durch den Dispatcher verfügbar.
Sent	Der Auftrag wurde erfolgreich an das Elevator-Gerät versendet.

NextAtPlace	Gibt an, dass der Auftrag der nächste ist, der an eine Öffnung läuft.
AtPlace	Gibt an, dass sich das angeforderte Tray an einer Auslagerposition befindet.
TaskDone	Der Auftrag wurde durch den Bediener oder Client quittiert.

Status bei Nutzung von externer Bestätigung

Accepted	Der Auftrag wurde durch Bediener und Client bestätigt.
AcceptedStillAtPlace	Der Auftrag wurde durch Bediener und Client bestätigt, wartet allerdings noch auf eine externe Quittierung durch den Client.
TaskDoneStillAtPlace	Der Auftrag wurde nur durch den Bediener, aber nicht durch den Client bestätigt.

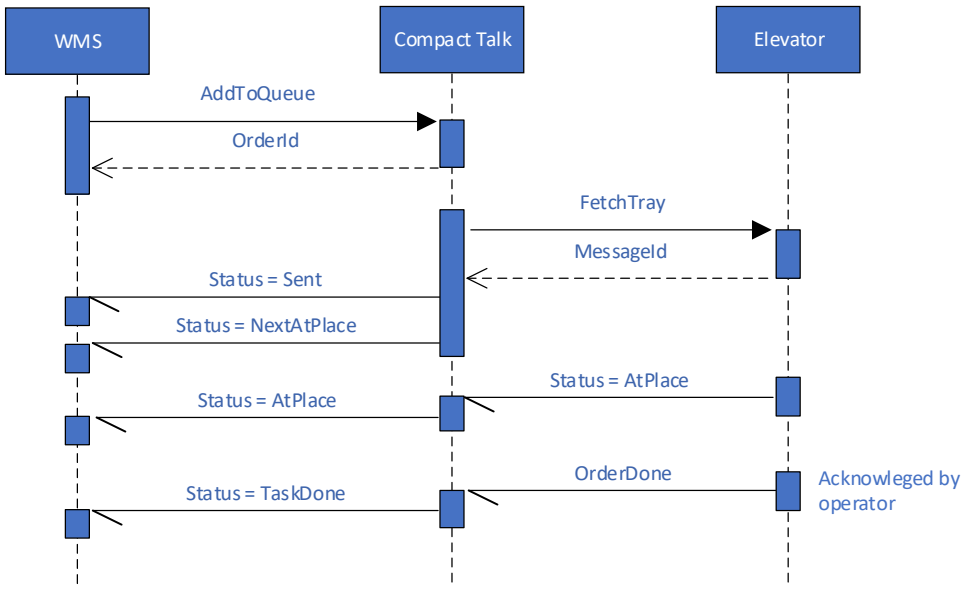
Fehlerstatus

Refused	Der Auftrag wurde durch das Elevator-Gerät abgelehnt. Der Auftrag gilt nicht als aktiv und kann gelöscht werden.

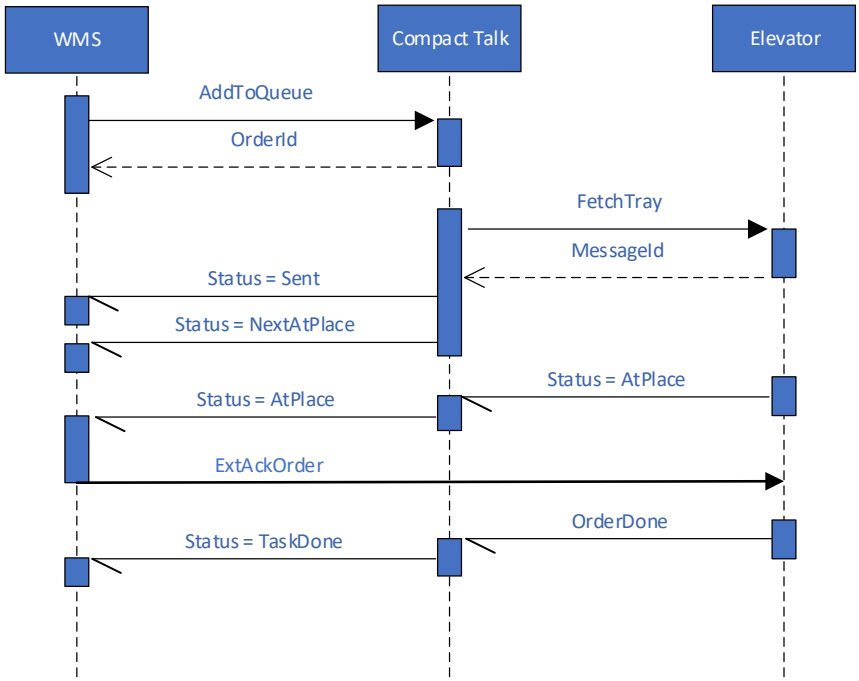
Ein Beispiel für die Inhalte in der Warteschleife könnte folgendermaßen aussehen:

Order	Tray	Status
1	10	AtPlace
2	11	NextAtPlace
3	12	Sent
4	13	Selected

5.1.1 Statusfluss quittiert durch Bediener

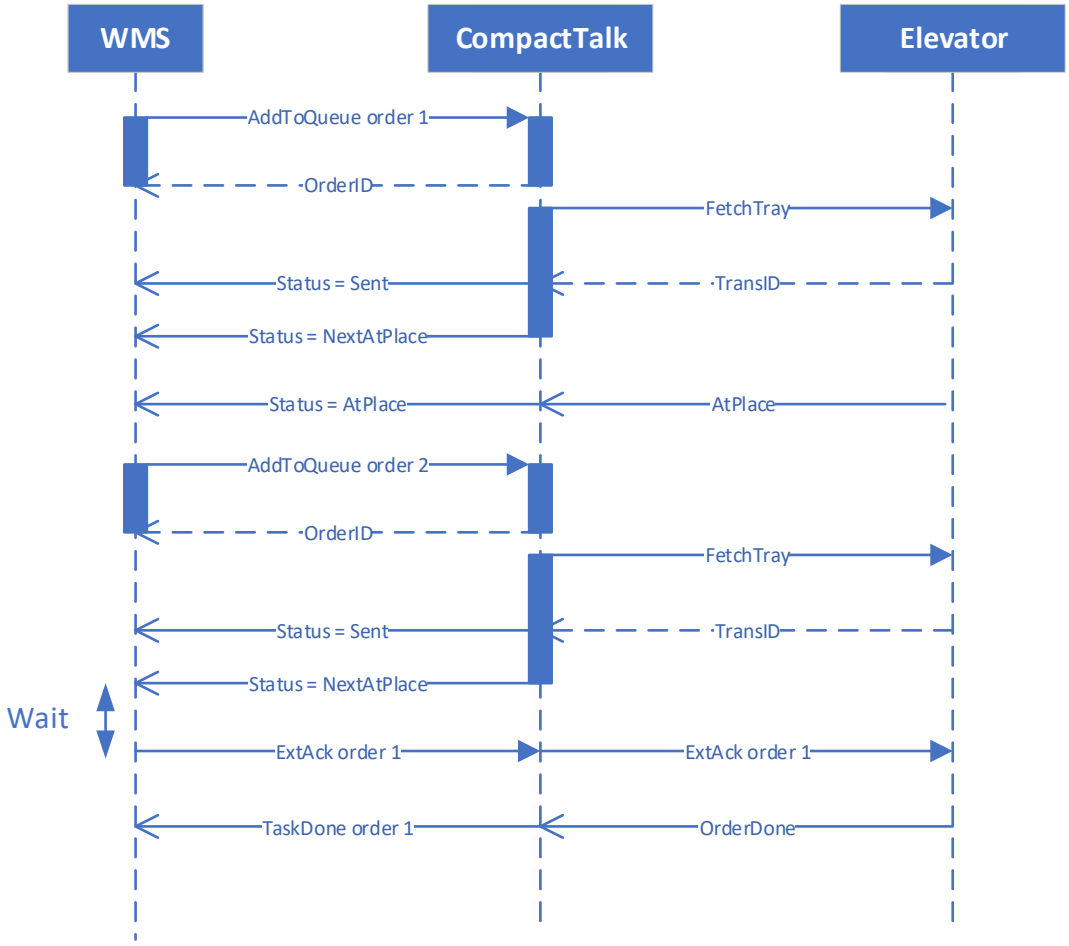


5.1.2 Statusfluss quittiert durch Client



5.2 Eventabwicklung

Die Statusänderungen von Aufträgen werden als Events an den Client versendet. Die Methode hängt von der Art der Verbindung ab. Die CTClient-API übernimmt für Sie diese Arbeit durch Abfrage der Eventwarteschleife und Senden dieser als Event an die Anwendung. Bei Verbindung mit dem Webservice-Interface müssen neue Events in der Eventwarteschleife explizit abgefragt werden.
Eine korrekte Eventabwicklung ist aufgrund von Timing-Problemen von großer Bedeutung. Diese Timing-Probleme treten auf, da Aufträge durch den Sender nicht direkt bei dem Aufruf *AddToQueue* an den Elevator gesendet werden.
Nehmen wir zum Beispiel an, Sie möchten immer nur einen aktiven Auftrag zur gleichen Zeit, aber Sie möchten nicht, dass das Tray zwischen den Aufträgen zurückgegeben wird, wenn diese sich auf dasselbe Tray beziehen. In diesem Fall müssen Sie sicherstellen, dass der zweite Auftrag den Elevator erreicht hat, bevor der erste Auftrag bestätigt wird, da der Elevator sonst nicht weiß, ob das Tray an der Öffnung belassen werden soll. Dies erfolgt durch das Abwarten eines Status von **Sent** oder höher am zweiten Auftrag. Siehe Diagramm:



5.3 Datenmodell

Beschreibung von Datentypen und Modellen

5.3.1 Tray

In einem Elevator kann es mehrere Trays geben. Jedes Tray hält spezifische Eigenschaften je nach Höhe der auf dem Tray platzieren Artikel und dem Gesamtgewicht darauf.
Das Brick-Layout wird in Compact Talk nicht gespeichert und kann nur von WMS erhalten werden.

Feldname	**Typ**	**Beschreibung**
Tray	object	Hält Informationen über ein Tray im Elevator
tray:Blocked	bool	Wenn Tray blockiert ist und nicht genutzt werden kann
tray:Height	int	Höhe des Trays in mm (belegter Platz in Höhe).
tray:Level	int	Elevator **G1**: Dargestellte Zugangsebene

		Elevator **G2**: Dargestellte Position eines Trays von der unteren Grenzposition bis zur oberen Höhe des aktuellen Trays in mm.
tray:Weight	int	Gewicht des Trays in Gramm
tray:Id	int	Numerische ID des Trays
tray:Borrowed	bool	Gibt an, dass ein Tray vom Elevator vorübergehend entnommen wurde und sich aktuell nicht im Elevator befindet.

6 Allgemeine Programmierfälle

In diesem Abschnitt werden allgemeine Integrationsfälle beschrieben, die ein empfohlenes Minimum an Funktionen abdecken, das bei der Integration mit Compact Talk implementiert werden sollte. Diese Fälle werden auch von der Beispielanwendung MiniWms abgedeckt.

6.1 Externe Quittierung

Die externe Quittierung *ExtAckOrder* ist nützlich, wenn das WMS-System vor der Rückgabe des Trays das letzte Wort haben soll. Um dies zu erreichen, muss der Elevator bei Hinzufügen des Auftrags die Information erhalten, dass vor der Rückgabe des Trays eine externe Quittierung erforderlich ist. Dies erfolgt durch ein Setzen des Arguments noReturnOfTray auf 1 in der Methode *AddToQueue*. Zur Durchführung der externen Quittierung am Auftrag ist ein Status erforderlich, der angibt, dass es sich auf dem Tisch im Elevator befindet.

6.2 Bestandsmanagement

Im nachfolgenden Code wird eine einfache Implementierung des Bestandsmanagements dargestellt. Bei Empfang des Events *CTOrderStatusChangeEvent* ist eine Aktivierung der Moduseigenschaft für den mit dem Event verbundenen Auftrag möglich. Der Modus des Auftrags wird gesetzt, wenn die Methode *AddToQueue* aufgerufen wird.

```
void OnOrderStatusChangedEvent(CTOrderStatusChangeEvent evt)
{
          if (evt.Order.Status == OrderStatus.TaskDone)
          {
                //Do stock management based on the OrderMode
                if (ctOrder.Mode == OrderMode.OUT)
                    //Subtract the evt.Order.AckQuantity from the quantity on the article record
                else if (ctOrder.Mode == OrderMode.IN)
                    //Add the evt.Order.AckQuantity to the quantity on your article record
                else if (ctOrder.Mode == OrderMode.INV)
                    //Replace the quantity on your article record with evt.Order.AckQuantity
          }
}
```

6.3 Auftragssynchronisierung

Die Auftragssynchronisierung ist nützlich, wenn das System abgeschaltet wurde, während sich in Compact Talk noch aktive Aufträge in der Warteschleife befunden haben. Während dieser Zeit kann es passieren, dass ein Auftrag durch einen Bediener quittiert wurde und das Event verpasst wurde, wenn der Status des

Auftrags auf *TaskDone* gesetzt wurde. Dies führt zu Problemen mit dem Bestandsmanagement, da die quittierte Menge vom Bediener nicht zur Verfügung stand. Der nachfolgende Code stellt dieses Problem dar und zeigt seine Lösung auf.

```
public class Class1()
{
        //Keep e list of orders sent to Compact Talk
        List<OrderRecord> m_OrderCache = new List<OrderRecord>();
        void SynchronizeOrders()
        {
            for (int i = 0; i < m_OrderCache.Count; i++)
            {
                //Retrieve the order from Compact Talk that has the same orderId as your cached order
                //The GetOrder method will first look for the order in Compact Talk's order queue, if
                //it's not found there it will look in the table of historical order data
                PickOrder ctOrder = m_CompactTalk.Command.GetOrder(m_OrderCache[i].CTOrderId);
                if (ctOrder.Status == OrderStatus.Historical)
                {
                    //A historical order means that it has gone trough a full status cycle and
                    //now only exists as an historical record in the database.
                    if (ctOrder.Mode == OrderMode.OUT)
                        //Subtract the evt.Order.AckQuantity from the quantity on the article record
                    else if (ctOrder.Mode == OrderMode.IN)
                        //Add the evt.Order.AckQuantity to the quantity on your article record
                    else if (ctOrder.Mode == OrderMode.INV)
                        //Replace the quantity on your article record with evt.Order.AckQuantity
                    //Remove the order from the cache and do implementation specific maintenance
                }
                else
                {
                    //Just uppdate the status on your cached order here
                    m_OrderCache[i].CTStatus = ctOrder.Status.ToString();
                }
            }
        }
}
```

Die Methode SynchronizeOrders in der obenstehenden Klasse sollte immer aufgerufen werden, wenn das System gestartet wird oder nachdem eine neue Verbindung mit Compact Talk nach einem Verbindungsverlust hergestellt wird.

6.4 Wartung

Zur Durchführung der nachfolgenden Wartungsmethoden müssen sich die Elevators im Zustand Pause befinden. Der Zustand Pause bedeutet, dass das Senden von neuen Aufträgen an den Elevator pausiert ist. Die zu nutzende Methode ist *PauseService*.
Die Wartungsmethoden besitzen das Argument servicePath zur Auswahl der Zielebene in der Gerätehierarchie. ServicePath kann auf alle Elevators, eine Partition oder einen spezifischen Elevator gerichtet sein.
Zur Auswahl aller Elevatoren dient der servicePath „Devices“, für eine Partition „Devices.<partitionid>“ und für einen spezifischen Elevator entweder „Devices.<partitionid>.<elevatorid>“ oder das servicePath-Alias, das der Elevator-ID entspricht. Für einen Elevator mit der ID „EI“ auf Partition „P1“ könnte der servicePath „Devices.P1.E1“ oder nur „E1“ sein.
Nach der Wartung ist der Aufruf *ResumeService* erforderlich.

6.4.1 ReturnTrays

Diese Methode dient dem Abbruch aller aktiver Aufträge im Elevator und erzwingt eine Rücksendung aktiver Trays durch den Elevator zur Lagerposition. Der Status

von aktiven Aufträgen wird auf „Selected“ zurückgesetzt. Das bedeutet, sie sind bereit für eine Auswahl bei Wiederaufnahme der Abwicklung.
Für die Rücksendung von Trays existieren zwei Vorgänge, die eine Rücksendung von Trays und einen Reset von Aufträgen auslösen.

- ReturnTrays, Unterstützung in G1, G2, Sim
- ReturnTraysByOpening, Unterstützung in G2

Hinweis: Der Service muss vor dem Senden des Befehls "Return Tray" pausiert werden.

6.4.2 ClearOrderQueue

Diese Methode dient dem Löschen aller Aufträge außer aktive Aufträge der durch servicePath abgedeckten Elevators. Zur Löschung aller durch servicePath abgedeckten Aufträge ist ein Abbruch der aktiven Aufträge durch Aufruf der oben beschriebenen Methode *ReturnTrays* erforderlich.
Für eine Portierung der Integration von CT v1 ist es wichtig zu wissen, dass diese Methode nicht dieselben Funktionen, wie die alte Methode besitzt. Um die Funktionen der alten Methode durchzuführen ist zunächst der Aufruf *PauseService* und anschließend *ReturnTrays* gefolgt von *ClearOrderQueue* erforderlich. Der Grund für den Aufruf von ReturnTrays ist das Rücksetzen des Status von aktiven Aufträgen auf „Selected“, da *ClearOrderQueue* diese andernfalls in der Warteschleife hält.

7 Zubehör und deren Konfiguration

Für Elevators steht eine Reihe an Zubehör wie LightBar, LaserPointer und PickDisplay zur Verfügung. Um deren korrekte Funktion zu gewährleisten ist eine Konfiguration in Compact Talk erforderlich. Je nach Art des genutzten Zubehörs stehen zwei Arten an Konfigurationen zur Verfügung: Single-Target-Box oder Full-Traylayout aus einer Liste an Boxen. Die Box verfügt auch über zwei benutzerdefinierte Eigenschaften, die durch das Zubehör genutzt werden können.
Die nachfolgende Tabelle gibt die erforderlichen Konfigurationen für das Zubehör an.

Zubehör	**Single-Box**	**Full-Layout**
LightBar	X (Eigenschaft zur Angabe der Tiefe muss bereitgestellt werden)	X (Eigenschaft Tiefe wird berechnet)
LaserPointer	X	X
PickDisplay		X

Zum Hinzufügen der erforderlichen Konfiguration stehen zwei Möglichkeiten zur Verfügung. Diese sind in der nachfolgenden Tabelle beschrieben.

Typ	**Methodenaufruf**	**Import von Auftrag**	**Import von Tray**
Single-Box	AddToQueueWithSingleBoxCoords	X	
Full-Layout	AddTrayConfig		X

HINWEIS: Die an das Zubehör PickDisplay gesendeten Artikelbilder müssen die folgenden Anforderungen erfüllen: 1920x1080px oder maximal 5MB/Bild.

7.1 Methodenaufruf

Methodenaufrufe erfolgen über das externe Interface und Methoden werden in der API-Referenzdokumentation [2] beschrieben.
Die Methode *AddToQueueWithSingleBoxCoords* dient der Hinzufügung des Auftrags und des Rechtecks, das der Zielbox in einem Aufruf entspricht.
AddTrayConfig dient der Hinzufügung des Full-Layouts eines Trays. Der Aufruf dieser Methode ist direkt vor einem Aufruf von *AddToQueue* möglich, um die Konfiguration des Zieltrays hinzuzufügen, oder sie kann beim Starten mehrere Male aufgerufen werden, um alle Konfigurationen auf einmal hinzuzufügen.
Zur Wiederherstellung der Konfiguration bei einem Neustart von Compact Talk mit unabgeschlossenen Aufträgen in der Warteschleife wird die Konfiguration in einer Cache-Datei gespeichert und von dort beim Starten geladen. Durch den Cache ist ein weiteres Hinzufügen der Tray-Konfiguration in Compact Talk nur nach einer Änderung erforderlich.

7.2 Import von Auftrag

Das Hinzufügen einer Single-Box-Konfiguration über die Methode zum Import von einem Auftrag wird in Abschnitt 11 und im Konfigurationshandbuch [1] beschrieben.

7.3 Import von Tray

Das Hinzufügen einer vollen Tray-Konfiguration über Tray-Import wird in Abschnitt 11.3 beschrieben.

8 CTCLIENT-API

Die Client-API [2] deckt die Bereiche Verbindungsmanagement, Fehlermanagement und synchronisierte Eventabwicklung ab. Ihre Implementierung erfolgt als NET-Assembly Weland.CompactTalk.Client.dll.
Der Zweck der Client-API ist die Vereinfachung der Integration für eine Implementation durch Dritte. Die Hauptklasse der API ist *CTConnection*.

8.1 CTConnection

Eigenschaften:

- **Command (CommandProxy)**
  Eine Referenz zum Proxy, die eine Verbindung zum Befehlsinterface des Service darstellt.
- **IsConnected (bool)**
  „True“, wenn eine Verbindung hergestellt wurde, andernfalls „false“.

Methoden:

- **void Connect(string host, int commandPort, int eventPort)**
  Verbindung mit dem Server über TCP/IP mit gegebenem Host und Portnummern.

- **void Connect(string host)**
  Verbindung mit dem Server über TCP/IP mit gegebenem Host und mit den standardmäßigen Portnummern.
- **void Connect()**
  Stellt eine lokale Verbindung mit dem Service über eine Named Pipe her.
- **void Disconnect()**
  Trennt vom Service.

Events:

- **event ClientQueueChanged OnQueueChanged**
  Wird ausgelöst bei Hinzufügen oder Entfernen eines Auftrags an der Warteschleife.
- **event ClientOrderStatusChanged OnOrderStatusChanged**
  Wird ausgelöst bei Änderung des Status an einem Auftrag.
- **event ClientServiceStateChanged OnServiceStateChanged**
  Wird ausgelöst bei Änderung des Zustands eines Service.
- **event ClientAckRequest OnClientAckRequest**
  Wird ausgelöst bei Aktivierung von externer Bestätigung (konfigurierbar) und bei Erreichen des Status TaskDone an einem Auftrag. Client muss mit einem Aufruf der Methode ConfirmAckRequest antworten oder der Auftrag bleibt in der Warteschleife.
- **event ClientOrderPriorityChanged OnOrderPriorityChanged**
  Wird ausgelöst bei Änderung der Priorität an einem Auftrag.
- **event ClientOpeningModeChanged OnOpeningModeChanged**
  Wird ausgelöst bei Änderung des Modus an einer spezifischen Öffnung.
- **event ClientOpeningUserChanged OnOpeningUserChanged**
  Wird ausgelöst bei Änderung des angemeldeten Benutzers an einer spezifischen Öffnung.
- **event ClientTrayWeightChanged OnTrayWeightChanged**
  Wird ausgelöst bei Änderung des Gewichts an einem spezifischen Tray.
- **event ClientTrayHeightChanged OnTrayHeightChanged**
  Wird ausgelöst bei Änderung der Höhe an einem spezifischen Tray.
- **event ClientOrderReturned OnOrderReturned**
  Wird ausgelöst bei Erreichen von TaskDone an einem Auftrag und bei Rücksendung des Trays zu seiner Position im Elevator. Wenn mehrere Aufträge durch denselben Tray ausgelagert werden, wird für jeden Auftrag ein Event gesendet. Das Event enthält die OrderID.
- **event ClientStopCodesChanged OnStopCodesChanged**
  Wird ausgelöst bei Änderung der Liste an StopCodes. Für weitere Informationen siehe Abschnitt 8.2.

8.2 StopCodes

8.2.1 Allgemeine Informationen

Stopp-Codes in Compact Talk bezieht sich auf eine Historienliste mit den letzten zehn StopCodes. Der letzte StopCode steht immer an erster Stelle, der älteste am Ende der Liste. Das Event OnStopCodesChanged wird ausgelöst bei Änderung der Liste an StopCodes. Das Event enthält die neue Liste mit Informationen zu StopCodes, List<StopCode>.

StopCode-Klasse:

```
public class StopCode
{
    //See stop code definition list for valid codes
    public int Code { get; set; }

    //Describes the cause of the stop
    public string Cause { get; set; }
}
```

8.2.2 StopCodes

Ursache für Stopp	**StopCode**
Änderung von Zustand	1
Änderung von Sub-Zustand	2
Maschinenneustart	3
Sequenzreset	4
E-Stopp	1000
Lichtschranke	1001
PIT-Sensor blockiert	1010
PIT-Sensor blockiert, von Öffnung holen	1011
PIT-Sensor blockiert, von Lager holen	1012
Bremsschütz 1 Fehler	1020
Bremsschütz 2 Fehler	1021
Bremsrelais 1 Fehler	1022
Bremsrelais 2 Fehler	1023
Verlust Encoder-Modul, obere Ebene	1030
Verlust Encoder-Modul, untere Ebene	1031
Umrichterposition geändert bei Reboot	1032
Referenzposition an Umrichter verloren	1033
Umrichter-Fehlercode aktiv	1034
Referenzsuche fehlgeschlagen, Elevatorebene	1040

Ungültige Position, obere Ebene	1041
Ungültige Position, untere Ebene	1042
Elevator an unterem Endschalter	1050
Elevator an oberem Endschalter	1051
Positionsfehler Elevator, nicht in Tray-Lagerposition	1060
Positionsfehler Elevator, nicht an Öffnung	1061
Positionsfehler Elevator, nicht in Unhook-Intervall	1062
Positionsfehler Elevator, nicht an Unhook-Position	1063
Positionsfehler Elevator, nicht an Zwischenlager	1064
Positionsfehler Elevator, nicht in Unhook-Intervall von Zwischenlager	1065
Positionsfehler Elevator, nicht an Unhook von Zwischenlager	1066
Positionsfehler, obere Ebene	1070
Positionsfehler, untere Ebene	1071
Fehler Gewichtsmessung	1080
Tray zurückgesendet zu Öffnung, zu schwer	1081
Fehler Höhenmessung	1090
Tray zu hoch für eingeschränkte Höhe an Öffnung	1091
Tray zurückgesendet zu Öffnung, zu hoch	1092
Fehler beim Holen von Tray an Öffnung, Ghost	1100
Fehler beim Holen von Tray an Lager, Ghost	1101
Ein Tray befindet sich bereits an Öffnung, Bewegen von Tray zu Öffnung nicht möglich	1110
Ein Tray befindet sich bereits an Öffnung, Twin-Umschaltung, Bewegen von Tray zu Öffnung nicht möglich	1111
Elevator-Bewegung durch Ebenenposition blockiert, nicht frei	1120
Untere Ebene nicht an Sensor, Holen von Öffnungs-Twin ohne Extraktor nicht möglich	1121
Obere Ebene nicht an Sensor, Holen von Öffnungs-Twin ohne Extraktor nicht möglich	1122
Keine Ebene an Auslagerposition, Holen von Öffnungs-Twin ohne Extraktor nicht möglich	1123
Beide Ebenen nicht an Sensor, Holen von Lager nicht möglich	1124
Ebene nicht an Sensor, Holen von Lager nicht möglich	1125
Bewegung von Elevator durch Extraktor oder Tür blockiert	1126
Ebene nicht an Sensor, Laden von Tray nicht möglich	1127
Ebene nicht an Sensor, Entladen von Tray nicht möglich	1128

Tray in Auslagerposition entspricht nicht Tray auf oberer Ebene oder Schritt	1130
Tray in Auslagerposition entspricht nicht Tray auf unterer Ebene oder Schritt	1131
Tray auf unterer Ebene entspricht nicht NextTray	1132
Tray auf oberer Ebene entspricht nicht NextTray	1133
Erreichen von Tray von beliebiger Ebene nicht möglich	1140
Erreichen von Tray nicht möglich	1141
Keine Lagerposition gesetzt für angefordertes Tray	1142
Ungültige Trayposition	1150
Ungültige Tray-Zwischenposition	1151
Ungültige Trayposition, nächste Sequenz holen, Reset	1152
Ungültige Tray-Unhook-Position	1153
Ungültige Tray-Unhook-Position, nächste Sequenz holen, Reset	1154
Ungültige Trayposition, Twin-Umschaltung an Lager nicht möglich	1155
Ungültige Tray-Unhook-position, Twin-Umschaltung an Lager nicht möglich	1156
Ungültige Tray-Unhook-Position, Zwischenlager	1157
Ungültige Tray-Unhook-Position, Sequenz holen, Reset	1158
Ungültige Trayposition, Sequenz holen, Reset	1159
Angefordertes Tray blockiert	1160
Benutzer besitzt keine Zugriffsrechte auf angefordertes Tray	1161
Benutzer besitzt keine Zugriffsrechte auf angefordertes NextTray	1162
Kein Zwischenlager gefunden	1170
Lager blockiert durch Zwischentray	1171
Kein Lager zum Rücksenden von Zwischentray, Optimierung	1172
Umrichterbewegung technisch blockiert	1180
Luftstrom (ATEX)	2000
Gaswarnung (ATEX)	2001
Gasalarm (ATEX)	2002
Feueralarm	2010
Sprinklerbereich	2011
Tray-Leihtrolley nicht entfernt	2020
Tray-Leihtrolley nicht in Position	2021
Tray bereits auf Tray-Leihtrolley	2022
Kein Tray auf Tray-Leihtrolley	2023
Angefordertes Tray geliehen	2024
Angefordertes Tray nicht geliehen	2025

Lasttraverse nicht installiert	2030
Tray nicht korrekt an Öffnungssensor positioniert	2040
Fehler beim Holen von Tray, Bedienstation	2050
Fehler Traypositionierung, Bedienstation	2051
Positionsfehler Elevator an Bedienstation	2052
Tray bereits in Bedienstation	2053
Ebene nicht an Sensor, Holen von Bedienstation nicht möglich	2054
Kein Tray in Bedienstation	2055
Fehler beim Holen von Tray, Transferstation	2060
Fehler Traypositionierung, Transferstation	2061
Positionsfehler Elevator an Transferstation	2062
Tray bereits in Transferstation	2063
Kette und Teleskop nicht in innerer Position	2064
Angefordertes Tray nicht geliehen	2065
Kein Tray in Transferstation	2066
Tray in Transferstation nicht korrekt positioniert	2067
Kein Lager zum Transfer von Tray gefunden	2068
Ebene nicht an Sensor, Holen von Transferstation nicht möglich	2069
USV, an Batterie	2080
Trayhalterung nicht verriegelt	2090
Seitentisch, Auftragsdiskrepanz	2100
Seitentisch, Bewegung Kette zu Elev. nicht erlaubt	2101
Seitentisch, Bewegung Kette zu Auslagerung nicht erlaubt	2102
Seitentisch, Bewegung Schraube zu Elev. nicht erlaubt	2103
Seitentisch, Bewegung Schraube zu Auslagerung nicht erlaubt	2104
Seitentisch, Bewegung Schraube zu Service nicht erlaubt	2105
Maschinenbereich nicht freigegeben	2120

8.2 Code-Samples

Alle Code-Samples sind im SDK-Release von Compact Talk verfügbar.
Zum Aufbau einer Client-Anwendung ist eine Referenzierung dieser Assemblies erforderlich:

- Weland.CompactTalk.Framework.dll
- Weland.CompactTalk.Client.dll
- System.ServiceModel

Diese Samples umfassen kein Fehler-Handling.

8.2.1 Sample 1

Dies ist eine einfache Implementierung zur Einbindung und zum Abruf der Version des Service Compact Talk.

```
CTConnection connection = new CTConnection();
connection.Connect();
Version version = connection.Command.Version;
connection.Disconnect();
```

8.2.2 Sample 2

Das zweite Sample ist ein etwas umfangreicherer Client, der den Service um einen Auslagerauftrag ergänzt und die Statusänderungen während der Durchführung druckt.

```
using System.Text;
using Weland.CompactTalk.Client;
using Weland.CompactTalk.Framework.OrderManagement;
using Weland.CompactTalk;

namespace SimpleClient
{
    class Program
    {
        static void Main(string[] args)
        {
            //Create an instance of the CTConnect type.
            CTConnection con = new CTConnection(null);

            //Connect to service.
            con.Connect();
            //Hook up the order status event.
            con.OnOrderStatusChanged += new ClientOrderStatusChanged(OnClientOrderStatusChanged);
            Console.WriteLine("Adding an order to Compact Talk service");
            //Add an order to the service
            con.Command.AddToQueue("SimpleClient:1", "Sim_1", 1, "", 1, "", "",
                OrderMode.OUT, 0, 1, "", 100, "", "", "", "", "", "", true);
            //Wait for input to terminate the program.
            Console.ReadKey();
            //Disconnect from the service.
            con.Disconnect();
        }
        //This method is called every time the status is changed on the order.
        static void OnClientOrderStatusChanged(CTOrderStatusChangeEvent evt)
        {
            Console.WriteLine("Status changed on order from " + evt.OldStatus.ToString()
                + " to " + evt.Order.Status.ToString());
        }
    }
}
```

Die Ausgabe der Konsole entspricht Abbildung 5.

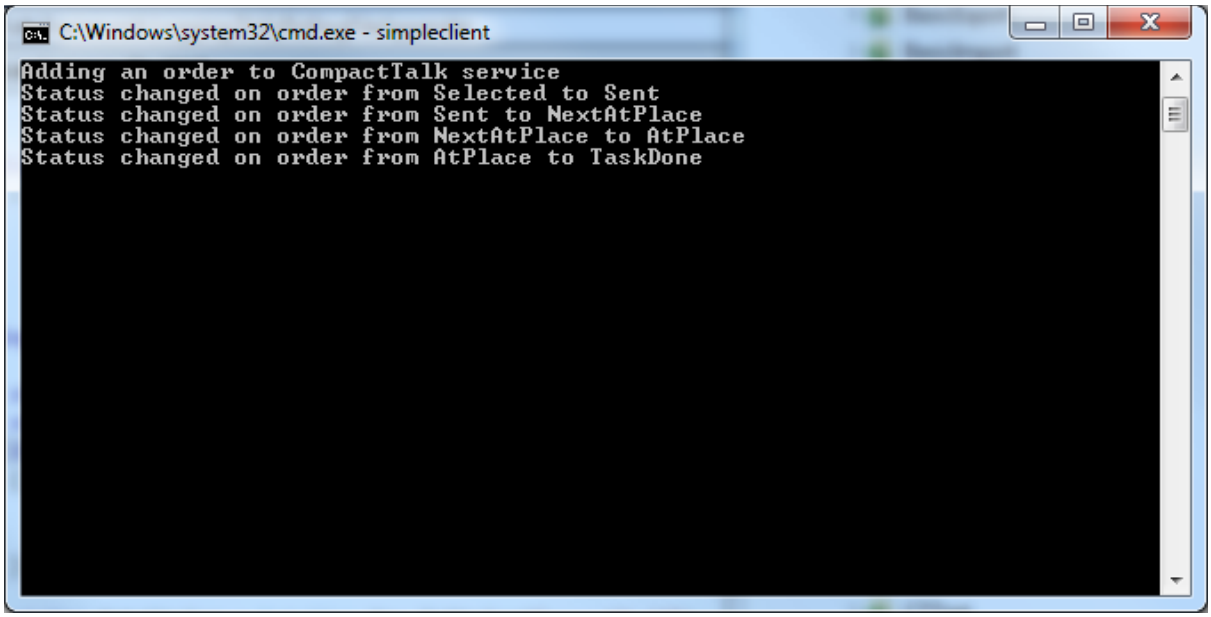


Abbildung 5, Ausgabe der Konsole von Sample 1

8.2.3 Ein umfangreicheres Sample, Mini WMS

Dies ist eine umfangreichere Sample-Anwendung zur Abdeckung der meisten Fälle. Diese einfache WMS-Anwendung ermöglicht ein Erstellen von Auslageraufträgen auf Basis einer Liste an verfügbaren Artikelaufzeichnungen.
Die dauerhafte Speicherung der Auftragsliste erfolgt in einer Datei anstatt in einer Datenbank.

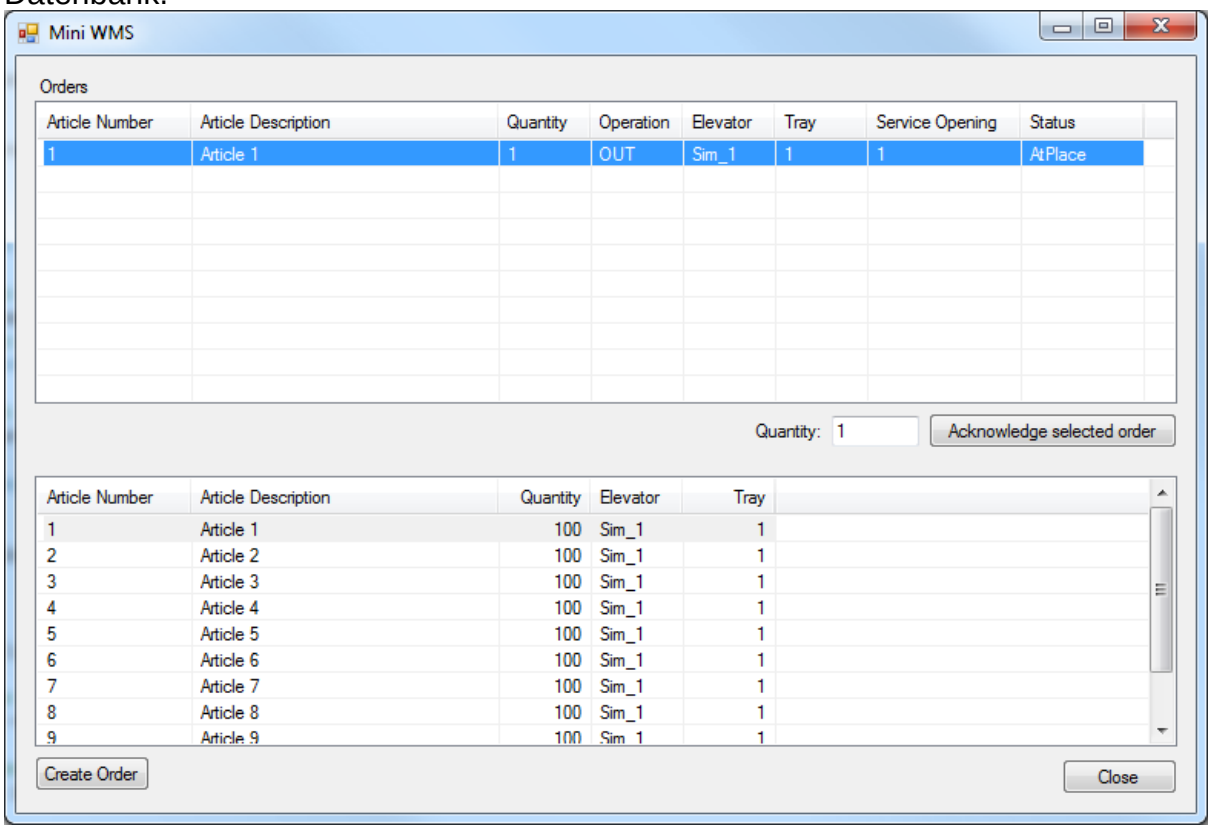


Abbildung 6, Hauptfenster MiniWMS

Das Hauptfenster der Anwendung umfasst eine Auftragsliste, eine Artikelliste, eine Schaltfläche zur Erstellung eines Auslagerauftrags und eine Schaltfläche zur externen Quittierung eines Auftrags.

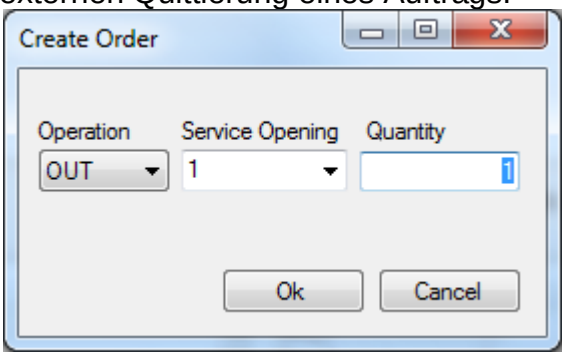


Abbildung 7, Dialog zur Eingabe von Informationen zu Auslagerauftrag

Der Dialog „Create Order“ dient zur Eingabe von Informationen zur Erstellung des Auslagerauftrags.
Die komplette Code-Liste des Hauptformulars der Anwendung ist unten verfügbar. Ein großer Anteil an Code bezieht sich zwar auf die Darstellung von Informationen in der Liste, die Kommentare sollten allerdings eine Isolierung der funktionellen Bits ermöglichen.

```
public partial class MainForm : Form
    {
        //List of orders sent to Compact Talk
        List<OrderRecord> m_MyOrders = new List<OrderRecord>();
        //Index of orders sent to Compact Talk
        Dictionary<int, OrderRecord> m_MyOrdersByCTId = new Dictionary<int, OrderRecord>();
        //List of article records
        List<ArticleRecord> m_MyArticleRecords = new List<ArticleRecord>();
        //Index of articles
        Dictionary<int, ArticleRecord> m_MyArticleRecordsById = new Dictionary<int, ArticleRecord>();
        //Client api connection
        CTConnection m_CompactTalk;
        public MainForm()
        {
            InitializeComponent();
            //Disable acknowledge button until we have a selected row in order view
            buttonExtAck.Enabled = false;
            //Generate 10 articles and add them to the list and index
            for (int i = 1; i <= 10; i++)
            {
                ArticleRecord ar = new ArticleRecord();
                ar.ArticleNo = i.ToString();
                ar.ArticleDesc = "Article " + i.ToString();
                ar.Elevator = "Sim_1";
                ar.Id = i;
                ar.Quantity = 100;
                ar.TrayNo = 1;

                m_MyArticleRecords.Add(ar);
                m_MyArticleRecordsById.Add(ar.Id, ar);
            }
            //Populate article view with the available article records
            for (int i = 0; i < m_MyArticleRecords.Count; i++)
            {
                ListViewItem lvItem = listViewArticles.Items.Add(m_MyArticleRecords[i].ArticleNo);
                lvItem.SubItems.Add(m_MyArticleRecords[i].ArticleDesc);
                lvItem.SubItems.Add(m_MyArticleRecords[i].Quantity.ToString());
                lvItem.SubItems.Add(m_MyArticleRecords[i].Elevator);
                lvItem.SubItems.Add(m_MyArticleRecords[i].TrayNo.ToString());
                lvItem.Tag = m_MyArticleRecords[i].Id;
            }
```

```
    //Load our orders from persistent storage
    LoadOrders();
    //Connect to Compact Talk
    try
    {
        m_CompactTalk = new CTConnection(this);
        m_CompactTalk.OnOrderStatusChanged += new
        ClientOrderStatusChanged(OnOrderStatusChangedEvent);
        m_CompactTalk.OnQueueChanged += new ClientQueueChanged(OnQueueChangedEvent);
        m_CompactTalk.Connect();
    }
    catch (Exception e)
    {
        MessageBox.Show("Failed to connect to Compact Talk");
        throw e;
    }
    //Do a recovery in case orders been acknowledge while this application was shut down
    SynchronizeOrders();
}
void LoadOrders()
{
    //Deserialize order list from file storage
    using (Stream file = File.Open("MyOrders.dat", FileMode.OpenOrCreate))
    {
        BinaryFormatter bformatter = new BinaryFormatter();
        if (file.Length > 0)
            m_MyOrders = (List<OrderRecord>)bformatter.Deserialize(file);
    }
    //Populate order view with available orders
    for (int i = 0; i < m_MyOrders.Count; i++)
    {
        ListViewItem lvItem = listViewOrders.Items.Add(m_MyOrders[i].ArticleNo);
        lvItem.SubItems.Add(m_MyOrders[i].ArticleDesc);
        lvItem.SubItems.Add(m_MyOrders[i].Quantity.ToString());
        lvItem.SubItems.Add(m_MyOrders[i].Operation);
        lvItem.SubItems.Add(m_MyOrders[i].Elevator);
        lvItem.SubItems.Add(m_MyOrders[i].TrayNo.ToString());
        lvItem.SubItems.Add(m_MyOrders[i].ServiceOpening.ToString());
        lvItem.SubItems.Add(m_MyOrders[i].CTStatus);
        lvItem.Tag = m_MyOrders[i].CTOrderId;
        //Add the order to the order index
        m_MyOrdersByCTId.Add(m_MyOrders[i].CTOrderId, m_MyOrders[i]);
    }
}
void SaveOrders()
{
    //Serialize order list to file storage
    using (Stream orderFile = File.Open("MyOrders.dat", FileMode.OpenOrCreate))
    {
        BinaryFormatter bformatter = new BinaryFormatter();
        bformatter.Serialize(orderFile, m_MyOrders);
    }
}
void SynchronizeOrders()
{
    //For each order in our order list, check to se if it matches Compact Talks order
    for (int i = 0; i < m_MyOrders.Count; i++)
    {
        PickOrder ctOrder = m_CompactTalk.Command.GetOrder(m_MyOrders[i].CTOrderId);
        if (ctOrder == null)
        {
            //Compakt Talk has never seen an order with this id. This should never happen.
            continue;
        }
        //The order has been acknowledged while we where shut down and now only exist in
        historical storage
        if (ctOrder.Status == OrderStatus.Historical)
        {
            //Do stock management based on the OrderMode
            if (ctOrder.Mode == OrderMode.OUT)
```

```
                        m_MyArticleRecordsById[m_MyOrders[i].ArticleId].Quantity
                        -= ctOrder.AckQuantity;
                    else if (ctOrder.Mode == OrderMode.IN)
                        m_MyArticleRecordsById[m_MyOrders[i].ArticleId].Quantity += ctOrder.AckQuantit
y;
                    else if (ctOrder.Mode == OrderMode.INV)
                        m_MyArticleRecordsById[m_MyOrders[i].ArticleId].Quantity = ctOrder.AckQuantity
;
                    //Locate the article in the article view and update quantity so we doesn't present
 stale data
                    for (int j = 0; j < listViewArticles.Items.Count; j++)
                    {
                        if (m_MyOrders[i].ArticleId == (int)listViewArticles.Items[j].Tag)
                        {
                            listViewArticles.Items[j].SubItems[2].Text = m_MyArticleRecordsById[m_MyOr
ders[i].ArticleId].Quantity.ToString();
                            break;
                        }
                    }
                    //Locate the order in the order view and remove it
                    for (int j = 0; j < listViewOrders.Items.Count; j++)
                    {
                        if (m_MyOrders[i].CTOrderId == (int)listViewOrders.Items[j].Tag)
                        {
                            listViewOrders.Items.RemoveAt(j);
                            break;
                        }
                    }
                    //Remove the order from the order list and order index
                    m_MyOrdersByCTId.Remove(m_MyOrders[i].CTOrderId);
                    m_MyOrders.Remove(m_MyOrders[i]);

                    i -= 1;
                }
                else
                {
                    //Update the status of the order in the order view and in the order list
                    listViewOrders.Items[i].SubItems[7].Text = ctOrder.Status.ToString();
                    m_MyOrders[i].CTStatus = ctOrder.Status.ToString();
                }
            }
            //Save changes to the order list in the storage file
            SaveOrders();
        }
        void OnOrderStatusChangedEvent(CTOrderStatusChangeEvent evt)
        {
            //Check to see if it's one of our orders we received the event for, if not just ignore the
            event
            if (!m_MyOrdersByCTId.ContainsKey(evt.Order.Id))
                return;
            OrderRecord order = m_MyOrdersByCTId[evt.Order.Id];
            //Search for the order in the order view
            for (int i = 0; i < listViewOrders.Items.Count; i++)
            {
                if (order.CTOrderId == (int)listViewOrders.Items[i].Tag)
                {
                    //Update the order in the order view and in the order list with the new status
                    listViewOrders.Items[i].SubItems[7].Text = evt.Order.Status.ToString();
                    order.CTStatus = evt.Order.Status.ToString();

                    //If the new status is TaskDone
                    if (evt.Order.Status == OrderStatus.TaskDone)
                    {
                        //Do stock management based on the OrderMode
                        if (evt.Order.Mode == OrderMode.OUT)
                            m_MyArticleRecordsById[order.ArticleId].Quantity -= evt.Order.AckQuantity;
                        else if (evt.Order.Mode == OrderMode.IN)
                            m_MyArticleRecordsById[order.ArticleId].Quantity += evt.Order.AckQuantity;
                        else if (evt.Order.Mode == OrderMode.INV)
                            m_MyArticleRecordsById[order.ArticleId].Quantity = evt.Order.AckQuantity;
```

```
                //Locate the article in the article view and update quantity so we doesn't
                present stale data
                for (int j = 0; j < listViewArticles.Items.Count; j++)
                {
                    if (order.ArticleId == (int)listViewArticles.Items[j].Tag)
                    {
                        listViewArticles.Items[j].SubItems[2].Text = m_MyArticleRecordsById[
                        order.ArticleId].Quantity.ToString();
                    }
                }
            }
            //Save changes to the order list in the storage file
            SaveOrders();
            break;
        }
    }
}
void OnQueueChangedEvent(CTQueueChangeEvent evt)
{
    //Check to see if it's one of our orders we received the event for, if not just ignore the
    event
    if (!m_MyOrdersByCTId.ContainsKey(evt.Order.Id))
        return;
    OrderRecord order = m_MyOrdersByCTId[evt.Order.Id];
    //If the queue chane type is OrderDeleted
    if (evt.ChangeType == OrderQueueChangeType.OrderDeleted)
    {
        //Search the order view for the order
        for (int i = 0; i < listViewOrders.Items.Count; i++)
        {
            if (order.CTOrderId == (int)listViewOrders.Items[i].Tag)
            {
                //Remove the order from the order view, the order list and the order index
                listViewOrders.Items.RemoveAt(i);
                m_MyOrdersByCTId.Remove(order.CTOrderId);
                for (int j = 0; j < m_MyOrders.Count; j++)
                {
                    if (m_MyOrders[j].CTOrderId == order.Id)
                    {
                        m_MyOrders.RemoveAt(j);
                        break;
                    }
                }
                //Save changes to the order list in the storage file
                SaveOrders();
                break;
            }
        }
    }
}
private void buttonCreateOrder_Click(object sender, EventArgs e)
{
    CreateOrderDlg dlg = new CreateOrderDlg();
    //Open the CreateOrderDlg to enter the pick order information
    if (dlg.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        ArticleRecord ar = m_MyArticleRecordsById[(int)listViewArticles.SelectedItems[0].Tag];
        //Create an order record
        OrderRecord order = new OrderRecord();
        order.ArticleId = ar.Id;
        order.ArticleNo = ar.ArticleNo;
        order.ArticleDesc = ar.ArticleDesc;
        order.Elevator = ar.Elevator;
        order.TrayNo = ar.TrayNo;
        order.Quantity = dlg.Quantity;
        order.ServiceOpening = dlg.ServiceOpening;
        order.Operation = dlg.Operation;
        try
        {
            //Call Compact Talk to add the order to the queue
```

```
            int orderId = m_CompactTalk.Command.AddToQueue(
                "MiniWMS:" + order.Id,
                order.Elevator,
                order.TrayNo,
                "",
                order.ServiceOpening,
                order.ArticleNo,
                order.ArticleDesc,
                (OrderMode)Enum.Parse(typeof(OrderMode), order.Operation),
                0,
                1,
                "",
                order.Quantity,
                "",
                "",
                "",
                "",
                "",
                "",
                "",
                true);
            order.CTOrderId = orderId;
            order.Id = orderId;
            order.CTStatus = "Selected";
        }
        catch (Exception ex)
        {
            MessageBox.Show("Failed to add order to Compact Talk\n\n" + ex.Message);
            return;
        }
        //Add the order to the order list and index
        m_MyOrders.Add(order);
        m_MyOrdersByCTId.Add(order.Id, order);
        //Save changes to the order list in the storage file
        SaveOrders();
        //Add a new ro to the order view
        ListViewItem lvItem = listViewOrders.Items.Add(order.ArticleNo);
        lvItem.SubItems.Add(order.ArticleDesc);
        lvItem.SubItems.Add(order.Quantity.ToString());
        lvItem.SubItems.Add(order.Operation);
        lvItem.SubItems.Add(order.Elevator);
        lvItem.SubItems.Add(order.TrayNo.ToString());
        lvItem.SubItems.Add(order.ServiceOpening.ToString());
        lvItem.SubItems.Add(order.CTStatus);
        lvItem.Tag = order.CTOrderId;
    }
}
private void buttonClose_Click(object sender, EventArgs e)
{
    //Just close the window so the application can exit
    Close();
}
private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
{
    //Disconnect from Compact Talk
    m_CompactTalk.Disconnect();
}
private void listViewOrders_SelectedIndexChanged(object sender, EventArgs e)
{
    //If no order selected in the order view, disable the ack button and empty the quantity
    field
    if (listViewOrders.SelectedIndices.Count < 1)
    {
        buttonExtAck.Enabled = false;
        textBoxQuantity.Text = "";
        return;
    }

    //If an order is selected in the order view, enable the ack button and
    //fill the quantity field with the quantity of the order
    buttonExtAck.Enabled = true;
```

```
            int orderId = (int)listViewOrders.SelectedItems[0].Tag;
            textBoxQuantity.Text = m_MyOrdersByCTId[orderId].Quantity.ToString();
        }
        private void buttonExtAck_Click(object sender, EventArgs e)
        {
            int orderId = (int)listViewOrders.SelectedItems[0].Tag;
            OrderRecord selectedOrder = m_MyOrdersByCTId[orderId];
            try
            {
                //Call Compact Talks ExtAckOrder method
                m_CompactTalk.Command.ExtAckOrder(
                    selectedOrder.Elevator,
                    selectedOrder.ServiceOpening,
                    float.Parse(textBoxQuantity.Text),
                    false
                    );
            }
            catch (Exception ex)
            {
                MessageBox.Show("Failed to acknowledge order. Error: " + ex.Message);
            }
        }
    }
```

Das Sample Mini WMS nutzt eine Datei zur Speicherung der Auftragsliste anstatt eine Datenbank. zur Synchronisierung von Aufträge während der Wiederherstellung müssen die Aufträge dauerhaft extern gespeichert werden.

8.2.4 Beispiel für Tray-Zugriff

Dieses Beispiel beschreibt einen Abruf von Elevator-Informationen und der Trays in diesem Elevator.

```
using System;
using Weland.CompactTalk;
using Weland.CompactTalk.Client;
using Weland.CompactTalk.Framework.Devices;
using Weland.CompactTalk.Framework.OrderManagement;
namespace SimpleClient
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            //Create an instance of the CTConnect type.
            CTConnection con = new CTConnection(null);
            //Connect to service a storageAddress (other then localhost)
            con.Connect(storageAddress);
            if (con.Connected)
            {
                //Hook up the order status event.
                con.OnOrderStatusChanged += new
                ClientOrderStatusChanged(OnClientOrderStatusChanged);
                Console.WriteLine("Adding an order to CompactTalk service");
                //Add an order to the service
                con.Command.AddToQueue("SimpleClient:1", "Sim_1", 1, "", 1, "", "",
                    OrderMode.OUT, 0, 1, "", 100, "", "", "", "", "", "", true);
                // Get Elevator Information position 0
                ElevatorInfo elevatorInfo = con.Command.GetElevatorInfo()[0];
                // Get all Trays from that Elevator Id
                Tray[] trays = con.Command.GetTrays(elevatorInfo.Id);
                //Wait for input to terminate the program.
                Console.ReadKey();
                //Disconnect from the service.
                con.Disconnect();
            }
        }
        //This method is called every time the status is changed on the order.
        private static void OnClientOrderStatusChanged(CTOrderStatusChangeEvent evt)
        {
            Console.WriteLine("Status changed on order from " +
            evt.OldStatus.ToString()
            + " to " + evt.Order.Status.ToString());
        }
    }
}
```

9 Webservice

Compact Talk verfügt über einen integrierten Webservice, der mit den meisten Client-Plattformen kompatibel ist. Compact Talk überwacht am Endpunkt http://<hostname>:20012/CommandConnection.
WSDL-Informationen sind unter der folgenden URL verfügbar: http://<hostname>:20012/CommandConnection?wsdl.
Der Webservice unterstützt SOAP 1.1 und folgt WS-I BP 1.1. Eine Installation von IIS ist nicht erforderlich.
Dieses Interface erfordert einen manuellen Aufruf für Events durch die Methoden ActivateEvents, ReActivateEvents, PollForEvent und PollForEvents.

9.1 Konvertierung von Client CT 1.x zu 2.x

Die Hauptänderungen im neuen Interface sind, dass Transaktionen nun Aufträge, bzw. „orders“ genannt werden, und durch den Typ *PickOrder* dargestellt werden. DeviceID ist nun CompatibilityID. Die im neuen Interface genutzte ID ist war „Name“ im alten Interface.
Das Event-Handlung wurde auch in ein Modell geändert, in dem die Erstellung einer Event-Warteschleife für Events explizit aktiviert werden muss.

9.1.1 Neues Event-Handling

Zur Erstellung von Events durch Compact Talk ist die Erstellung einer Eventwarteschleife durch Aufruf von *ctivateEvents* oder *ReActivateEvents*, je nach der Implementierung des Clients oder der Situation, erforderlich.
ActivateEvents erstellt eine Warteschleife und gibt eine eindeutige ID der Warteschleife aus. *ActivateEvents* sollte nur einmal für jede Client-Session aufgerufen werden.
ReActivateEvents reaktiviert oder erstellt eine Warteschleife mit der gegebenen ID. Die erstellte Warteschleife erhält einen „Keep Alive“-Timer, der aktuell auf 15 Minuten gesetzt ist und bei jedem Aufruf einer Abfragemethode zurückgesetzt wird. Nach Ablauf des Timers wird die Warteschleife zerstört, um Memory-Leaks zu vermeiden.
Die Abfrage von Events erfolgt entweder über *PollForEvent* oder *PollForEvents* je nach eindeutiger ID der Warteschleife.

9.1.2 Methoden-Übersetzungstabelle

Die folgende Liste beschreibt verfügbare Tabellenänderungen in Methoden und deren Nutzung.

Alte Methode	Neue Methode	Beschreibung
`AddToQueue`	`[AddTrayConfig] AddToQueue`	Mit dem Methodenaufruf AddToQueue wird die Tray-Konfiguration nicht mehr hinzugefügt.
`AddToQueue + SetTransBoxCoordinates`	`AddToQueueWithSingleBoxCoords`	Diese Kombination von Methoden kommt normalerweise zum Einsatz, wenn Zubehör wie LightBar oder LaserPointer genutzt wird. Die alte Methode ist fehleranfällig.
Init		Im neuen Interface nicht vorhanden

Start	ResumeService("Devices")	Eine bessere Lösung ist eine Konzentration auf individuelle Elevators anstelle von allen. Beispiel: ResumeService(„H1“) Nur der Elevator mit der ID „H1“ wird wieder gestartet.
Stop (bool returnTrays)	PauseService("Devices") [ReturnTrays("Devices")]	Siehe Methode oben.
ClearTransQueue	PauseService("Devices") ReturnTrays("Devices") ClearOrderQueue("Devices") ResumeService(“Devices”)	Siehe Methode oben.
DeleteTransaction	DeleteOrdersByCondition	Die Anwendung dieser Methode ist sehr komplex, da das Format der Bedingung vom Provider des Lagers abhängt. Die entsprechenden Eigenschaften wurden umbenannt, d. h. „ELEVATORNAME“ ist nun „ELEVATOR“.
GetSize	GetOrderCount	
Sort		Wird durch das neue Interface nicht unterstützt.
TransActivate	ActivateOrder	
GetAllTransactions	GetOrders	
GetTransactionAtOpening	GetOrderAtOpening	Diese Methode sollte nicht zur Überwachung des Auftragsflusses genutzt werden. Events, deren Signalstatus sich am Auftrag ändert, sind der richtige Weg.
ConfirmAckTransaction	ConfirmOrderAck	Diese Methode ist nur nützlich, wenn die Bestätigung von Aufträgen in der

		Konfiguration aktiviert wurde, um das alte Verhalten zu emulieren.
ExtAckTransaction	ExtAckOrder	
AreAllDevicesReadyForRunning		Im neuen Interface nicht unterstützt.
SetLoggMask ReadLoggMask	SetLogThreshold	
GetDeviceVersion GetDeviceSignature	GetElevatorInfo GetSpecificElevatorInfo(string elevatorId)	Im neuen Interface wird ein Objekt mit dem Typ ElevatorInfo mit allen Eigenschaften des Elevators abgerufen.
GetMaxTrays	GetMaxTrayCount	
GetTrayStatus	GetTray	GetTray gibt ein Objekt mit dem Typ Tray aus, wenn das Tray existiert, andernfalls nichts.
GetTrayBlockedStatus GetTrayLevel GetTrayWeight GetTrayHeight	GetTray	Im neuen Interface wird ein Objekt mit dem Typ Tray mit allen Eigenschaften des Trays abgerufen.
SetTraySize		Im neuen Interface nicht unterstützt.
GetAllDevices	GetElevatorInfo	
GetDetailedDeviceInfo	GetSpecificElevatorInfo	ElevatorInfo enthält weniger als der alte Typ DeviceInfoDetailed. Siehe Dokumentation von ElevatorInfo in CompactTalkAPI.chm für weitere Informationen.
GetField	GetOrder	Durch Abruf eines Objekts vom Typ PickOrder werden alle Eigenschaften des Auftrags abgerufen.
WaitForEvent		Wird im neuen Interface nicht unterstützt.

Altes Event	Neues Event	Beschreibung
CTAckEventArgs	CTAckRequestEvent	
CTDeviceEventArgs	CTServiceStateChangedEvent	
CTStatusEventArgs	CTOrderStatusChangeEvent	
CTElevatorStatusEventArgs	CTOpeningModeChangedEvent CTOpeningUserChangedEvent	

Interface 10 XML

XML-Interface ermöglicht Aufruf von Methoden über XML-Dateien. Bei Nutzung dieses Interfaces muss durch Compact Talk ein einzelner Auftragsmodus genutzt werden. Das heißt, dass an jedem Elevator und an jeder Öffnung immer nur ein Auftrag akzeptiert wird. Ein Auftrag mit dem Status AtPlace in der Warteschleife wird durch Compact Talk automatisch quittiert, bevor ein neuer Auftrag hinzugefügt wird. Die Dokumentation für dieses Interface beginnt mit einer Übersicht der verfügbaren Methoden und fährt mit weiteren Einzelheiten zu den Methoden und einigen Beispielen zu deren Anwendung fort.

10.1 Übersicht

Das XML-Interface verfügt über die folgenden Befehle/Methoden:

Befehl	Beschreibung
AddToQueue	AddToQueue dient dem Setzen von Aufträgen in Warteschleife zu Compact Talk.
ExtAckOrder	ExtAckOrder dient der externen Quittierung/Bestätigung von Aufträgen. Die entsprechende Funktion kann über AddToQueue in Verbindung mit Tray = 0 genutzt werden.
AddTrayConfig	Muss genutzt werden, wenn ein Zubehör genutzt wird, das das Full-Layout aus dem Tray erfordert. (Ein Beispiel für ein derartiges Zubehör wäre das „PickDisplay“
ResetElevator	Diese Methode dient dem Abbruch aller aktiver Aufträge im Elevator und erzwingt eine Rücksendung aktiver Trays durch den Elevator.

	Die entsprechende Funktion kann über AddToQueue in Verbindung mit Tray = 1000 und einem Parameter für die entsprechende Öffnung genutzt werden.

10.2 Befehle

10.2.1 Methode AddToQueue

AddToQueue erzeugt einen neuen Auftrag in der Warteschleife von Compact Talk. Alle empfangenen Aufträge werden in der Sequenz, in der Sie empfangen wurden, ausgeführt.
Liste verfügbarer Parameter bei Nutzung von AddToQueue

Parameter	**Zwingend**	**Beschreibung**
TransId	Ja/Nein	Wenn Rückmeldungen verwendet werden, ist dieses Element zwingend. Zur Verknüpfung von Befehl und Rückmeldung.
ElevatorId	Ja	ID des Elevators, an den der Auftrag gerichtet ist. Muss der in der Gerätekonfiguration gesetzten ID entsprechen.
Tray	Ja	Die ID des zu holenden Trays. Muss einem existierenden Tray in der Maschine entsprechen. Hinweis! Zwei Traynummern sind reserviert und können für spezielle Funktionen genutzt werden Bei Nutzung von Tray = 0 wird dieses durch Compact Talk als Methode ExtAckOrder behandelt. Bei Nutzung von Tray = 1000 wird dieses durch Compact Talk als Methode ResetElevator behandelt.
Öffnung	Ja	Nummer der Serviceöffnung am Elevator. Kann 1, 2 oder 3 sein, je nach der Anzahl der Öffnungen an der Maschine. Hinweis! Für die Funktion ResetElevator steht die Öffnungsnummer „99“ zur Verfügung.

NoReturnOfTray	Nein	Gibt dem Elevator vor, was mit dem Tray bei Quittierung durch den Bediener am Bedienfeld passieren soll. 0 = Rücksendung von Tray bei Bestätigung von Auftrag an Bedienfeld 1 = Keine Rücksendung von Tray bei Bestätigung von Auftrag an Bedienfeld; Warten auf externe Quittierung von WMS. Wenn kein Wert angegeben wird, wird der Standardwert 1 genutzt.
NextTray	Nein	Gibt die ID des nächsten Trays an (1-n). Muss genutzt werden, wenn die Twin-Funktion erwünscht ist. Der Standardwert ist 0.
ArtNo	Nein	Artikelnummer Angezeigt am Bedienfeld und am PickDisplay-Zubehör zur Anleitung des Bedieners.
ArtDescr	Nein	Artikelbeschreibung Angezeigt am Bedienfeld und am PickDisplay-Zubehör zur Anleitung des Bedieners.
Quantity	Nein	Die Menge des Auftrags. Angezeigt am Bedienfeld und am PickDisplay-Zubehör zur Anleitung des Bedieners.
Zusatzinformationen für Zubehör		
Info1	Nein	Zusätzliche Auftragsinformationen angezeigt am PickDisplay.
Info2	Nein	Zusätzliche Auftragsinformationen angezeigt am PickDisplay.
Info3	Nein	Zusätzliche Auftragsinformationen angezeigt am PickDisplay.
Info4	Nein	Zusätzliche Auftragsinformationen angezeigt am PickDisplay.
Info5	Nein	Zusätzliche Auftragsinformationen angezeigt am Bedienfeld und PickDisplay.
CurrentBoxName	Ja, wenn PickDisplay, andernfalls Nein	Position an Tray. Genutzt von PickDisplay.
Modus	Ja, wenn PickDisplay, andernfalls Nein	Dieser Wert hat keine logische Funktion in Compact Talk. Allerdings wird dies genutzt, um dem Bediener den durchzuführenden Vorgang zu zeigen. Je nach dem Wert wird dem Bediener

		am PickDisplay ein Symbol angezeigt. Verfügbare Modi IN = Einlagerung, OUT = Auslagerung, INV = Inventar.
TrayCoord	Nein	Information, die am Bedienfeld angezeigt werden kann.
Job	Nein	Name des Jobs, zu dem der Auftrag gehört. Angezeigt am PickDisplay zur Anleitung des Bedieners.
XPos	Ja, wenn „LaserPointer/LightBar“ genutzt werden sollen, andernfalls Nein.	X-Position der Box Bei Nutzung von AddTrayConfig ist dies nicht erforderlich.
YPos	Ja mit LaserPointer, andernfalls Nein.	Y-Position der Box Bei Nutzung von AddTrayConfig ist dies nicht erforderlich.
XSize	Ja mit LightBar, andernfalls Nein.	Breite der Box Bei Nutzung von AddTrayConfig ist dies nicht erforderlich.
Para1	Ja mit LightBar, andernfalls Nein.	Bei Nutzung von LightBar sollte dieser Wert auf die Anzeige im Y-Digit-Display gesetzt werden Bei Nutzung von AddTrayConfig ist dies nicht erforderlich.

Beispiele für AddToQueue
Dieser Abschnitt zeigt einige Beispiele zum Aufruf der Methoden über das XML-Interface.

Beispiel 1 - Einfachen Tray holen (kein Twin)
Dieses Beispiel zeigt ein einfaches Holen eines Trays an einem Elevator ohne Zubehör

```
<AddToQueue>
  <ElevatorId>E1</ElevatorId>
  <Tray>1</Tray>
  <Opening>1</Opening>
</AddToQueue>
```

Das obere Beispiel fügt einen Auftrag hinzu: Traynummer 1 zu Öffnung 1 für Elevator E1 holen.

Beim Senden des nächsten Auftrags passiert folgendes.

1. Wenn der nächste Auftrag dieselbe Traynummer nutzt, die bereits in der Öffnung am Elevator vorhanden ist. Das Tray verbleibt an der Öffnung. Trotz

der Quittierung des bestehenden Auftrags.

2. Wenn der nächste Auftrag eine andere Traynummer betrifft, wird der Auftrag quittiert und das Tray entsprechend der beauftragten Nummer geändert.

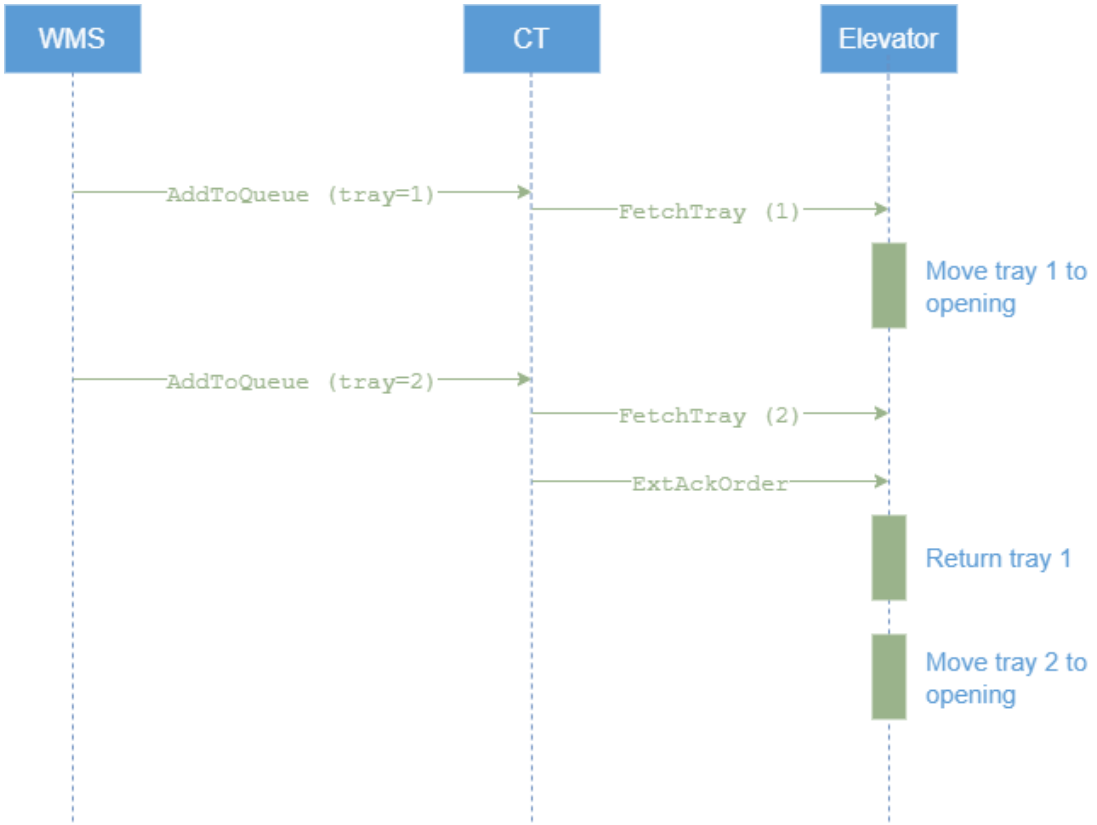


Abbildung 1. Sequenzdiagramm für Elevator ohne Twin-Funktion

Beispiel 2 – AddToQueue mit Twin-Funktion

Um die Maschine mit Twin-Funktion zu nutzen, müssen zwei Traynummern über die Methode AddToQueue gesendet werden. Dies erfolgt gleichzeitig über <Tray> und <NextTray>.

```
<AddToQueue>
  <ElevatorId>E1</ElevatorId>
  <Tray>1</Tray>
  <Opening>1</Opening>
  <NextTray>2</NextTray>
</AddToQueue>
```

Das obere Beispiel zeigt das Hinzufügen eines Auftrags zum Holen von Traynummer 1 zu Öffnung 1 an Elevator E1. Außerdem wird die Maschine angewiesen, Traynummer 2 für den Twin-Modus zu holen. Die Maschine wartet anschließend auf Auftrag von WMS.

Szenario 1 – Normaler Twin

Wenn der nächste Auftrag dieselbe Traynummer betrifft, die zuvor im Parameter <NextTray> gesendet wurde, wechselt die Maschine auf diesen Tray (Normaler Twin).

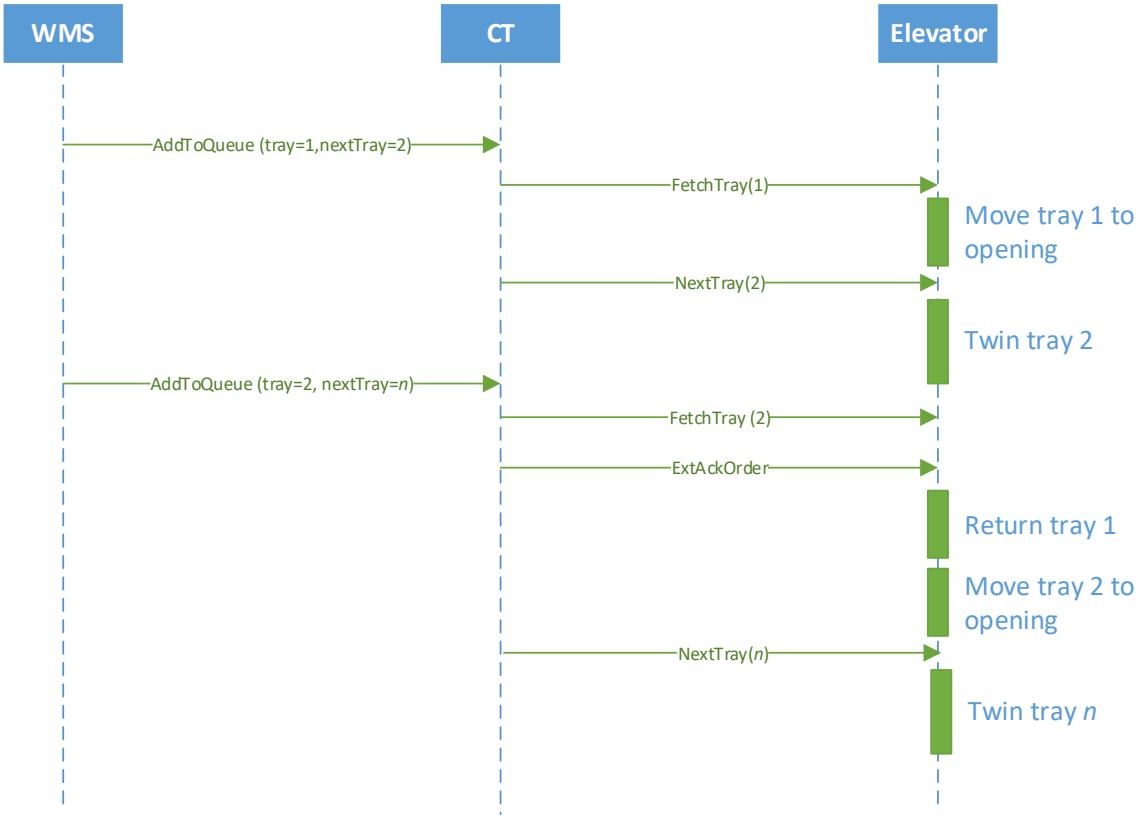


Abbildung 2. Sequenzdiagramm für Vorgang für normalen Twin

Szenario 2 – Erwarteter Twin nicht gewünscht

Wenn die nächste Traynummer einer anderen als der vorherigen entspricht, wird <NextTray> gesendet. Die Tray in Warteposition wird durch die Maschine zurückgesendet und die neu bestellte Traynummer wird geholt. Die Maschine wechselt dann auf die neue Traynummer. Siehe Sequenzbeschreibung.

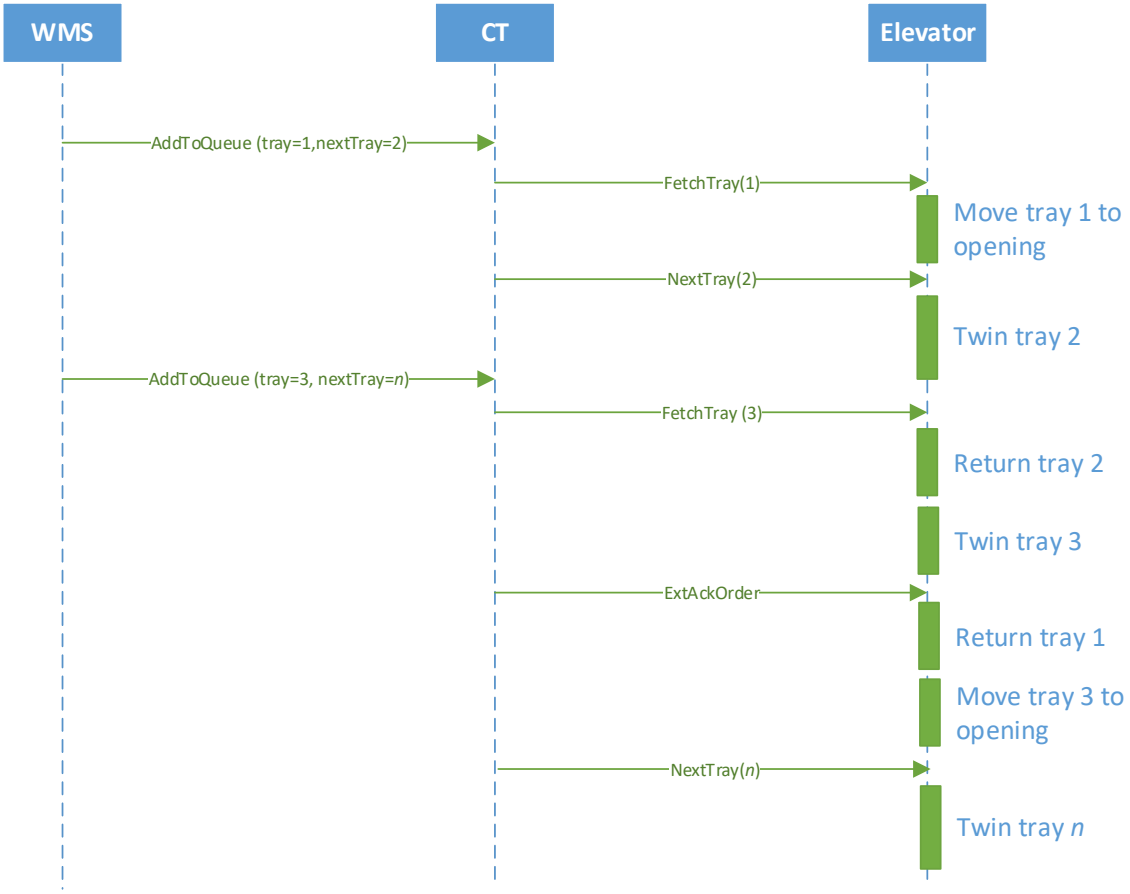


Abbildung 3. Sequenzdiagramm für abgebrochenen Vorgang für Twin

10.2.2 ExtAckOrder

Die Methode ExtAckOrder dient der externen Quittierung/Bestätigung von Aufträgen. Wenn Eigenschaft NoReturnOfTray auf 1 gesetzt wird. Eine externe Quittierung ist erforderlich.

Parameter	Zwingend	Beschreibung
ElevatorId	Ja	ID des Elevators, an den der Auftrag gerichtet ist. Sollte der in der Gerätekonfiguration gesetzten ID entsprechen.
Öffnung	Ja	Nummer der Serviceöffnung 1 – n
TransId	Ja/Nein	Wenn Rückmeldungen verwendet werden, ist dieses Element zwingend. Zur Verknüpfung von Befehl und Rückmeldung.

Beispiel 1

Das folgende Beispiel zeigt eine externe Quittierung an Elevator E1, Öffnung 1.

```
<ExtAckOrder>
  <ElevatorId>E1</ElevatorId>
  <Opening>1</Opening>
</ExtAckOrder>
```

Beispiel 2
Dieselbe Funktion kann durch den Befehl AddToQueue über die Einstellung von Tray auf 0 erreicht werden.

```
<AddToQueue>
  <ElevatorId>E1</ElevatorId>
  <Tray>0</Tray>
  <Opening>1</Opening>
</AddToQueue>
```

10.2.3 AddTrayConfig

Zur Nutzung von Zubehör wie PickDisplay ist die Methode AddTrayConfig erforderlich. Dies stellt die Zubehörinformation zum Layout des Trays zur Verfügung. Die an Compact Talk zur Tray-Konfiguration gesendeten Daten werden im Cache gespeichert. Durch den Cache ist ein weiteres Hinzufügen Konfiguration des Tray-Layouts in Compact Talk nur nach einer Änderung erforderlich.

Parameter	Zwingend	Beschreibung
ElevatorId	Ja	ID des Elevators der das Ziel ist.
Tray	Ja	Die ID des zu holenden Trays. Muss einem existierenden Tray in der Maschine entsprechen.
Boxes	Ja	Eine Liste an Boxen, die den Lagerbereich auf dem Tray darstellen.
TransId	Ja/Nein	Wenn Rückmeldungen verwendet werden, ist dieses Element zwingend. Zur Verknüpfung von Befehl und Rückmeldung.

Beispiel
Dieses Beispiel beschreibt die Nutzung der Methode AddTrayConfig für einen Tray (Tray 1) im Elevator E1. Das letzte Beispiel beschreibt die Nutzung des hinzugefügten Tray-Layouts in Kombination mit AddToQueue.

```
<AddTrayConfig>
  <ElevatorId>E1</ElevatorId>
  <Tray>1</Tray>
  <Boxes>
    <Box>
      <Name>A-1</Name>
      <XPos>0</XPos>
      <YPos>0</YPos>
      <XSize>244</XSize>
      <YSize>164</YSize>
    </Box>
    <Box>
      <Name>A-2</Name>
      <XPos>0</XPos>
      <YPos>164</YPos>
      <XSize>244</XSize>
      <YSize>164</YSize>
    </Box>
  </Boxes>
</AddTrayConfig>
<AddToQueue>
  <ElevatorId>E1</ElevatorId>
```

```
  <Tray>1</Tray>
  <Opening>1</Opening>
  <NoReturnOfTray>0</NoReturnOfTray>
  <CurrentBoxName>A-1</CurrentBoxName>
</AddToQueue>
```

CurrentBoxName wird im Beispiel auf „A-1“ gesetzt, was aktuell in AddTrayConfig beschrieben wird.

10.2.4 ResetElevator

Wartungsfunktion zur Wiederherstellung von Aufträgen und Elevators die spezifische Öffnungen

Parameter	Zwingend	Beschreibung
ElevatorId	Ja	ID des Elevators der das Ziel ist.
Öffnung	Nein	Nummer der Serviceöffnung. Öffnung 1 – n verwenden. 99 steht für kompletten Elevator/alle Öffnungen am Elevator. Falls kein Wert gesetzt ist, wird 99 standardmäßig verwendet.
TransId	Ja/Nein	Wenn Rückmeldungen verwendet werden, ist dieses Element zwingend. Zur Verknüpfung von Befehl und Rückmeldung.

Beispiel 1

Das folgende Beispiel zeigt einen „Reset von“ Öffnung 1 an Elevator E1.

```
<ResetElevator>
  <ElevatorId>E1</ElevatorId>
  <Opening>1</Opening>
</ResetElevator>
```

Beispiel 2

Dieselbe Funktion kann durch den Befehl AddToQueue über die Einstellung von Traynummer 1000 erreicht werden.

```
<AddToQueue>
  <ElevatorId>E1</ElevatorId>
  <Tray>1000</Tray>
  <Opening>1</Opening>
</AddToQueue>
```

10.3 Rückmeldungen

Wenn Rückmeldungen erforderlich sind, kann die gewünschte Ebene an Feedback in der Konfiguration gewählt werden.

10.3.1 CommandResponse

Parameter	Beschreibung
Command	Name des Befehls
Result	Bei 0 ist der Befehl fehlgeschlagen
TransId	Wenn Rückmeldungen verwendet werden, ist dieses Element zwingend. Zur Verknüpfung von Befehl und Rückmeldung.
ErrorMessage	Enthält eine Beschreibung des aufgetretenen Fehlers. Nur bei **Result** von 0.

Beispiel 1

Beispiel für einen erfolgreichen Befehl.

```
<CompactTalkResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Response xsi:type="CommandResponse">
    <TransId>1</TransId>
    <Command>AddToQueue</Command>
    <Result>916</Result>
  </Response>
</CompactTalkResponse>
```

Beispiel 1

Beispiel für einen nicht erfolgreichen Befehl.

```
<CompactTalkResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Response xsi:type="CommandResponse">
    <TransId>3</TransId>
    <Command>AddToQueue</Command>
    <Result>0</Result>
    <ErrorMessage>[E=G2_1,T=333] ValidateOrderData: Tray number 333 does not exist
within elevator G2_1</ErrorMessage>
  </Response>
</CompactTalkResponse>
```

10.3.2 OrderStatusResponse

Parameter	Beschreibung
TransId	Wenn Rückmeldungen verwendet werden, ist dieses Element zwingend. Zur Verknüpfung von Befehl und Rückmeldung.
Status	Entweder Sent, NextAtPlace oder AtPlace, je nach Konfiguration.

```
<?xml version="1.0" encoding="utf-8"?>
<CompactTalkResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Response xsi:type="OrderStatusResponse">
    <TransId>2</TransId>
    <Status>Sent</Status>
  </Response>
</CompactTalkResponse>
```

10.3.3 TaskDoneResponse

Parameter	Beschreibung
TransId	Wenn Rückmeldungen verwendet werden, ist dieses Element zwingend. Zur Verknüpfung von Befehl und Rückmeldung.
Modus	Entweder IN, OUT oder INV. Derselbe Wert, der im Befehl AddToQueue genutzt wurde.
AckQuantity	Die durch den Bediener gehandhabte Menge an Material. Enthält nur einen Wert, wenn **NoReturnOfTray** im Aufruf **AddToQueue** auf 0 gesetzt wurde.

```
<?xml version="1.0" encoding="utf-8"?>
<CompactTalkResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Response xsi:type="TaskDoneResponse">
    <TransId>2</TransId>
    <Mode>OUT</Mode>
    <AckQuantity>0</AckQuantity>
  </Response>
</CompactTalkResponse>
```

11 Import und Export

Die Import- und Exportfunktionen von Compact Talk sind eine Lösung auf Basis eines Plugins, das in der Standardinstallation Plugins für den Import und Export von Flat-Files umfasst, wobei auch weitere Möglichkeiten durch andere Parteien hinzugefügt werden können.

11.1 Import

Für den Import von Dateien in Compact Talk müssen drei Punkte erledigt werden: Hinzufügen einer Instanz eines Importers in der Konfiguration, Basiskonfiguration und Definition des Formats für den Dateiinhalt. Dies erfolgt über das Konfigurationstool. Siehe Konfigurationshandbuch [1] für weitere Informationen und Beispiele.

11.2 Export

Für den Export von Dateien in Compact Talk müssen drei Punkte erledigt werden: Hinzufügen einer Instanz eines Exporters in der Konfiguration, Basiskonfiguration und Definition des Formats für den Dateiinhalt. Dies erfolgt über das Konfigurationstool. Siehe Konfigurationshandbuch [1] für weitere Informationen und Beispiele.

11.3 Import von Tray-Konfiguration

Der Import der Traykonfiguration ist ein spezieller Typ an Import, der eine Flat-File für die Tray-Layoutkonfigurationen verwendet. Zur Unterstützung des Bedieners wird diese Konfiguration durch Compact Talk zur Steuerung von Zubehörgeräten durch eine Anzeige der Boxen, in denen Artikel positioniert sind, verwendet. Das Client-Interface bietet Methoden mit derselben Funktion. Allerdings sollte die Importmethode in manchen Fällen bevorzugt werden.

11.3.1 Konfiguration

Verwenden Sie das Konfigurationstool zur Auswahl der Datei zum Import. Siehe Konfigurationshandbuch [1] für Einzelheiten.

11.3.2 Format

Die Tray-Konfigurationsdatei muss über eine Box pro Zeile verfügen; aufgeteilt auf 9 Spalten, davon 2 optional, geteilt mit dem Pipe-Zeichen „|“.

Die neun Spalten sind:

1. Elevator ID
2. Traynummer
3. Name der Box Bei Hinzufügen eines Auftrags zur Identifikation der Box mit Position des Artikels.
4. Position in X (mm)
5. Position in Y (mm)
6. Größe in X (mm)
7. Größe in Y (mm)
8. Optionaler Nummernwert. Spezifisch für Implementierung.
9. Optionaler Textwert. Spezifisch für Implementierung.

Beispiel:
Elevator_1|1|A-1|101|1|100|200
Dieses Beispiel zeigt eine Box für **Elevator_1** Tray **1** mit dem Namen **A-1**, mit Koordinaten **101, 1**, Breite **100** und Höhe **200**.