

# EOS



## MSG Systemhandbuch

**Speicherverwaltung für sichere, zuverlässige  
und strukturierte Echtzeit-Anwendungen**

Die in diesen Unterlagen enthaltenen Informationen sind Eigentum der SMA Regelsysteme GmbH. Die Veröffentlichung, ganz oder in Teilen, bedarf der schriftlichen Zustimmung der SMA Regelsysteme GmbH. Eine innerbetriebliche Vervielfältigung, die zur Evaluierung des Produktes oder zum sachgemäßen Einsatz bestimmt ist, ist erlaubt und nicht genehmigungspflichtig.

## Haftungsausschluss

Es gelten als Grundsatz die AGB der SMA Regelsysteme GmbH.

Der Inhalt dieser Unterlagen wird fortlaufend überprüft und gegebenenfalls angepasst. Trotzdem können Abweichungen nicht ausgeschlossen werden. Es wird keine Gewähr für Vollständigkeit gegeben. Die jeweils aktuelle Version ist im Internet unter [www.SMA.de](http://www.SMA.de) abrufbar oder über die üblichen Vertriebswege zu beziehen.

Gewährleistungs- und Haftungsansprüche bei Schäden jeglicher Art sind ausgeschlossen, wenn sie auf eine oder mehrere der folgenden Ursachen zurückzuführen sind:

- Unsachgemäße oder nicht bestimmungsgemäße Verwendung der Baugruppe
- Betreiben der Baugruppe in einer nicht vorgesehenen Umgebung
- Betreiben der Baugruppe unter Nichtberücksichtigung der am Einsatzort relevanten gesetzlichen Sicherheitsvorschriften
- Nichtbeachten der Warn- und Sicherheitshinweise in allen für die Baugruppe relevanten Unterlagen
- Betreiben der Baugruppe unter fehlerhaften Sicherheits- und Schutzbedingungen
- Eigenmächtiges Verändern der Baugruppe oder der mitgelieferten Software
- Fehlverhalten der Baugruppe durch Einwirkung angeschlossener oder benachbarter Geräte außerhalb der gesetzlich zulässigen Grenzwerte
- Katastrophenfälle und höhere Gewalt

## Softwarelizenzierung

Die Nutzung der mitgelieferten von der SMA Regelsysteme GmbH hergestellten Software unterliegt folgenden Bedingungen:

Die Software darf für innerbetriebliche Zwecke vervielfältigt und auf beliebig vielen Computern installiert werden. Mitgelieferte Quellcodes dürfen, dem innerbetrieblichen Verwendungszweck entsprechend, in Eigenverantwortung verändert und angepasst werden. Ebenso dürfen Treiber auf andere Betriebssysteme portiert werden. Jegliche Veröffentlichung der Quellcodes ist nur mit schriftlicher Zustimmung der SMA Regelsysteme GmbH zulässig. Eine Unterlizenzierung der Software ist nicht zulässig.

**Haftungsbeschränkung:** Die SMA Regelsysteme GmbH lehnt jegliche Haftung von direkten oder indirekten Folgeschäden, die sich aus der Verwendung der von SMA Regelsysteme GmbH erstellten Software ergeben, ab. Dies gilt auch für die Leistung beziehungsweise Nicht-Leistung von Support-Tätigkeiten.

Mitgelieferte Software, die nicht von der SMA Regelsysteme GmbH erstellt wurde, unterliegt den jeweiligen Lizenz- und Haftungsvereinbarungen des Herstellers.

## Warenzeichen

Alle Warenzeichen werden anerkannt, auch wenn diese nicht gesondert gekennzeichnet sind. Fehlende Kennzeichnung bedeutet nicht, eine Ware oder ein Zeichen seien frei.

SMA Regelsysteme GmbH  
Hannoversche Straße 1-5  
34266 Niestetal  
Deutschland

Tel. +49 561 9522-0  
Fax +49 561 9522-100  
[www.SMA.de](http://www.SMA.de)  
E-Mail: [info@SMA.de](mailto:info@SMA.de)

© 2003 SMA Regelsysteme GmbH. Alle Rechte vorbehalten.

# Inhaltsverzeichnis

<b>1</b>	<b>Allgemeine Hinweise</b>	<b>5</b>
1.1	Erweiterte Copyright Hinweise	5
1.2	Verwendete Symbole	5
1.3	Änderungsübersicht	6
<b>2</b>	<b>Einleitung</b>	<b>7</b>
<b>3</b>	<b>Das MSG Systemkonzept</b>	<b>10</b>
3.1	Message	10
3.2	Mailboxen	11
3.3	Fehlererkennung, Interaktionsregeln	11
3.4	Statisches Design	12
3.5	Fehlerhandling	13
3.6	Messages und Funktionen	13
3.7	Module und Komponenten	14
3.8	Bibliotheken und Frameworks	14
<b>4</b>	<b>Systemparameter und Systemfunktionen</b>	<b>16</b>
4.1	Error Subsystem (ERR)	16
4.2	Message Subsystem (MSG)	16
<b>5</b>	<b>Das User Interface (API)</b>	<b>19</b>
5.1	Funktionsaufrufe und Returncodes	19
5.2	Der EOS Nullpointer (EOS_NULL)	20
<b>6</b>	<b>EOS Fehlerhandling</b>	<b>21</b>
6.1	eos_err_log	22
6.2	EosErrCnt	22
6.3	EosErrTab [EOS_ERR_TAB]	22
6.4	EosUsrErr [EOS_USR_ERR + 1]	22
6.5	EosErrLog [EOS_ERR_LOG]	23
6.6	EosErrLogWr	23
6.7	EOS_ErrInit (void)	23
6.8	EOS_Err (int Err, int Usr, int Mbx, void* Msg)	23
6.9	EOS_TRACE (Trc, Usr, Mbx, Ptr)	23
<b>7</b>	<b>EOS Messages und Mailboxen</b>	<b>24</b>
7.1	EOS_NULL (EosNull)	24
7.2	EosMsgClr	25
7.3	EOS_MsgInit (void)	25
7.4	EOS_MbxCreate (unsigned Own)	25
7.5	EOS_UsrBnd (int Usr, int Bnd)	25
7.6	EOS_Alloc (unsigned Siz, int Usr)	26
7.7	EOS_Free (void* Msg, int Usr)	26
7.8	EOS_Put (int Mbx, void* Msg, int Usr)	26

7.9	EOS_Get (int Mbx, int Usr) . . . . .	26
7.10	EOS_MbxPurge (int Mbx, unsigned Cnt, ...) . . . . .	26
7.11	EOS_MsgPurge (int Usr) . . . . .	27
7.12	EOS_MbxInfo (int Mbx, int* Cnt, int* Bnd, ...) . . . . .	27
7.13	EOS_MsgInfo (void* Msg, int* Len, int* Usr, ...) . . . . .	28
7.14	EOS_UsrInfo (int Usr, int* Mem, int* Bnd) . . . . .	28
7.15	EOS_IdxInfo (int Idx, int* Len, int* Usr, ...) . . . . .	28
8	Ein Beispiel . . . . .	29
9	Fehlercodes, Ursachen, Fehlerbehebung . . . . .	38
9.1	MSG Subsystem . . . . .	38
10	Die MSG Implementierung . . . . .	44
10.1	IniTab und die Anlage des Memory-Pools . . . . .	44
10.2	Die Message-Tabelle (MsgTab). . . . .	46
10.3	Message- und Index-Prüfungen . . . . .	47
10.4	Mailboxen . . . . .	48
10.5	Alloc und Free . . . . .	49
10.6	Zugriffsrechte und Besitzverhältnisse . . . . .	50
10.7	Get und Put . . . . .	50
10.8	MsgPurge . . . . .	52
11	Der Sourcecode . . . . .	53
11.1	EOS_ERR.H. . . . .	53
11.2	EOS_ERR.C. . . . .	57
11.3	EOS_MSG.H. . . . .	59
11.4	EOS_MSG.C. . . . .	62

# 1 Allgemeine Hinweise

## 1.1 Erweiterte Copyright Hinweise

© Copyright 2003 by SMA Regelsysteme GmbH

Das Copyright für EOS und das Subsystem MSG und alle sonstigen Rechte an der Konzeption und dem Sourcecode liegen bei SMA.

Der Sourcecode darf im Rahmen und zu Zwecken der Forschung und Lehre eingesetzt werden. Er darf dabei auch modifiziert und mit anderen Systemen kombiniert werden, sofern SMA über die Erfahrungen und Ergebnisse informiert wird. In Veröffentlichungen sind EOS (MSG) und SMA entsprechend zu benennen.

Die Weitergabe von EOS (MSG), die Integration in Applikationssysteme und jede Form kommerzieller Nutzung ist nur mit schriftlicher Genehmigung durch SMA zulässig.

Umschlagfoto: P. Gootzen

## 1.2 Verwendete Symbole

Um einen sicheren, optimalen Gebrauch dieses Systemhandbuches und der Baugruppe zu gewährleisten, werden folgende Symbole in dieser Anleitung verwendet.

*Unter dem Symbol Hinweis wird ein Sachverhalt aufgeführt, dessen Nichtbeachtung zu einem Verlust an Komfort oder zur Beeinträchtigung der Funktion führen kann.*



Dieses Symbol kennzeichnet ein Beispiel.



## 1.3 Änderungsübersicht

Dokumentennummer	Ausgabe	Änderungen
EOSMSG-11-SD3803	1.0	Erstausgabe
EOSMSG-11-SD4703	1.1	Überarbeitung für SW Version 1.1

## 2 Einleitung

Die funktionalen Anforderungen an Steuerungssysteme wachsen kontinuierlich und mit ihnen die Komplexität der Software und der Kostendruck. Steigende Anforderungen an das Echtzeitverhalten kommen hinzu. Und für sicherheitsrelevante Anwendungen wird gleichzeitig noch ein klar strukturiertes und einfaches Design gefordert. Dieses Handbuch beschreibt eine Lösung, die alle genannten Aspekte berücksichtigt, mit erstaunlich geringem Aufwand implementiert werden kann und sehr effektiv ist.

Klar strukturierte und einfache Designs sind natürlich immer gut. In der Regel sind die Entwürfe für neu zu entwickelnde Systeme auch so konzipiert. Aber im Laufe der Entwicklung verliert sich dann mit immer neuen Forderungen an die Funktionalität oft nicht nur die Einfachheit, sondern mangels geeigneter Methoden zur Bewältigung der zunehmenden Komplexität auch die Klarheit.

Objektorientierte Ansätze versprechen hier Abhilfe durch Kapselung, strikte Methodenorientierung und wiederverwendbare Entwurfsmuster (Design-Pattern). Sie führen aber durch ihre Verzahnung mit dem Laufzeitsystem (Speichermanagement, Fehlerhandling, Garbage Collection) oft zu Problemen im Echtzeitverhalten.

- Um ein adäquates Gesamtkonzept für diese sicherlich nicht einfache Situation zu entwickeln, stellen wir zunächst die wichtigsten Anforderungen bei der Entwicklung einer sicherheitskritischen Echtzeitanwendung zusammen. Die Betonung liegt dabei nicht vorrangig auf sicherheitskritisch, denn Zuverlässigkeit, Robustheit und hohe Verfügbarkeit sind sicherlich allgemeingültige Qualitätsmerkmale.
- Die Echtzeitbedingungen müssen erfüllt werden
- Der zeitliche Programmablauf muss vorhersagbar sein
- Fehler müssen erkannt und beherrscht werden
- Die zeitliche und funktionale Struktur muss transparent sein
- Spezifizierte Anforderungen müssen verifizierbar sein
- Die Komplexität der Software muss methodisch beherrschbar sein

Diese Anforderungen sind die Leitlinien für das von SMA entwickelte und in vielen Projekten erfolgreich eingesetzte Embedded Operating System (EOS). Es handelt sich dabei um ein in sich geschlossenes, kompaktes und sehr überschaubares Laufzeitsystem mit

- eigenständigen Software-Komponenten (Tasks),
- Nachrichten (Messages) und
- Kommunikations-Kanälen (Mailboxen).

Das EOS Gesamtsystem ist aus Subsystemen aufgebaut, die weitgehend unabhängig voneinander sind und unabhängig voneinander eingesetzt werden können.

- Fehlerhandling (ERR)
- Message- und Mailboxsystem (MSG)
- Multitasking System (TSK)
- User Interface (API)

In diesem Handbuch sind die Messages und Mailboxen (MSG) von EOS beschrieben und das Fehlerhandling (ERR).

Die Beschreibung des Multitaskings (TSK) und des User-Interfaces (API) von EOS finden Sie im TSK Systemhandbuch.

MSG enthält eine Speicherverwaltung, die sehr effektiv, robust und kompakt ist. In Verbindung mit den Mailboxfunktionen put und get bietet MSG nicht nur beste Echtzeiteigenschaften, sondern auch ideale Voraussetzungen zum Einsatz in sicherheitsrelevanten Anwendungen.

Die Messages sind dabei nichts anderes als dynamische Speicherbereiche (Puffer), mit denen in EOS gleichermaßen der Datenfluss und der Aktionsfluss organisiert wird. Der Aktionsfluss erfolgt dabei zwischen verschiedenen Software-Komponenten, der Datenfluss wahlweise zwischen oder innerhalb dieser Komponenten. Der Aktionsfluss erfolgt meist mit kürzeren Messages (Nachrichten oder Kommandos mit wenigen Parametern), der Datenfluss mit längeren Messages (Nutzdaten-Pakete). Aus der Sicht von MSG spielt diese inhaltliche Unterscheidung aber keine Rolle, da alle Messages gleich behandelt werden.

Den Kern des MSG Message- und Mailboxsystems bildet eine Implementierung der beiden Funktionen malloc und free, die dynamischen Speicher unabhängig vom Laufzeitsystem des Compilers bzw. Betriebssystems verwalten, denn die Heap-Verwaltung vieler Laufzeitumgebungen ist wegen mangelndem Determinismus für Echtzeitsysteme nicht geeignet. Darüber hinaus werden viele dynamische Speicher-Verwaltungen durch Anwendungsfehler schnell korrumpiert.

Wir stellen deshalb eine Heapverwaltung mit malloc und free vor, die für den Anwender völlig transparent bleibt, aber deterministisches Zeitverhalten hat und übliche Fehler sicher erkennt und definiert behandelt.

- Puffer-Überlauf
- Ungültige Pointer, insbesondere Null-Pointer
- Mehrfaches free
- Fehlendes free (Memory Leaks)



Damit kann die zeitlich unbegrenzte Stabilität, Konsistenz und Vollständigkeit des MSG Speicherpools sichergestellt werden. Zur Strukturierung des Daten- und Kontrollflusses innerhalb von Applikationen stellt das MSG Pattern dem Anwender zusätzlich ein Mailboxsystem mit den beiden Operationen Put und Get (auf denen auch malloc und free basieren) zur Verfügung.

- Unbegrenzte Kapazität aller Mailboxen
- Konstante und sehr kurze Ausführungszeit der Zugriffs-Funktionen
- Zero Copy Prinzip
- Fehlererkennung, Fehlermanagement, Fehlererfassung
- Zugriffsverwaltung, Statistik, Monitoring

Das nachrichtenbasierte Kommunikations-Konzept von MSG mit Messages und Mailboxen verbindet einzelne Software-Komponenten (Funktionen, Module, Tasks) auf einfache und transparente Weise untereinander und ist darüber hinaus prädestiniert für Mehrprozessor-Anwendungen und verteilte Systeme. Der Overhead an Speicher ist mit 8 Byte pro Message (zusätzlich zu den mit malloc allozierten Nutzdaten), 16 Bytes pro Mailbox und ca. 500 Zeilen C-Sourcecode minimal, so dass MSG sowohl den 64K-Speicher eines 8-Bit Controllers als auch Gigabytepools eines 32-Bit Prozessors verwalten kann.

Das MSG Pattern wurde von der Firma SMA mehrfach und in sehr verschiedenen Anwendungen eingesetzt und hat sich sowohl unter harten Echtzeitbedingungen als auch unter hohen Sicherheitsanforderungen bestens bewährt. Darüber hinaus bildet MSG eine ideale Basis für die Entwicklung und den Einsatz wiederverwendbarer Softwarekomponenten, da inhaltlich in sich abgeschlossene Teilprozesse als eigenständige Komponenten mit definiertem Interface entwickelt, vollständig getestet und problemlos in verschiedenen Projekten eingesetzt werden können.

MSG steht vollständig im Sourcecode zur Verfügung und kann für eigene Anwendungen adaptiert, erweitert und modifiziert werden. Beachten Sie hierzu den Copyrightvermerk und die rechtlichen Hinweise am Anfang.

## 3 Das MSG Systemkonzept

Innerhalb eines Echtzeitsystems sind die Aufgaben und Funktionen einzelner Tasks oder anderer Komponenten miteinander verknüpft, indem etwa Task A Datenpakete empfängt und Task B diese interpretiert und auswertet. Die zeitliche und funktionale Struktur des Gesamtsystems bleibt dabei nur erhalten, wenn Komponente A hierbei so mit Komponente B in Kontakt tritt, dass bereits verifizierte Eigenschaften von A (kann alle 5 ms ein Paket empfangen) auch bei Änderungen in Komponente B erhalten bleiben. Dies lässt sich nicht mit Funktionsaufrufen erreichen, sondern nur mit Message-Passing (funktionale Entkopplung) über Mailboxen (zeitliche Entkopplung).

### 3.1 Message

Eine Message ist ein Datenpuffer zur Kommunikation zwischen Software-Komponenten oder auch zwischen logischen Programmteilen innerhalb dieser Komponenten. Sie wird von einer Datenquelle unter Angabe der gewünschten Nutzdatenlänge angefordert (Alloc), zur Weitergabe von Informationen in Mailboxen abgelegt (Put) und daraus wieder entnommen (Get) und am Ende ihres Weges von einer Datensenke wieder an das MSG-System zurückgegeben (Free).

Jede Message hat zu jeder Zeit einen Besitzer (User), das ist entweder die Mailbox, in der sie sich gerade befindet, oder die Software-Komponente, die sie aus einer Mailbox entnommen und noch nicht in einer anderen Mailbox abgelegt oder zurückgegeben hat.

Technisch sind Messages Pointer auf einen Pufferbereich. Mit Alloc oder Get wird einer Task der Pointer übergeben. Sie kann dann auf die Daten im zugehörigen Puffer zugreifen und den Pointer anschließend mit Put oder Free weitergeben. Physikalisch wird dabei nur der Pointer bewegt, nicht aber die Daten (Zero Copy Prinzip). Messages sind zum Transport von Daten zwischen Mailboxen gedacht, aber auch einem Einsatz zum Aufbau klassischer dynamischer Datenstrukturen, etwa binärer Suchbäume, steht nichts entgegen.

In der Regel sollte die Entgegennahme, die Verarbeitung und die Weitergabe einer Message in einem Schritt, d.h. ohne zwischenzeitlichen Kontext-Wechsel, erfolgen. Ausnahmen ergeben sich oft an physikalischen Schnittstellen, da die Übertragung von Daten Zeit benötigt, oder bei sehr komplexen Verarbeitungsschritten, die auf mehrere Taskzyklen aufgeteilt werden.

Die robuste Message-Verwaltung ist das Herzstück von EOS. Fehlerhafte Pointer und die Überschreitung von Puffergrenzen werden zuverlässig erkannt und definiert behandelt. Damit kann die zeitlich unbegrenzte Stabilität, Konsistenz und Vollständigkeit des EOS Message-Pools sichergestellt werden.

## 3.2 Mailboxen

Die EOS Mailboxen sind Warteschlangen für Messages und arbeiten nach dem FIFO-Prinzip (First In, First Out). Sie werden mit `MbxCreate` eingerichtet und stehen dann für die Aufnahme von Messages (`Put`) bzw. deren Entnahme (`Get`) bereit. Mailboxen können ausschließlich Messages aufnehmen, davon aber unbegrenzt viele.

Beim Einrichten erhält jede Mailbox eine Mailbox-Nummer, mit der sie später angesprochen wird, und ihr wird eine Task als Besitzer zugeordnet. Mailboxen mit dem Besitzer 0 heißen öffentliche Mailboxen, die anderen sind private Mailboxen.

Auch die zur Initialisierungszeit (`MsgInit`) eingerichteten Messages werden in Mailboxen abgelegt. Für jede in der Initialisierungstabelle aufgeführte Puffer-Länge wird dabei eine eigene Mailbox angelegt. Diese Mailboxen heißen Home-Mailboxen und sind öffentlich. Auch die Systemfunktionen `EOS_Alloc` und `EOS_Free` arbeiten mit Mailboxen, indem sie Messages aus den Home-Mailboxen entnehmen bzw. dort ablegen.

Die Mailboxen sind also die Wächter an den Ein- und Ausgängen des MSG Message-Pools und garantieren dessen Sicherheit. Technisch sind Mailboxen nichts weiter als ein Verweis auf den Anfang und das Ende einer verketteten Liste von Messages, nämlich jener, die sich zur Zeit in der Mailbox befinden. Die Funktion der Mailboxen basiert also wesentlich auf der Message-Verwaltung.

## 3.3 Fehlererkennung, Interaktionsregeln

In EOS werden alle Parameter der API-Funktionen in jedem Aufruf vollständig auf Gültigkeit geprüft. Dabei wird (im Gegensatz zu einem `ASSERT`-Macro) nicht zwischen einer Entwicklungs- und einer Laufzeit-Version unterschieden. Dadurch ist die zeitlich und inhaltlich unbegrenzte Zuverlässigkeit und Verfügbarkeit des MSG-Systems durch die MSG-Implementierung sichergestellt und nicht von der Testgüte des Anwendungssystems abhängig.

Darüber hinaus gibt es zwischen den SW-Komponenten, Messages und Mailboxen zusätzliche Interaktions-Regeln, mit denen viele weitere typische Softwarefehler erkannt und damit von EOS behandelt und vom Anwender behoben werden können. Zusätzlich kann der Verbleib aller Messages jederzeit verifiziert und gegebenenfalls korrigiert werden, um die Langzeit-Verfügbarkeit des Systems auch bei Software- oder Design-Fehlern sicherzustellen.

Eine Komponente kann eine Message nur dann in einer Mailbox ablegen, wenn sie Besitzer (User) der Message ist. Das verhindert mehrfache `Put`- oder `Free`-Aufrufe mit der gleichen Message (Gefahr der Inkonsistenz im Message-Pool), weil die Message beim ersten Aufruf von `Free`/`Put` ihren Besitzer wechselt (vom User zur Mailbox), damit keinem User mehr gehört und damit auch nicht mehr in einer Mailbox abgelegt werden kann.

Ein User kann eine Messages nur dann aus einer Mailbox entnehmen, wenn er der Besitzer (Owner) der Mailbox ist oder die Mailbox öffentlich ist. Das verhindert fehlerhaftes Lesen aus fremden Mailboxen durch Indizierungsfehler.

Eine Message, die in einer nicht existenten Mailbox abgelegt werden soll, wird stattdessen automatisch ihrer Home-Mailbox zurückgegeben. Das verhindert den Verlust von Messages (Memory Leak) durch fehlerhafte Mailbox-Nummern, garantiert die Langzeit-Verfügbarkeit des MSG Message-Pools und erspart der Applikation ein eigenes Fehlerhandling.

Ungültige Message-Pointer, die also zwischen der Entnahme mit Alloc/Get und der Weitergabe mit Free/Put verändert wurden, werden von EOS (Put) erkannt und abgewiesen. Damit werden Inkonsistenzen in der Message-Verwaltung auf Grund von Pointer-Fehlern verhindert. Der (ursprüngliche) Pufferbereich geht dem System zunächst verloren (Memory Leak), kann aber erkannt (MsgPurge) und seiner Home-Mailbox zurückgegeben werden.

Messages im Besitz einer Komponente können dieser weggenommen (MsgPurge) und in ihre Home-Mailboxen zurückgeführt werden. Das kann notwendig werden, wenn die Komponente "vergisst", verarbeitete Messages zurückzugeben, d.h. Messages verliert oder Message-Pointer vor der Rückgabe verändert und diese dadurch von MSG nicht akzeptiert werden. Damit können auch Memory Leaks, das Fehler-Pendant zur mehrfachen Rückgabe, erkannt und behandelt werden.

Messages in einer Applikations-Mailbox können dieser weggenommen (MbxPurge) und in ihre Home-Mailboxen zurückgeführt werden. Das kann notwendig werden, wenn der zuständige Besitzer (Owner) der Mailbox die Messages nicht mehr (Fehler) oder nicht schnell genug (Performance, Zykluszeit) verarbeitet, sich dadurch zu viele Messages in der Mailbox ansammeln und so die Gefahr eines Speichermangels entsteht.

Der Speicherplatz, den eine Komponente aktuell per Alloc belegt haben darf, kann begrenzt werden (UsrBnd). Damit kann verhindert werden, dass eine sicherheitsrelevante oder hochpriorie Komponente keinen Speicher allozieren kann, weil eine speicherhungrige niederpriorie oder eine fehlerhafte Komponente diesen blockiert.

## 3.4 Statisches Design

MSG ist so konzipiert, dass der Speicherplatz für Messages bereits bei der Initialisierung (Init) vollständig angelegt wird anhand einer Tabelle, die vom Anwender im Sourcecode editiert wird. Ein späteres Hinzufügen weiterer Messages ist nicht möglich. Auch die Tabellen für Mailboxen werden zur Übersetzungszeit fest dimensioniert.

Die Anwendung ihrerseits sollte zu Beginn alle benötigten Mailboxen öffnen und mit einer anschliessenden Fehlerabfrage (ErrCnt) feststellen, ob alle benötigten Ressourcen erfolgreich angelegt wurden. Andernfalls ist MSG für die Anwendung falsch dimensioniert und muss entsprechend angepasst werden.

Dadurch vereinfacht sich die Implementierung von MSG, die Fehlermöglichkeiten und damit das Fehlerhandling innerhalb der Anwendung werden reduziert und Laufzeitprobleme durch die Heapverwaltung der Standard C-Library (malloc, free) werden vermieden. Konsequenterweise ist das Schließen von Mailboxen in MSG nicht vorgesehen.

Für sicherheitsrelevante Echtzeitanwendungen in embedded Systemen ist ein solches statisches Design in der Regel das einzig angemessene.

## 3.5 Fehlerhandling

Das MSG-System führt alle notwendigen Prüfungen durch, um mögliche Fehler bei der Ausführung einer Systemfunktion zu erkennen und so zu behandeln, dass Folgefehler möglichst vermieden oder zumindest begrenzt werden. Alle Fehler werden über das EOS Subsystem ERR gemeldet und registriert (EosErrCnt, EosErrTab, EosUserErr, EosErrLog). Die Anwendung hat vollständigen Zugriff auf die Fehlererfassung von EOS und kann auf eine eigene Fehlerbehandlung somit verzichten.

Fehler sollten während der Entwicklung zu einem sofortigen Stopp und anschliessender Fehlerbeseitigung, zur Laufzeit zur Protokollierung und einem Wartungshinweis führen. Jeder von EOS gemeldete Fehler signalisiert eine Fehlfunktion oder zumindest Nachlässigkeit der Anwendungs-Software und gefährdet damit potentiell deren Zuverlässigkeit, deren Verfügbarkeit oder deren Stabilität.

Beim Umfang der Fehlerprüfungen wird in EOS bewusst nicht unterschieden zwischen Entwicklung und Laufzeit, weil nur dadurch eine gleichbleibende Fehlererkennung und definierte Fehlerbehandlung garantiert ist.

Die teilweise recht umfangreiche Fehlerprüfung in MSG, die z.B. bei EOS\_Put ähnlich aufwendig wie die eigentliche Kernfunktion ist, erhöht die Laufzeit der EOS System-Funktionen um weniger als 20 Prozent. Angesichts des dadurch erzielten Sicherheitsstandards ist das eher gering.

## 3.6 Messages und Funktionen

Eine Funktion erbringt eine Leistung, die eine Task zur Bedienung ihres Teilprozesses benötigt, auf die sie also wartet, damit sie mit dem Funktionsergebnis ihre Arbeit fortsetzen kann. Die Leistung wird auf der Zeitachse des Auftraggebers (aufrufende Komponente) erbracht.

Eine Message erteilt einen Auftrag zur Erbringung einer Leistung an eine andere Komponente. Ob der Absender die Bedienung seines Teilprozesses danach fortsetzt oder unterbricht, hängt von der Ablaufstruktur des Prozesses ab, aber nicht von der Auswirkung des erteilten Auftrags. Die Leistung wird auf der Zeitachse des Auftragnehmers (Message-Empfänger) erbracht.

Nachrichten sollten in der Regel nicht zur Simulation von Funktionsaufrufen (ich schicke dir die Parameter und warte auf die Nachricht mit dem Ergebnis) missbraucht werden. Ausnahmen von dieser Regel können Sinn machen, wenn die simulierte Funktion mit Hardware-Ressourcen arbeitet (und ihr Ergebnis zur Weiterarbeit notwendig ist) oder grossen Speicherbedarf hat, der nicht mehrfach vorgehalten werden soll.

Generell sollte (nicht nur) in einem nachrichtenbasierten System wie MSG die asynchrone Kopplung von Komponenten immer bevorzugt werden, da sie die Zahl der Kontextwechsel reduziert, die Verfügbarkeit erhöht und die Echtzeiteigenschaften des Systems verbessert.

## 3.7 Module und Komponenten

Die klassische strukturierte Programmierung unterteilt die Software in einzelne Module, in denen eine klar umrissene Teilaufgabe eines Problems gelöst wird. Die implementierten Leistungen werden wirksam, indem die im Interface des Moduls bereitgestellten Funktionen aufgerufen werden. Komponenten realisieren eine klar umrissene Teilaufgabe eines Prozesses. Die implementierten Leistungen werden wirksam, indem die Komponente aktiviert oder von sich aus aktiv wird oder von einer anderen Komponente per Message dazu beauftragt wird.

Die Aufteilung in Komponenten und Module erfolgt also unter ganz verschiedenen Gesichtspunkten, nämlich ablaforientierte Strukturierung in Komponenten und logisch orientierte Strukturierung in Module. Funktional macht es Sinn, Sicherungsalgorithmen (z.B. CRC) in einem Modul zusammenzufassen. Im Ablauf werden diese Funktionen eventuell in verschiedenen Teilprozessen (Komponenten) benötigt, z.B. Prüfung eintreffender Datenpakete und Sicherung abgehender Pakete. Verschiedene Komponenten werden also bei Bedarf die Funktionen des gleichen CRC-Moduls aufrufen. Umgekehrt wird eine Komponente oftmals Funktionen aus mehreren Modulen aufrufen.

Logik und Ablauf fallen zusammen, wenn es um Hardware-Ressourcen geht, weil diese spezifisches Handling erfordern und zu jeder Zeit nur von einem Teilprozess genutzt werden können. Ein Interface zum CAN-Bus wird also sinnvollerweise als eigenständige Komponente realisiert, die als einzige auf die Funktionen eines CAN-Moduls zugreift und von anderen Komponenten Sendeaufträge per Message erhält, die dadurch auf natürliche Weise serialisiert werden. Dies gilt auch für einfachere Ressourcen wie etwa einen Timer, eine Watchdog oder einen Signalgeber.

## 3.8 Bibliotheken und Frameworks

Module und Tasks als logikorientierte und ablaforientierte Gliederung lassen sich in anderer Terminologie auch so beschreiben, dass Module das Ergebnis eines bibliotheksbasierten Ansatzes sind, die EOS Komponenten hingegen das Ergebnis eines Framework orientierten Ansatzes. Wir benutzen den Begriff des Frameworks hier im allgemeinen Sinn von Komponenten.

Bei der traditionellen Anwendung von Software-Bibliotheken steht die Entwicklung einer Applikationssoftware im Vordergrund, welche auf existierende Basisroutinen zugreift. Die Ablaufkontrolle liegt dabei vorrangig in der Applikation. Das Ausmaß der Wiederverwendung der Bibliotheksroutinen ist hierbei kaum vorhersagbar und deren architekturkonforme Anwendung stark vom einzelnen Entwickler abhängig.

Frameworks dagegen beinhalten bereits einen vollständigen Applikationsrahmen, der um eigene Komponenten ergänzt wird, die innerhalb des bestehenden Frameworks aktiviert werden. In diesem Sinne ist MSG das EOS Message-Framework und TSK das EOS Task-Framework. Beide nutzen ihrerseits das EOS Error Framework ERR.

Dies bedeutet, dass eine EOS Applikation sich auf die inhaltliche Ausgestaltung der im Framework bereits vorhandenen abstrakten Funktionalitäten konzentrieren kann. Damit wird eine deutlich höhere Software-Wiederverwendbarkeit ermöglicht. Die Software-Qualität steigt, da die neuralgischen Punkte wie Komponenten-Modell, Kontrollfluss, Datenfluss und Fehlerhandling bereits im Framework definiert sind und damit außerhalb des projektspezifischen Software-Anteils liegen.

## 4 Systemparameter und Systemfunktionen

Vor dem Arbeiten mit dem MSG System in einem neuen Projekt müssen einige Konstanten und Macros (im Sourcecode mit XXX gekennzeichnet) an die Erfordernisse der Applikation angepasst werden. Anschließend müssen die betroffenen Dateien neu übersetzt werden.

Die Anpassung der System-Parameter erfolgt in der Regel einmalig zu Beginn eines Projektes. Von da an kann mit den Objektdateien von EOS gearbeitet werden, um versehentliche Änderungen im Sourcecode auszuschließen.

### 4.1 Error Subsystem (ERR)

Die Anpassungen für dieses Subsystem werden in EOS\_ERR.H vorgenommen. Zunächst können alle Einträge belassen werden wie sie sind, Wenn Zeitstempel gewünscht sind, muss EOS\_TIM angepasst werden.

#### 4.1.1 EOS\_ERR\_LOG

Diese Konstante legt fest, wie viele Fehlereinträge der Ringpuffer EosErrLog aufzeichnen kann. Der Minimalwert ist 1 und der Wert sollte eine Zweierpotenz sein.

#### 4.1.2 EOS\_USR\_ERR

Diese Konstante legt fest, für welche User (1...EOS\_USR\_ERR) eine individuelle Zählung ihrer Fehler in der Tabelle EosUsrErr durchgeführt wird. Größere Usr-Werte sind erlaubt, es erfolgt aber nur eine globale Zählung ihrer Fehler in EosUsrErr [0].

#### 4.1.3 EOS\_TIM()

Dieses Macro wird benutzt, um die Fehlereinträge mit einem Zeitstempel zu versehen. In Verbindung mit dem TSK Subsystem wird die Variable EosTim verwendet. Ohne TSK Subsystem kann 0 benutzt werden oder eine andere Systemzeit.

### 4.2 Message Subsystem (MSG)

Die Anpassungen für dieses Subsystem werden in EOS\_MSG.H und EOS\_MSG.C vorgenommen. Die zunächst wesentlichen Einträge sind MBX\_MAX und IniTab, für das TSK Subsystem auch EOS\_SLEEP und EOS\_WAKE.

#### 4.2.1 EOS\_ENTER(), EOS\_LEAVE()

Diese Macros sind leer. Sie sind für den Einsatz von MSG in preemptiven Umgebungen bzw. in Interrupt-Handlern vorgesehen. Hierzu gibt es einige technische Hinweise im Sourcecode.



## 4.2.2 EOS\_WAKE()

Dieses Macro wird benutzt, um eine suspendierte Task zu wecken (EOS\_Put). In Verbindung mit dem TSK Subsystem wird die Funktion EOS\_Wake verwendet. Ohne TSK Subsystem bleibt der Eintrag leer.

## 4.2.3 MBX\_MAX

Diese Konstante steht in EOS\_MSG.C. Sie legt die Anzahl der Mailboxen fest, die mit EOS\_MbxCreate eingerichtet werden können. Die Mailboxen werden von 1 bis MBX\_MAX nummeriert.



*Für jede in IniTab spezifizierte Message-Länge wird von EOS eine zugehörige Mailbox (Home-Mailbox) eingerichtet.*

MBX\_MAX muss also so festgelegt werden, dass alle Home-Mailboxen angelegt werden können und darüber hinaus noch alle Mailboxen, die von der Applikation benötigt werden.

## 4.2.4 MSG\_TRC

Diese Konstante legt fest, ob die Put- und Get-Operationen des MSG Subsystems in EosErrLog (zusätzlich zu den Fehlern) protokolliert werden. Der Wert 0 schaltet die Protokollierung ab, alle anderen Werte schalten sie ein. Bei eingeschalteter Protokollierung sollte EosErrLog per EOS\_ERR\_LOG ausreichend groß dimensioniert werden. Diese Protokollierung (Trace) ist sehr hilfreich zur Fehlersuche, kostet aber etwas Performance des Systems (vgl. Beispiel).

## 4.2.5 USR\_MAX

Diese Konstante legt die Dimension der Tabelle UsrTab fest. Darin erfolgt für die User (1...EOS\_USR\_MAX) die Zählung der in ihrem Besitz befindlichen Messages und des zugehörigen Speichers sowie die Verwaltung des Speicherlimits (Bnd) pro User. Größere Usr-Werte sind erlaubt, es erfolgt dann aber nur eine globale Zählung ihrer Messages in UsrTab [0].

## 4.2.6 index\_t

Der Datentyp index\_t in EOS\_MSG.C ist definiert als unsigned short. Er dient zur Nummerierung der Messages und kann somit für die Indizierung von bis zu ca. 65500 Messages genutzt werden. Sollen mehr Messages via IniTab angelegt werden, muss index\_t als 32-Bit Typ (unsigned int) definiert werden.

## 4.2.7 IniTab

Diese Tabelle wird in EOS\_MSG.C initialisiert. Sie legt pro Zeile eine Länge und eine Anzahl an Messages mit dieser Länge fest. Der erste und der letzte Eintrag (-1) dürfen nicht verändert werden. Die Gesamtheit aller in IniTab definierten Messages wird in EOS\_MsgInit angelegt.

Die Längen sollten aufsteigend sortiert sein, weil EOS\_Alloc diese Tabelle vom Index Eins an durchläuft, um eine freie Message zu finden, deren Länge größer oder gleich der angeforderten Puffergröße ist.

Messages mit der Länge 0 sind möglich, können aber keine Information transportieren. Sie können zu Triggerzwecken oder als Semaphore benutzt werden.

Eine sinnvolle und angemessene Aufteilung der Messages in Längen und zugehörige Anzahlen kann nur applikationsspezifisch festgelegt werden. Manche Anwendungen benötigen viele kurze und viele lange Nachrichten, manche kommen mit einer einzigen Länge aus und andere benötigen eine ganze Palette verschiedener Längen (6 bis 8 Einträge sollten aber stets ausreichend sein).

*Mit der Anzahl der Einträge (Zeilen) in IniTab steigt der Suchaufwand in EOS\_Alloc, weil dort ein lineares Suchverfahren verwendet wird.*



*Pro Eintrag (Zeile) in IniTab wird eine Mailbox (die Home-Mailbox für Messages der betreffenden Länge) eingerichtet. Das ist in MBX\_MAX zu berücksichtigen.*



## 5 Das User Interface (API)

Zur Nutzung des MSG Subsystems müssen die Dateien EOS\_ERR.H und EOS\_MSG.H per #include in jede Datei der Applikation eingebunden werden, in der MSG Funktionen genutzt werden sollen, und die Dateien EOS\_ERR.C und EOS\_MSG.C müssen in das Projekt eingebunden werden.

Zu Beginn des Hauptprogramms wird das EOS System mit EOS\_ErrInit und EOS\_MsgInit initialisiert. Anschließend werden die Mailboxen eingerichtet mit EOS\_MbxCreate.

Eine anschließende Überprüfung von EosErrCnt sollte sicher stellen, dass bis hierher alles fehlerfrei abgelaufen ist (vgl. Beispiel). Andernfalls muss das System oder die Anwendung entsprechend angepasst oder korrigiert werden.

### 5.1 Funktionsaufrufe und Returncodes

Die MSG-Funktionen haben den Ergebnistyp int oder void\*.

- Die int-Funktionen liefern im Fehlerfall ein negatives Ergebnis zurück. Dies ist der negative Wert des zugehörigen Fehlercodes aus der Tabelle EOS\_ERR in EOS\_ERR.H. Null und positive Rückgabewerte zeigen einen erfolgreichen Aufruf an. Positive Ergebnisse haben zusätzlich eine inhaltliche Bedeutung, die von der jeweiligen Funktion abhängt und dort beschrieben ist.
- Die void\*-Funktionen liefern einen Message-Pointer oder EOS\_NULL.

Funktionsergebnisse sollten stets dort ausgewertet werden, wo aus inhaltlichen Gründen auch im normalen (fehlerfreien) Programmablauf differenziert werden muss. Dies betrifft z.B. die Funktion EOS\_Get, die wahlweise einen Message-Pointer liefert oder EOS\_Null, wenn die betreffende Mailbox leer war.

Echte Fehler (z.B. ungültiger Mbx-Parameter im EOS\_Put Aufruf) sollten nicht von der Applikation verfolgt werden, sondern zentral mit dem EOS Error Subsystem. Das entlastet die Applikation deutlich und hält sie übersichtlicher. Dies bedeutet keinen Verlust an Zuverlässigkeit, weil MSG selbst eine vollständige Fehlererkennung, Fehlerprotokollierung und Fehlerbehandlung enthält. Im Falle des ungültigen Mbx-Parameters z.B. wird die Message von MSG automatisch in ihre Home-Mailbox zurückgegeben.

## 5.2 Der EOS Nullpointer (EOS\_NULL)

EOS\_NULL ist das EOS Pendant zum Nullpointer NULL der C-Bibliothek. Dieser Wert wird von allen EOS-Funktionen, die einen Message-Pointer (void\*) als Ergebnis haben, zurückgeliefert, wenn die Funktion nicht erfolgreich war, d.h. keine Message vorhanden oder ein Parameter fehlerhaft war.

*Anstelle von EOS\_NULL darf keinesfalls NULL oder 0 oder (void\*)0 oder ähnliches verwendet werden, da EOS\_NULL mit keinem dieser Werte übereinstimmt.*



Wenn z.B. EOS\_Alloc eine Anforderung nicht erfüllen kann oder EOS\_Get eine leere Mailbox vorfindet, wird als Ergebnis EOS\_NULL zurückgeliefert. Ein Aufruf von EOS\_Put oder EOS\_Free mit EOS\_NULL als Message wird als Fehler erkannt.

EOS\_NULL bietet gegenüber NULL wesentliche Vorteile, da EOS\_NULL (im Gegensatz zu NULL) nicht wirklich den Wert 0 hat. EOS\_NULL ist ein Pointer auf einen Puffer, dessen Länge dem größten Längen-Eintrag in IniTab entspricht. Beim (fehlerhaften) Schreibzugriff auf Nullpointer werden also keine anderen Variablen oder Systemdaten überschrieben, sondern ein speziell dafür vorgesehener Puffer, und es werden keine Zugriffsrechte verletzt.

Softwaretechnisch ist EOS\_NULL keine echte Konstante, sondern ein Synonym für die Variable EosNull. Ein Überschreiben von EosNull wird von EOS bemerkt, korrigiert und als Fehler gemeldet.

## 6 EOS Fehlerhandling

Das EOS Fehlerhandling (ERR) wird sowohl vom EOS Message Subsystem (MSG) als auch vom EOS Task Subsystem (TSK) benutzt. Zusätzlich kann es vom Anwender (APP) genutzt werden, wenn er die Liste EOS\_ERR dazu entsprechend erweitert.

Das User-Interface (API) zum EOS Fehlerhandling besteht aus dem globalen Fehlerzähler EosErrCnt, der nach Fehlercodes aufgeschlüsselten Fehlertabelle EosErrTab, der nach Usern aufgeschlüsselten Fehlertabelle EosUsrErr, einem Ringpuffer EosErrLog als Fehler- und Trace-Logger und dem zugehörigen Schreibindex EosErrLogWr. Die Fehlercodes sind in der Tabelle EOS\_ERR aufgelistet.

Zur Fehlersuche sind die Einträge in der Log-Tabelle EosErrLog am wirkungsvollsten, da sie neben dem Fehlercode wichtige Zusatzinformationen enthalten. Es ist also, insbesondere während der Entwicklung, sehr sinnvoll, die Einträge in EosErrLog von einer eigenen Task online auf einem Bildschirm oder Drucker protokollieren zu lassen.

Während der Entwicklung sollte die Applikation bei einem Fehler sofort angehalten werden (z.B. durch einen Aufruf direkt aus EOS\_Error heraus), um die Ursache unmittelbar erkennen und beheben zu können. Andernfalls kann es zu Folgefehlern kommen, die für die Erkennung der eigentlichen Fehlerursache eher hinderlich sind. Die zusätzliche Aktivierung der MSG Trace Funktion (MSG\_TRC) kann bei der Fehlersuche sehr hilfreich sein.

Die in EOS\_ERR definierten Fehlercodes sind in einem eigenen Kapitel beschrieben in Verbindung mit möglichen Ursachen und Gegenmaßnahmen.

## 6.1 eos\_err\_log

Diese Datenstruktur wird zur Aufzeichnung von Fehlern und Trace-Ereignissen innerhalb des Ringpuffers EosErrLog benutzt.

```
typedef struct
{
    unsigned int    Tim;           /* event time (scheduler time) */
    unsigned short  Cnt;          /* error counter                */
    short           Err;          /* error code from list EOS_ERR */
    short           Usr;          /* actual usr when event occured */
    short           Mbx;          /* involved mailbox             */
    void*           Msg;          /* involved message             */
}
eos_err_log;                  /* error logging descriptor */
/*
```

- Tim ist die Zeit, zu der das Ereignis eingetreten ist.
- Cnt ist ein laufender Zähler. Lücken in Cnt zeigen einen Überlauf in EosErrLog an.
- Err ist die Nummer des Ereignisses gemäß der Tabelle EOS\_ERR.
- Usr ist der User (Komponente), der den Fehler bzw. das Ereignis verursacht hat.
- Mbx ist die betroffene Mailbox
- Msg ist die betroffene Message

In Einzelfällen können die Komponenten auch eine etwas andere Information beinhalten. Genaue Auskunft gibt der Sourcecode.

## 6.2 EosErrCnt

Dies ist der globale EOS Fehlerzähler. Er wird bei jedem Fehler um 1 erhöht. Bei Bedarf kann der Zähler vom Anwenderprogramm zwischenzeitlich auf 0 gesetzt werden.

## 6.3 EosErrTab [EOS\_ERR\_TAB]

Diese Tabelle enthält individuelle Zähler für jeden in der Liste EOS\_ERR definierten Fehler. Der Eintrag 0 (EOS\_ALLOC\_FIT) zählt nicht als Fehler und ist folglich nicht in EosErrCnt mitgezählt. Bei Bedarf können diese Zählwerte vom Anwendungsprogramm zwischenzeitlich auf 0 gesetzt werden.

## 6.4 EosUsrErr [EOS\_USR\_ERR + 1]

Diese Tabelle enthält individuelle Fehler-Zähler für die User 1 bis EOS\_USR\_ERR. Für alle anderen User werden die Fehler global in EosUsrErr [0] summiert. Bei Bedarf können diese Zählwerte vom Anwendungsprogramm zwischenzeitlich auf 0 gesetzt werden.

## 6.5 EosErrLog [EOS\_ERR\_LOG]

In diesem Ringpuffer werden Fehler und gegebenenfalls MSG Trace Ereignisse eingetragen. Neue Einträge werden durch den Schreibindex EosErrLogWr angezeigt. Die übliche Auswertung sieht wie folgt aus.

```
while (MyErrLogRd != EosErrLogWr)
{
    /* Eintrag mit Index MyErrLogRd auswerten */
    MyErrLogRd = (MyErrLogRd + 1) % EOS_ERR_LOG;
}
```

## 6.6 EosErrLogWr

Aktueller EOS Schreibindex im Ringpuffer EosErrLog (siehe dort).

## 6.7 EOS\_ErrInit (void)

Diese Funktion initialisiert das ERR Subsystem. Sie sollte als erste Funktion aufgerufen werden, um die Zählwerte auf 0 zu setzen. Sie kann auch zwischenzeitlich aufgerufen werden, um die Fehlerzähler EosErrCnt, EosErrTab und EosUsrErr gemeinsam auf 0 zurückzusetzen.

## 6.8 EOS\_Err (int Err, int Usr, int Mbx, void\* Msg)

Mit dieser Funktion wird ein neuer Fehler gemeldet. Er wird in EosErrCnt, EosErrTab und EosUsrErr gezählt und in EosErrLog eingetragen. Diese Funktion wird von den Subsystemen MSG und TSK genutzt und steht auch dem Anwendungssystem (APP) zur Verfügung.

## 6.9 EOS\_TRACE (Trc, Usr, Mbx, Ptr)

Dieses Macro nimmt einen Eintrag in EosErrLog vor. Es wird von dem MSG Subsystem für die MSG Trace Funktion genutzt und steht auch dem Anwendungssystem (APP) zur Verfügung. Die Parameter werden in die Komponenten Err, Usr, Mbx und Msg der Datenstruktur eos\_err\_log eingetragen. Für Trc sollten von APP deshalb Werte verwendet werden, die nicht als EOS Fehlercodes vorkommen.

## 7 EOS Messages und Mailboxen

Das EOS Message Subsystem (MSG) ist das Herzstück der EOS Laufzeitumgebung zur Speicherverwaltung und zur Kommunikation zwischen Software-Komponenten. MSG nutzt die Funktionen des EOS Fehlerhandlings (ERR) und kann eigenständig (standalone) oder in Verbindung mit dem TSK Subsystem eingesetzt werden. Es gibt viele Anwendungen, in denen ein standalone-Betrieb des MSG Subsystems sinnvoll ist.

Zur Strukturierung der Software in einzelne, überschaubare Komponenten, die über Mailboxen miteinander kommunizieren, unterstützt MSG das Konzept mehrerer User. Ein solcher User kann eine Funktion sein, ein Modul oder eine auf andere Art in sich abgeschlossene Komponente des Anwendungs-Systems. Für jeden User werden von MSG die in seinem Besitz befindlichen Messages und von ERR seine Fehler separat gezählt. In Verbindung mit dem TSK-Subsystem ist es sicherlich sinnvoll, die TSK Tasks mit den MSG Usern gleichzusetzen. Deshalb verwendet TSK eine dementsprechend modifizierte Implementierung von MSG.

`EOS_Alloc` und `EOS_Free` sind aus Sicht der Anwendung identisch mit den Funktionen `malloc` und `free` der C-Standardbibliothek zur Verwaltung des dynamischen Speichers auf dem Heap. Die Parameter haben die gleiche Bedeutung und die Funktionen liefern die gleichen Ergebnisse.

In EOS wird aber nicht mit dem Heap des C-Laufzeitsystems gearbeitet, sondern mit einem EOS eigenen Message-Pool. Das ist ein zusammenhängendes Stück Speicher, das von `EOS_MsgInit` angelegt und in einzelne Puffer-Bereiche (Messages, gemäß `IniTab`) aufgeteilt wird. Gegenüber dem Heap ist der Message-Pool von EOS schneller, hat deterministische Zugriffszeiten, ist wesentlich sicherer und ist zuverlässiger, da er keine Fragmentierung kennt und somit keine Defragmentierung benötigt.

Die im Folgenden angegebenen Rückgabewerte der MSG Funktionen beziehen sich immer auf den fehlerfreien Fall. Die Returncodes im Fehlerfall sind im Kapitel Fehlercodes beschrieben.

### 7.1 EOS\_NULL (EosNull)

`EOS_NULL` ist das EOS Pendant zum Nullpointer `NULL` der C-Bibliothek. Dieser Wert wird von allen EOS-Funktionen, die einen Message-Pointer als Ergebnis haben, zurückgeliefert, wenn die Funktion nicht erfolgreich war, d.h. keine Message vorhanden oder ein Parameter fehlerhaft war.

*Anstelle von `EOS_NULL` darf keinesfalls `NULL` oder `0` oder `(void*)0` oder ähnliches verwendet werden, da `EOS_NULL` mit keinem dieser Werte übereinstimmt.*





EOS\_NULL ist ein Pointer auf einen Puffer, dessen Länge der größten Länge in IniTab entspricht. Beim (fehlerhaften) Schreibzugriff auf Nullpointer werden also keine Systemdaten überschrieben, sondern ein speziell dafür vorgesehener Puffer, und es werden keine Zugriffsrechte verletzt.

Softwaretechnisch ist EOS\_NULL keine echte Konstante, sondern ein Synonym für die Variable EosNull.

## 7.2 EosMsgClr

Diese Variable legt fest, ob die Funktion EOS\_Alloc alle Datenpuffer vor der Auslieferung mit dem Wert 0 initialisiert. Mit EosMsgClr = 0 (Defaultwert) ist bzw. wird die Initialisierung ausgeschaltet, andernfalls ist bzw. wird sie eingeschaltet. Sie kann jederzeit ein- oder ausgeschaltet werden. Wenn EosMsgClr bereits vor dem Aufruf von EOS\_MsgInit auf 1 gesetzt wird, wird der gesamte MSG Message-Pool vorab mit 0 initialisiert.

Das Initialisieren der Messages erleichtert die Fehlersuche und verhindert nicht-deterministische Effekte beim Zugriff auf undefinierte Bereiche einer Message, kostet aber einige Performance (vgl. Beispiel).

Auch in Kommunikationsanwendungen kann das Initialisieren der Messages sinnvoll sein, um im Fehlerfall ein Versenden kritischer Informationen (z.B. Passwörter) zu verhindern.

## 7.3 EOS\_MsgInit (void)

Diese Funktion initialisiert alle Variablen des MSG Subsystems und richtet die in IniTab festgelegten Messages ein. Sie darf nur einmal aufgerufen werden und ist die Voraussetzung für alle anderen MSG Aufrufe.

## 7.4 EOS\_MbxCreate (unsigned Own)

Diese Funktion richtet eine neue Mailbox ein. Own legt einen User als Besitzer der Mailbox fest.

Die Funktion liefert den Identifier (Nummer) der Mailbox zurück.

Nur der Besitzer einer Mailbox darf aus ihr lesen (EOS\_Get). Mailboxen mit dem Besitzer 0 (z.B. die Home-Mailboxen der Messages) sind öffentliche Mailboxen. Aus ihnen darf jeder lesen. Alle von der Applikation eingerichteten Mailboxen sollten deshalb nicht öffentlich sein, weil dann alle Lesezugriffe anderer User auf diese Mailbox von MSG als Fehler erkannt und gemeldet werden.

## 7.5 EOS\_UsrBnd (int Usr, int Bnd)

Diese Funktion legt für den User Usr ein Speicherlimit für allozierte Messages fest, dass von EOS\_Alloc nicht überschritten wird. Per Default sind diese Grenzen alle auf maximalen Wert gesetzt, d.h. eine Begrenzung ist nicht wirksam.

Wenn das mit `EOS_UsrBnd` neu eingetragene Limit (`Bnd`) den aktuellen Verbrauch (`Mem`, vgl. `EOS_UsrInfo`) überschreitet, werden keine weiteren Maßnahmen ergriffen, neue Messages können aber erst wieder nach unterschreiten der Grenze `Bnd` alloziert werden.

## 7.6 EOS\_Alloc (unsigned Siz, int Usr)

Es wird ein Pufferbereich (Message) der Länge `Siz` von der Anwendungs-Komponente `Usr` angefordert. `Siz` ist dabei die Anzahl der Bytes, die der Anwender für seine Nutzdaten benötigt. Die Message darf später nur von der Anwendungs-Komponente `Usr` zurückgegeben (`Put`, `Free`) werden.

Die Funktion liefert einen Pointer auf die Message oder `EOS_NULL`, wenn kein Puffer ausreichender Länge verfügbar war.

## 7.7 EOS\_Free (void\* Msg, int Usr)

Diese Funktion gibt eine zuvor von der Komponente `Usr` per `Alloc` oder `Get` erhaltene `Msg` an die EOS `Msg`-Verwaltung (Pool) zurück.

Die Funktion liefert 0 zurück.

## 7.8 EOS\_Put (int Mbx, void\* Msg, int Usr)

Diese Funktion legt eine zuvor von der Anwendungs-Komponente `Usr` per `Alloc` oder `Get` erhaltene `Msg` in der Mailbox `Mbx` ab. Die Mailbox darf keine Home-Mailbox sein.

Die Funktion liefert 0 zurück..

## 7.9 EOS\_Get (int Mbx, int Usr)

Die Anwendungs-Komponente `Usr` möchte eine Message aus der Mailbox `Mbx` entnehmen. Dazu muss `Usr` Besitzer (Owner) der Mailbox sein, sofern die Mailbox nicht öffentlich ist.

Die Funktion liefert einen Message Pointer oder `EOS_NULL`, wenn die Mailbox leer war.

## 7.10 EOS\_MbxPurge (int Mbx, unsigned Cnt, ...)

Diese Funktion entfernt Messages aus der Mailbox `Mbx` (per `Get` und `Free`), solange sich mehr als `Cnt` Messages in `Mbx` befinden. Wenn der letzte Lesezugriff auf `Mbx` länger als `Tim` Zeiteinheiten zurückliegt, werden alle Messages aus `Mbx` entfernt (da sie veraltet sind). Mit `Tim < 0` wird die Zeitbedingung deaktiviert. Jede entfernte Message wird als Fehlerereignis gemeldet.

Die Funktion liefert als Ergebnis die Anzahl der entfernten Messages.

Diese Funktion repariert die Folgen vorangehender Probleme innerhalb der Applikation, indem sie die übermäßige Kumulation von Messages in einer Mailbox auflöst und dadurch die langfristige Verfügbarkeit des Systems erhält. Da es sich um einen recht massiven Eingriff in das System (Notbetrieb) handelt, sollten die Parameter Cnt und Tim deutlich über den Werten im Normalbetrieb liegen. Das vorrangige Ziel muss immer die Vermeidung der ursprünglichen Probleme sein.

MbxPurge kann von den einzelnen Anwendungs-Komponenten für deren eigene Mailboxen eingesetzt werden (dezentrale Überwachung) oder von einer dedizierten Überwachungs-Komponente für alle Mailboxen (zentrale Überwachung). Die dezentrale Überwachung kann oft genauer auf die aktuelle Situation abgestimmt werden, die zentrale Überwachung ist leichter wartbar.

## 7.11 EOS\_MsgPurge (int Usr)

Diese Funktion gibt alle Messages, die sich im Besitz der Anwendungs-Komponente Usr befinden, an den EOS Message-Pool zurück. Dies sind Messages, die von Usr per Alloc oder Get aus dem Pool entnommen und bisher nicht wieder zurückgegeben wurden. Jede entfernte Message wird als Fehlerereignis gemeldet.

Die Funktion liefert als Ergebnis die Anzahl der entfernten Messages.

Die Funktion MsgPurge repariert die Folgen zuvor aufgetretener Fehler, indem sie den endgültigen Verlust von Puffern (Memory Leak) aus dem Message-Pool verhindert und dadurch die langfristige Verfügbarkeit des Systems erhält. Das vorrangige Ziel muss immer die Vermeidung der ursprünglichen Fehler sein.

MsgPurge sollte nur aufgerufen werden, wenn zuvor (z.B. per EosUsrMsg) festgestellt wurde, dass die betreffende Komponente mehr Messages in ihrem Besitz hat, als im Software-Konzept vorgesehen ist. Da es sich um einen recht massiven Eingriff in das System (Notbetrieb) handelt (und die gesamte Message-Tabelle inspiziert werden muss), sollte die Aufrufbedingung (EosUsrMsg) deutlich über den Werten im Normalbetrieb liegen.

MsgPurge kann von den einzelnen Anwendungs-Komponenten für sich selbst eingesetzt werden (dezentrale Überwachung) oder von einer dedizierten Überwachungs-Komponente für alle Komponenten (zentrale Überwachung). Die dezentrale Überwachung kann oft genauer auf die aktuelle Situation abgestimmt werden, die zentrale Überwachung ist leichter wartbar.

## 7.12 EOS\_MbxInfo (int Mbx, int \* Cnt, int \* Bnd, ...)

Diese Funktion liefert alle in der Parameterliste angegebenen Informationen über die Mailbox Mbx. Die Bedeutung der Informationen ergibt sich aus der Datenstruktur msg\_dsc.

Als Ergebnis liefert die Funktion 0 oder die Nummer der wartenden Task.

## 7.13 EOS\_MsgInfo (void\* Msg, int\* Len, int\* Usr, ...)

Diese Funktion liefert alle in der Parameterliste angegebenen Informationen über die Message mit der Adresse Msg, insbesondere auch deren Index (Idx). Mit dem Index können gezielt vorhergehende und nachfolgende Messages im Pool untersucht werden, z.B. nach einem Datenfehler (Overflow oder Underflow). Die Bedeutung der Informationen ergibt sich aus der Datenstruktur msg\_dsc.

Als Ergebnis liefert die Funktion die Home-Mailbox der Message.

## 7.14 EOS\_UsrInfo (int Usr, int\* Mem, int\* Bnd)

Diese Funktion liefert alle in der Parameterliste angegebenen Informationen über den User mit der Nummer Usr (0...USR\_MAX). Die Bedeutung der Informationen ergibt sich aus der Datenstruktur usr\_dsc.

Als Ergebnis liefert die Funktion die Anzahl der von Usr zur Zeit allozierten Messages (Komponente Cnt in usr\_dsc).

Dieses Funktionsergebnis kann zur Prüfung der Systemkonsistenz (verlorene, d.h. nicht zurückgegebene Messages) genutzt werden, insbesondere in Verbindung mit EOS\_MsgPurge (vgl. Beispiel).

In der zugehörigen Tabelle (UsrTab) werden die Messages gezählt, die sich zur Zeit im Besitz der User Eins bis EOS\_USR\_MSG befinden, für alle anderen User werden die Messages global in UsrTab[0] gezählt.

## 7.15 EOS\_IdxInfo (int Idx, int\* Len, int\* Usr, ...)

Diese Funktion liefert alle in der Parameterliste angegebenen Informationen über die Message mit dem Index Idx. Die Bedeutung der Informationen ergibt sich aus der Datenstruktur msg\_dsc.

Als Ergebnis liefert die Funktion die Home-Mailbox der Message.

## 8 Ein Beispiel

Das folgende Beispiel enthält ein kleines Applikationssystem ohne inhaltlich ausgestaltete Funktionalität mit einigen Komponenten (Funktionen), die über Mailboxen miteinander kommunizieren.

- Komponente 1 produziert Messages. Das kann z.B. ein Hardware-Handler sein, der Zeichen per Interrupt empfängt und sie paketweise weiterleitet. Die Daten-Pakete werden via Mailbox 2 an Komponente 2 weitergeleitet. Die Komponente benötigt keine Empfangs-Mailbox.
- Komponente 2 bearbeitet Daten-Pakete. Dazu alloziert sie einen neuen (größeren) Puffer und transformiert die Information aus der empfangenen Message in den neuen Puffer. Anschließend wird das empfangene Paket in den Pool zurückgegeben und die neue Message via Mailbox 5b an Komponente 5 geschickt.
- Komponente 3 arbeitet zyklisch in jeder Zeiteinheit. Sie sendet Nachrichten (Statusabfragen) an Komponente 5 via Mailbox 5a und gibt als Adresse für die Antwort ihre Mailbox Mb3 an.
- Komponente 4 arbeitet zyklisch alle 2 Zeiteinheiten. Sie sendet Nachrichten (Kommandos) an Komponente 5 via Mailbox 5a. Komponente 4 will auf ihre Kommandos keine Antwort und gibt als Absender Mailbox 0 (Synonym für Home-Mailbox) in der Message an. Mailbox 4 ist deshalb zur Zeit ungenutzt.
- Komponente 5 verarbeitet Nachrichten aus Mailbox 5a und Daten-Pakete aus Mailbox 5b. Datenpakete werden anschließend an den Pool zurückgegeben, Nachrichten werden an die in der Nachricht genannte Adresse (Mailbox) beantwortet. Dies kann z.B. ein Hardware-Handler sein, der Daten-Pakete überträgt, per Kommando den Controller konfiguriert und auf Anfrage Statusinformationen liefert.
- Komponente 6 ist eine zentrale Überwachungs-Komponente für alle Mailboxen und User.

```

/*****
* Copyright(C) : S M A Regelsysteme GmbH, 34266 Niestetal, Germany
*****/
* project      : EOS embedded operating system
*****/
* filename     : MAIN.C
*****/
* description  : demonstration program for MSG subsystem
*****/
* history     : author  date    ver / description
*
*****/
*              Frees   20.09.03 1.0 / creation
*****/

```

```
#include <stdio.h>
#include <time.h>

#include "EOS_MSG.H"
#include "EOS_ERR.H"

/*****

int Cnt1 [8];                /* component loop counter    */
int Cnt2 [8];                /* component msg  counter    */

/*****
/*
the MSG mailboxes
*/
int Mbx2;
int Mbx3;
int Mbx4;
int Mbx5a;
int Mbx5b;
/*
the system time
*/
unsigned EosTim = 0;

/*****
/* the application components */
/*****

void Tl (void)
/*
high speed tsk (producer)
*/
{
void* Msg;

Cnt1[1]++;
/*
alloc, fill, put
*/
Msg = EOS_Alloc (50, 1);
```

```

    /* ... */
    EOS_Put (Mbx2, Msg, 1);
    Cnt2[1]++;
    /*
    alloc, fill, put
    */
    Msg = EOS_Alloc (50, 1);
    /* ... */
    EOS_Put (Mbx2, Msg, 1);
    Cnt2[1]++;
}

/*****

void T2 (void)
    /*
    event driven by one mbx (transformer)
    */
{
    void* Msg1;
    void* Msg2;

    while (1)
    {
        Cnt1[2]++;
        /*
        get, alloc, transform, put, free
        */
        Msg1 = EOS_Get (Mbx2, 2);
        if (Msg1 == EOS_NULL) return;
        Msg2 = EOS_Alloc (100, 2);
        /* ... */
        EOS_Free (Msg1, 2);
        EOS_Put (Mbx5b, Msg2, 2);
        Cnt2[2] += 2;
    }
}

/*****

void T3 (void)
    /*

```

```

    cyclic tsk (producer/consumer)
    */
{
    static unsigned MyTim = 0;
    int* Msg;

    while ((int)(MyTim - EosTim) <= 0)
    {
        MyTim += 1;
        Cnt1[3]++;
        /*
        alloc, fill, put
        */
        Msg = EOS_Alloc (10, 3);
        Msg[0] = Mbx3;                /* send the answer to mbx3 */
        /* ... */
        EOS_Put (Mbx5a, Msg, 3);
        Cnt2[3]++;
        /*
        get, read, free the answers
        */
        while (1)
        {
            Msg = EOS_Get (Mbx3, 3);
            if (Msg == EOS_NULL) break;
            /*...*/
            EOS_Free (Msg, 3);
            Cnt2[3]++;
        }
    }
}

/*****

void T4 (void)
    /*
    cyclic tsk (producer)
    */
{
    static unsigned MyTim = 0;
    int* Msg;

```



```

while ((int)(MyTim - EosTim) <= 0)
{
    MyTim += 2;
    Cnt1[4]++;
    /*
    alloc, fill, put
    */
    Msg = EOS_Alloc (15, 4);
    Msg[0] = 0;                      /* no answer please */
    /* ... */
    EOS_Put (Mbx5a, Msg, 4);
    Cnt2[4]++;
}
}

/*****

void T5 (void)
/*
event driven by two mbxs (consumer)
*/
{
    int  Busy = 1;
    int  Mbx;
    int* Msg;

    while (Busy)
    {
        Cnt1[5]++;
        Busy = 0;
        /*
        first mbx (commands)
        get, handle, free
        */
        Msg = EOS_Get (Mbx5a, 5);
        if (Msg != EOS_NULL)
        {
            Mbx = Msg[0];
            /* ... */
            EOS_Put (Mbx, Msg, 5);
            Cnt2[5]++;
            Busy = 1;
        }
    }
}

```

```

    }
    /*
    second mbx (data)
    get, handle, free
    */
    Msg = EOS_Get (Mbx5b, 5);
    if (Msg != EOS_NULL)
    {
        /* ... */
        EOS_Free (Msg, 5);
        Cnt2[5]++;
        Busy = 1;
    }
}

}

/*****

void T6 (void)
/*
msg supervisor
*/
{
    static unsigned MyTim = 0;
    int Mem;
    int Bnd;

    while ((int)(MyTim - EosTim) <= 0)
    {
        MyTim += 50;
        Cnt1[6]++;
        /*
        look for cumulation and timing problems within APP mbxs
        */
        EOS_MbxPurge (Mbx2, 10, 100);
        EOS_MbxPurge (Mbx3, 15, 100);
        EOS_MbxPurge (Mbx4, 0, 0);    /*not used */
        EOS_MbxPurge (Mbx5a, 10, 100);
        EOS_MbxPurge (Mbx5b, 10, 100);
        /*
        look for lost msgs within APP components
        */

```

```

        if (EOS_UsrInfo (1, &Mem, &Bnd) > 10) EOS_MsgPurge (1);
        if (EOS_UsrInfo (2, &Mem, &Bnd) > 10) EOS_MsgPurge (2);
        if (EOS_UsrInfo (3, &Mem, &Bnd) > 10) EOS_MsgPurge (3);
        if (EOS_UsrInfo (4, &Mem, &Bnd) > 10) EOS_MsgPurge (4);
        if (EOS_UsrInfo (5, &Mem, &Bnd) > 10) EOS_MsgPurge (5);
    }
}

/*****

int main (void)
{
    unsigned Tlm1;                /* start time          */
    int i;

    /*
    init
    */
    for (i = 0; i < 8; i++) Cntl[i] = Cnt2[i] = 0;
    /*
    init EOS system
    */
    EosMsgClr = 0;
    EOS_ErrInit();
    EOS_MsgInit();
    /*
    create all mailboxes
    */
    Mbx2  = EOS_MbxCreate (2);
    Mbx3  = EOS_MbxCreate (3);
    Mbx4  = EOS_MbxCreate (4);
    Mbx5a = EOS_MbxCreate (5);
    Mbx5b = EOS_MbxCreate (5);
    /*
    check for errors
    */
    if (EosErrCnt != 0) printf ("Errors: %i ", EosErrCnt);
    /*
    run application (for 1 sec)
    */
    printf ("\n\rSTART\n\r");

```

```
Tim1    = clock();
EosTim  = Tim1;

while ((EosTim - Tim1) <= 1000)
{
    T1();
    T2();
    T3();
    T4();
    T5();
    T6();
    EosTim = clock();
    Cnt1[0]++;
}

printf ("\n\rSTOP %i\n\r", Tim2 - Tim1);

/*
print results
*/
printf ("\n\r");
if (EosErrCnt != 0) printf ("Errors: %i\n\r", EosErrCnt);

for (i = 0; i <= 6; i++)
{
    printf ("%i %8i %8i\n\r", i, Cnt1[i], Cnt2[i]);
}
}
/***** EOF *****/
```

**Auf einem Pentium IV System mit 1,8 GHz unter Windows 2000 werden in 1 Sekunde folgende Werte erreicht.**

0	838774	0
1	838774	1677548
2	2516322	3355096
3	1002	1993
4	501	501
5	2517627	1679053
6	21	0

**Insgesamt handelt es sich hierbei um 6.713.202 vollständige Message-Operationen mit Alloc/Get und Put/Free.**

**Mit eingeschalteter Message Trace Funktion werden diese Werte erreicht.**

0	718741	0
1	718741	1437482
2	2156223	2874964
3	993	1974
4	497	497
5	2157513	1438972
6	20	0

**Mit eingeschalteter Initialisierung der Pufferbereiche werden diese Werte erreicht.**

0	586680	0
1	586680	1173360
2	1760040	2346720
3	993	1974
4	497	497
5	1761330	1174850
6	20	0

**Mit eingeschalteter Message Trace Funktion und eingeschalteter Initialisierung der Pufferbereiche werden diese Werte erreicht.**

0	531758	0
1	531758	1063516
2	1595274	2127032
3	1002	1993
4	501	501
5	1596577	1065019
6	21	0

## 9 Fehlercodes, Ursachen, Fehlerbehebung

In diesem Kapitel sind alle MSG Fehlercodes beschrieben und es werden Hinweise zur möglichen Fehlerursache und zur Fehlerbeseitigung gegeben. Alle Fehlercodes sind in der Tabelle EOS\_ERR in der Datei EOS\_ERR.H definiert.

### 9.1 MSG Subsystem

Die Meldungen des MSG-Subsystems sind nach Funktionen bzw. Funktionsgruppen gegliedert.

#### 9.1.1 EOS\_Alloc

##### **EOS\_ALLOC\_FIT**

Dies ist kein Fehlercode, sondern der Index 0, unter dem in EosErrTab gezählt wird, wie oft eine Alloc-Anforderung aus einer Home-Mailbox mit ausreichender Länge nicht erfüllt werden konnte, weil diese leer war. Einträgen unter diesem Index kann durch entsprechende Anpassungen in IniTab begegnet werden. In manchen Anwendungen kann es sinnvoll sein, die Alloc-Strategie zu ändern und bei leeren Home-Mailboxen nicht auf Home-Mailboxen mit größeren Puffern zurückzugreifen.

##### **EOS\_ALLOC\_SIZ**

Die angeforderte Puffergrösse überschreitet die größte Längenangabe in IniTab. Korrigieren Sie den Alloc-Aufruf oder passen Sie IniTab an.

##### **EOS\_ALLOC\_USR**

Der übergebene Parameter muss größer als Null sein. Überprüfen Sie den Aufruf. Im Programm sollte eine Konstante oder eine einfache Variable verwendet werden.

##### **EOS\_ALLOC\_BND**

Der insgesamt von Usr allozierte Speicher (Mem) würde die Grenze (Bnd, vgl. EOS\_UsrBnd) überschreiten. In Mem wird nicht die angeforderte Länge, sondern die interne Länge der Message gezählt. Die Komponente muss ihre Messages schneller wieder frei geben oder die Message-Längen müssen in IniTab besser an die angeforderten Längen angepasst werden.

##### **EOS\_ALLOC\_NUL**

Es ist kein Puffer mehr vorhanden, der (mindestens) die angeforderte Länge hat. Passen Sie (per IniTab) den Message-Pool an (Länge oder Anzahl oder beides) oder geben Sie angeforderten Speicherplatz schneller wieder frei. Ermitteln Sie anhand von EosErrTab [EOS\_ALLOC\_FIL] und der Statistik-Werte Cnt und Bnd in den Home-Mailboxen, ob zu jeder Länge ausreichend viele Messages reserviert sind.

Prüfen Sie per EosMsgCnt und der Statistikwerte der Mailboxen (Cnt, Tim), ob Messages verlorengehen oder in Mailboxen festsitzen, und führen Sie diese gegebenenfalls mit EOS\_MsgPurge und EOS\_MbxPurge dem Pool wieder zu.

Eventuell ist es sinnvoll, die Alloc-Strategie zu ändern und bei fehlenden kurzen Messages nicht auf längere zurückzugreifen.

## 9.1.2 EOS\_Put

Diese Funktion ist die sensibelste im EOS System. Die übergebenen Msg-Pointer und auch die anderen Parameter können beliebig falsch sein. Alle Parameter werden deshalb sorgfältig geprüft und gegebenenfalls angepasst, um Schutzverletzungen auszuschließen, die Konsistenz der Message-Pool Verwaltung sicherzustellen und Datenverluste (Memory Leak) so weit wie möglich zu vermeiden.

### EOS\_PUT\_MSG\_LOW

Der übergebene Msg-Parameter ist zu klein, d.h. kleiner als die Startadresse des Message-Pools, und wurde verworfen. Eventuell wurde der Pointer verändert oder nicht von einem Alloc- oder Get-Aufruf geliefert.

### EOS\_PUT\_MSG\_NUL

Der übergebene Msg-Parameter ist die Adresse des EOS Null-Pointers (EOS\_NULL). Vermutlich hat ein vorausgehender Alloc- oder Get-Aufruf EOS\_NULL geliefert. Testen Sie das Ergebnis von Get auf EOS\_NULL. Testen Sie das Ergebnis von Alloc auf EOS\_NULL oder passen Sie IniTab an, wenn Sie davon ausgehen, dass Alloc stets einen gültigen Puffer liefert.

### EOS\_PUT\_MSG\_HIG

Der übergebene Msg-Parameter ist zu groß, d.h. größer als die Endadresse des Message-Pools, und wurde verworfen. Eventuell wurde der Pointer verändert oder nicht von einem Alloc- oder Get-Aufruf geliefert.

### EOS\_PUT\_MSG\_INV

Der übergebene Msg-Parameter ist keine gültige Msg-Adresse, d.h. kommt innerhalb des Pools nicht als Msg-Adresse vor, und wurde verworfen. Eventuell wurde der Pointer verändert oder nicht von einem Alloc- oder Get-Aufruf geliefert.

### EOS\_PUT\_PUT

Die Message befindet sich bereits in einer Mailbox. Vermutlich wurde sie zuvor schon einmal zurückgegeben. Überprüfen Sie ihr Software-Konzept daraufhin, ob stets klar ist, wann welche Funktion innerhalb der Anwendungs-Komponente für die Weiterleitung einer Msg zuständig ist. Idealerweise finden Alloc/Get und Put/Free in der gleichen Funktion statt (Transparenz des Datenflusses). Zwischengeschaltete Unter-Funktionen teilen als Ergebnis mit, wohin die Msg weitergeleitet oder ob sie zurückgegeben werden soll (dies kann als Mailbox 0 mitgeteilt werden).

### EOS\_PUT\_USR

Der übergebene Parameter muss größer als Null sein. Die Msg wurde in ihre Home-Mbx umgeleitet. Überprüfen Sie den Aufruf. Im Programm sollte eine Konstante oder eine einfache Variable verwendet werden.

## **EOS\_PUT\_ACC**

Der im Parameter angegebene User ist nicht der intern (MsgTab) eingetragene User, der den zugehörigen Alloc- bzw. Get-Aufruf durchgeführt hat. Die Msg wurde in ihre Home-Mbx umgeleitet. Prüfen Sie die User-Parameter im Alloc/Get- und im Put-Aufruf. Es sollte eine Konstante oder eine einfache Variable verwendet werden. Die direkte Weitergabe einer Msg von einem User an einen anderen ist nicht zulässig.

## **EOS\_PUT\_DAT\_ACT**

Der Index-Eintrag im Pool unmittelbar vor dem Nutzdaten-Bereich der Message ist überschrieben und wurde korrigiert. Entweder hat der User selbst Einträge vor der Msg-Adresse vorgenommen (Underflow) oder ein User einer der vorhergehenden Messages hat über seine Message hinaus geschrieben (Buffer Overflow).

Da der (eigene) Index in jedem Alloc/Get-Aufruf geprüft (und gegebenenfalls korrigiert) wird, kann ein Overflow nur eintreten, wenn zwischen Alloc/Get und Put ein Kontextwechsel liegt. Dies sollte, wenn es aus zeitlichen Gründen nicht zwingend erforderlich ist, in der Regel vermieden werden (Alloc/Get - Verarbeitung - Put/Free - return).

Zur Fehleranalyse sollte der Message-Pool mit MsgInfo und IdxInfo im Detail untersucht werden und zusätzlich die MSG-Trace-Funktion (MSG\_TRC) aktiviert werden. Dazu muss so zeitnah wie möglich (am besten unmittelbar aus EOS\_Error heraus) auf den Fehler reagiert werden.

## **EOS\_PUT\_DAT\_NXT**

Der Index-Eintrag im Pool unmittelbar hinter dem Nutzdaten-Bereich der Message (d.h. der Index der nachfolgenden Msg) ist überschrieben und wurde korrigiert. Vermutlich hat der User der an Put übergebenen Message über seinen Datenbereich hinaus geschrieben (Buffer Overflow).

Zur Fehleranalyse sollte der Message-Pool mit MsgInfo und IdxInfo im Detail untersucht werden und zusätzlich die MSG-Trace-Funktion (MSG\_TRC) aktiviert werden. Dazu muss so zeitnah wie möglich (am besten unmittelbar aus EOS\_Error heraus) auf den Fehler reagiert werden.

## **EOS\_PUT\_MBX**

Der Mbx-Parameter ist größer als der größte gültige Mailbox-Identifizierer oder kleiner als Null. Die Msg wurde stattdessen in ihrer Home-Mbx abgelegt. Im Programm sollte für Mbx eine Konstante oder eine einfache Variable verwendet werden.

## **EOS\_PUT\_HOM**

Der Mbx-Parameter verweist auf eine Home-Mailbox. Diese dürfen aber nicht direkt angesprochen werden, sondern nur über Free (oder den Mbx-Parameter 0). Das System hat die Msg in der zuständigen Home-Mbx abgelegt.



## 9.1.3 EOS\_Get

### EOS\_GET\_MBX

Der Mbx-Parameter ist größer als der größte gültige Mailbox-Identifizierer oder kleiner als 0. Im Programmcode sollte eine Konstante oder eine einfache Variable verwendet werden.

### EOS\_GET\_USR

Der übergebene Parameter muss größer als Null sein. Es sollte eine Konstante oder eine einfache Variable verwendet werden.

### EOS\_GET\_OWEN

Die Mailbox ist nicht öffentlich und der User ist nicht Owner der Mailbox. Prüfen Sie die Mbx- und Usr-Parameter im Get-Aufruf sowie den Own-Parameter im MbxCreate-Aufruf. Es sollte eine Konstante oder eine einfache Variable verwendet werden.

### EOS\_GET\_DAT

Der Index-Eintrag im Pool unmittelbar vor dem Nutzdaten-Bereich der Message war überschrieben und wurde korrigiert. Vermutlich hat ein User einer der vorhergehenden Messages über seinen Datenbereich hinaus geschrieben (Buffer Overflow). Zur Fehleranalyse sollte der Message-Pool mit MsgInfo und IdxInfo im Detail untersucht werden und zusätzlich die MSG-Trace-Funktion (MSG\_TRC) aktiviert werden. Es sollte so zeitnah wie möglich (am besten direkt aus EOS\_Error heraus) auf den Fehler reagiert werden.

### EOS\_GET\_NUL

Der EOS Null-Pointer EOS\_NULL war überschrieben und wurde korrigiert. Es hat also eine (unerlaubte) Zuweisung an EosNull stattgefunden, eventuell wurde versehentlich Zuweisung und Vergleich verwechselt.

## 9.1.4 EOS\_MsgPurge

### EOS\_MSGPURGE\_USR

Der übergebene Parameter muss größer als Null sein. Prüfen Sie den Parameter, es sollte eine Konstante oder eine einfache Variable verwendet werden.

### EOS\_MSGPURGE\_MSG

Es wurde eine Message aus dem Besitz des Users in den Pool zurückgegeben. Dies ist eine Notmaßnahme zur Erhaltung der Verfügbarkeit des Systems, die im Normalbetrieb nicht notwendig sein sollte. Stellen Sie auf jeden Fall fest, warum der betreffende User die Msg nicht selbst zurückgegeben hat oder passen Sie die Aufrufbedingungen der Funktion an.

## 9.1.5 EOS\_MbxPurge

### EOS\_MBXPURGE\_MBX

Der Mbx-Parameter ist größer als der größte gültige Mailbox-Identifizierer oder kleiner als 0 oder ist eine Home-Mailbox. Prüfen Sie den Parameter, es sollte eine Konstante oder eine einfache Variable verwendet werden.

### EOS\_MBXPURGE\_MSG

Es wurde eine Message aus der Mailbox in den Pool zurückgegeben. Dies ist eine Notmaßnahme zur Erhaltung der Verfügbarkeit des Systems, die im Normalbetrieb nicht notwendig sein sollte. Stellen Sie auf jeden Fall fest, warum die Mailbox zu viele oder zu alte Messages enthielt, d.h. warum der Besitzer (Owner) diese nicht ausgelesen hat.

### EOS\_MBXPURGE\_SYS

Es wurden mehr Messages aus der Mailbox entfernt, als im gesamten System vorhanden sind. Dies deutet stark auf einen Fehler in der MSG Implementierung hin.

## 9.1.6 Info-Funktionen

### EOS\_MBX\_MBX

Der Mbx-Parameter ist größer als der größte gültige Mailbox-Identifizierer oder kleiner als 0. Prüfen Sie den Parameter, es sollte eine Konstante oder eine einfache Variable verwendet werden.

### EOS\_MSG\_MSG

Der Message Pointer liegt außerhalb des Message-Pools oder ist kein gültiger Pointer der EOS Message Verwaltung (MsgTab). Der Pointer wurde nicht von Alloc, Get oder IdxInfo geliefert oder zwischenzeitlich verändert.

### EOS\_USR\_USR

Der Idx Parameter ist kleiner als 0 (bzw. 1) oder größer als USR\_MAX.

### EOS\_MSG\_IDX

Der Idx-Parameter ist größer als der größte gültige Message-Identifizierer. Der Index wurde nicht von MsgInfo geliefert oder zwischenzeitlich verändert.

## 9.1.7 EOS\_MbxOpen

### EOS\_MBXOPEN\_INI

Die Funktion EOS\_Msglnit wurde bisher nicht aufgerufen. Rufen Sie Msglnit einmal vor allen anderen MSG-Funktionen auf.

## **EOS\_MBXOPEN\_NON**

Aus Platzmangel kann keine weitere Mailbox angelegt werden. Passen Sie MBX\_MAX an.

## **9.1.8 EOS\_MsgInit**

### **EOS\_MSGINIT\_CNT**

Die Anzahl der in IniTab angeforderten Msgs übersteigt den Bereich des Typs index\_t. Stellen Sie index\_t auf einen 32-Bit Typ um oder reduzieren Sie die Zahl der Messages in IniTab.

### **EOS\_MSGINIT\_MEM**

Der Speicherplatz für den Message-Pool oder die Message-Tabelle (MsgTab) konnte vom Laufzeitsystem nicht bereitgestellt werden. Reduzieren Sie den Speicherplatzbedarf durch Anpassung von IniTab oder erweitern Sie den verfügbaren Speicher auf dem Heap.

### **EOS\_MSGINIT\_SYS\_1**

Der Index der Home-Mailbox stimmt nicht mit dem Index in IniTab überein. Dies ist ein Fehler in der Implementierung.

### **EOS\_MSGINIT\_SYS\_2**

Berechneter und tatsächlicher Speicherbedarf stimmen nicht überein. Dies ist ein Fehler in der Implementierung.

### **EOS\_MSGINIT\_DON**

Die Funktion EOS\_MsgInit wurde bereits aufgerufen. Mehrfacher Aufruf ist nicht erlaubt.

# 10 Die MSG Implementierung

In diesem Kapitel werden die Grundzüge der Implementierung des Message- und Mailbox-Systems von EOS (MSG Subsystem) dargestellt. Wir folgen dabei der Anlage und Initialisierung des Message-Pools in EOS\_MsgInit bis hin zum Aufruf der API-Funktionen Alloc/Get und Put/Free durch die Applikation.

## 10.1 IniTab und die Anlage des Memory-Pools

Wir gehen von folgender IniTab-Tabelle in der Datei EOS\_MSG.C aus.

```
/*
XXX message pool (first entry len, second entry cnt)
*/
static ini_dsc IniTab [] =
{
    { -1 /*!!!*/},          /* !!! first entry must be < 0 !! */

    { 8    3 },             /* XXX entries can be modified, */
    { 16,  4 },             /* XXX added or deleted          */
    { 32,  5 },             /* XXX len should be in ascending */

    { -1 /*!!!*/}          /* !!! last entry must be < 0 !! */
};
```

Es sollen also 3 Messages mit einer Länge von 8 Bytes, 4 der Länge 16 und 5 der Länge 32 angelegt werden. Das sind insgesamt 12 Messages. Pro Message kommt ein Index vom Typ `index_t` (short, 2 Bytes) hinzu und zusätzlich wird eine weitere Message der größten in IniTab vorkommenden Länge (32) als Nullpointer (EOS\_NULL) benötigt.

Daraus berechnet sich folgender Speicherbedarf (in Bytes) für den Memory-Pool.

$$(8 + 2) * 3 + (16 + 2) * 4 + (32 + 2) * 5 + (32 + 2) = 306$$

Dieser Speicherplatz wird in EOS\_MsgInit per malloc von der Heap-Verwaltung des C-Laufzeitsystems angefordert und der Erfolg anhand der zurückgelieferten Startadresse des Pools geprüft (wir gehen im Folgenden von der Startadresse 1000 aus). In kleineren embedded Systemen (Controller) wird die Startadresse gegebenenfalls explizit zugewiesen. Nun strukturiert EOS den Speicherplatz als EOS Message-Pool.

<b>RAM-Speicher (Message Pool)</b>		
<b>Adresse</b>	<b>Wert</b>	<b>Funktion</b>
1000	0	Index Msg 0
1002		Nutzdaten Msg 0 (8 Bytes)
1010	1	Index Msg 1
1012		Nutzdaten Msg 1 (8 Bytes)
1020	2	Index Msg 2
1022		Nutzdaten Msg 2 (8 Bytes)
1030	3	Index Msg 3
1032		Nutzdaten Msg 3 (16 Bytes)
1048	4	Index Msg 4
1050		Nutzdaten Msg 4 (16 Bytes)
1066	5	Index Msg 5
1068		Nutzdaten Msg 5 (16 Bytes)
1084	6	Index Msg 6
1086		Nutzdaten Msg 6 (16 Bytes)
1102	7	Index Msg 7
1104		Nutzdaten Msg 7 (32 Bytes)
1136	8	Index Msg 8
1138		Nutzdaten Msg 8 (32 Bytes)
1170	9	Index Msg 9
1172		Nutzdaten Msg 9 (32 Bytes)
1204	10	Index Msg 10
1206		Nutzdaten Msg 10 (32 Bytes)
1238	11	Index Msg 11
1240		Nutzdaten Msg 11 (32 Bytes)
1272	12	Index Msg 12
1274 - 1305		EOS_NULL (32 Bytes)

## 10.2 Die Message-Tabelle (MsgTab)

Die Startadressen der Messages (d.h. der Nutzdaten) werden in der Message-Tabelle MsgTab eingetragen. MsgTab hat dazu so viele Einträge vom Typ msg\_dsc, wie Messages im System vorhanden sind, im Beispiel also 13.

Die Datenstruktur msg\_dsc hat zwei Komponenten vom Typ index\_t (Msg, Nxt) und zwei Komponenten vom Typ short (Hom, Usr). In der EOS Standard-Implementierung hat sie somit eine Größe von 8 Bytes. Für jede im Pool angelegte Message werden also neben den Nutzdaten zusätzliche 10 Bytes (2 für den Index, 8 für den Eintrag in MsgTab) benötigt.

Da die Größe von MsgTab von IniTab abhängt, wird MsgTab gleichfalls per malloc auf dem Heap angelegt (in der Implementierung vor dem Message-Pool).

<b>MsgTab (Message Verwertung)</b>				
<b>Index</b>	<b>Msg</b>	<b>Hom (Ini)</b>	<b>Nxt</b>	<b>Usr</b>
0	1002	1		
1	1012	1		
2	1022	1		
3	1032	2		
4	1050	2		
5	1068	2		
6	1086	2		
7	1104	3		
8	1138	3		
9	1172	3		
10	1206	3		
11	1240	3		
12	1274	0		

Msg ist die Adresse, an der die Nutzdaten der zum betreffenden Index gehörigen Message beginnen (vgl. Message-Pool).

Der Eintrag Hom (Ini) ist ein Verweis auf den zugehörigen Index in IniTab. Darüber lässt sich die Länge der Message ermitteln. Gleichzeitig ist es der weiter unten beschriebene Verweis auf die zuständige Home-Mailbox.

## 10.3 Message- und Index-Prüfungen

Mit dem bisher beschriebenen Teil der Message-Tabelle (MsgTab) werden im MSG Laufzeitsystem (EOS\_Put) folgende Prüfungen und Fehlerkorrekturen durchgeführt.

### (1) Adressbereich von Message Pointern

Für jeden Message-Pointer Msg1 werden folgende Bereichs-Prüfungen durchgeführt.

- Die Adresse darf nicht kleiner als die der ersten Message sein.

```
if (Msg1) < MsgTab[0].Msg) EOS_Error(..)
```

- Die Adresse muss kleiner als EOS\_NULL sein.

```
if (Msg1 >= EOS_NULL) EOS_Error(..)
```

Im Fehlerfall wird der Zugriff verweigert.

### (2) Gültigkeit eines Message Pointers

- Über den Message-Pointer wird im Pool der zugehörige Index gelesen.

```
Idx1 = * ( ( (index_t*)Msg1 ) - 1 )
```

- Über den Index wird der in der Message-Tabelle hinterlegte Pointer gelesen.

```
Msg2 = MsgTab[Idx1].Msg)
```

Wenn Msg1 ungleich Msg2 ist, stimmt der Index Idx1 oder der Pointer Msg1 nicht. Also wurde entweder Idx1 im Pool überschrieben (Datenfehler, Overflow) oder Msg1 wurde im Anwendungsprogramm manipuliert (Pointer Fehler) oder verwechselt.

### (3) Rekonstruktion eines überschriebenen Index

Dieser Schritt ist nur nach einem in (2) erkannten Fehler (Msg1 != Msg2) notwendig.

- Msg1 wird in MsgTab gesucht. Da die Pointer (Msg) dort sortiert stehen, ist hierfür in MSG eine sehr schnelle binäre Suche implementiert.

```
Idx1 = Msg_Idx (Msg1)
```

- Wenn der Pointer Msg1 gefunden wurde, kann der Index-Eintrag im Pool entsprechend korrigiert werden, andernfalls wird der Zugriff verweigert.

```
if (Idx1 != -1) * ( ( (index_t*)Msg1 ) - 1 ) = Idx1
```

## (4) Prüfung des Folge-Index

Dieser Schritt ist nur möglich, wenn in (2) kein Fehler erkannt wurde oder dieser in (3) korrigiert werden konnte.

- Über den Index wird in MsgTab die Adresse des nächsten Puffers bestimmt.

```
Msg2 = MsgTab[Idx1 + 1].Msg
```

- Über die Adresse wird im Pool der zugehörige Index gelesen.

```
Idx2 = * ( ( (index_t*)Msg2 ) - 1 )
```

- Wenn Idx2 ungleich (Idx1 + 1) ist, wurde der Index hinter den Nutzdaten von Msg1 im Pool überschrieben (Overflow) und wird von MSG korrigiert.

```
if ( Idx2 != (Idx1 +1) )
    * ( ( (index_t*)Msg2 ) - 1 ) = (Idx1 + 1)
```

MSG nutzt auf diese Weise die Index-Einträge im Message-Pool zur Verifikation der Message-Pointer (via MsgTab) und gleichzeitig zur Verifikation der Datenkonsistenz.

## 10.4 Mailboxen

Die Implementierung der MSG Mailboxen basiert im Kern auf den beiden Komponenten First und Last der Datenstruktur mbx\_dsc in der Tabelle MbxTab. Dies sind die Indices der ersten (First) und der letzten (Last) Message in der Mailbox. Die Verkettung der Messages als lineare Liste (von First nach Last) erfolgt in MsgTab (Nxt).

In EOS\_MsgInit wird zu jedem Eintrag in IniTab (d.h. zu jeder Länge) eine Mailbox angelegt. Diese Mailboxen werden Home-Mailboxen genannt. In den Home-Mailboxen werden von EOS die freien (d.h. per Alloc verfügbaren) Messages der betreffenden Länge verwaltet. Deshalb werden anschließend alle Messages (nach aufsteigendem Index) per Msg\_Put in der für sie zuständigen Home-Mailbox abgelegt. Daraus resultiert folgende Situation (MbxTab).

<b>MbxTab (Mailbox Verwertung)</b>				
<b>Nummer</b>	<b>First</b>	<b>Last</b>	<b>Cnt</b>	<b>Own</b>
1	0	2	3	0
2	3	6	4	0
3	7	11	5	0

Cnt gibt die aktuelle Anzahl der Messages in der Mailbox an, Own ist der Besitzer der Mailbox. Die Home-Mailboxen sind öffentliche (public) Mailboxen, weil jeder User (per Alloc oder direkt) darauf zugreifen darf, und haben deshalb den Besitzer 0.



Beim Einfügen einer Message in einer Mailbox werden in MsgTab die Einträge Nxt (Verkettung innerhalb der Mailbox) und Usr (aktueller Besitzer/Aufenthaltort) aktualisiert. Nach dem Ablegen aller Messages in ihren Home-Mailboxen führt dies zu folgender Situation in MsgTab.

<b>MsgTab (Message Verwaltung)</b>				
<b>Index</b>	<b>Msg</b>	<b>Hom</b>	<b>Nxt</b>	<b>Usr</b>
0	1002	1	1	-1
1	1012	1	2	-1
2	1022	1	X	-1
3	1032	2	4	-2
4	1050	2	5	-2
5	1068	2	6	-2
6	1086	2	X	-2
7	1104	3	8	-3
8	1138	3	9	-3
9	1172	3	10	-3
10	1206	3	11	-3
11	1240	3	X	-3
12	1274	0	X	0

Der Eintrag X (I\_NONE) bei Nxt zeigt an, dass diese Message keinen Nachfolger hat.

Die Komponente Usr beschreibt den aktuellen Aufenthaltsort einer Message. Negative Werte bezeichnen dabei Mailboxen, positive Werte einen externen User (SW-Komponente oder Task).

## 10.5 Alloc und Free

Die Implementierung der Funktionen EOS\_Alloc und EOS\_Free basiert vollständig auf dem unterlagerten Mailbox-System.

EOS\_Alloc vergleicht die angeforderte Puffer-Länge sukzessive mit den Längenangaben in IniTab und entnimmt bei ausreichender Länge per Get eine Message aus der zugehörigen Home-Mailbox. Die Nummer der Home-Mailbox ist dabei identisch mit dem Index in IniTab (das vereinfacht die Implementierung).

EOS\_Free leitet jeden Aufruf unmittelbar an EOS\_Put weiter und gibt dabei als Ziel die Mailbox 0 an. Die Mailbox 0 wird von Put in die zuständige Home-Mailbox umgesetzt.

## 10.6 Zugriffsrechte und Besitzverhältnisse

Mit der vollständigen Message-Tabelle (MsgTab) und den Mailboxen werden im EOS Laufzeitsystem (EOS\_Put, EOS\_Get) folgende Prüfungen und Fehlerkorrekturen durchgeführt.

### (1) Prüfung des Aufenthaltsortes

Put: Der Aufenthaltsort der Message muss größer als 0 sein, sonst befindet sie sich bereits in einer Mailbox und der Zugriff wird verweigert (Verhinderung mehrfacher Rückgabe).

### (2) Prüfung des Users

Put: Der im Put-Aufruf genannte User und der in MsgTab von Get eingetragene (Usr) müssen übereinstimmen, andernfalls wird die Message nicht in der gewünschten Mailbox, sondern in ihrer Home-Mailbox (Hom) abgelegt.

### (3) Existenz der Mailbox

Put: Wenn die im Put-Aufruf genannte Mailbox nicht existiert, wird die Message stattdessen in ihrer Home-Mailbox (Hom) abgelegt. Dadurch wird der Verlust von Message-Puffern (Memory Leak) vermieden.

### (4) Konsistenz der Home-Mailboxen

Put: Auf die Home-Mailboxen darf (und kann) vom Anwender nicht direkt zugegriffen werden, weil MSG zur Wahrung der Konsistenz die zuständige Home-Mailbox immer selbst bestimmt.

### (5) Besitzer einer Mailbox

Get: Der im Aufruf genannte User und der in MbxTab von Create eingetragene Besitzer (Own) der Mailbox müssen übereinstimmen (Zugriffs-Berechtigung), oder die Mailbox muss öffentlich sein.

## 10.7 Get und Put

Die API-Funktionen EOS\_Get und EOS\_Put erledigen die gesamte Parameter-Prüfung und rufen dann die zugehörigen Kernel-Funktionen Msg\_Get bzw. Msg\_Put auf. Die Implementierung dieser Kernel-Funktionen ist straightforward und basiert auf der Technik einfach verketteter Listen.

Messages werden am Anfang der Mailboxen (First) entnommen und am Ende (Last) angefügt (FIFO). Eine Ausnahme bilden die Home-Mailboxen, bei denen Put die Message am Anfang anfügt. Das kann bei grossen Systemen mit Cache und Memory Management Unit (MMU) von Vorteil sein, weil der lokale Gebrauch des Speichers

gefördert wird. Andernfalls rotieren aufeinanderfolgende Alloc/Free Aufrufe durch den ganzen Pool (oder zumindest den Teilbereich aller Messages mit der betreffenden Länge).

Zur Verdeutlichung des Modells zeigen wir hier noch den Effekt einiger Alloc und Put-Aufrufe durch den User 19, nachdem der User 18 eine Mailbox für sich eingerichtet hat.

- Mbx18 = EOS\_MbxCreate (18)
- Msg1 = EOS\_Alloc (10, 19)
- Msg2 = EOS\_Alloc (20, 19)
- Msg3 = EOS\_Alloc (20, 19)
- EOS\_Put (Mbx18, Msg3, 19)
- EOS\_Put (Mbx18, Msg1, 19)

Mbx18 erhält den Wert 4 (nächster freier Mailbox Index). Das Alloc der Größe 10 wird aus der Mailbox 2 (Länge 16) bedient, die Allocs der Größe 20 aus der Mailbox 3 (Länge 32). Daraus resultiert folgende Situation in MbxTab und MsgTab (in Klammern stehen die Werte vor Ausführung der Kommando-Sequenz).

<b>MbxTab (Mailbox Verwaltung)</b>				
<b>Nummer</b>	<b>First</b>	<b>Last</b>	<b>Cnt</b>	<b>Own</b>
1	0	2	3	0
2	4 (3)	6	3 (4)	0
3	9 (7)	11	3 (5)	0
Mbx18 = 4	8 (X)	3 (X)	2 (0)	18

<b>MsgTab (Message Verwaltung)</b>				
<b>Index</b>	<b>Msg</b>	<b>Ini/Hom</b>	<b>Nxt</b>	<b>Usr</b>
0	1002	1	1	-1
1	1012	1	2	-1
2	1022	1	X	-1
3	Msg1 = 1032	2	X (4)	-4 (-2)
4	1050	2	5	-2
5	1068	2	6	-2
6	1086	2	X	-2
7	Msg2 = 1104	3	X (8)	19 (-3)
8	Msg3 = 1138	3	3 (9)	-4 (-3)
9	1172	3	10	-3
10	1206	3	11	-3
11	1240	3	X	-3
12	1274	0	X	0

## 10.8 MsgPurge

User 19 hat in der Kommandosequenz des vorhergehenden Abschnitts vermutlich vergessen, seine Message Msg2 gleichfalls in der Mailbox Mbx18 abzulegen.

- EOS\_MsgPurge (19)

Dieser Aufruf findet Msg2 bei der Inspektion der Tabelle MsgTab unter Index 7 und führt die Message in den Message-Pool (Mailbox 3) zurück.

# 11 Der Sourcecode

In diesem Kapitel ist der komplette Sourcecode des MSG und des ERR Subsystems von EOS wiedergegeben.

## 11.1 EOS\_ERR.H

```

/*****
 * Copyright(C) : S M A Regelsysteme GmbH, 34266 Niestetal, Germany
 *****/
 * project      : EOS embedded operating system
 *****/
 * filename     : EOS_ERR.H
 *****/
 * description  : error handling subsystem of EOS
 *****/
 * history      : author  date      ver / description
 *
 *****/
 *              Frees   20.09.03 1.0 / creation
 *              Frees   05.11.03 1.1 / EOS_ALLOC_BND, EOS_USR_USR
 *****/

#define EOS_ERR_LOG 16          /* XXX size of EosErrLog buffer */
                                /* should be a power of 2 */
/*****
/*
XXX number of users
*/
#define EOS_USR_ERR  63

/*
This constant determines the size of EosUsrErr, which shows the act
number of errors for the users 1...EOS_USR_ERR. It is not an error to
use greater user numbers, but their errors are all counted in
EosUsrErr[0].

In the EOS multitasking environment, you should use the same value
as for TSK_MAX in EOS_TSK.C (if your users are tasks).
*/

/*****
/* error and event handling structures and variables */
/
*****/

```

```

/*
data types
*/
typedef struct
{
    unsigned int    Tim;           /* event time (scheduler time) */
    unsigned short  Cnt;           /* error counter                */
    short           Err;           /* error code from list EOS_ERR */
    short           Usr;           /* actual usr when event occured */
    short           Mbx;           /* involved mailbox              */
    void*           Msg;           /* involved message              */
}
eos_log_dsc;                     /* error logging descriptor      */
/*
variables
*/
extern unsigned    EosErrCnt;     /* total error counter           */
extern unsigned    EosErrTab[];  /* err spec. error counter table */
extern unsigned    EosUsrErr[];  /* usr spec. error counter table */
extern eos_log_dsc EosErrLog[];  /* error logging buffer [LOG_MAX] */
extern unsigned    EosErrLogWr;  /* write index in EosErrLog      */

/*****
/* API functions */
*****/

extern int EOS_ErrInit (void);
extern int EOS_Err      (int Err, int Usr, int Mbx, void* Msg);

/*****
/* event logging (trace) in EosErrLog */
*****/

#define EOS_TRACE(Trc,Usr,Mbx,Ptr) \
{ \
    EosErrLog[EosErrLogWr].Tim = EOS_TIM(); \
    EosErrLog[EosErrLogWr].Cnt = (short)EosErrCnt; \
    EosErrLog[EosErrLogWr].Err = (short)(Trc); \
    EosErrLog[EosErrLogWr].Usr = (short)(Usr); \
    EosErrLog[EosErrLogWr].Mbx = (short)(Mbx); \
    EosErrLog[EosErrLogWr].Msg = Ptr; \
    EosErrLogWr = (EosErrLogWr + 1) % (EOS_ERR_LOG); \
}

```

```
}

/*****
/* error codes */
*****/

enum EOS_ERR
{
    EOS_ALLOC_FIT,          // fitting home mbx empty (no error)

    /*****
    MSG: EOS message and mailbox system
    *****/

    EOS_ALLOC_SIZ,          // size exceeds longest msg
    EOS_ALLOC_USR,          // usr parameter is not positive
    EOS_ALLOC_BND,          // allocated memory exceeds bound
    EOS_ALLOC_NUL,          // no msg available

    EOS_PUT_MSG_LOW,        // msg parameter is too low
    EOS_PUT_MSG_NUL,        // msg parameter is EOS_NULL
    EOS_PUT_MSG_HIG,        // msg parameter is too high
    EOS_PUT_MSG_INV,        // invalid msg pointer
    EOS_PUT_PUT,            // msg is member of a mbx
    EOS_PUT_USR,            // invalid usr parameter
    EOS_PUT_ACC,            // usr is not registered for the msg
    EOS_PUT_DAT_ACT,        // invalid index in actual msg buffer
    EOS_PUT_DAT_NXT,        // invalid index in next msg buffer
    EOS_PUT_MBX,            // mbx identifier out of range
    EOS_PUT_HOM,            // mbx identifier is a home mbx

    EOS_GET_MBX,            // invalid mbx identifier
    EOS_GET_USR,            // invalid usr parameter
    EOS_GET_OWN,            // usr is not owner of the mbx
    EOS_GET_DAT,            // invalid index in msg (overwritten)
    EOS_GET_NUL,            // EOS_NULL was overwritten

    EOS_MSGPURGE_USR,       // invalid usr identifier
    EOS_MSGPURGE_MSG,       // msg purged from usr

    EOS_MBXPURGE_MBX,       // invalid mbx identifier
    EOS_MBXPURGE_MSG,       // msg purged from mbx
```

```

EOS_MBXPURGE_SYS,      // implementation error (msg count)

EOS_MBX_MBX,          // invalid mbx identifier
EOS_MSG_MSG,          // invalid msg pointer
EOS_MSG_IDX,          // invalid msg index
EOS_USR_USR,          // invalid usr identifier

EOS_MBXOPEN_INI,      // EOS system not initialized
EOS_MBXOPEN_NON,      // no more mbx available

EOS_MSGINIT_CNT,      // too many msgs for index_t
EOS_MSGINIT_MEM,      // memory could not be allocated
EOS_MSGINIT_SYS_1,    // implementation error (mbx index)
EOS_MSGINIT_SYS_2,    // implementation error (mem count)
EOS_MSGINIT_DON,      // MsgInit was already called

/*****
APP: application specific errors
*****/

/*****
the end (don't change)
*****/

EOS_ERR_TAB,          // dimension of EosErrTab
};

/*****
/* XXX system environment functions */
*****/
/*
get actual system time
*/
#define EOS_TIM()      EosTim

extern unsigned EosTim;
/*
extern unsigned EosTim;
In the EOS multitasking environment, you can use the variable
EosTim from EOS_TSK.H
*/
/***** EOF *****/

```



## 11.2 EOS\_ERR.C

```

/*****
* Copyright(C) : S M A Regelsysteme GmbH, 34266 Niestetal, Germany
*****/
* project      : EOS embedded operating system
*****/
* filename     : EOS_ERR.C
*****/
* description  : error handling of EOS
*****/
* history      : author   date       ver / description
*
*****/
*              Frees   20.09.03 1.0 / creation
*****/
/*****

#include "EOS_ERR.H"

/*****
/* global variables */
/*****

unsigned   EosErrCnt = 0;           /* total error counter      */
unsigned   EosErrTab [EOS_ERR_TAB]; /* err error counter table */
unsigned   EosUsrErr [EOS_USR_ERR+1]; /* usr err counter table   */
eos_log_dsc EosErrLog [EOS_ERR_LOG]; /* error logging buffer     */
unsigned   EosErrLogWr = 0;        /* write index in EosErrLog */

/*****
/* error and event handling */
/*****

int EOS_Err (int Err, int Usr, int Mbx, void* Msg)
/*
error and event handler
*/
{
    int i = EosErrLogWr;
    /*
increment global error counter
*/
    EosErrCnt++;

```

```

    /*
    count error in the error specific table
    */
    if (Err < EOS_ERR_TAB) EosErrTab[Err]++;
    /*
    count error in the user specific table
    */
    if (Usr <= EOS_USR_ERR) EosUsrErr[Usr]++; else EosUsrErr[0]++;
    /*
    store error information in the log buffer
    */
    EosErrLog[i].Cnt = (short)EosErrCnt;
    EosErrLog[i].Tim = EOS_TIM();
    EosErrLog[i].Err = (short)Err;
    EosErrLog[i].Usr = (short)Usr;
    EosErrLog[i].Mbx = (short)Mbx;
    EosErrLog[i].Msg = Msg;
    /*
    signal new log entry to the user API
    */
    EosErrLogWr = (i + 1) % EOS_ERR_LOG;
    /*
    result is negativ err number
    */
    return (-Err);
}

/*****
/* initialization and creation */
*****/

int EOS_ErrInit (void)
/*
initialize EOS error system (may be called several times)
*/
{
    int i;
    /*
    global error counter
    */
    EosErrCnt = 0;
    /*

```

```

    error specific table
    */
    for (i = 0; i <  EOS_ERR_TAB; i++) EosErrTab[i] = 0;
    /*
    user specific table
    */
    for (i = 0; i <=  EOS_USR_ERR; i++) EosUsrErr[i] = 0;

    return (0);
}

```

```

/***** EOF *****/

```

## 11.3 EOS\_MSG.H

```

/*****
* Copyright(C) : S M A Regelsysteme GmbH, 34266 Niestetal, Germany
*****/
* project      : EOS embedded operating system
*****/
* filename     : EOS_MSG.H
*****/
* description  : message and mailbox subsystem of EOS
*****/
* history     : author  date      ver / description
*
*****/
*              Frees   20.09.03 1.0 / creation
*              Frees   05.11.03 1.1 / UsrTab, EOS_UsrBnd,
EOS_UsrInfo
*****/
/*
the universal EOS Null-Pointer
*/
#define EOS_NULL    (EosNull)

/*
This is the value of a NULL-pointer within EOS. Don't use 0 or NULL
or anything else, it won't work!
*/

/*****
*/ global variables */

```

```

/*****/

extern unsigned      EosMsgClr;      /* import: buffer init. flag      */
extern void*         EosNull;        /* export: EOS null pointer       */

/*****/
/* API functions */
/*****/

int   EOS_MsgInit      (void);
int   EOS_MbxCreate    (unsigned Own);
int   EOS_UsrBnd       (int Usr, int Bnd);

void* EOS_Alloc        (unsigned Siz, int Usr);
int   EOS_Free         (void *   Msg, int Usr);

int   EOS_Put          (int Mbx, void* Msg, int Usr);
void* EOS_Get          (int Mbx, int Usr);

int   EOS_MbxPurge     (int Mbx, unsigned Cnt, unsigned Tim);
int   EOS_MsgPurge     (int Usr);

int   EOS_MbxInfo      (int Mbx, int* Cnt, int* Bnd, int* Tim, int*
Own);
int   EOS_MsgInfo      (void* Msg, int* Len, int* Usr, int* Idx);
int   EOS_IdxInfo      (int Idx, int* Len, int* Usr, void**Msg);
int   EOS_UsrInfo      (int Usr, int* Mem, int* Bnd);

/*****/
/* XXX system environment functions */
/*****/
/*
enter critical section for mbx nr i
leave critical section for mbx nr i
*/
#define EOS_ENTER(i)
#define EOS_LEAVE(i)

/*
(1) You dont need EOS_ENTER and EOS_LEAVE in cooperative Systems
(EOS).
In this case EOS_Put and EOS_Get are not interrupt-save.

```

(2) If you use disabling (cli) and enabling (sti) of hardware-interrupts as EOS\_ENTER and EOS\_LEAVE, the functions EOS\_Alloc, EOS\_Free, EOS\_Put, EOS\_Get, EOS\_GetTmd, EOS\_MbxTsk are save within interrupts (and preemptive task-switches). If your system allows it, this method is the best. The additional interrupt latency is really minimal

(3) If you use EOS\_Put in interrupt-functions, be aware of waiting tasks

at the mailbox. In this case EOS\_WAKE must be interrupt-save. For EOS\_TskWake in connection with EOS\_Schedul this is the case.

(4) If you use semaphores or mutexes as EOS\_ENTER and EOS\_LEAVE,

EOS\_Put and EOS\_Get are save within preemptive task switches, but may not be used in interrupts.

```
*/
```

```
/******
```

```
/*
```

```
get actual time
```

```
*/
```

```
#define EOS_TIM() EosTim
```

```
extern unsigned EosTim;
```

```
/*
```

```
extern unsigned EosTim;
```

In the EOS multitasking environment, you can use the variable EosTim from EOS\_TSK.H

```
*/
```

```
/******
```

```
/*
```

```
wake a task
```

```
*/
```

```
#define EOS_WAKE(Tsk)
```

```
/*
```

```
int EOS_Wake (int Tsk);
```

In the EOS multitasking environment, you can use the function EOS\_Wake from EOS\_TSK.H

```
*/
```

```
/* ***** EOF ***** */
```

## 11.4 EOS\_MSG.C

```
/*
*****
* Copyright(C) : S M A Regelsysteme GmbH, 34266 Niestetal, Germany
*****
* project      : EOS embedded operating system
*****
* filename     : EOS_MSG.C
*****
* description  : message and mailbox subsystem of EOS
*****
* history     : author date ver / description
*
*****
*              Frees   20.09.03 1.0 / creation
*              Frees   05.11.03 1.1 / UsrTab, EOS_UsrBnd,
EOS_UsrInfo*****
*/
```

```
#include <stdlib.h>          /* malloc, abs          */
#include <mem.h>              /* memset              */
```

```
#include "EOS_ERR.H"
#include "EOS_MSG.H"
```

```
/* *****
/* !! adjust all entries marked with XXX for your application !!
/* ***** */
```

```
#define MBX_MAX 25           /* XXX number of mailboxes */
#define MSG_TRC 0           /* XXX enable/disable trace log */
```

```
/* ***** */
```

```
#define USR_MAX 63          /* XXX number of usrs      */
/*
```

This constant determines the size of UsrTab, which shows the act number of msgs used by the usrs 1...USR\_MAX. It is not an error to use greater usr numbers, but their msgs are all counted in UsrTab[0]. In the EOS multitasking environment, you should use the same value as for TSK\_MAX in EOS\_TSK.C (if your users are tasks).

```

*/
/*****

typedef unsigned short index_t;    /* XXX msg index in every buffer */
#define I_NONE ((index_t)-1)      /* value for invalid msg index */

/*****

#undef  EOS_NULL /*EosNull*/        /* undef  public version          */
#define EOS_NULL (MsgNull)         /* define private version        */

/*****
/*
descriptor of a message pool entry (IniTab)
*/
typedef struct
{
    int      Len;                  /* length of message             */
    int      Cnt;                  /* number of msgs of this length */
}
ini_dsc;
/*
descriptor of a message (MsgTab)
*/
typedef struct
{
    index_t* Msg;                  /* adr of msg's user data buffer */
    index_t  Nxt;                  /* link to next msg while in mbx  */
    short    Hom;                  /* home mbx and index in IniTab   */
    short     Uxr;                  /* ext. user if > 0, -mbx if < 0  */
}
msg_dsc;
/*
descriptor of a mailbox (MbxTab)
*/
typedef struct
{
    index_t  First;                /* index of first msg in mbx      */
    index_t  Last;                /* index of last  msg in mbx      */
    index_t  Cnt;                  /* message counter                */
    index_t  Bnd;                  /* min for home mbx, else max     */
    short    Own;                  /* owner (0 for public)           */
}

```

```

    short    Tsk;                /* waiting task (GetTmd, Put)    */
    unsigned Tim;               /* time of last EOS_Get         */
}
mbx_dsc;
/*
descriptor of a user (UsrTab)
*/
typedef struct
{
    int Cnt;                    /* nr of allocated msgs         */
    int Mem;                    /* amount of allocated memory    */
    int Bnd;                    /* memory bound for alloc       */
}
usr_dsc;

/*****
/* global variables */
*****/

void*      EosNull   = 0;       /* public null pointer          */
unsigned   EosMsgClr = 0;       /* buffer init. flag            */

/*****
/* local variables */
*****/
/*
XXX message pool (first entry len, second entry cnt)
*/
static ini_dsc IniTab [] =
{
    { -1 /*!!!*/,                /* !!! first entry must be < 0 !! */

    { 4,    250 },                /* XXX entries can be modified,    */
    { 16,   450 },                /* XXX added or deleted            */
    { 64,   350 },                /* XXX len should be in ascending  */
    { 128,  100 },                /* XXX order                      */

    { -1 /*!!!*/                /* !!! last entry must be < 0 !! */
};
static int    IniMax = -1;       /* last valid entry in MsgIni      */
static unsigned SizMax = 0;     /* greatest length in MsgIni      */
/*

```



```

message table
*/
static msg_dsc* MsgTab;           /* allocated in Msg_Init           */
static int      MsgMax = -1;      /* last valid entry in MsgTab      */
static void*    MsgNull = 0;     /* private EOS null pointer       */
/*
mailbox table
*/
static mbx_dsc  MbxCtx[MBX_MAX+1]; /* mailbox descriptor array        */
static int      MbxCtxMax = 0;     /* last valid entry in MbxCtx      */
/*
user table
*/
usr_dsc         UsrcTab[USR_MAX+1]; /* usr msg counter table          */
/*
init
*/
static int      EosMsgInit = 0;

/*****
/* local functions (kernel functions) */
*****/

static void      MbxCtx_Put (int MbxCtx, void* Msg);
static void*     MbxCtx_Get (int MbxCtx, int  Tsk, unsigned Usrc);
static index_t   MbxCtx_Idx (index_t* Msg);

/*****
/* Alloc, Free, Put, Get */
*****/

void* EOS_Alloc (unsigned Siz, int Usrc)
/*
  get a msg for usr of (at least) the specified size
*/
{
    index_t* Msg;
    int      i;
    /*
      Size must not exceed greatest size in IniTb
    */
    if (Siz > SizMax)

```

```
{
    EOS_Err(EOS_ALLOC_SIZ, Usr, Siz, 0);

    return (EOS_NULL);
}
/*
Usr must be greater zero
*/
if (Usr < 1)
{
    EOS_Err(EOS_ALLOC_USR, Usr, Siz, 0);
    return (EOS_NULL);
}
/*
Mem must not exceed Bnd
*/
if (Usr <= USR_MAX)
{
    if ((UsrTab[Usr].Mem + Siz) > UsrTab[Usr].Bnd)
    {
        EOS_Err(EOS_ALLOC_BND, Usr, UsrTab[Usr].Bnd, 0);
        return (EOS_NULL);
    }
}
/*
the choice of the start point can be improved
(lookup table, nested if-then-else clauses)
*/
i = 1;
/*
search a message with length greater or equal to Siz
*/
while (1)
{
    if (IniTab[i].Len >= Siz)
    {
        /*
        try to get a message from this home mbx
        set buffer to zero if IniMem is set
        */
        Msg = Mbx_Get(i, 0, Usr);
        if (Msg != EOS_NULL)
```

```
{
    if (EosMsgClr > 0) memset (Msg, 0, IniTab[i].Len);
    return (Msg);
}
/*
Len is sufficient, but mbx is empty (not an error)
if you don't want to use other home-mbxs, break here
*/
EosErrTab[EOS_ALLOC_FIT]++;
}
/*
switch to next mbx
*/
if ((++i) > IniMax) break;
}
/*
no message available
*/
EOS_Err (EOS_ALLOC_NUL, Usrc, Siz, 0);
return (EOS_NULL);
}

/*****

int EOS_Free (void* Msg, int Usrc)
/*
returns a msg into its home mbx
*/
{
/*
EOS_Put does the job (0 indicates home mbx)
*/
return (EOS_Put (0, Msg, Usrc));
}

/*****

int EOS_Put (int Mbx, void* Msg, int Usrc)
/*
put a msg into the specified mbx
direct access to home mbxs is forbidden (use 0 for msg's home mbx)
*/
```

```
{
    index_t Idx;
    int      Err = 0;
    /*
    check null pointer and address range
    */
    if (Msg > EOS_NULL)      Err = EOS_PUT_MSG_HIG; else
    if (Msg == EOS_NULL)    Err = EOS_PUT_MSG_NUL; else
    if (Msg < MsgTab[0].Msg) Err = EOS_PUT_MSG_LOW;
    /*
    break on error
    */
    if (Err) return (EOS_Err (Err, Usr, Mbx, Msg));
    /*
    now it's save to access the address where msg points to
    check idx range, restore idx if out of range
    compare msg with ptr in MsgTab, restore idx if different
    */
    Idx = *(((index_t*)Msg) - 1);
    if (Idx > MsgMax)      Idx = Msg_Idx (Msg); else
    if (Msg != MsgTab[Idx].Msg) Idx = Msg_Idx (Msg);
    /*
    now we must have a valid idx for msg, break if not
    */
    if (Idx == I_NONE)
        return (EOS_Err(EOS_PUT_MSG_INV, Usr, Mbx, Msg));
    /*
    usr must be greater zero (else he could "use" mbx msgs)
    usr must be the registered usr of the msg (detects multiple put)
    */
    if (Usr < 1)          Err = EOS_PUT_USR; else
    if (MsgTab[Idx].Usr != Usr) Err = EOS_PUT_ACC;
    /*
    if err: report it and direct msg to it's home mbx
    msg must be in external use (else multiple put)
    */
    if (Err)
    {
        if (MsgTab[Idx].Usr <= 0)
            return (EOS_Err(EOS_PUT_PUT, Usr, Mbx, Msg));
        EOS_Err (Err, Usr, Mbx, Msg);
        Mbx = 0;
    }
}
```

```
}
/*
check integrity of next msg (indicates buffer overflow)
restore index in buffer if it's overwritten
*/
if (*(MsgTab[Idx + 1].Msg - 1) != (index_t)(Idx + 1))
{
    EOS_Err (EOS_PUT_DAT_NXT, Usr, Mbx, Msg);
    *(MsgTab[Idx + 1].Msg - 1) = (index_t)(Idx + 1);
}
/*
check mbx (0 addresses msg's home mbx)
*/
if (Mbx == 0) Mbx = MsgTab[Idx].Hom; else
{
    /*
    mbx must be in the valid range
    direct access to home mbxs is not allowed
    */
    if ((unsigned)Mbx > MbxMax) Err = EOS_PUT_MBX;
    if ((unsigned)Mbx <= IniMax) Err = EOS_PUT_HOM;
    /*
    if err: report it and direct msg to it's home mbx
    */
    if (Err)
    {
        EOS_Err (Err, Usr, Mbx, Msg);
        Mbx = MsgTab[Idx].Hom;
    }
}
/*
it was a long way, but now it's save to
call the kernel function
*/
Mbx_Put (Mbx, Msg);
/*
no error
*/
return (0);
}
```

```
/*****
```

```
void* EOS_Get (int Mbx, int Usr)

/*
get a msg from the specified mbx (or EOS_NULL)
direct access to home mbxs is allowed
*/
{
    void* Msg;
    /*
    we check if the "constant" EOS_NULL is constant
    otherwise we restore it from our local copy (MsgNull)
    remember: in this module EOS_NULL is MsgNull
    */
    if (EosNull != MsgNull)
    {
        EOS_Err (EOS_GET_NUL, Usr, Mbx, EosNull);
        EosNull = MsgNull;
    }
    /*
    the mbx index must be in the valid range
    */
    if ((unsigned)Mbx > MbxMax)
    {
        EOS_Err (EOS_GET_MBX, Usr, Mbx, 0);
        return (EOS_NULL);
    }
    /*
    usr must be greater zero
    */
    if (Usr < 1)
    {
        EOS_Err (EOS_GET_USR, Usr, Mbx, 0);
        return (EOS_NULL);
    }
    /*
    access rule: if mbx is not public, usr must be its owner
    */
    if (MbxTab[Mbx].Own != 0)
    {
        if (MbxTab[Mbx].Own != Usr)
        {
```

```

        EOS_Err (EOS_GET_OWN, Usrc, Mbx, 0);
        return (EOS_NULL);
    }
}
/*
call the kernel function (normal call with Tsk = 0)
*/
Msg = Mbx_Get (Mbx, 0, Usrc);
return (Msg);
}

/*****
/* mbx kernel functions (only for use within EOS functions) */
*****/

static void* Mbx_Get (int Mbx, int Tsk, unsigned Usrc)
/*
get a msg from the specified mbx (or EOS_NULL) for usr
Tsk > 0: register tsk if mbx empty, else get msg (EOS_GetTmd)
Tsk == 0: get msg (normal call) (EOS_Get)
Tsk < 0: register tsk if mbx empty, don't get msg (EOS_MbxTsk)
*/
{
    index_t Idx;                                /* index of new Msg */
    /*
take the first message out of the mbx (empty if I_NONE)
set First to next msg in mbx
*/
    EOS_ENTER (Mbx);
    {
        Idx = MbxTab[Mbx].First;
        if (Idx == I_NONE)
        {
            /*
            mbx is empty, register waiting tsk (dummy if tsk 0)
            */
            MbxTab[Mbx].Tsk = (short)abs(Tsk);
            EOS_LEAVE (Mbx);
            return (EOS_NULL);
        }
        /*
        mbx is not empty

```

```
calls with tsk < 0 (EOS_MbxTsk) don't want a msg
*/
if (Tsk < 0)
{
    EOS_LEAVE (Mbx);
    return ((void*)0);
}
/*
get next msg
*/
MbxTab[Mbx].First = MsgTab[Idx].Nxt;
/*
supervision
*/
MbxTab[Mbx].Cnt--;
if (Usr <= USR_MAX)
{
    UsrTab[Usr].Cnt++;
    UsrTab[Usr].Mem += IniTab[MsgTab[Idx].Hom].Len;
}
else
{
    UsrTab[0].Cnt++;
}
}
EOS_LEAVE (Mbx);
/*
update MsgTab and MbxTab
*/
MsgTab[Idx].Nxt = I_NONE;
MsgTab[Idx].Usr = (short)Usr;
MbxTab[Mbx].Tim = EOS_TIM();
/*
check integrity of index in msg buffer,
restore it if it's invalid
*/
if ( *(MsgTab[Idx].Msg - 1) != Idx)
{
    EOS_Err (EOS_GET_DAT, Usr, Mbx, MsgTab[Idx].Msg);
    *(MsgTab[Idx].Msg - 1) = Idx;
}
/*
```



```

    statistics (minimal cnt for home mbxs)
    */
    if (Mbx <= IniMax)
    {
        if (MbxTab[Mbx].Bnd > MbxTab[Mbx].Cnt) MbxTab[Mbx].Bnd--;
    }
    /*
    optional MSG trace
    */
    #if (MSG_TRC)
        EOS_TRACE(100, Mbx, Usr, MsgTab[Idx].Msg);
    #endif
    /*
    msg found in mbx
    */
    return (MsgTab[Idx].Msg);
}

/*****

static void Mbx_Put (int Mbx, void* Msg)
    /*
    put the msg into the specified mbx
    */
{
    index_t  Idx = (((index_t*)Msg) - 1);    /* MsgTab index of msg */
    int      Tsk = 0;                        /* tsk to wake (if > 0) */
    unsigned Usr = MsgTab[Idx].Usr;          /* original usr of msg */
    /*
    initialize the dsc of the incoming msg
    */
    MsgTab[Idx].Nxt = I_NONE;
    MsgTab[Idx].Usr = (short)(-Mbx);
    /*
    insert the new msg in the mbx
    */
    if (Mbx <= IniMax)
    {
        /*
        home mbx (LIFO): add the new msg at the front of the mbx
        this strategy (stack) improves the local usage of memory
        */

```

```

EOS_ENTER (Mbx);
{
    if (MbxTab[Mbx].First == I_NONE)
    {
        MbxTab[Mbx].First = Idx;
        MbxTab[Mbx].Last  = Idx;
    }
    else
    {
        MsgTab[Idx].Nxt    = MbxTab[Mbx].First;
        MbxTab[Mbx].First = Idx;
    }
    /*
    supervision
    */
    MbxTab[Mbx].Cnt++;
    if (Usr <= USR_MAX)
    {
        UsrTab[Usr].Cnt--;
        UsrTab[Usr].Mem -= IniTab[MsgTab[Idx].Hom].Len;
    }
    else
    {
        UsrTab[0].Cnt--;
    }
}
EOS_LEAVE (Mbx);
}
else
{
    /*
    user mbx (FIFO): add the new msg at the end of the mbx
    this strategy (queue) preserves the order of the msgs
    */
    EOS_ENTER (Mbx);
    {
        if (MbxTab[Mbx].First == I_NONE)
        {
            MbxTab[Mbx].First = Idx;
            MbxTab[Mbx].Last  = Idx;
        }
        /*
        first msg in a private mbx can lead to a wake

```

```

        */
        Tsk = MbxTab[Mbx].Tsk;
        MbxTab[Mbx].Tsk = 0;
    }
    else
    {
        MsgTab[MbxTab[Mbx].Last].Nxt = Idx;
        MbxTab[Mbx].Last              = Idx;
    }
    /*
    supervision
    */
    MbxTab[Mbx].Cnt++;
    if (Usr <= USR_MAX)
    {
        UsrTab[Usr].Cnt--;
        UsrTab[Usr].Mem -= IniTab[MsgTab[Idx].Hom].Len;
    }
    else
    {
        UsrTab[0].Cnt--;
    }
}
EOS_LEAVE (Mbx);
/*
wake a waiting tsk
*/
if (Tsk > 0) EOS_WAKE (Tsk);
}
/*
statistics (maximal cnt for usr mbxs)
*/
if (Mbx > IniMax)
{
    if (MbxTab[Mbx].Bnd < MbxTab[Mbx].Cnt) MbxTab[Mbx].Bnd++;
}
/*
optional MSG trace
*/
#ifdef MSG_TRC
    EOS_TRACE(101, Mbx, Usr, Msg);
#endif

```

```

}

/*****

static index_t Msg_Idx (index_t* Msg)
/*
    search Msg in MsgTab and restore the index in its data buffer
*/
{
    unsigned Lo = 0;                /* low bound for search          */
    unsigned Hi = MsgMax;           /* high bound (EOS_NULL)        */
    unsigned Me;                   /* medium index between lo and Hi */
    /*
    binary search for Msg
    */
    while (Lo <= Hi)
    {
        Me = (Hi + Lo) / 2;
        if (Msg == MsgTab[Me].Msg)
        {
            EOS_Err (EOS_PUT_DAT_ACT, 0, 0, Msg);
            *(Msg - 1) = (index_t)Me;
            return ((index_t)Me);
        }
        if (Msg > MsgTab[Me].Msg) Lo = Me + 1; else Hi = Me - 1;
    }
    /*
    unknown pointer
    */
    return (I_NONE);
}

/*****
/* administration, supervision */
/*****/

int EOS_MbxPurge (int Mbx, unsigned Cnt, unsigned Tim)
/*
    purge msgs in mbx if more than cnt or last access is too long ago
    returns nr of purged msgs
    */
{

```

```

index_t* Msg;                /* pointer to purged msg buffer */
int      Hom;                /* home mbx of purged msg */
int      MsgCnt = 0;         /* nr of purged msgs */
/*
test mbx index (home mbxs can't be purged)
*/
if ((Mbx <= IniMax) || (Mbx > MbxMax))
    return (EOS_Err(EOS_MBXPURGE_MBX, 0, Mbx, 0));
/*
purge all if more time than Tim is elapsed since last EOS_Get
*/
if ((EOS_TIM() - MbxTab[Mbx].Tim) > Tim) Cnt = 0;
/*
reduce nr of msgs in mbx down to Cnt
*/
while (MbxTab[Mbx].Cnt > Cnt)
{
    Msg = Mbx_Get (Mbx, 0, 0);
    Hom = MsgTab[* (Msg-1)].Hom;
    Mbx_Put (Hom, Msg);
    MsgCnt++;
    /*
    error detection and reporting
    */
    if (MsgCnt > MsgMax)
        return (EOS_Err (EOS_MBXPURGE_SYS, 0, Mbx, 0));
    EOS_Err (EOS_MBXPURGE_MSG, 0, Mbx, Msg);
}
/*
nr of purged msgs
*/
return (MsgCnt);
}

/*****

int EOS_MsgPurge (int Usr)
/*
free all msgs owned by Usr
returns nr of purged msgs
*/
{

```

```

    int    MsgCnt = 0;                /* nr of purged msgs          */
    void*  Msg;                      /* act msg to purge          */
    int    i;                        /* scan index in MsgTab      */
    /*

    check usr
    */
    if (Usr < 1) return (EOS_Err (EOS_MSGPURGE_USR, Usr, 0, 0));
    /*
    put all msgs owned by usr back into their home mbxs
    */
    for (i = 0; i < MsgMax; i++)
    {
        if (MsgTab[i].Usr == Usr)
        {
            Msg = MsgTab[i].Msg;
            Mbx_Put (MsgTab[i].Hom, Msg);
            EOS_Err (EOS_MSGPURGE_MSG, Usr, 0, Msg);
            MsgCnt++;
        }
    }
    /*
    result
    */
    return (MsgCnt);
}

/*****
/* information, monitoring */
*****/

int EOS_MbxInfo (int Mbx, int* Cnt, int* Bnd, int* Tim, int* Own)
/*
    get mbx info,
    returns waiting tsk (0 for none)
    */

{
    *Cnt = 0;                        /* actual msg counter        */
    *Bnd = 0;                        /* min cnt if home mbx, else max */
    *Tim = 0;                        /* last tim of EOS_Get       */
    *Own = 0;

```

```

/*
test mbx index
*/
if ((unsigned)Mbx > MbxMax)
    return (EOS_Err (EOS_MBX_MBX, 0, Mbx, 0));
/*
info
*/
*Cnt = MbxTab [Mbx].Cnt;
*Bnd = MbxTab [Mbx].Bnd;
*Tim = MbxTab [Mbx].Tim;
*Own = MbxTab [Mbx].Own;

return (MbxTab[Mbx].Tsk);
}

/*****

int EOS_MsgInfo (void* Msg, int* Len, int* Usr, int* Ind)
/*
get msg info by msg pointer,
returns home mbx of the msg
*/
{
    index_t Idx;

    *Len = 0;                                /* msg length                                */
    *Usr = 0;                                /* msg owner (tsk>0, mbx<=0)                */
    *Ind = 0;                                /* msg index in MsgTab                       */
    /*
    check error conditions, search Msg if index is invalid
    EOS_NULL is allowed
    */
    if (Msg < MsgTab[0].Msg) return (EOS_Err (EOS_MSG_MSG, 0, 0, Msg));
    if (Msg > EOS_NULL)      return (EOS_Err (EOS_MSG_MSG, 0, 0, Msg));

    Idx = *(((index_t*)Msg) - 1);
    if (Idx > MsgMax)        Idx = Msg_Idx (Msg); else
    if (Msg != MsgTab[Idx].Msg) Idx = Msg_Idx (Msg);

    if (Idx == I_NONE) return (EOS_Err(EOS_MSG_MSG, 0, 0, Msg));
    /*

```

```

    info
    */
    *Usr = MsgTab [Idx].Usr;
    *Ind = Idx;
    *Len = IniTab[MsgTab[Idx].Hom].Len;

    return (MsgTab[Idx].Hom);
}

/*****

int EOS_IdxInfo (int Idx, int* Len, int* Usr, void** Msg)
    /*
    get msg info by index,
    returns home mbx of the msg
    */
{
    *Len = 0;                                /* msg length                                */
    *Usr = 0;                                /* msg user (ext > 0, mbx < 0)            */
    *Msg = EOS_NULL;                         /* msg pointer in msg pool                */
    /*
    errors
    */
    if ((unsigned)Idx > MsgMax)
        return (EOS_Err (EOS_MSG_IDX, 0, 0, 0));
    /*
    info
    */
    *Len = IniTab [MsgTab[Idx].Hom].Len;
    *Usr = MsgTab [Idx].Usr;
    *Msg = MsgTab [Idx].Msg;

    return (MsgTab[Idx].Hom);
}

/*****

int EOS_UsrInfo (int Usr, int* Mem, int* Bnd)
{
    *Mem = 0;
    *Bnd = 0;
    /*

```



```
    test usr index
    */
    if ((unsigned)Usr > USR_MAX)
        return (EOS_Err (EOS_USR_USR, 0, Usr, 0));
    /*
    info
    */
    *Mem = UsrTab[Usr].Mem;
    *Bnd = UsrTab[Usr].Bnd;

    return (UsrTab[Usr].Cnt);
}

/*****
/* creation, initialization */
*****/

int EOS_MbxCreate (unsigned Own)
    /*
    open a new mbx for own and return its identifier
    */
{
    /*
    check for errors
    */
    if (EosMsgInit == 0)
        return (EOS_Err (EOS_MBXOPEN_INI, Own, 0, 0));
    if (MbxMax >= MBX_MAX)
        return (EOS_Err (EOS_MBXOPEN_NON, Own, 0, 0));
    /*
    init. next entry in MbxTab
    */
    MbxMax++;
    MbxTab[MbxMax].First = I_NONE;
    MbxTab[MbxMax].Own   = (short)Own;
    MbxTab[MbxMax].Tsk   = 0;
    MbxTab[MbxMax].Cnt   = 0;
    MbxTab[MbxMax].Bnd   = 0;
    /*
    mbx identifier
    */
    return (MbxMax);
}
```

```

}

/*****/

int EOS_UsrBnd (int Usr, int Bnd)
{
    if ((Usr < 1) || (Usr > USR_MAX))
        return (EOS_Err (EOS_USR_USR, 0, Usr, 0));

    UsrTab[Usr].Bnd = Bnd;
    return (0);
}

/*****/

int EOS_MsgInit (void)
{
    index_t Idx;                /* actual msg index */
    char* MsgAdr;               /* actual addr within msg pool */
    index_t* Msg;               /* actual msg within msg pool */
    int i;                      /* actual len index */
    int j;                      /* actual cnt index */
    unsigned Mem = 0;           /* total memory space for msgs */
    unsigned Cnt = 0;           /* total number of messages */
    /*
    check and set initialization state
    */
    if (EosMsgInit > 0) return (EOS_Err(EOS_MSGINIT_DON, 0, 0, 0));
    EosMsgInit = 1;
    /*
    clear all mbx descriptors
    */
    memset (MbxTab, 0, sizeof (MbxTab));
    /*
    clear all usr descriptors, set mem bnd to max
    */
    memset (UsrTab, 0, sizeof (UsrTab));
    for (i = 1; i <= USR_MAX; i++) UsrTab[i].Bnd = -1;
    /*
    calculate the number of messages (Cnt) and the whole amount of
    memory space (Mem) needed for the message pool
    open a (public) home mbx for every message len

```

```
*/
for (i = 1; IniTab[i].Len >= 0; i++)
{
    /*
    home mbx identifier must be the index in IniTab
    */
    j = EOS_MbxCreate(0);
    if (j != i) return (EOS_Err (EOS_MSGINIT_SYS_1, i, j, 0));
    /*
    amount per msg: index plus user space
    */
    Mem += IniTab[i].Cnt * (sizeof(index_t) + IniTab[i].Len);
    Cnt += IniTab[i].Cnt;
    IniMax = i;
    /*
    determine greatest length
    */
    if (IniTab[i].Len > SizMax) SizMax = IniTab[i].Len;
}
/*
EOS_NULL: add entry in MsgTab and add memory at the end of the pool
use as length of EOS_NULL the length of the longest msg
*/
MsgMax = Cnt;
Mem += sizeof(index_t) + SizMax;
Cnt++;
/*
check if nr of msgs exceeds the range of index_t
*/
if (Cnt != (index_t)Cnt)
    return (EOS_Err(EOS_MSGINIT_CNT, 0, 0, 0));
/*
alloc system memory for MsgTab (list of msg descriptors)
*/
MsgTab = (msg_dsc*) malloc (Cnt * sizeof(msg_dsc));
if (MsgTab == 0)
    return (EOS_Err (EOS_MSGINIT_MEM, 0, 0, 0));
/*
alloc system memory for the message pool
*/
MsgAdr = (char*) malloc (Mem);
if (MsgAdr == 0)
```

```
    return (EOS_Err (EOS_MSGINIT_MEM, 0, 0, 0));
/*
init message pool with zero if EosMsgClr indicates it
*/
if (EosMsgClr > 0) memset (MsgAdr, 0, Mem);
/*
index of first msg in MsgTab
*/
Idx = 0;
/*
do the partitioning of the message pool into seperate messages
according to IniTab and put them into their home mbxs
*/
for (i = 1; IniTab[i].Len >= 0; i++)
{
    for (j = 0; j < IniTab[i].Cnt; j++)
    {
        /*
        initialize the msg buffer and its dsc
        */
        Msg                = (index_t*) MsgAdr;
        Msg[0]              = Idx;
        MsgTab [Idx].Msg = Msg + 1;
        MsgTab [Idx].Hom = (short)i;
        MsgTab [Idx].Usr = 0;
        /*
        put the msg into its home mbx
        */
        Mbx_Put (i, Msg + 1);
        /*
        increase MsgAdr by length of index and user space
        */
        MsgAdr += (sizeof(index_t) + IniTab[i].Len);
        Idx++;
    }
    MbxTab[i].Bnd = (index_t)IniTab[i].Cnt;
}
/*
clr EosUsrMsg[0] (here the msgs were counted)
*/
EosUsrMsg[0] = 0;
/*
```

```
    last index is for EOS_NULL
*/
Msg                = (index_t*)MsgAdr;
Msg[0]             = Idx;
MsgTab [Idx].Msg = Msg + 1;
MsgTab [Idx].Hom = (unsigned short)IniMax;
MsgTab [Idx].Usr = 0;
/*
initialize the EOS_NULL pointers (private and public)
*/
MsgNull = Msg + 1;
EosNull = MsgNull;
/*
consistency test of our calculations and assignments
Idx must be equal to MsgMax
*/
Cnt = (unsigned)MsgTab[MsgMax].Msg - (unsigned)MsgTab[0].Msg;
Cnt = Cnt + sizeof(index_t) + SizMax;
if (Cnt != Mem) EOS_Err(EOS_MSGINIT_SYS_2, 0, 0, 0);
/*
it was a long way
*/
return (0);
}

/***** EOF *****/
```

