Konzeption und Implementierung portabler Anwendungssoftware zur grafischen Kapazitätsdisposition im Rahmen der Produktionsplanung

Diplomarbeit

"Konzeption und Implementierung portabler Anwendungssoftware zur grafischen Kapazitätsdisposition im Rahmen der Produktionsplanung"

Diplomarbeit am Fachbereich Informatik der Universität Dortmund

Autor: Andreas Heidemann

Betreuer: Professor Dr. Karl Kurbel, Inhaber des Lehrstuhls für Wirtschaftsinformatik am Fachbereich Wirtschafts- und Sozialwissenschaften der Universität Dortmund.

Erklärung :

Ich habe diese Diplomarbeit ohne fremde Hilfe und ohne andere als die im Literatur-verzeichnis angegebenen Quellen erstellt.

Dortmund, den 30. Mai 1989

Andreas Heidemann

An dieser Stelle möchte ich meinem Betreuer Herrn Prof. Dr. Karl Kurbel und Herrn Dipl. Math. Jürgen Meynert für Anregungen und konstruktive Kritik meinen Dank aussprechen.

Ich widme diese Arbeit meinen Eltern und Großeltern, die mir das Studium erst ermöglicht haben.

Der Anfang ist
schon die Hälfte
des Ganzen

Aristoteles

Inhaltsverzeichnis

1. Einleitung

Diese Arbeit beschäftigt sich mit der **Konzeption und Implementierung portabler Anwendungssoftware zur grafischen Kapazitätsdisposition im Rahmen der Produktionsplanung**.

Ziel der Arbeit ist die Konzeption eines Softwaresystems für die Kapazitätsdisposition bei der Produktionsplanung mit Hilfe grafischer Interaktionsformen. Belastungsdiagramme, die die Auslastung von Maschinengruppen periodisch darstellen, sollen auf Bildschirmgrafiken abgebildet und mit einem Pointing-Device direkt manipuliert werden. Diese Manipulationen stoßen unmittelbar Funktionen der Produktionsplanung an.

Die Tragfähigkeit der Konzepte soll durch eine exemplarische Implementierung unter Beweis gestellt werden. Der dialogorientierte Charakter dieser Funktionen impliziert hohe Anforderungen an die Performance des Systems. Dafür müssen Lösungen entwickelt werden, wie die darzustellenden Daten am günstigsten zu organisieren sind und wie die Grafikfunktionen effizient implementiert werden können. Für die Grafikfunktionen sollen existierende Funktionsbibliotheken auf ihre Eignung für diese Aufgabe untersucht werden.

Ein besonderer Schwerpunkt liegt auf der Anforderung, daß das Programmsystem mit möglichst geringem Aufwand auf unterschiedliche Rechnersysteme portierbar sein soll. Dies ist sowohl im Hinblick auf das zugrundeliegende Betriebssystem (UNIX, MS-DOS,...), wie auch auf die unterschiedliche Hardwareausstattung der einzelnen Rechnersysteme zu berücksichtigen. Dieser Aspekt grenzt die Kandidaten für geeignete Grafikbibliotheken weiter ein (GKS, X-WINDOWS, ...). Dabei sollen die Window-Systeme X-WINDOWS der UNIX-Welt und das Microsoft-Produkt MS-WINDOWS bezüglich der Konzepte und Leistungen miteinander verglichen werden.

In Kapitel zwei wird die Einordnung der Kapazitätsdisposition innerhalb eines Modells der gesamten Produktionsplanung kurz dargestellt. Das Kapitel drei enthält grundlegende Überlegungen zur Konzeption der manuellen Kapazitätsplanung. Dabei finden moderne grafische Interaktionsformen besondere Berücksichtigung. In Kapitel vier werden Aspekte der Portabilität und ihre Umsetzung bei der Implementierung behandelt. In Kapitel fünf werden Entwicklungsumgebungen und Bibliotheken untersucht, die grafische Interaktionen unterstützen. In Kapitel sechs schließlich wird die konkrete Implementation der grafischen Kapazitätsdisposition beschrieben. Die Dokumentation des erstellten Programms schließt in Kapitel sieben an. Den Anhang bilden dann die Quelltexte und das Literaturverzeichnis.

2. Kapazitätsdisposition im Rahmen der Produktionsplanung

Der Begriff der Produktionsplanung umfaßt viele unterschiedliche Planungsaktivitäten für verschiedene Zeithorizonte. Innerhalb der gesamten Unternehmensplanung beinhaltet die Produktionsplanung "die Festlegung des Produktionsgeschehens für einen zukünftigen Zeitraum"[1]. In der Literatur gibt es unterschiedliche Ansätze, die Produktionsplanung zu gliedern. Eine gute Zusammenfassung ist von Kurbel (1983) erschienen.

Für die computergestützte Planung wird eine Gliederung nach Funktion und Zeitbezug gewählt. Die Produktionsplanung gliedert sich danach in die strategische, die taktische und die operative Produktionsplanung.

Die strategische Planung hat einen langfristigen Planungshorizont. Hier fallen Entscheidungen über die generellen Produktfelder und die Organisation der Produktion. Die Auswahl der Fertigungsverfahren und die Entscheidung über die grundsätzliche Auslegung der Fertigungskapazitäten gehören ebenfalls zur strategischen Planung.

Im mittelfristigen Planungshorizont liegt die taktische Produktionsplanung. Hier werden Produktionsprogramm und Produkte konkretisiert und die Ausstattung mit Personal und Produktionsanlagen festgelegt.

---

[1] s. Kurbel 1983, S.12

Die operative Produktionsplanung schließlich plant die Durchführung des Produktionsprozesses auf kurzfristige Sicht. Hier wird im **Produktionsprogramm** festgelegt, welche Mengen von welchem Produkt gefertigt werden. Der Bedarf an Einzelteilen sowie Baugruppen und Zwischenerzeugnissen etc. wird in der **Bedarfsplanung** festgelegt. Das **Termingerüst**, welches von Durchlaufzeiten und Maschinenkapazitäten mitbestimmt wird und letztlich die **Maschinenbelegungsplanung**, d.h. die genaue Planung, wann welche Maschine welches Produkt fertigt, gehören ebenfalls in den Rahmen der operativen Produktionsplanung.

Die Erstellung des Termingerüstes wird auch mit dem Begriff **Kapazitätsdisposition** bezeichnet. Die Kapazitätsdisposition soll für eine gleichmäßige Auslastung der Kapazitäten sorgen und sowohl Überbelastung als auch Unterbelastung vermeiden. Da die Entscheidung über die verfügbare Maschinenkapazität aber bereits innerhalb der taktischen Produktionsplanung für einen längeren Zeitraum fällt, muß die Kapazitätsdisposition mit den gegebenen Kapazitäten auskommen.

Im Wesentlichen handelt es sich also um eine Glättung der Kapazitätsanforderungen. Diese Glättung soll durch Verschiebung von einzelnen Arbeitsaufträgen erreicht werden. Diese Arbeitsaufträge befinden sich aber innerhalb eines Auftragsnetzes, dessen hinterer Eckpunkt der Liefertermin und dessen vorderer Eckpunkt der Zeitpunkt ist, an dem frühestens alle benötigten Teile für diesen Arbeitsgang bereitgestellt sein können. Während der hintere Ecktermin (Liefertermin) ungern überschritten wird, da Termintreue heute in vielen Branchen den Kunden gegenüber ein wichtiges Qualitätsmerkmal ist, ist es beim vorderen Ecktermin physikalisch unmöglich, ihn zu unterschreiten.

Innerhalb dieser vorgegebenen Grenzen (und notfalls auch über den Liefertermin hinaus) besteht aber oft ein großer Dispositionsspielraum. Um diesen Spielraum auszunutzen bedarf es sowohl des Sachverstandes des im speziellen Betrieb und in dessen Betriebsabläufen eingearbeiteten Sachbearbeiters, als auch ausgefeilter Computerprogramme, um dem Sachbearbeiter Routineaufgaben, langwierige Berechnungen und auch umständliche Terminierungen und Umterminierungen abzunehmen. Die Aufgaben dieses Computerprogramms werden im Rahmen dieser Arbeit eingegrenzt und beispielhaft implementiert.

3. Konzeption einer manuellen Kapazitätsdisposition mit Hilfe grafischer Interaktionsformen

Herkömmliche Kapazitätsdisposition ist immer ein Arbeiten mit langen Zahlenkolonnen. Wenn der Sachbearbeiter mit Computerunterstützung arbeitet, so war bisher entweder eine aufwendige Optimierung vom Computer oder wieder der Umgang mit den einzelnen Belastungszahlen vorgesehen. Der Nachteil der Optimierungsverfahren ist vor allem der hohe Aufwand, der für sie notwendig ist, und komplexe Randbedingungen der Produktion, die sich nicht immer mit vertretbarem Aufwand im Rechnermodell erfassen lassen. Dies gilt vor allem für kleine und mittlere Unternehmen. Hier soll nun versucht werden, die Problemlösungskompetenz des einzelnen Sachbearbeiters durch geeignete Computerunterstützung wieder in den Vordergrund zu stellen[2].

Ausgangspunkt für eine manuelle Kapazitätsdisposition mit grafischen Interaktionsformen ist die grafische Darstellung der Kapazitätsbedarfe und des Kapazitätsangebotes: das Kapazitätsgebirge. In dieser Grafik kann der Sachbearbeiter Überlast und Überkapazität direkt sehen. So ist schon auf den ersten Blick die Entscheidung möglich, ob ein Ausgleich herbeigeführt werden muß und ob dieser Ausgleich durch Verschieben errreicht werden kann, oder ob andere Maßnahmen ergriffen werden müssen (z.B. "verlängerte Werkbank" od. Überstunden).

---

[2] vgl. Kurbel 1983, S.82,83

Die Darstellung der Kapazitäten als Kapazitätsgebirge ist nicht neu. Bisher allerdings waren diese Grafiken nur darstellendes Medium, und direkte Manipulationen mit Auswirkungen auf die Betriebsplanung waren nicht möglich. Hier liegt nun der Ansatzpunkt für einen effizienten Mensch-Maschine Dialog. Die Grafik wird vom Sachbearbeiter mit geeigneten Mitteln direkt so umgestaltet, daß die gewünschte Glättung der Kapazitätsanforderungen erreicht wird und weder Über- noch Unterbelastung auftritt.

Dabei werden direkt in der Kapazitätsgrafik Arbeitsgänge erkannt, die Überlastungen auslösen. Sie können mit der Maus "angefaßt", und dann an Stellen geschoben werden, an denen die Kapazität noch nicht ausgelastet ist.

Soll ein solches Verschieben durch direkte Manipulation an der Grafik möglich sein, so muß das Programm folgenden Funktionsumfang haben :

1. Die im Betrieb vorhandenen Betriebsmittel müssen einzeln bzw. als Betriebsmittelgruppe für die grafische Darstellung ausgewählt werden können.

2. Die für die Darstellung notwendigen Daten sollen möglichst direkt aus einer vorhandenen Datenbank gelesen und die Änderung wieder in diese Datenbank geschrieben werden. Wenn für die grafische Kapazitätsdisposition eine eigene Datenbank angelegt würde, so wäre die Konsistenz der Daten innerhalb des Betriebes gefährdet. Die Kapazitätsdisposition muß also an vorhandene Datenbanken angepaßt werden können.

3. Einzelne Aufträge müssen in der Grafik identifiziert und angewählt werden können. Dies geschieht im Idealfall mittels eines pointing device, z.B. Maus oder Lichtgriffel. Wichtige Informationen über den Auftrag, wie zum Beispiel Auftragsnummer, Status des Auftrages oder die Ecktermine, müssen entweder direkt erkennbar sein, oder auf Abruf am Bildschirm erscheinen.

4. Das Verschieben selber muß ebenfalls mit dem pointing device direkt in der Grafik ausgeführt werden können. Die entsprechenden Änderungen sollen in der Grafik anschließend sichtbar sein.

5. Da eine neue Terminierung sehr zeitaufwendig sein kann, wird sie bei Bedarf vom Benutzer explizit angestoßen. Die Terminierung selber ist nicht Bestandteil dieses Programms zur grafischen Kapazitätsdisposition, sondern eine externe Funktion.

Aus diesen Überlegungen ergeben sich die einzelnen Funktionen für das Programm: Für den Zugriffschutz der Datenbank können Benutzername und ein Password angegeben werden (**LOGIN**). Mit einem Kommando **Betriebsmittelauswahl** werden Betriebsmittelgruppen (oder evtl. Einzelbetriebsmittel), die in der Datenbank vorhanden sind, zur Auswahl am Bildschirm angezeigt. Nach der Wahl einer Betriebsmittelgruppe wird das Kapazitätsgebirge auf dem Bildschirm angezeigt.

Die Darstellung am Bildschirm kann mit zwei Funktionen geändert werden. Zunächst kann die Zeitachse so eingestellt werden, daß jeweils ein Tag, eine Woche oder ein Monat als Basiseinheit verwendet, und die Grafik entsprechend dargestellt wird. Außerdem kann zwischen unterschiedlichen Zeitgrenzen[3] für Beginn und Ende, die in der Datenbank vorhanden sind, umgeschaltet werden.

Die Arbeitsgänge werden jeweils als horizontale Folge übereinander gestapelter Blöcke dargestellt. Die Höhe entspricht dabei der voraussichtlichen zeitlichen Belastung des angezeigten Betriebsmittels für den ausgewählten Basiszeitraum (Tag, Woche oder Monat); die Farbe soll den Status darstellen. Die verfügbare Kapazität wird als eine Linie in der entsprechenden Höhe gezeichnet.

Mit Hilfe der Maus kann ein Block angewählt werden; die Arbeitsgangnummer und die zugrundeliegenden Zeiten werden dann angezeigt und alle zu diesem Arbeitsgang gehörenden Blöcke entsprechend schraffiert. Bei Bedarf können aus der Datenbank zusätzliche Informationen zu diesem Arbeitsgang angefordert und angezeigt werden.

Die Umplanung von Arbeitsgängen geschieht ebenfalls mit der Maus durch Anwählen, Festhalten und Verschieben von Blöcken. Auf der Grafik kann die Auswirkung der Verschiebung auf die Auslastung direkt beobachtet werden. Erst wenn dieses Ergebnis zufriedenstellend ist, soll die möglicherweise zeitaufwendige Neuplanung erfolgen.

---

[3] Im vorhandenen PPS-System handelt es sich um den frühesten und spätesten Start- und Endtermin

Eine Neuplanung der verschobenen Arbeitsgänge (mit den Auswirkungen auf das gesamte Auftragsnetz) mit den vom Sachberbeiter durch das Verschieben vorgegebenen Zeiten kann nach unterschiedlichen Terminierungsalgorithmen angestoßen werden.

Die im Programm verfügbare Hilfe besteht aus einer Legende der grafischen Symbole und der Farben und Schraffuren, sowie aus der Anzeige von Texten zur Erläuterung der wichtigsten Funktionen.

Als letzte Funktion muß schließlich auch noch das Beenden des Programms möglich sein. Der Sachbearbeiter wird vorher noch darauf hingewiesen, daß seit der letzten Neuplanung keine Änderungen in die Datenbank übernommen sind.

3.1 Qualitätskriterien

Wichtige Qualitätskriterien bei diesem Programm sind gute Performance und eine möglichst hohe Portabilität. Diesen Kriterien ist schon bei der Konzeption des Programms Beachtung zu schenken.

Aufgrund des interaktiven Charakters der gestellten Aufgabe sind akzeptable Antwortzeiten unabdingbar, wie dies auch schon in der Aufgabenbeschreibung ausgeführt ist. Diesem Ziel müssen sich gegebenenfalls andere Ziele unterordnen. Wenn das Programm aufgrund nicht akzeptabler Antwortzeiten nicht eingesetzt werden kann, sind alle anderen Entwicklungsziele obsolet.

Darüberhinaus ist die Arbeitsgeschwindigkeit des Programms auch ein Kostenfaktor. In gewissem Rahmen läßt sich ein langsames Programm durch besonders schnelle Hardware kompensieren, so daß die Akzeptanz durch den Anwender wieder gegeben ist. Diese höhere Geschwindigkeit der Hardware wird aber in der Regel durch überproportional höhere Anschaffungskosten der Hardware erkauft. Einige Programme erleben erst dann ihren Durchbruch am Markt, wenn die dafür nötige Hardware erschwinglich wird.

Die Forderung nach hoher Portabilität ergibt sich aus der Entwicklungsumgebung und den möglichen Einsatzrechnern. Entwickelt wird das Programm auf einem IBM-AT kompatiblen Rechner unter MS-DOS. Der mögliche Einsatz auf UNIX-Workstations und auf PS/2 - Rechnern soll aber so leicht wie möglich gemacht werden. Darüberhinaus stellt die Kapazitätsdisposition einen Teil eines elektronischen Fertigungsleitstandes dar, der gemäß seiner Konzeption jeweils an schon bestehende, unterschiedliche PPS-Systeme angepaßt werden soll.

Neben diesen vorrangingen Anforderungen dürfen aber auch weitere Qualitätsmerkmale nicht vernachlässigt werden. Dazu zählen besonders die Adaptibilität und die Wartbarkeit des Programms. Diese beiden Anforderungen ergeben sich vor allem aus dem prototypenhaften Charakter dieses Programms. Auf Wünsche möglicher Benutzer bezüglich der Oberfläche oder der realisierten Funktionen sollte schnell und ohne großen Aufwand reagiert werden können. Da diese möglichen Änderungen mit großer Wahrscheinlichkeit zunächst mit neuen Fehlern im Programm verbunden sind, ist auch eine gute Wartbarkeit vonnöten.

Die Forderung nach Universalität braucht für dieses Programm hingegen nicht erhoben zu werden. Es geht in erster Linie um das Aufzeigen der Möglichkeiten, mittels grafischer Interaktionen die Kapazitätsplanung für den Sachbearbeiter einfacher und übersichtlicher zu gestalten. Durch die Forderung nach Adaptibilität besteht anschließend auch die Möglichkeit, fehlende Funktionen schnell und einfach zu realisieren. Der Nachteil schlechterer Effizienz durch Universalität, wenn nicht benötigte Funktionen das Programm verlangsamen, soll schließlich vermieden werden.

4. Ausführungen zur Portierbarkeit im Hinblick auf die gestellte Aufgabe

Unter Portieren versteht man im Allgemeinen das Übertragen eines Programms von einer Umgebung (Hardware, Betriebssystem, Softwarebibliotheken etc.) in eine andere Umgebung[4]. Je höher der Aufwand für eine derartige Übertragung ist, desto geringer ist die **Portabilität** eines Programms.

Diesen Übertragungsaufwand muß man insbesondere unter Kostengesichtspunkten betrachten. Spätestens dann, wenn es billiger ist, das Programm komplett neu zu schreiben, wird man an Portierung nicht mehr denken. Während bei gegebener Problemstellung die Kosten einer Neuimplentierung durch vorhergehende Implementierungen nicht beeinflußt werden, kann man die Kosten einer Portierung durch geeignete Maßnahmen durchaus senken.

Der Aufwand für eine Portierung ist auch direkt abhängig von der Zielumgebung. Ein Programm kann z.B. innerhalb eines Betriebssystems oder einer Betriebssystemgruppe leicht portierbar sein, die Umstellung auf ein ganz anderes Betriebssystem kann dagegen einen unvertretbar hohen Aufwand erforderlich machen.

Die Übertragung eines Programms von einer Programmiersprache in eine andere wird naturgemäß dem Aufwand für eine Neuimplementierung nahekommen. Aus diesem Grund wird für die folgenden Überlegungen davon ausgegangen, daß die Programmiersprache auch in der Zielkonfiguration verfügbar ist. Diese Voraussetzung schränkt den Kreis der verwendbaren Programmiersprachen schon ein. Maschinensprache und Assemblerprogrammierung fallen hier aus, da sie nur auf jeweils einem Maschinentyp verfügbar sind.

---

[4] vgl. Balzert 1982, S.12 u. Kurbel 1983 S.112

Bei der Auswahl der Programmiersprache ist ferner darauf zu achten, daß sie eine hinreichend hohe Verbreitung hat und auch in Zukunft von den Herstellern unterstützt wird.

Der Aufwand, der für eine Portierung nötig ist, ergibt sich aus der Abhängigkeit eines Programms von bestimmten Komponenten der konkreten Umgebung. Das können sowohl Hardwarekomponenten als auch Betriebssystem und ergänzende Systemsoftware sein.

Jede einzelne Quelle der Abhängigkeit ist bei der Untersuchung der Portierbarkeit zunächst getrennt zu betrachten. Man kann für fertiggestellte Programme einen Abhängigkeitsgrafen zeichnen, aus dem der Portierungsaufwand sowohl insgesamt als auch für einzelne Bereiche ersichtlich wird (Abb. 1). Jede einzelne Achse steht dabei für eine Quelle der Abhängigkeit des Programms bzw. für eine Kostenquelle bei einer möglichen Portierung. Je weiter außen der Graf die Achse schneidet, desto höher sind die entsprechenden Kosten. Die vom Graf beschriebene Fläche entspricht also den Portierungskosten auf eine komplett neue und andere Umgebung.

Abb. 1

Entsprechend diesem Diagramm kann man jetzt auch ein Anforderungsprofil zeichnen, in dem festgehalten wird, wie wichtig für jeden einzelnen Aspekt die Unabhängigkeit bzw. Portierbarkeit ist.

Bei dem Entwurf eines Programms ist den unterschiedlichen Aspekten der Portabilität von vornherein genügend Aufmerksamkeit zu schenken. Folgende Fragen müssen geklärt werden :

1. Auf welche möglichen Zielsysteme / Zielkonfigurationen soll das Programm später portiert werden ?

2. Von welchen Umgebungskomponenten soll das Programm unabhängig sein bzw. von welchen Komponenten darf ggf. eine Abhängigkeit bestehen ?

3. Welche verfügbaren Module sollen für das Programm benutzt werden und welche Funktionsbereiche werden selber programmiert ?

Frage 1 :

Wenn die Zielsysteme einer möglichen Portierung bekannt sind, kann die spätere Portierung schon bei der Programmerstellung gezielt vorbereitet werden. Dazu werden zunächst die Unterschiede der Zielsysteme untersucht und festgehalten. Anschließend ist bei der Programmierung sorgsam darauf zu achten, daß die Unterschiede entweder gar nicht tangiert werden, oder mindestens softwaretechnisch abgekapselt werden.

Diese Kapselung wäre im Idealfall ein Modul; denkbar ist aber auch das Konzentrieren der betroffenen Anweisungen an einer eingegrenzten Stelle im Quelltext, oder die textuelle Auslagerung in eine Include-Datei.

Frage 2 :

Die Frage nach den Komponenten ist eng verbunden mit der ersten Frage nach den möglichen Zielsystemen. Von einer Komponente, die auf allen Zielsystemen vorhanden ist, kann man sich abhängig machen, ohne Portabilität in Richtung auf die Zielsysteme zu verlieren. Es ist aber dennoch eine eigenständige Entscheidung, sich auf Dauer an eine bestimmte Komponente - sei sie auch noch so verbreitet - zu binden, oder sich auch hier unabhängig zu halten.

Frage 3 :

Einer der scheinbar einfachsten Wege, herstellerunabhängig zu sein, ist die Eigenprogrammierung der benötigten Module. Aber in Wirklichkeit macht man sich dadurch nur von sich selbst abhängig, da die Erstellung und Portierung des Moduls einen erheblichen eigenständigen Aufwand erfordern kann.

Hier ist eine Abschätzung erforderlich zwischen den Kosten der Module (und deren Verbreitung) einerseits und den Kosten einer Eigenimplementierung andererseits. Als Kosten treten dabei sowohl Kauf-/Implementierungskosten als auch die Kosten einer etwaigen Portierung auf.

4.1 Portabilität und andere Entwicklungsziele

Die Bestrebungen nach Portabilität tangieren auch andere Entwicklungsziele und Qualitätsmerkmale[5]. So steht eine hohe Portabilität im Widerspruch zu einer optimalen Performance, da auf der einen Seite Systemunterschiede bewußt nicht benutzt werden, zur Erzielung optimaler Laufzeiten aber das Ausnutzen von diesen Unterschieden wichtig ist. Ein möglicher Ansatzpunkt, hier einen Kompromiß zu finden, wird im Folgenden aufgezeigt.

Größere Programme benutzen in der Regel nicht nur die Hardware als extern hergestelltes "Teil", sondern auch unterschiedliche Bibliotheken von Standardsoftware. Dazu gehören im vorliegenden Fall Grafik- und Datenbanksoftware. In dieser Standardsoftware liegt der Schlüssel zu Effizienz und Portabilität. Da diese Bibliotheken eine wesentlich höherere Verbreitung erzielen als spezielle Anwendungsprogramme, lohnt es sich für die Hersteller dieser Software eher, speziell angepaßte Versionen für die optimale Ausnutzung unterschiedlicher Hardware zu entwickeln. Auf der Ebene des Anwendungsprogramms muß zur Erhaltung der Portabilität nur noch sichergestellt werden, daß die benutzte Software auf hinreichend vielen Rechnersystememem verfügbar ist, bzw. daß das Programm bezüglich dieser benutzten Software leicht änderbar ist.

Andere Ansätze zur Erreichung hoher Portabilität vernachlässigen die Erfordernisse guter Effizienz teilweise erheblich. So ist auch der Versuch zu verstehen, einen eigenen allgemeinen Modul zur Erledigung der nötigen Aufgaben zu schreiben, welcher seinerseits dann Standardsoftware benutzt. Die Vorteile liegen auf der Hand: dieser Modul kann in mehreren Programmen verwendet werden und bei Portierungen genügen Änderungen im entsprechenden Modul.

---

[5] vgl. Kurbel 1983, S.132,141

Dieses Verfahren birgt allerdings beträchtliche Nachteile. So erfordert schon die Entwicklung dieses allgemeinen Moduls mehr Aufwand als eine einfache aufgabenspezifische Schnittstelle. Dieser Aufwand rentiert sich nur dann, wenn der Modul wirklich in mehreren unterschiedlichen Programmen zum Einsatz kommt. Aufgrund des beträchtlichen Overheads durch die Allgemeinheit des Moduls wird sich die Effizienz deutlich verschlechtern und bei Portierungen muß der gesamte Umfang des Moduls portiert werden, obwohl nur jeweils ein Teil davon überhaupt benötigt wird.

Verbesserungen anderer Qualitätsmerkmale gehen zum Teil mit einer hohen Portabilität Hand in Hand. So erhöhen sich durch die klare Abgrenzung und Gliederung die Adaptibilität[6] und die Wartbarkeit[7].

[6] vgl. Kurbel 1983, S.134
[7] vgl. Kurbel 1983, S.141

5. Entwicklungsumgebungen für grafische Interaktionsformen und ihre Eignung für die spezifizierten Aufgaben

Im Folgenden werden beispielhaft zwei Produkte (Microsoft Windows und X Windows) auf ihre Eignung für die spezifizierten Aufgaben untersucht. Obwohl beide Produkte sowohl grafische Funktionen als auch ein Fensterkonzept beinhalten, werden diese Bereiche hier getrennt behandelt, da es sich um völlig unterschiedliche Funktionsbereiche handelt. Es gibt Grafikbibliotheken, die keinerlei Fensterfunktionen beinhalten (z.B. Grafik aus No Limit[8]) und es sind fensterorientierte Systeme denkbar, die nur mit zeichenorientierter Grafik arbeiten, also ohne Grafik im eigentlichen Sinne auskommen. Darüberhinaus sollten unterschiedliche Funktionsbereiche auch aus Gründen für eine Verbesserung der Portierbarkeit durch eine übersichtliche Modularisierung hier getrennt untersucht werden.

5.1 Grafische Bibliotheken

Der grafische Teil von MS-Windows bzw. X Windows ist eine Programmbibliothek im klassischen Sinn, d.h. eine Sammlung von Unterprogrammen, welche grafische Ausgaben auf Bildschirm und anderen Medien unterstützen. An dieser Stelle werden zum einen beide Produkte auf ihre Eignung für die grafischen Aufgaben des zu behandelnden Problems untersucht, zum anderen werden hier auch schon Erkenntnisse gesammelt für die portable Implementation. Funktionen, die sich als spezielle Eigenheiten eines Produktes herausstellen, sollten dabei für die Implementation nicht genutzt werden, da sich daraus bei späteren Portierungen Probleme bzw. zusätzlicher vermeidbarer Aufwand ergeben.

---

[8] vgl. MEF Environmental 1986, S.12-25

Die grafischen Anforderungen, die eine Kapazitätsdisposition stellt, sind im Vergleich zu anderen grafischen Programmen (z.B. CAD) gering. Für ein Kapazitätsgebirge müssen Blöcke unterschiedlicher Farbe bzw. Schraffur, ein Skalenkreuz mit Beschriftung und die Linie der verfügbaren Kapazität dargestellt werden. Für die Manipulation der Darstellung muß ein pointing device wie z.B. eine Maus unterstützt werden. Diese Anforderungen sollten von jeder Grafikbibliothek abgedeckt werden können.

Weitere Aspekte für die Eignung ergeben sich aus dem interaktiven Charakter des Programms und der Absicht, möglichst leicht portierbar zu sein. Die Geschwindigkeit der grafischen Operationen sollte hinreichend groß sein, um die Akzeptanz beim Anwender zu erhöhen, bzw. überhaupt erst zu ermöglichen. Wenn der Anwender die Enstehung der Grafik bequem mit dem Auge verfolgen kann, so wird schnell beim Arbeiten ein schädliches Ungeduldsgefühl entstehen. Das wird dazu führen, daß in hektischen Situationen eher ohne den Computer und am Computer vorbei geplant wird. Das aber würde den Einsatz des Programms konterkarieren, da es ja gerade die Produktivität eines Sachbearbeiters quantitativ erhöhen und qualitativ verbessern soll.

Um das Ziel der guten Portierbarkeit zu erreichen, kann man insbesondere auch auf eine hohe Verbreitung der benutzten Bibliothek achten. Wenn für die neue Zielmaschine die benutzte Grafikbibliothek existiert, liegt es auf der Hand, daß es einfacher ist, für die neue Zielmaschine die identische Grafiksoftware zu beschaffen, als das Programm an eine andere Grafiksoftware anzupassen. Daher muß in eine Beurteilung der Eignung auch die derzeitige Verbreitung und eine Einschätzung über die zukünftige Verbreitung - auch auf neuer Hardware - eingehen.

5.1.1 MS-Windows

Der oben erwähnte geringe benötigte Funktionsumfang ist in der Grafikbibliothek von MS-Windows vollständig enthalten[9].

MS-Windows stellt zum Zeichnen unterschiedliche Abstraktionsebenen zur Verfügung. Auf der einfachsten Ebene wird in Pixeln auf Basis der realen Bildschirmgröße gezeichnet. Der Ursprung des Koordinatensystems befindet sich dabei in der linken oberen Fensterecke. Es kann auch eine Zeichenfläche definiert werden, aus der ein kleiner, verschiebbarer Ausschnitt auf dem Bildschirm dargestellt wird. Zusätzlich kann auch ein beliebiger Maßstab statt der Pixel benutzt werden.

Farben und Schraffierungen können frei gewählt, die Schraffierungen auch vom Benutzerprogramm frei definiert werden. Sie werden für einen bestimmten Zeichenbereich ausgewählt und gelten dann für alle folgenden Zeichenoperationen.

Die Zeichengeschwindigkeit von MS-Windows wurde mit einfachen Beispielprogrammen getestet und kann als durchaus zufriedenstellend angesehen werden. Insbesondere das Füllen von Flächen, welches sich bei anderen PC-Grafikprogrammen als unangenehm langsam erwiesen hat (z.B. die Füllfunktion aus der No Limit[10] Bibliothek), ergibt hier keine merklichen Verzögerungen mehr.

---

9 vgl. Microsoft 1985

10 MEF Environmental 1986

MS-Windows läuft unter MS-DOS auf allen IBM-kompatiblen Personalcomputern. Zumindest für den kommerziellen Bereich bedeutet das, daß MS-Windows für fast alle installierten Personalcomputer erhältlich ist. Dadurch bräuchte man nur für die PC-Welt keinerlei Überlegungen bezüglich Portierung anstellen. Auch die Zukunft von MS-Windows erscheint sicher, da Microsoft eng mit IBM zusammenarbeitet und daher als Marktführer auf dem PC-Softwaremarkt einzustufen ist. MS-Windows wird ständig weiterentwickelt und auch für neue Prozessoren für eine optimale Leistungsfähigkeit angepaßt[11]. Auch das neue Betriebssystem für die IBM PS/2 Serie (BS/2 bzw. OS/2) enthält eine von Microsoft entwickelte Window-Software (Presentation Manager), die sehr eng an MS-Windows angelehnt ist.

5.1.2 X Windows

Der oben erwähnte geringe benötigte Funktionsumfang ist auch in der Grafikbibliothek von X Windows vollständig enthalten[12].

In X Windows werden alle Maße ausschließlich in Pixel angegeben. Der Ursprung des Koordinatensystems liegt in der linken oberen Ecke des definierten Fensters. Andere gewünschte Koordinatensysteme müssen in X Windows vom Anwendungsprogramm realisiert werden.

Wie in MS-Windows so können auch in X Windows Farben und Schraffierungen definiert und für einen bestimmten Zeichenbereich ausgewählt werden. Die Vorgehensweise dafür ist in X Windows etwas komplizierter aber dennoch prinzipiell gleich.

---

[11] vgl. Duncan 1988
[12] vgl. Nye 1988

Die Arbeitsgeschwindigkeit der Grafik in X Windows konnte nicht eingehend untersucht werden, da keine Version des Programms zur Verfügung stand. Da aber zum Beispiel die Grafiksoftware der SUN Workstations - unter anderem für den CAD Einsatz - auf X Windows basiert, besteht ein berechtigter Grund für die Annahme, daß die Geschwindigkeit für die hier geforderten Zwecke zumindest ausreichend ist.

X Windows ist ein Fenstersystem, das extra als Standard herstellerunabhängig konzipiert worden ist. Alle namhaften Rechnerhersteller im UNIX-Bereich (HP, DEC, SUN, Apollo, etc.) arbeiten inzwischen an der Weiterentwicklung von X Windows mit. Damit erscheint die Zukunft zumindest im UNIX-Bereich sicher. Auch PC-Implementationen von X Windows sind inzwischen erhältlich, so daß man davon ausgehen kann, daß eine Implementation auf Basis von X Windows bezüglich möglicher Portierungen keine falsche Entscheidung ist.

5.2 Window-Systeme

Das Fenstersystem von MS-Windows bzw. X/Windows kann man nicht nur als Programmbibliothek begreifen. Zum einen werden hier Funktionen angeboten, mit deren Hilfe sich die Fenster anzeigen, verschieben oder löschen lassen, zum anderen wird aber für das Fensterkonzept eine neue Kontrollflußphilosophie benutzt.

Die Betrachtungen hinsichtlich der Verbreitung und absehbaren Zukunft der beiden Produkte wurden schon im vorigen Kapitel angestellt und werden hier nicht wiederholt. Sie sind selbstverständlich auch für die hier besprochenen Fensterkomponenten von X Windows und MS-Windows gültig.

5.2.1 MS-Windows

Programme in MS-Windows besitzen eine Kontrollfluß-strategie, die von MS-Windows vorgegeben wird. Der Programmablauf wird auf der oberen Ebene durch Ereignisse gesteuert, die entweder von außen kommen (beispielsweise Tastatureingaben oder Mausbewegungen) oder vom Programm selber generiert werden. Dies geschieht in einer Warteschleife, in der das Hauptprogramm auf Ereignisse wartet. Bei Eintreten eines beliebigen Ereignisses wird es vom Hauptprogramm aus auf Unterprogramme verteilt, die für jedes Fenster einzeln definiert diese Ereignisse behandeln. Die Reaktionen auf diese Ereignisse werden dann in herkömmlicher Art und Weise prozedural formuliert.

Benutzereingaben erfolgen innerhalb von MS-Windows über Maus oder Tastatur und sind immer einem bestimmten Fenster zugeordnet. Eine besondere Möglichkeit, Benutzereingaben zu machen, bieten die Menüs. Sie werden vom Programmierer definiert und erzeugen spezielle Ereignisse, wenn sie vom Benutzer angewählt werden. Auf diese Art und Weise läßt sich einfach eine komfortable Benutzeroberfläche programmieren.

Zusätzlich zum üblichen Quellcode benötigt MS-Windows noch eine Datei in der die Fensterarten und -größen und die Menüs definiert werden. Die hierin enthaltenen Informationen werden von einem speziellen Ressourcecompiler übersetzt und beim Linken zum eigentlichen Programm hinzugefügt.

5.2.2 X Windows

Auch X Windows besitzt wie MS-Windows eine ereignisgesteuerte Kontrollflußphilosophie. Die Verarbeitung der Ereignisse erfolgt wie beim Microsoft Produkt im Hauptprogramm in einer Ereigniswarteschlange, die die Ereignisse dann weitergibt an die bearbeitenden Funktionen.

Alle Eingaben von Tastatur und Maus sind diesem Konzept zufolge Ereignisse. Etwas komplizierter ist in X Windows die Behandlung von Menüs. Die Menüs sind bewußt aus X Windows ausgeklammert worden und bilden eine Aufgabe für den sogenannten Window Manager. Dieser Window Manager kann sich bei verschiedenen Implementationen von X Windows unterscheiden und so Ursache für notwendige Änderungen bei Portierungen sein. Allerdings wird ein einfacher Beispiel-Window Manager mitgeliefert (uwm[13]).

5.3 Resümee

Beide Produkte sind für die beabsichtigten Zwecke offensichtlich gut geeignet. Auf Personalcomputern besitzt MS-Windows mindestens zur Zeit noch Vorteile durch die höhere Verbreitung und die Erfahrungen, die mit dieser Software schon gemacht wurden.

Dies kann in Zukunft anders aussehen, wenn die Möglichkeiten der erheblich einfacheren Portierung vom PC auf UNIX-Systeme und umgekehrt durch Benutzung von X Windows an Bedeutung zunimmt und X Windows dadurch auf PC-Ebene eine höhere Verbreitung erfährt.

Ein Nachteil von X Windows bezüglich hoher Portabilität ist sicherlich das Fehlen eines standardisierten Window Managers. Dies kann nur durch sorgfältige Programmierung abgefangen werden, in dem man das Programm auf den Einsatz mit beliebigen Window Managern vorbereitet.

---

[13] vgl. Nye 1988, S.5 u. Gancarz 1986

6. Beschreibung der Implementierung

6.1 Auswahl der Umgebung

Das Programm wurde auf einem MCI-AT04 und einem IBM PS/2 Modell 80 implementiert. Die zugrundeliegende Softwareumgebung besteht aus dem Betriebssystem MS-DOS 3.3, dem Datenbanksystem ORACLE, dem Fenstersystem MS-Windows und der Programmiersprache C (Compiler von Microsoft, Version 5.1).

Die Hardware ergibt sich aus der Ausstattung des Lehrstuhls für Wirtschaftsinformatik. Die Entscheidung für MS-Windows und MS-DOS fiel aus praktischen Gründen der Verfügbarkeit. Zu Beginn dieser Arbeit war X Windows noch nicht am Lehrstuhl verfügbar. Durch die Entscheidung für MS-Windows ist dann auch die Entscheidung für MS-DOS vorgezeichnet, da andere Implementationen von MS-Windows nicht vorliegen und auch nicht angekündigt sind.

Die Entscheidung für das Datenbanksystem ORACLE hat im Wesentlichen zwei Gründe. Erstens ist ORACLE mit der gleichen Schnittstelle (bei steigendem Leistungsumfang) vom PC bis zum Mainframe erhältlich. Zudem erfolgen die ORACLE-Aufrufe in der standardisierten Abfragesprache SQL. Für eine spätere Portierbarkeit bietet diese Eigenschaft beste Vorraussetzungen. Zweitens ist das Datenbanksystem ORACLE Grundlage für das am Lehrstuhl für Wirtschaftsinformatik entwickelte PPS-System, dessen Datenbank die erste Arbeitsgrundlage für die Kapazitätsdisposition darstellt.

Die Wahl der Programmiersprache C begründet sich in vielen unterschiedlichen Aspekten. Zunächst spielen wieder Überlegungen zur Portabilität eine Rolle. Die Sprache C ist heute auf fast allen Rechnern verfügbar und wird das in absehbarer Zukunft vermutlich auch bleiben. C ist Implementierungssprache für das Betriebssystem UNIX und partizipiert aus diesem Grund an dem Aufschwung von UNIX. Zudem ist C wegen des geringen Sprachumfangs sowohl leicht zu lernen als auch leicht zu implementieren. Einzig Mängel in der Klarheit des Quellcodes und im Modularisierungskonzept sprechen gegen C; diese sind aber durch disziplinierte Programmierung in den Griff zu bekommen.

Weitere Gründe für die Verwendung von C liegen in den Schnittstellen zu verschiedenen Softwarebibliotheken. Für keine Sprache gibt es im PC- und UNIX-Bereich eine derart große Auswahl an Bibliotheken. Das verwendete Datenbanksystem ORACLE und auch beide Windowsysteme haben C-Schnittstellen, so daß die logische Wahl auf diese Programmiersprache fällt.

6.2 Modularisierung

Die vollständige Aufgabe muß zu Beginn in Teilaufgaben zur vernünftigen Modularisierung zerlegt werden. Dabei spielen einerseits die Überlegungen zur Portabilität eine Rolle, d.h. Berührungspunkte zu benutzten Softwarebibliotheken sollten möglichst lokal innerhalbs eines Moduls auftreten. Das bedeutet für die Modularisierung eine Moduleinteilung nach benutzter Software. Andererseits sollte sich die Modularisierung auch immer nach den Aufgaben richten, die innerhalb eines Moduls bearbeitet werden. Das wäre eine Moduleinteilung nach der Leistung der Software.

Glücklicherweise widersprechen sich diese beiden Konzepte zur Moduleinteilung nicht, sondern sie ergänzen sich gegenseitig. Wichtig ist, in diesem Zusammenhang noch einmal an die grundsätzliche Zweiteilung der Aufgaben von MS-Windows zu erinnern: auf der einen Seite sind die grafischen Funktionen zu sehen, und auf der anderen Seite, getrennt von der Grafik, die Fensterverwaltung und Ereignissteuerung.

Aufgrund der oben angeführten Überlegungen wird die Gesamtaufgabe auf folgende Module verteilt :

- KAP_PLAN Hauptprogramm
- PAINT Zeichenfunktionen
- DB_SQL Datenbankaufrufe
- DATEN Aufbereitung der Datenbankwerte

Die Module KAP_PLAN, DB_SQL und DATEN werden als getrennt zu übersetzende C-Module realisiert. Bei dem Modul PAINT wird beispielhaft aufgezeigt, wie eine Modularisierung auch durch Aufteilung in Include-Dateien erfolgen kann. Durch die Einschachtelung von PAINT in das Hauptprogramm KAP_PLAN wird gleichzeitig erreicht, daß sich das gesamte Windowsystem inclusive Grafik aus der Sicht des Compilers auf einen Modul beschränkt.

Extern benutzte Software ist einmal das Datenbanksystem ORACLE und das Fenstersystem MS-Windows. Ein weiterer externer Modul ist die Terminierung. Die Funktionen zur Terminierung gehören nicht in den Rahmen dieser Diplomarbeit.

Bei den Modulschnittstellen muß auf eine aufgabenadäquate Definition geachtet werden. Das bedeutet z.B. für die Datenbankfunktionen, daß nicht nur um jeden Datenbankaufruf eine neue Prozedurkapsel geschrieben wird, um für andere Programmteile unabhängig von der speziellen Datenbanksoftware zu werden. Vielmehr wird mit den Prozeduren und Datenstrukturen der Schnittstelle eine neue, ganz speziell auf die Aufgabe zugeschnittene Sicht auf die Datenbank definiert. Durch die reduzierte Anzahl der Prozeduraufrufe wegen des speziellen Zuschnitts der Modulschnittstelle erhält man eine bessere Effizienz. Denn gerade Prozeduraufrufe kosten in der Regel viel Prozessorzeit (Kontextwechsel etc.). Die Vorteile der Modularisierung bezüglich der Portabilität bleiben dennoch erhalten.

6.3 Vorgehensweise

Zunächst wurde die grafische Darstellung des Kapazitätsgebirges implementiert. Dafür wurde eine Dummyversion des Datenbankzugriffsmoduls geschrieben, um die grafische Darstellung unabhängig von eventuellen Fehlern in der Datenbank schreiben und testen zu können.

Für die Daten, die in der Darstellung des Kapazitätsgebirges benötigt werden, wurde eine umfassende Datenstruktur implementiert[14]. Durch Übergabe von einem einzelnen Zeiger auf diese Struktur werden die Funktionsaufrufe für die Grafikfunktionen optimiert.

[14] s. Quelltext (KAP_PLAN.H, Zl.137 ff.)

Noch ohne die Datenbank wurde die Geschwindigkeit der einzelnen zeichnenden Funktionen getestet. Wo es notwendig war, wurden Datenstrukturen verbessert und Algorithmen umgestellt oder verändert, bis die Performance der Grafik zufriedenstellend war. Anschließend wurden auch die Datenbankfunktionen aus dem Modul DB_SQL eingebunden und das Gesamtsystem getestet.

Um während der Implementierung bei auftretenden Fehlern die im Programm durchlaufenen Pfade an kritischen Stellen feststellen zu können, wurden Testhilfeausgaben auf die logische Datei **stddbg** eingefügt. Die Übersetzung dieser Testhilfen kann durch Anweisungen an den C-Preprozessor gesteuert werden, so daß diese Testhilfen das fertige Programm später nicht belasten.

Die Arbeit mit MS-Windows und ORACLE hat bis an die Grenzen der Belastbarkeit des Betriebssystems geführt. Nach Reduktion des Platzbedarfes für das Datensegment durch Auslagern von größeren Variablen in eigene Segmente mit Hilfe einer Option im C-Compiler und der Einführung der neuesten Version des Windows-Development-Toolkits (Version 2.1) waren die Speicherprobleme aber behoben.

7. Dokumentation des erstellten Programms

7.1 Beschreibung der Module

Das Programm Kapazitätsplanung ist in mehrere Module unterteilt. Die Einteilung in die Gruppen erfolgte aufgrund der Art der im jeweiligen Modul implementierten Funktionen sowie aufgrund der für die Implementierung benötigten Softwarebibliotheken.

Die drei Module sind im Einzelnen : **KAP_PLAN**, **DATEN** und **DB_SQL**. Der Hauptmodul KAP_PLAN ist dabei noch durch Verwendung mehrerer Source Code Dateien weiter unterteilt.

7.1.1 Das Hauptprogramm KAP_PLAN

Im Hauptprogramm ist die für Windows benötigte Ablaufsteuerung implementiert. Das sind alle Funktionen, die von Windows aus als Reaktionen auf Benutzereingaben aufgerufen werden.

In jeweils einer eigenen Datei innerhalb des Hauptprogramms sind die globalen Variablen (**KPL_DCL.INC**) deklariert und die Zeichenfunktionen (**PAINT.INC**) implementiert.

7.1.1.1 Fensterverwaltung in KAP_PLAN

Die Funktionen für die Verwaltung der Fenster und die Reaktionen auf Benutzereingaben stehen direkt im Hauptprogramm. Die Beschreibung der benutzten Fenster und die Beschreibung des Menüs des Hauptfensters steht im Anhang unter **KAP_PLAN.RC**.

7.1.1.2 Zeichenfunktionen in PAINT.INC

In der Datei **PAINT.INC** sind die Zeichenfunktionen zusammengefaßt. Änderungen in der Schnittstelle zu den benutzten Grafikroutinen sollten sich daher im Wesentlichen auf Änderungen in dieser Datei beschränken.

7.1.2 Der Modul DB_SQL

Im Modul DB_SQL sind alle Aufrufe an die verwendete Datenbank[15] zusammengefaßt. Aus dem Modul herausgereicht werden für jede Sicht auf die Datenbank drei Funktionen. Eine Funktion dient zum Einleiten einer Datenbankanfrage (INIT...List), eine Funktion zum Beenden der eingeleiteten Datenbankanfrage (END...List) und die dritte Funktion liefert Datensatz für Datensatz die gewünschten Daten aus der Datenbank (NEXT...List).

Es gibt drei verschiedene Sichten auf die Datenbank im Modul DB_SQL: eine Liste der vorhandenen Betriebsmittelgruppen (...BMGList), eine Liste der für jeden einzelnen Tag zur Verfügung stehenden Maschinenkapazität einer Betriebsmittelgruppe (...BestList) und eine Liste der Arbeitsgänge für eine Betriebsmittelgruppe (...FAGList).

Zusätzlich gibt es noch Funktionen zum Ein- bzw. Ausloggen aus der Datenbank (Login, Logout) und zum Abbrechen bzw. Bestätigen einer Transaktion (Rollback, Commit).

Die Datumsfunktionen (stdtowoche, stdtodate, datetostd) sind ebenfalls im Modul DB_SQL implementiert, um die Abhängigkeit der Datumsdarstellung zur verwendeten Datenbank nicht nach außen sichtbar werden zu lassen. Die Datumsfunktionen verwenden allerdings aus Performancegründen keinen Datenbankaufruf.

---

[15] Zur Zeit wird das Datenbanksystem ORACLE benutzt.

Die Fehlernummer bei Fehlern des Datenbanksystems wird transparent durchgereicht. Den entsprechenden Text zu einer Fehlernummer liefert die Prozedur **ErrorText** aus dem Modul heraus.

7.1.3 Der Modul DATEN

Im Modul DATEN sind alle Operationen zusammengefaßt, die den Datentransfer aus der Datenbank über den Modul DB_SQL abwickeln, oder die zur Aggregation oder Manipulation der Daten aus der Datenbank benötigt werden. Insbesondere werden von diesem Modul aus auch die Funktionen zur Neuterminierung aufgerufen.

7.2 Beschreibung der Aufrufstruktur in WINDOWS

Die Kommunikation zwischen der Windows Benutzeroberfläche und dem Programm KAP_PLAN geschieht mit jeweils einer eigenen Funktion für jedes Fenster, um Benutzereingaben für dieses Fenster zu behandeln. Da diese Funktionen nicht direkt voneinander aufgerufen werden, sondern über Windows, müssen für die Kommunikation zwischen den Prozeduren globale Variable verwendet werden.

7.3 Beschreibung der Datenstrukturen

In diesem Programm werden vier eigene Datenstrukturen verwendet, deren Aufbau im Folgenden beschrieben wird[16]:

7.3.1 BLOCK

Die Datenstruktur **BLOCK** beschreibt ein Kästchen der Darstellung eines Fertigungsarbeitsganges. Die Komponenten **bl_slot** und **bl_auftrag** geben die Position dieses Kästchens in der Datenstruktur **KAPDAT** an, wo weitere Daten zu diesem Kästchen stehen. Die Komponente **bl_basis** gibt den unteren Rand des Kästchens direkt an, der sonst erst aus Angaben in **KAPDAT** errechnet werden müßte.

7.3.2 FA_DATEN

Die Datenstruktur **FA_DATEN** enthält Daten eines Fertigungsauftrages. Diese Daten werden benötigt, wenn genauere Informationen zu einem Fertigungsauftrag abgefragt werden.

Die Bedeutung der einzelnen Komponenten ergibt sich aus der Definition der Datenbank, den Namen der Bezeichner aus der Datenbank wurde hier lediglich ein "fad_" vorangestellt.

---

[16] Die genaue Definition der Datenstrukturen steht im Quelltext (KAP_PLAN.H).

7.3.3 FAG

Die Datenstruktur **FAG** enthält Daten eines Fertigungsarbeitsganges. Diese Daten werden benötigt, um den Arbeitsgang grafisch darzustellen.

Die Bedeutung der einzelnen Komponenten ergibt sich aus der Definition der Datenbank, den Namen der Bezeichner aus der Datenbank wurde hier lediglich ein "fag_" vorangestellt.

Es gibt noch zwei zusätzliche Bezeichner: **fag_veraendert** zeigt an, ob dieser Arbeitsgang schon einmal grafisch umgeplant wurde, ohne eine erneute Terminierung anzustoßen, und **fag_next** ist ein Zeiger auf den nächsten Fertigungsarbeitsgang.

7.3.4 KAPDAT

In der Datenstruktur **KAPDAT** sind alle Informationen über die Fertigungsarbeitsgänge, die Maschinenkapazität und die grafische Darstellung zusammengefaßt.

Die Liste der Fertigungsarbeitsgänge fängt bei der Komponente **fag_liste** an, die Maschinenkapazität ist in der Komponente **kap_best** abgespeichert und die grafische Darstellung ist in dem Array **kap_bed** abgelegt. Die weiteren Komponenten dienen der Speicherung der Koordinaten der Statuszeile (**statuszeile**), enthalten die Koordinaten des Hauptfensters oder andere Hilfsinformationen, die jeweils zur Berechnung und zur Darstellung des Kapazitätsgebirges benötigt werden.

7.4 Beschreibung der Testausgaben

Während der Erprobungszeit des Programms können als Testhilfe verschiedene Testausgaben mitübersetzt werden. Die Testausgaben werden in eine Debug-Datei geschrieben. Die Übersetzung der Testausgaben und der Name der Debug-Datei werden über define-Konstanten in der Datei KAP_PLAN.H[17] gesteuert.

7.5 Beschreibung der Funktionen

7.5.1 KAP_PLAN

7.5.1.1 ErrorMsg

Funktionswert : void

Parameter : char *msg in
char *titel in

globale Variable : HWND hActWnd in

Beschreibung : Die Funktion **ErrorMsg** gibt den Text **msg** mit der Überschrift **titel** in einem Fenster aus. Zusätzlich wird ein Warnton ausgegeben. Der Benutzer muß die Fehlermeldung quittieren.

[17] s. Quelltext (KAP_PLAN.H)

7.5.1.2 ClearMsgQueue

Funktionswert : void

Parameter : HWND hWnd in/out

globale Variable : keine

Beschreibung : Löscht die Message-Warteschlange, wenn bei langer Wartezeit dort Messages stehen, die vermutlich nicht mehr aktuell sind.

7.5.1.3 AboutKapPlan

Funktionswert : BOOL

Parameter : HWND hDlg in
unsigned message in
WORD wParam in
LONG lParam in

globale Variable : keine

Beschreibung : Steuert den Empfang der Benutzerbestätigung beim Anzeigen des **About**-Fensters. Funktionswert ist **FALSE** wenn Fehler beim Fenster auftreten, sonst **TRUE**.

7.5.1.4 DebugCmd

Funktionswert : int

Parameter : HWND hDlg in
WORD wParam in
LONG lParam in

globale Variable : keine

Beschreibung : Steuert den Empfang der Benutzerbestätigung beim Anzeigen des **Debug**-Fensters. Liefert als Funkti-

onswert immer TRUE.

7.5.1.5 DebugInfo

Funktionswert : BOOL

Parameter :	HWND	hDlg	in
	unsigned	message	in
	WORD	wParam	in
	LONG	lParam	in

globale Variable :	KAPDAT	*hKapDat	in

Beschreibung : Diese Funktion zeigt im zugeordneten Fenster ausgewählte Daten für Kontrollzwecke an; die Funktion ist nur für die Zeit der Programmentwicklung bestimmt. Funktionswert ist **FALSE**, wenn Fehler beim Fenster auftreten, sonst **TRUE**.

7.5.1.6 MarkDlg

Funktionswert : BOOL

Parameter :	HWND	hDlg	in
	unsigned	message	in
	WORD	wParam	in
	LONG	lParam	in

globale Variable :	FILE	*stddbg[18]	out
	FAG	*hAktFAG	in/out
	KAPDAT	*hKapDat	in

Beschreibung : Diese Funktion steuert die Benutzereingabe für das **Markierungs**-Fenster. Funktionswert ist **FALSE**, wenn Fehler beim Fenster auftreten, sonst **TRUE**.

[18] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.1.7 HelpDlg

Funktionswert : BOOL

Parameter :	HWND	hDlg	in
	unsigned	message	in
	WORD	wParam	in
	LONG	lParam	in

globale Variable :	FILE	*help_file	in
	TEXTMETRIC	tmStuff	in

Beschreibung : Diese Funktion wird von Windows aufgerufen, wenn das **Hilfe**-Fenster angezeigt wird, und steuert die Reaktion auf Benutzereingaben in diesem Fenster. Die Hilfetexte werden aus der Hilfedatei gelesen und im Fenster angezeigt. Funktionswert ist **FALSE**, wenn Fehler beim Fenster auftreten, sonst **TRUE**.

7.5.1.8 LegendeDlg

Funktionswert : BOOL

Parameter :	HWND	hDlg	in
	unsigned	message	in
	WORD	wParam	in
	LONG	lParam	in

globale Variable :	TEXTMETRIC	tmStuff	in
	HBRUSH	hBrNormal	in
	HBRUSH	hBrAktuell	in
	HBRUSH	hBrAngebot	in
	HBRUSH	hBrGrobInkons	in
	HBRUSH	hBrFeingeplant	in
	HBRUSH	hBrFeinInkons	in
	HBRUSH	hBrVerspaetet	in
	HBRUSH	hBrFreigegeben	in
	HBRUSH	hBrBegonnen	in

HBRUSH	hBrTeilRueck	in
HBRUSH	hBrGestoert	in
HBRUSH	hBrWartung	in
HPEN	hPenNormal	in
HPEN	hPenAktuell	in
int	RopAktuell	in

Beschreibung : Diese Funktion zeigt in dem entsprechenden Fenster die unterschiedlichen Farben der Arbeitsgänge und ihre Bedeutung an. Der Benutzer muß die Anzeige bestätigen. Funktionswert ist **FALSE**, wenn Fehler beim Fenster auftreten, sonst **TRUE**.

7.5.1.9 BMAuswahlCmd

Funktionswert : void

Parameter :	HWND	hDlg	in
	WORD	wParam	in
	LONG	lParam	in

globale Variable :	KAPDAT	*hKapDat	out

Beschreibung : Diese Funktion steuert die Auswahl einer Betriebsmittelgruppe aus der Liste der zulässigen Betriebsmittelgruppen.

7.5.1.10 BMAuswahlDlg

Funktionswert : BOOL

Parameter : HWND hDlg in
unsigned message in
WORD wParam in
LONG lParam in

globale Variable : HCURSOR hArrowCursor in

Beschreibung : Diese Funktion zeigt im Betriebsmittelauswahlfenster eine Liste der zulässigen Betriebsmittelgruppen an und steuert die notwendige Benutzereingabe in dieses Fenster. Funktionswert ist **FALSE**, wenn Fehler beim Fenster auftreten, sonst **TRUE**.

7.5.1.11 DatenDlg

Funktionswert : BOOL

Parameter : HWND hDlg in
unsigned message in
WORD wParam in
LONG lParam in

globale Variable : FAG *hAktFAG in
HCURSOR hArrowCursor in
TEXTMETRIC tmStuff in

Beschreibung : Diese Funktion zeigt im Datenfenster ausgewählte Daten des aktuellen Fertigungsarbeitsganges, die vorher aus der Datenbank gelesen werden. Anschließend muß der Benutzer den Empfang der Daten bestätigen. Funktionswert ist **FALSE**, wenn Fehler beim Fenster auftreten, sonst **TRUE**.

7.5.1.12 LoginDlg

Funktionswert : BOOL

Parameter :			
	HWND	hDlg	in
	unsigned	message	in
	WORD	wParam	in
	LONG	lParam	in

globale Variable :			
	FILE	*stddbg[19]	out
	KAPDAT	*hKapDat	out

Beschreibung : Diese Funktion fragt aus dem Login-Fenster die Benutzeridentifikation für das Datenbanksystem ab (userid & password) und stellt eine Verbindung zur Datenbank her. Der Funktionswert ist **FALSE**, wenn Fehler auftreten, sonst **TRUE**.

7.5.1.13 InitKapDat

Funktionswert : void

Parameter :			
	KAPDAT	*hKapDat	out

globale Variable : keine

Beschreibung : Diese Funktion initialisiert die Daten in hKapDat für den ersten Gebrauch.

[19] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.1.14 KapPlanInit

Funktionswert : BOOL

Parameter : HANDLE hInstance in

globale Variable : int MessageLength out
char *szAppName out
char *szMessage out
char *szAbout out

Beschreibung : Diese Funktion übernimmt Initialisierungen für Window-Funktionen. Der Funktionswert ist **TRUE** wenn keine Fehler aufgetreten sind, sonst **FALSE**.

7.5.1.15 WinMain

Funktionswert : int

Parameter : HANDLE hInstance in
HANDLE hPrevInstance in
LPSTR lpszCmdLine in
int cmdShow in

globale Variable : FILE *stddbg[20] out
HBRUSH hBrNormal out
HBRUSH hBrAktuell out
HBRUSH hBrAngebot out
HBRUSH hBrGrobInkons out
HBRUSH hBrFeingeplant out
HBRUSH hBrFeinInkons out
HBRUSH hBrVerspaetet out
HBRUSH hBrFreigegeben out
HBRUSH hBrBegonnen out
HBRUSH hBrTeilRueck out

[20] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

HBRUSH	hBrGestoert	out
HBRUSH	hBrWartung	out
HPEN	hPenNormal	out
HPEN	hPenAktuell	out
int	RopAktuell	out
HANDLE	hInst	out
HWND	hAktWnd	out
FAG	*hAktFAG	out
HCURSOR	hArrowCursor	out
HCURSOR	hWaitCursor	out
FARPROC	lpprocMarkDlg	out
FARPROC	lpprocAbout	out
FARPROC	lpprocHelpDlg	out
FARPROC	lpprocDebug	out
FARPROC	lpprocBMAuswahl	out
FARPROC	lpprocLoginDlg	out
FARPROC	lpprocDatenDlg	out
FARPROC	lpprocLegende	out
char	*szAppName	in
char	*szMessage	in
KAPDAT	*hKapDat	out

Beschreibung : Diese Funktion ist der prozedurale Kern dieses Programms. Alle globalen Initialisierungen werden hier durchgeführt oder angestoßen. Anschließend geht diese Funktion in eine Warteschleife zur Behandlung und Weitergabe von Messages des Window-Systems.

7.5.1.16 process_scroll_cmds

Funktionswert : BOOL

Parameter : WORD cmd in
LONG pos in
KAPDAT *hKapDat in/out

globale Variable : FILE *stddbg[21] out

Beschreibung : Diese Funktion verarbeitet vom Benutzer im Hauptfenster eingegebene Kommandos zum Verschieben der Darstellung. Im Fehlerfall ist der Funktionswert **FALSE**, sonst **TRUE**.

[21] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.1.17 KapPlanWndProc

Funktionswert : long int

Parameter :	HWND	hWnd	in
	unsigned	message	in
	WORD	wParam	in
	LONG	lParam	in

globale Variable :	FILE	*stddbg[22]	out
	KAPDAT	*hKapDat	in/out
	FAG	*hAktFag	in/out
	HBRUSH	hBrAktuell	in
	HPEN	hPenAktuell	in
	int	RopAktuell	in

Beschreibung : Diese Funktion bearbeitet alle Meldungen und Benutzereingaben des Hauptfensters. Der Funktionswert ist 0 im fehlerfreien Fall, sonst wird eine Fehlernummer zurückgegeben.

7.5.1.18 TestEnde

Funktionswert : BOOL

Parameter :	HWND	hWnd	in

globale Variable : keine

Beschreibung : Diese Funktion fragt den Benutzer in einem Mitteilungsfenster, ob das Programm wirklich beendet werden soll. Wenn das Programm beendet werden soll, ist der Funktionswert TRUE, sonst FALSE.

---

[22] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.1.19 Menu

Funktionswert : void

Parameter :	int	cmd	in
	HWND	hWnd	in
	KAPDAT	*hKapDat	in/out

globale Variable :	FILE	*stddbg[23]	out
	HCURSOR	hArrowCursor	in
	HCURSOR	hWaitCursor	in
	FAG	*hAktFAG	in/out
	FARPROC	lpprocMarkDlg	in
	FARPROC	lpprocAbout	in
	FARPROC	lpprocHelpDlg	in
	FARPROC	lpprocDebug	in
	FARPROC	lpprocBMAuswahl	in
	FARPROC	lpprocLoginDlg	in
	FARPROC	lpprocDatenDlg	in
	FARPROC	lpprocLegende	in

Beschreibung : Diese Funktion bearbeitet jede Menüauswahl eines Benutzers im Hauptfenster.

[23] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.2 PAINT

7.5.2.1 SelectBlockObjects

Funktionswert : void

Parameter :	HDC	hDC	in/out
	int	Status	in

globale Variable :	HBRUSH	hBrNormal	in
	HBRUSH	hBrAngebot	in
	HBRUSH	hBrGrobInkons	in
	HBRUSH	hBrFeingeplant	in
	HBRUSH	hBrFeinInkons	in
	HBRUSH	hBrVerspaetet	in
	HBRUSH	hBrFreigegeben	in
	HBRUSH	hBrBegonnen	in
	HBRUSH	hBrTeilRueck	in
	HBRUSH	hBrGestoert	in
	HBRUSH	hBrWartung	in
	HPEN	hPenNormal	in

Beschreibung : Diese Funktion setzt die Voreinstellungen zu Zeichnen (Farbe u. Schraffur) so, wie es der Status erfordert.

7.5.2.2 SkalaZeichnen

Funktionswert : void

Parameter : HWND hWnd in
HDC hDC in
KAPDAT *hKapDat in

globale Variable : FILE *stddbg[24] out
TEXTMETRIC tmStuff in

Beschreibung : Diese Funktion zeichnet das Koordinatenkreuz und die Skalierung und Beschriftung daran.

7.5.2.3 KapBestZeichnen

Funktionswert : void

Parameter : HWND hWnd in
HDC hDC in
KAPDAT *hKapDat in

globale Variable : HPEN hLinePen in

Beschreibung : Diese Funktion zeichnet die Linie der verfügbaren Maschinenkapazität.

[24] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.2.4 KapBedZeichnen

Funktionswert : void

Parameter : HWND hWnd in
HDC hDC in
KAPDAT *hKapDat in

globale Variable : keine

Beschreibung : Diese Funktion zeichnet die Blöcke der eingeplanten Fertigungsarbeitsgänge.

7.5.2.5 StatFormZeichnen

Funktionswert : void

Parameter : HWND hWnd in
HDC hDC in
KAPDAT *hKapDat in/out

globale Variable : HPEN hPenNormal in
TEXTMETRIC tmStuff in

Beschreibung : Diese Funktion berechnet und zeichnet den Rahmen und die stehende Beschriftung für die Statuszeile.

7.5.2.6 StatRepZeichnen

Funktionswert : void

Parameter :	HDC	hDC	in
	KAPDAT	*hKapDat	in
	FAG	*hFAG	in

globale Variable :	FILE	*stddbg[25]	out

Beschreibung : Diese Funktion zeichnet die variablen Teile der Statuszeile für die Fertigungsarbeitsgänge.

7.5.2.7 StatBMNRZeichnen

Funktionswert : void

Parameter :	HDC	hDC	in
	KAPDAT	*hKapDat	in

globale Variable : keine

Beschreibung : Diese Funktion zeichnet den Eintrag in der Statuszeile für die Betriebsmittelgruppe.

[25] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.2.8 KapPlanPaint

Funktionswert : void

Parameter :

HWND	hWnd	in
HDC	hDC	in
KAPDAT	*hKapDat	in/out

globale Variable :

FILE	*stddbg[26]	out
TEXTMETRIC	tmStuff	in/out

Beschreibung : Diese Funktion zeichnet das Bild im Hauptfenster nach den Daten in hKapDat neu.

7.5.2.9 IdentifyBlock

Funktionswert : void

Parameter :

HWND	hWnd	in
KAPDAT	*hKapDat	in
long	lPos	in
BLOCK	*hBlock	out

globale Variable : keine

Beschreibung : Diese Funktion bestimmt an Hand der Mausposition in lPos den Block auf den die Maus gerade zeigt, und den dazugehörigen Fertigungsarbeitsgang.

[26] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.2.10 ZeichneBlock

Funktionswert : void

Parameter :	HDC	hDC	in
	KAPDAT	*hKapDat	in
	BLOCK	*hBlock	in

globale Variable : keine

Beschreibung : Diese Funktion zeichnet den angegebenen Block in der eingestellten Farbe.

7.5.2.11 ZeichneFag

Funktionswert : void

Parameter :	HDC	hDC	in
	KAPDAT	*hKapDat	in
	FAG	*hFAG	in

globale Variable :	FILE	*stddbg[27]	out

Beschreibung : Diese Funktion zeichnet den angegebenen Fertigungsarbeitsgang in der eingestellten Farbe.

[27] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.2.12 BlockVerschieben

Funktionswert : void

Parameter : long lParam in
KAPDAT *hKapDat in/out
HWND hWnd in

globale Variable : FAG *hAktFAG in/out

Beschreibung : Diese Funktion überprüft, ob die gewünschte Verschiebung auf den durch lParam angegebenen Punkt durchführbar ist[28] und ändert gegebenenfalls die entspechenden Daten des Arbeitsganges.

7.5.2.13 BlockSchieben

Funktionswert : void

Parameter : HWND hWnd in
long lParam in
KAPDAT *hKapDat in

globale Variable : FAG *hAktFAG in

Beschreibung : Diese Funktion zeigt während des Hin- und Herschiebens eines Arbeitsganges die jeweiligen Start- und Endezeiten in der Statuszeile an.

[28] Es werden nur einfache Randbedingungen überprüft; es wird keine Terminierung durchgeführt.

7.5.2.14 TrackMousePosition

Funktionswert : void

Parameter : HWND hWnd in
long lParam in
KAPDAT *hKapDat in

globale Variable : keine

Beschreibung : Diese Funktion aktualisiert bei Bewegungen der Maus den Eintrag in der Statuszeile, so daß immer die Daten des Arbeitsganges angezeigt werden, auf den die Maus zeigt, bzw. die Daten des aktuellen Arbeitsganges.

7.5.3 DATEN

7.5.3.1 UpdateFAG

Funktionswert : BOOL

Parameter : FAG *hFag in/out
int cmd in
int *error out

globale Variable : FILE *stddbg[29] out

Beschreibung : Diese Funktion stößt für den angegebenen Fertigungsarbeitsgang eine Umterminierung an. Die Art der Terminierung steht in cmd. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst ist er **FALSE** und in **error** wird die Fehlernummer zurückgegeben.

[29] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.3.2 umplanen

Funktionswert : void

Parameter : KAPDAT *hKapDat in/out
int cmd in

globale Variable : FILE *stddbg[30] out

Beschreibung : Diese Funktion führt für alle geänderten Arbeitsgänge eine Umterminierung durch. Fehler beim Umterminieren werden dem Benutzer sofort mitgeteilt.

7.5.3.3 read_fags

Funktionswert : BOOL

Parameter : KAPDAT *hKapDat in/out

globale Variable : FILE *stddbg[31] out

Beschreibung : Diese Funktion liest aus der Datenbank die benötigten Fertigungsarbeitsgänge. Dabei wird dynamisch neuer Speicherplatz angefordert, um die Daten im Hauptspeicher zu halten. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**.

---

[30] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

[31] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.3.4 read_kap_best

Funktionswert : BOOL

Parameter : KAPDAT *hKapDat in/out

globale Variable : keine

Beschreibung : Diese Funktion liest aus der Datenbank die benötigten Maschinenkapazitäten. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**.

7.5.3.5 darstellung_berechnen

Funktionswert : void

Parameter : KAPDAT *hKapDat in/out

globale Variable : FILE *stddbg[32] out

Beschreibung : Diese Funktion berechnet aus den Daten der Fertigungsarbeitsgänge und den eingestellten Optionen die in Blöcken aufgeteilte Darstellung der Maschinenauslastung.

[32] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4 DB_SQL

Die große Zahl der globalen Variablen bei den nachfolgend beschriebenen Funktionen ist durch das Datenbanksystem ORACLE bedingt. Kommunikation mit dem Datenbanksystem kann nur über globale Variable erfolgen.

7.5.4.1 GetUID

Funktionswert : BOOL

Parameter : char *uid out
char *pwd out

globale Variable : VARCHAR sql_uid in
VARCHAR sql_pwd in

Beschreibung : Diese Funktion liefert die aktuelle Benutzeridentifikation für die Datenbank zurück. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**.

7.5.4.2 InitBMGList

Funktionswert : BOOL

Parameter : int *error out

globale Variable : SQLCA sqlca in

Beschreibung : Diese Funktion leitet die Datenbankrecherche nach den vorhandenen Betriebsmittelgruppen ein. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.3 EndBMGList

Funktionswert : BOOL

Parameter : int *error out

globale Variable : SQLCA sqlca in

Beschreibung : Diese Funktion schließt die Datenbankrecherche nach den vorhandenen Betriebsmittelgruppen ab. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.4 NextBMGList

Funktionswert : BOOL

Parameter : BMNR bmnr out
int *error out

globale Variable : SQLCA sqlca in
VARCHAR sql_bmnr in

Beschreibung : Diese Funktion liefert die nächste Betriebsmittelgruppe aus der aktuellen Datenbankrecherche. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.5 InitFAGList

Funktionswert :	BOOL			
Parameter :	char	bmnr []		in
	long int	startzeit	in	
	long int	endzeit	in	
	int	semode	in	
	int	error	out	
globale Variable :	FILE	*stddbg[33]		out
	SQLCA	sqlca	in	
	VARCHAR	sql_bmnr	out	

Beschreibung : Diese Funktion leitet die Datenbankrecherche nach den angeforderten Fertigungsarbeitsgängen ein. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.6 EndFAGList

Funktionswert :	BOOL			
Parameter :	int	*error		out
globale Variable :	SQLCA	sqlca		in

Beschreibung : Diese Funktion schließt die Datenbankrecherche nach den Fertigungsarbeitsgängen ab. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

---

[33] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4.7 NextFAGList

Funktionswert : BOOL

Parameter :	FAG	*hFag	out
	int	*error	out
globale Variable :	FILE	*stddbg[34]	out
	SQLCA	sqlca	in
	VARCHAR	sql_fanr	in
	VARCHAR	sql_fstart	in
	VARCHAR	sql_fend	in
	VARCHAR	sql_lstart	in
	VARCHAR	sql_lend	in
	long	sql_bedarf	in
	int	sql_fagnr	in
	int	sql_status	in

Beschreibung : Diese Funktion liefert den nächsten Fertigungsarbeitsgang aus der aktuellen Datenbankrecherche. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

[34] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4.8 InitBestList

Funktionswert : BOOL

Parameter :			
	char	bmnr []	in
	long int	startzeit	in
	long int	endzeit	in
	int	error	out

globale Variable :			
	FILE	*stddbg[35]	out
	SQLCA	sqlca	in
	long int	sql_akttag	in/out
	long int	sql_endtag	in/out
	VARCHAR	sql_bmnr	in/out
	long int	sql_guelvon	in/out
	long int	sql_guelbis	in/out
	int	sql_bmcount	in/out
	VARCHAR	sql_kalname	in/out
	long int	sql_fabtag	in/out

Beschreibung : Diese Funktion leitet die Datenbankrecherche nach den angeforderten Maschinenkapazitäten ein. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

[35] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4.9 EndBestList

Funktionswert : BOOL

Parameter :	int	*error	out

globale Variable :	FILE	*stddbg[36]	out
	SQLCA	sqlca	in

Beschreibung : Diese Funktion schließt die Datenbankrecherche nach den Maschinenkapazitäten ab. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.10 NextBestList

Funktionswert : BOOL

Parameter :	long int	*hBestand	out
	int	*error	out

globale Variable :	FILE	*stddbg[37]	out
	SQLCA	sqlca	in
	int	sql_bmcount	in
	VARCHAR	sql_bmnr	in
	long int	sql_akttag	in/out
	long int	sql_endtag	in
	long int	sql_guelvon	in/out
	long int	sql_guelbis	in/out
	int	sql_smodnr	in/out
	int	sql_minprotag	in/out
	long int	sql_fabtag	in/out

[36] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

[37] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

Beschreibung : Diese Funktion liefert die nächste Maschinenkapazität aus der aktuellen Datenbankrecherche. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.11 Login

Funktionswert : BOOL

Parameter :			
	char	name []	in
	char	passwd []	in
	KAPDAT	*hKapDat	out
	int	*error	out

globale Variable :			
	FILE	*stddbg[38]	out
	SQLCA	sqlca	in
	VARCHAR	sql_uid	in/out
	VARCHAR	sql_pwd	in/out
	long int	sql_sysdate	out
	long int	sql_starttag	out

Beschreibung : Diese Funktion stellt den Zugang zur Datenbank mit Hilfe der angegebenen Benutzeridentifikation her. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

[38] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4.12 Logout

Funktionswert : BOOL

Parameter : int *error out

globale Variable : FILE *stddbg[39] out
SQLCA sqlca in

Beschreibung : Diese Funktion beendet die aktuelle Verbindung zur Datenbank. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.13 Rollback

Funktionswert : BOOL

Parameter : int *error out

globale Variable : FILE *stddbg[40] out
SQLCA sqlca in

Beschreibung : Diese Funktion macht die Änderungen in der Datenbank während der letzten Transaktion wieder rückgängig. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

---

[39] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

[40] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4.14 Commit

Funktionswert : BOOL

Parameter : int *error out

globale Variable : FILE *stddbg[41] out
SQLCA sqlca in

Beschreibung : Diese Funktion bestätigt die Änderungen in der Datenbank während der letzten Transaktion und schließt damit diese Transaktion ab. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.15 FADaten

Funktionswert : BOOL

Parameter : FA_DATEN *fad in/out
int *error out

globale Variable : FILE *stddbg[42] out

SQLCA	sqlca	in
VARCHAR	sql_fanr	in/out
int	sql_fagnr	in/out
VARCHAR	sql_agbez	in/out
int	sql_agstat	in/out
long int	sql_gbzeit	in/out
VARCHAR	sql_kanr	in/out
int	sql_posnr	in/out
VARCHAR	sql_teilenr	in/out

[41] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

[42] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

VARCHAR	sql_var	in/out
VARCHAR	sql_me	in/out
int	sql_prio	in/out
VARCHAR	sql_bmengestr	in/out

Beschreibung : Diese Funktion liest zum angegebenen Fertigungsauftrag weitere benötigte Daten aus der Datenbank. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**. Die Fehlernummer wird in **error** zurückgegeben.

7.5.4.16 ErrorText

Funktionswert : BOOL

Parameter :	int	error	in
	char	errortxt []	out
globale Variable :	SQLCA	sqlca	in

Beschreibung : Diese Funktion liefert zu der angegebenen Fehlernummer den entsprechenden Fehlertext des Datenbanksystems. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**.

7.5.4.17 datetostd

Funktionswert : BOOL

Parameter :			
long	*std		out
int	tt	in	
int	mm	in	
int	jj	in	
int	ss	in	
int	min	in	
int	*error	out	

globale Variable :			
FILE	*stddbg[43]		out
struct tm	beg_time	in	

Beschreibung : Die Funktion rechnet den angegebenen Zeitpunkt (Tag **tt**, Monat **mm**, Jahr 19**jj**, Stunde **ss**, Minute **min**) um in Stunden seit dem 1. 1. 1988 0:00 Uhr (**beg_time**). Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**.

[43] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4.18 stdtodate

Funktionswert : BOOL

Parameter :	long	std	in
	int	*tt	out
	int	*mm	out
	int	*jj	out
	int	*ss	out
	int	*min	out
	int	*error	out

globale Variable :	FILE	*stddbg[44]	out
	struct tm	beg_time	in

Beschreibung : Die Funktion rechnet die angegebenen Stunden seit dem 1. 1. 1988 0:00 Uhr (**beg_time**) um in einen Zeitpunkt (Tag **tt**, Monat **mm**, Jahr 19**jj**, Stunde **ss**, Minute **min**). Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**.

[44] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.5.4.19 stdtowoche

Funktionswert : BOOL

Parameter :	long	std	in
	int	*woche	out
	int	*err	out

globale Variable :	FILE	*stddbg[45]	out
	struct tm	beg_time	in

Beschreibung : Die Funktion rechnet die angegebenen Stunden seit dem 1. 1. 1988 0:00 (**beg_time**) um in die entsprechende Kalenderwoche. Im fehlerfreien Fall ist der Funktionswert **TRUE**, sonst **FALSE**.

[45] Diese Variable wird nur dann benutzt, wenn das Programm mit den Testhilfen übersetzt wird. (s.a. "Beschreibung der Testausgaben")

7.6 Aufrufstruktur der Funktionen;[46]

7.6.1 AboutKapPlan

Die Funktion AboutKapPlan ruft keine weiteren Funktionen der Kapazitätsplanung auf.

7.6.2 BMAuswahlDlg

```
BMAuswahlDlg
  Ã- BMAuswahlCmd
  Ã- ClearMsgQueue
  Ã- EndBMGList
  Ã- ErrorTxt
  Ã- InitBMGList
  À- NextBMGList
```

7.6.3 DatenDlg

```
DatenDlg
  Ã- FADaten
  Ã- ErrorText
  À- ClearMsgQueue
```

7.6.4 DebugInfo

```
DebugInfo
  Ã- DebugCmd
  À- GetUID
```

7.6.5 HelpDlg

Die Funktion HelpDlg ruft keine weiteren Funktionen der Kapazitätsplanung auf.

7.6.6 LegendeDlg

Die Funktion LegendeDlg ruft keine weiteren Funktionen der Kapazitätsplanung auf.

---

[46] Aufrufe von Funktionen, die zu C, ORACLE oder Windows gehören, werden hier nicht mit aufgelistet.

7.6.7 LoginDlg

```
LoginDlg
  Ã- Login
  À- ErrorText
```

7.6.8 MarkDlg

```
MarkDlg
  À- ClearMsgQueue
```

7.6.9 KapPlanWndProc

```
KapPlanWndProc
  Ã- Login
  Ã- Menu
  ³    Ã- Logout
  ³    Ã- ErrorText
  ³    Ã- darstellung_berechnen
  ³    Ã- ClearMsgQueue
  ³    Ã- read_fags
  ³    ³    Ã- InitFAGList
  ³    ³    Ã- NextFAGList
  ³    ³    Ã- EndFAGList
  ³    ³    Ã- ErrorText
  ³    ³    À- ErrorMsg
  ³    Ã- read_kap_best
  ³    ³    Ã- InitBestList
  ³    ³    Ã- NextBestList
  ³    ³    Ã- EndBestList
  ³    ³    Ã- ErrorText
  ³    ³    À- ErrorMsg
  ³    Ã- umplanen
  ³    ³    Ã- UpdateFAG
  ³    ³    ³    Ã- stdtodate
  ³    ³    ³    Ã- AGUMTERM
  ³    ³    ³    Ã- ENGTERM
  ³    ³    ³    À- datetostd
  ³    ³    Ã- ErrorMsg
  ³    ³    Ã- Rollback
  ³    ³    À- Commit
  ³    À- TestEnde
  Ã- BlockSchieben
  ³    À- stdtodate
  Ã- TrackMousePosition
  ³    Ã- IdentifyBlock
  ³    À- StatRepZeichnen
```

```
³         À- stdtodate
Ã- BlockVerschieben
Ã- darstellung_berechnen
Ã- ClearMsgQueue
Ã- IdentifyBlock
```

```
  Ã- ZeichneFAG
  ³    À- ZeichneBlock
  Ã- KapBestZeichnen
  Ã- SelectBlockObjects
  Ã- StatRepZeichnen
  ³    À- stdtodate
  Ã- process_scroll_cmds
  À- KapPlanPaint
       Ã- StatFormZeichnen
       Ã- StatBMNRZeichnen
       Ã- SkalaZeichnen
       ³    Ã- stdtodate
       ³    À- stdtowoche
       Ã- KapBedZeichnen
       ³    À- SelectBlockObjects
       À- KapBestZeichnen
```

7.6.9.1 WinMain

```
WinMain
  Ã- InitKapDat
  À- KapPlanInit
```

7.7 Benutzte Relationen der Datenbank

Relation	Feld
BM	BMGRP
	BMNR
	GRPFLG
BMBETR	BIS
	BMNR
	SMODNR
	VON
DUAL	SYSDATE
FA	BMENGE
	FANR
	ME
	PRIO
	TEILENR
	VAR
FABKAL	DATUM
	FABTAG
	NAME

FAG	AGBEZ
	AGSTAT
	FAGNR
	FANR
	FEND
	FSTART
	GBZEIT
	LEND
	LSTART
FAGBEL	BMNR
	FAGNR
	FANR
FKABEZ	FANR
	KANR
	POSNR
SCHICHT	ABBEG
	ABEND
	SMODNR

7.8 Liste der zugehörigen Dateien;[47]

KAP_PLAN	Makefile für das Programm
KAP_PLAN.LNK	Eingabedatei für LINK
KAP_PLAN.DEF	Programmbeschreibung für Windows
KAP_PLAN.RC	Ressourcenbeschreibung für Windows
KAP_PLAN.ICO	Window-Icon für das Programm
KAP_PLAN.HLP	Hilfetexte für die OnLine Hilfe
KPL_DEF.H	Headerdatei mit Konstantendefinitionen
KAP_PLAN.H	Headerdatei für KAP_PLAN
KAP_PLAN.DCL	Funktionsdeklarationen für das Hauptprogramm
DCL_VAR.INC	Include-Datei mit Variablendeklarationen
PAINT.INC	Include-Datei für die Zeichenfunktionen
KAP_PLAN.C	Quellcode für das Hauptprogramm
DB_SQL.DCL	Funktionsdeklarationen für den Modul DB_SQL
DB_SQL.PC	Quellcode für den Modul DB_SQL
DATEN.H	Headerdatei für den Modul DATEN
DATEN.DCL	Funktionsdeklarationen für den Modul DATEN
DATEN.C	Quellcode für den Modul DATEN
EGT_MS.LIB[48]	Bibliothek mit den Terminierungsfunktionen

[47] Dateien, die zum Betriebssystem, zum C-Compiler, zu Windows oder zum Datenbanksystem ORACLE gehören, werden hier nicht mit aufgelistet.

[48] Die Terminierungsfunktionen sind gesondert dokumentiert.

8. Anhang

8.1 Windows - Definitionsdateien

8.1.1 KAP_PLAN.DEF

```
 1 DESCRIPTION 'Kapazitaetsplanung '
 2
 3 STUB    'WINSTUB.EXE'
 4
 5 CODE PRELOAD MOVEABLE DISCARDABLE
 6 DATA PRELOAD FIXED MULTIPLE
 7
 8 HEAPSIZE   32768
 9 STACKSIZE  12000
10
11 SEGMENTS
12    _MOD1    PRELOAD MOVEABLE
13    _SQL1    PRELOAD MOVEABLE DISCARDABLE
14    _SQL2    LOADONCALL MOVEABLE DISCARDABLE
15
16 EXPORTS
17     KapPlanWndProc @1
18     AboutKapPlan   @2
19     DebugInfo      @3
20     LegendeDlg     @4
21     BMAuswahlDlg   @5
22     DatenDlg       @6
23     LoginDlg       @7
24     HelpDlg        @8
25     MarkDlg        @9
```

8.1.2 KAP_PLAN.RC

```
#include "windows.h"
#include "kpl_def.h"

KapPlan   ICON    kap_plan.ico

STRINGTABLE
BEGIN
    IDSNAME,    "KapPlan"
    IDSABOUT,   "Informationen"
    IDSTITLE,   "Leitstand L1 : Kapazitaetsdisposition"
END

KapPlan MENU
BEGIN
   MENUITEM    "&Login",                    MENU_LOGIN
   MENUITEM    SEPARATOR
   POPUP       "&Zeit",              INACTIVE
   BEGIN
      MENUITEM    "&Tage",                  MENU_Z_TAGE, CHECKED
      MENUITEM    "&Wochen",                MENU_Z_WOCHEN
      MENUITEM    "&Monate",                MENU_Z_MONATE
   END
   POPUP       "&Modus"
   BEGIN
      MENUITEM    "F&S-LE",                   MENU_FSLE, CHECKED
      MENUITEM    "&FS-FE",                   MENU_FSFE
      MENUITEM    "&LS-LE",                   MENU_LSLE
   END
   MENUITEM    "&BM-Auswahl",     MENU_BM,          INACTIVE
   MENUITEM    "M&arkieren",      MENU_MARK
   MENUITEM    SEPARATOR
   MENUITEM    "&Daten"               MENU_DATENANZEIGE,INACTIVE
   POPUP       "&Neuplanung"
   BEGIN
      MENUITEM    "&Umterminierung",         MENU_UMPLAN,      INACTIVE
      MENUITEM    "&Engpassterminierung",    MENU_ENGPASS,     INACTIVE
   END
   POPUP       "&Hilfe"
   BEGIN
      MENUITEM    "&Info ...",               MENU_ABOUT
      MENUITEM    "&Legende",                MENU_LEGENDE
      MENUITEM    "&Erlaeuterungen",         MENU_ERLAEUTERN
/*    MENUITEM    SEPARATOR                                    */
/*    MENUITEM    "&Debug",                  MENU_DEBUG        */
   END
   MENUITEM    "&Ende",                     MENU_ENDE
END

ABOUTBOX DIALOG 22, 17, 144, 75
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
    CTEXT "Andreas Heidemann",                    -1, 37,  5, 68, 8
    ICON  "KapPlan",                              -1,  9, 23,  0, 0
    CTEXT "Leitstand L1 : Kapazitaetsdisposition", -1,  0, 14, 144, 8
    CTEXT "Version 1.9",                          -1, 38, 34, 64, 8
    CTEXT "Copyright ® 1988, 1989 by A.Heidemann", -1,  0, 47, 144, 9
    DEFPUSHBUTTON "Ok",                          IDOK, 53, 59, 32, 14, WS_GROUP
END

LEGENDEBOX DIALOG 190, 14, 130, 200
STYLE WS_POPUP | WS_DLGFRAME
CAPTION "Farben & Muster"
BEGIN
   DEFPUSHBUTTON "OK",                       IDOK, 40, 2, 20, 15, WS_TABSTOP |
```

```
WS_GROUP
  65 END
  66
  67 HELPBOX DIALOG 100, 14, 200, 80
  68 STYLE WS_POPUP | WS_DLGFRAME
  69 CAPTION "Erlaeuterungen"
  70 BEGIN
  71    PUSHBUTTON    "OK",            IDOK,  33, 60, 50, 15, WS_TABSTOP | WS_GROUP
  72    DEFPUSHBUTTON "WEITER",   ID_WEITER, 117, 60, 50, 15, WS_TABSTOP | WS_GROUP
  73 END
  74
  75 DEBUGBOX DIALOG 12, 16, 160,  90
  76 STYLE WS_POPUP | WS_DLGFRAME
  77 BEGIN
  78    DEFPUSHBUTTON "OK",                        IDOK,  70,  70, 20, 16, WS_TABSTOP |
WS_GROUP
  79    EDITTEXT DBG_XSIZE,                              105,  9,  50, 12, WS_TABSTOP |
WS_GROUP
  80    EDITTEXT DBG_YSIZE,                              105, 24,  50, 12, WS_TABSTOP |
WS_GROUP
  81    EDITTEXT DBG_TXT,                                 10, 55, 100, 12, WS_TABSTOP |
WS_GROUP
  82    LTEXT "Groesse in X - Richtung :",           -1,   0, 10, 100, 10
  83    LTEXT "Groesse in Y - Richtung :",           -1,   0, 25, 100, 10
  84    LTEXT "Text :",                              -1,   0, 45, 100, 10
  85 END
  86
  87 BMAUSWAHL DIALOG 50, 31, 131, 103
  88 STYLE 0x90c00000
  89 CAPTION "Betriebsmittelauswahl"
  90 Begin
  91 Control "",                IDBMALIST,   "ListBox",  0x50a00003,   5,   4,  59, 102
  92 Control "ausgewaehlt :",   IDNULL,      "Static",   0x50020000,  69,  11,  60,  10
  93 Control "",                IDBMA,       "Edit",     0x50810080,  68,  22,  56,  12
  94 Control "OK",              IDOK,        "Button",   0x50010001,  81,  44,  34,  17
  95 Control "Cancel",          IDBMACANCEL, "Button",   0x50010000,  81,  70,  34,  18
  96 End
  97
  98 DATENBOX DIALOG 50, 10, 220, 92
  99 STYLE WS_POPUP | WS_DLGFRAME
 100 CAPTION "Fertigungsauftragsdaten :"
 101 BEGIN
 102     DEFPUSHBUTTON "gelesen", IDOK, 94, 76, 32, 14, WS_GROUP
 103 END
 104
 105 LOGINBOX DIALOG 99, 44, 120, 64
 106 STYLE 0x90c00000
 107 CAPTION "LOGIN"
 108 Begin
 109 Control "",                  IDLUSER,  "Edit", 0x50810080, 50, 17, 63, 12
 110 Control "",                  IDLPW,    "Edit", 0x50810080, 50, 40, 63, 4
 111 Control "User :",            IDNULL, "Static", 0x50020000, 4, 19, 40, 10
 112 Control "Password :",        IDNULL, "Static", 0x50020000, 2, 39, 40, 10
 113 Control "OK",                IDOK  , "Button", 0x50010001, 18, 50, 31, 13
 114 Control "Cancel",            IDCANCEL, "Button", 0x50010000, 70, 50, 31, 12
 115 Control "Datenbankzugriff",  IDNULL, "Static", 0x50020000, 27, 3, 66, 10
 116 End
 117
 118 MARKBOX DIALOG LOADONCALL MOVEABLE DISCARDABLE 15, 26, 190, 70
 119 CAPTION "Markiere Auftrag"
 120 STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
 121 BEGIN
 122           CONTROL "Fertigungsauftrag :", 0, "static", SS_LEFT | WS_CHILD, 5, 5,
81, 12
 123           CONTROL "Arbeitsgangnummer :", 0, "static", SS_LEFT | WS_CHILD, 5, 24,
81, 12
```

```
 124            CONTROL "           4711", IDFANR, "edit", ES_LEFT | WS_BORDER |
WS_GROUP | WS_TABSTOP
                                                                   | WS_CHILD, 90, 5,
91, 12
 125            CONTROL "10", IDFAGNR, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP |
WS_CHILD, 90, 22,

91, 12
 126            CONTROL "Markiere", IDMARK, "button", BS_DEFPUSHBUTTON | WS_GROUP |
WS_TABSTOP
                                                                       | WS_CHILD, 41,
46, 40, 14
 127            CONTROL "Abbruch", IDABBRUCH, "button", BS_PUSHBUTTON | WS_TABSTOP |
WS_CHILD, 105, 46,

40, 14
 128 END
```

8.2 Programmtext

8.2.1 KAP_PLAN.H

```
 1 /* Header File fuer Kap_plan */
 2
 3 #undef  DEBUGFILE   /* steuert die Deklaration des stddbg - Files*/
 4 #undef  DEBUG       /* steuert die Uebersetzung von Testausgaben */
 5 #undef  DEBUG_MARK  /* steuert die Uebersetzung von Testausgaben */
 6 #undef  DEBUG_DATE  /* steuert die Uebersetzung von Testausgaben */
 7 #undef  DEBUG_FEIN  /* steuert die Uebersetzung von Testausgaben */
 8 #undef  MOUSEDEBUG  /* steuert Testausgaben fuer Mausbewegungen  */
 9
10 #include "kpl_def.h"    /* Konstantendefinitionen für Windows */
11
12 /* array size constants */
13 #define  MAX_SLOTS         32
14 #define  MAX_AUFTRAEGE     60
15 #define  MAX_BESTS         3600
16 #define  ERRTXTLEN         100
17 #define  BMNR_SIZE         15
18 #define  FANR_SIZE         15
19 #define  FAGNR_SIZE        4
20 #define  TNR_SIZE          15
21 #define  TVR_SIZE          15
22 #define  KANR_SIZE         15
23 #define  AGBEZ_SIZE        30
24 #define  ME_SIZE           5
25 #define  USERNAME_SIZE     20
26 #define  PASSWORD_SIZE     20
27 #define  FAG_QUERY_SIZE    500
28
29 /* Umrechnungskonstanten */
30 #define  STD_PRO_TAG         24L
31 #define  STD_PRO_WOCHE      168L
32 #define  STD_PRO_MONAT      720L
33 #define  STD_PRO_QUARTAL   2160L
34 #define  STD_PRO_JAHR      8640L
35 #define  MAX_LONG          2147483647L
36
37 /* Fertigungsauftragsstati : */
38 #define  FAS_NT            0        /* nicht terminiert */
39 #define  FAS_AT            1        /* Angebotstermin */
40 #define  FAS_GI            5        /* Grobterminierung inkonsistent */
41 #define  FAS_GV            10       /* Grobterminiert vorwaerts */
42 #define  FAS_W1            14       /* Wartung nicht terminiert */
43 #define  FAS_GR            15       /* Grobterminiert rueckwaerts */
44 #define  FAS_FI            20       /* Feinplanung inkonsistent */
45 #define  FAS_FT            25       /* Feingeplant */
46 #define  FAS_AV            30       /* Arbeitsgang verspaetet */
47 #define  FAS_AF            35       /* Arbeitsgang freigegeben */
48 #define  FAS_AB            40       /* Arbeitsgang begonnen */
49 #define  FAS_AG            45       /* Arbeitsgang gestoert */
50 #define  FAS_TR            50       /* Arbeitsgang teilweise rueckgemeldet */
51 #define  FAS_W2            55       /* Wartung terminiert */
52
53 /* Fehlermeldungen */
54 #define  WRONG_DATE        9001
55 #define  MEMORY_OUT        9011
56 #define  ERR_TERMIN        9901
57
58 /* Farben : */
59 #define  ROT         (RGB (255,0,0))
60 #define  TIEFROT     (RGB (127,0,0))
```

```
 61 #define  GRUEN       (RGB (0,255,0))
 62 #define  DUNKELGRUEN (RGB (0,127,0))
 63 #define  BLAU        (RGB (0,0,255))
 64 #define  DUNKELBLAU  (RGB (0,0,127))
 65 #define  GELB        (RGB (255,255,0))
 66 #define  BRAUN       (RGB (127,127,0))
 67    /* Magenta = Cyan ?? */
 68 #define  MAGENTA     (RGB (0,255,255))
 69 #define  DARKMAGENTA (RGB (0,127,127))
 70    /* Pink = Magenta ?? */
 71 #define  PINK        (RGB (255,0,255))
 72 #define  DARKPINK    (RGB (127,0,127))
 73 #define  WEISS       (RGB (255,255,255))
 74 #define  GRAU        (RGB (127,127,127))
 75 #define  SCHWARZ     (RGB (0,0,0))
 76
 77
 78 /* Dateinamen */
 79 #define STDDBG          "C:\\SCRATCH\\KAP_PLAN.DBG"
 80 #define HELP_FILE_NAME  "KAP_PLAN.HLP"
 81
 82 /* Makrodefinitionen */
 83 #define  MAKE_STD          (hKapDat->kd_zeit==MENU_Z_TAGE   ? STD_PRO_TAG\
 84                            :hKapDat->kd_zeit==MENU_Z_WOCHEN ? STD_PRO_WOCHE\
 85                            :hKapDat->kd_zeit==MENU_Z_MONATE ? STD_PRO_MONAT\
 86                            :   /* else */             STD_PRO_QUARTAL)
 87 #define  MAKE_WIDTH        (hKapDat->kd_zeit==MENU_Z_TAGE   ? 31L * STD_PRO_TAG\
 88                            :hKapDat->kd_zeit==MENU_Z_WOCHEN ? 28L * STD_PRO_WOCHE\
 89                            :hKapDat->kd_zeit==MENU_Z_MONATE ? 24L * STD_PRO_MONAT\
 90                            :   /* else */             16L * STD_PRO_QUARTAL)
 91 #define  MAKE_SLOTS        (hKapDat->kd_zeit==MENU_Z_TAGE     ? 31L\
 92                            :hKapDat->kd_zeit==MENU_Z_WOCHEN   ? 28L\
 93                            :hKapDat->kd_zeit==MENU_Z_MONATE   ? 24L\
 94                            :   /* else */             16L)
 95 #define  TAGE_PRO_SLOT     (hKapDat->kd_zeit==MENU_Z_TAGE     ? 1\
 96                            :hKapDat->kd_zeit==MENU_Z_WOCHEN   ?    7\
 97                            :hKapDat->kd_zeit==MENU_Z_MONATE   ?    30\
 98                            :   /* else */                90)
 99
100 /* Typdefinitionen */
101 typedef struct BLOCKstruct {
102    int   bl_slot,
103          bl_auftrag,
104          bl_basis;
105 } BLOCK;
106
107 typedef char BMNR [15+1];
108
109 typedef struct FA_DATENstruct { /* Daten fuer Fertigungsauftrag */
110    char     fad_fanr  [FANR_SIZE+1],
111             fad_tnr   [TNR_SIZE+1],
112             fad_tvr   [TVR_SIZE+1],
113             fad_kanr  [KANR_SIZE+1],
114             fad_agbez [AGBEZ_SIZE+1],
115             fad_me    [ME_SIZE+1];
116    short    fad_kapos,
117             fad_fagnr,
118             fad_agstat,
119             fad_prio;
120    long     fad_gbzeit;
121    double   fad_bmenge;
122 } FA_DATEN;
123
124 typedef struct FAGstruct {
125    char  fag_fanr [FANR_SIZE+1];
126    long  fag_fstart,
```

```
127          fag_lstart,
128          fag_fend,
129          fag_lend,
130          fag_bedarf;
131    short fag_status,
132          fag_nr,
133          fag_veraendert;
134    struct FAGstruct *fag_next;
135 } FAG;
136
137 typedef struct KAPDATstruct { /* KAPDAT */
138    int   x_size,
139          y_size,
140          x_start,
141          x_ende,
142          y_start,
143          y_ende,
144          x_einheit;
145    float y_einheit;
146    int   kd_slotzahl;
147    long  kd_startzeit,
148          kd_endzeit,
149          kd_sysdate,
150          kd_minzeit,
151          kd_maxzeit;
152    struct {
153       int   y_header,
154             y_werte,
155             x_fanr,
156             x_fagnr,
157             x_fstart,
158             x_lend,
159             x_bmnr;
160    }     status_zeile;
161    RECT  winsize;
162    struct {
163       int   y_header,
164             y_werte,
165             x_fanr,
166             x_fagnr,
167             x_fstart,
168             x_lend,
169             x_bmnr;
170    }     statuszeile;
171    int   kd_zeit,
172          kd_semode;
173    BMNR  kd_bmnr;
174    int   kd_eintraege;
175    BOOL  kd_daten_vorhanden;
176    long  max_y,
177          kd_beststart,
178          kap_best [MAX_BESTS];
179    struct {
180       int      kb_bed,
181                kb_status;
182       FAG      * kb_fag;
183    }     kap_bed  [MAX_SLOTS][MAX_AUFTRAEGE];
184    FAG * fag_liste;
185 } KAPDAT;
```

8.2.2 KPL_DEF.H

```
 1 /* String table constants  */
 2 #define IDSNAME     100
 3 #define IDSABOUT    200
 4 #define IDSTITLE    300
 5
 6 /* dialog box resource id's */
 7 #define ABOUTBOX     10
 8 #define DEBUGBOX     20
 9 #define LEGENDEBOX   40
10 #define BMAUSWAHL    50
11 #define DATENBOX     60
12 #define LOGINBOX     70
13 #define HELPBOX      80
14 #define MARKBOX      90
15
16 /* dialog box control item id's */
17 #define IDNULL           0
18 #define DBG_XSIZE       21
19 #define DBG_YSIZE       22
20 #define DBG_TXT         23
21 #define IDBREITEEINH    24
22 #define IDBREITEBLOCK   25
23 #define IDSCHICHT       26
24 #define IDSTUNDE        27
25 #define IDTAG           28
26 #define IDWOCHE         29
27 #define IDBMALIST       30
28 #define IDBMA           32
29 #define IDBMACANCEL     33
30 #define IDLUSER         34
31 #define IDLPW           35
32 #define ID_WEITER       36
33 #define IDFANR          37
34 #define IDFAGNR         38
35 #define IDMARK          39
36 #define IDABBRUCH       40
37
38
39 /* MENUE - Eintraege : */
40 #define  MENU_LOGIN        1001
41 #define  MENU_ZEIT_POS        2
42 #define  MENU_Z_WOCHEN     1002
43 #define  MENU_Z_MONATE     1003
44 #define  MENU_Z_QUARTALE   1004
45 #define  MENU_BM           1005
46 #define  MENU_UMPLAN       1006
47 #define  MENU_LEGENDE      1007
48 #define  MENU_ERLAEUTERN   1008
49 #define  MENU_DEBUG        1009
50 #define  MENU_ENDE         1010
51 #define  MENU_LOGOUT       1012
52 #define  MENU_DATENANZEIGE 1013
53 #define  MENU_Z_TAGE       1014
54 #define  MENU_ABOUT        1015
55 #define  MENU_FSLE         1016
56 #define  MENU_FSFE         1017
57 #define  MENU_LSLE         1018
58 #define  MENU_ENGPASS      1019
59 #define  MENU_MARK         1020
```

8.2.3 DCL_VAR.INC

```
 1 int  __argc;
 2 char **__argv;
 3
 4 char far szAppName [ 8],
 5      far szAbout   [30],
 6      far szMessage [40];
 7
 8 int MessageLength,
 9     iXdist,
10     iYdist,
11     iX,
12     iY;
13
14 TEXTMETRIC tmStuff;
15
16 static HANDLE hInst;
17
18 FARPROC lpprocAbout,
19         lpprocDebug,
20         lpprocLegende,
21         lpprocDatenDlg,
22         lpprocBMAuswahl,
23         lpprocHelpDlg,
24         lpprocMarkDlg,
25         lpprocLoginDlg;
26
27 HWND     hActWnd;
28
29 HCURSOR  hArrowCursor,
30          hWaitCursor;
31
32 BLOCK    Block;
33
34 FAG      *hAktFAG;
35
36 BOOL     verschieben = FALSE;
37
38 KAPDAT   *hKapDat,
39          KapDat;
40
41 int      RopAktuell;
42
43 HBRUSH   hBrNormal,
44          hBrAktuell,
45          hBrFeingeplant,
46          hBrFreigegeben,
47          hBrBegonnen,
48          hBrVerspaetet,
49          hBrTeilRueck,
50          hBrWartung,
51          hBrAngebot,
52          hBrFeinInkons,
53          hBrGrobInkons,
54          hBrGestoert;
55
56 HPEN     hPenNormal,
57          hPenAktuell,
58          hLinePen,
59          hScalePen;
60
61 #ifdef DEBUGFILE
62    FILE     *stddbg;
63 #endif
```

8.2.4 KAP_PLAN.C

```
/*  Kap_Plan.c

    Kapazitätsplanung   Version 1.9
    Copyright (c) A.Heidemann 1988,1989

    Windows Application
    Windows Version 2.0
    Copyright (c) Microsoft 1985        */

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include <process.h>
#define  NOMINMAX
#include "windows.h"
#include "Kap_Plan.h"
#include "daten.h"
#include "dcl_var.inc"
#include "kap_plan.dcl"       /* Prozedurkoepfe fuer Kap_Plan */

extern BOOL GetUID (char *, char *);
extern BOOL InitBMGList (int *);
extern BOOL EndBMGList (int *);
extern BOOL NextBMGList (char [], int *);
extern BOOL FADaten (FA_DATEN *, int *);
extern BOOL ErrorText (int, char []);
extern BOOL Login (char[], char[], KAPDAT *, int *);
extern BOOL Logout (int *);
extern BOOL datetostd (long int *, int, int, int, int, int, int *);
extern BOOL stdtodate (long int, int *, int *, int *, int *, int *, int *);
extern BOOL stdtowoche (long int, int *, int *);

extern FILE *fopen ();

long FAR PASCAL KapPlanWndProc (HWND, unsigned, WORD, LONG);
void PASCAL Menu (int, HWND, KAPDAT*);

void ErrorMsg (msg, titel)
char  *msg,
      *titel;
{
   MessageBeep (MB_OK);
   MessageBox (hActWnd, (LPSTR) msg, (LPSTR) titel, MB_OK);
   SetCursor (hWaitCursor);
} /* end ErrorMsg */

void ClearMsgQueue (hWnd)
HWND  hWnd;
{                          /* loescht wg. langer Wartezeit die MsgQueue */
   /* tut z. Zt. noch gar nichts, wird aber demnächst aktiv !! */
} /* end ClearMsgQueue */

BOOL FAR PASCAL AboutKapPlan ( hDlg, message, wParam, lParam )
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
    if (message == WM_COMMAND) {
        EndDialog( hDlg, TRUE );
        return TRUE;
```

```
 65         }
 66     else
 67     if (message == WM_INITDIALOG)
 68         return TRUE;
 69     else
 70         return FALSE;
 71 }/* end AboutKapPlan */
 72
 73
 74 int DebugCmd (hDlg, wParam, lParam)
 75 HWND hDlg;
 76 WORD wParam;
 77 LONG lParam;
 78 {
 79    if (wParam == IDOK)
 80       EndDialog (hDlg, TRUE);
 81    else printf ("%c", (char) 7);
 82    return (TRUE);
 83 } /* End DebugCmd */
 84
 85 BOOL FAR PASCAL DebugInfo (hDlg, message, wParam, lParam)
 86 HWND hDlg;
 87 unsigned message;
 88 WORD wParam;
 89 LONG lParam;
 90 {
 91     char buffer [30],
 92          uid [30],
 93          pwd [30];
 94
 95     if (message == WM_COMMAND) {
 96         DebugCmd (hDlg, wParam, lParam);
 97         return (TRUE);
 98         }
 99     else
100     if (message == WM_INITDIALOG)
101     {
102         GetUID (uid, pwd);
103         sprintf (buffer, "%d", hKapDat->x_size);
104         SetDlgItemText (hDlg, DBG_XSIZE, (LPSTR) buffer);
105         sprintf (buffer, "%d", hKapDat->y_size);
106         SetDlgItemText (hDlg, DBG_YSIZE, (LPSTR) buffer);
107         sprintf (buffer, "memavl = %lu, uid = <%s/%s>",
108                   GlobalCompact (0L), uid, pwd );
109         SetDlgItemText (hDlg, DBG_TXT, (LPSTR) buffer);
110         return TRUE;
111     }
112     else
113         return FALSE;
114 } /* end DebugInfo */
115
116 BOOL FAR PASCAL MarkDlg (hDlg, message, wParam, lParam)
117 HWND hDlg;
118 unsigned message;
119 WORD wParam;
120 LONG lParam;
121 {
122    FAG      fag,
123             *hFag;
124    BOOL     err;
125
126     switch (message) {
127     case WM_INITDIALOG  :
128       ClearMsgQueue (hDlg);
129       if (hAktFAG != (FAG*) NULL){
130          SetDlgItemText (hDlg, IDFANR, hAktFAG->fag_fanr);
```

```
131          SetDlgItemInt (hDlg, IDFAGNR, hAktFAG->fag_nr, FALSE);
132       } /* end if */
133       break;
134     case WM_COMMAND :
135       switch (wParam) {
136       case IDMARK :
137          GetDlgItemText (hDlg, IDFANR, (LPSTR) fag.fag_fanr, FANR_SIZE+1);
138          fag.fag_nr = GetDlgItemInt (hDlg, IDFAGNR, (BOOL FAR *) &err, FALSE);
139 #ifdef DEBUG_MARK
140          fprintf (stddbg, "Mark gelesen FANR = <%s>, FAGNR = <%d>",
141                           fag.fag_fanr, fag.fag_nr);
142          fflush (stddbg);
143 #endif
144          /* FAG suchen : */
145          hFag = hKapDat->fag_liste;
146          while ((hFag != (FAG*) NULL)
147                & !((strcmp (fag.fag_fanr, hFag->fag_fanr) == 0)
148                   &(hFag->fag_nr == fag.fag_nr))) {
149             hFag = hFag->fag_next;
150          } /* end while */
151          if (hFag != (FAG*) NULL) {
152             hAktFAG = hFag;
153             EndDialog (hDlg, TRUE);
154          } /* end if */
155          else {
156             EndDialog (hDlg, FALSE);
157          } /* end else */
158          break;
159       case IDABBRUCH :
160          EndDialog (hDlg, FALSE);
161          break;
162       default :
163          break;
164       } /* end switch */
165       break;
166     default:
167         return (FALSE);
168         break;
169     };
170     return (TRUE);
171 } /* end MarkDlg */
172
173 BOOL FAR PASCAL HelpDlg (hDlg, message, wParam, lParam)
174 HWND hDlg;
175 unsigned message;
176 WORD wParam;
177 LONG lParam;
178 {
179    static FILE    * help_file;
180    char           txt [80];
181    int            i,
182                   j,
183                   y;
184    HDC            hDC;
185    PAINTSTRUCT    ps;
186
187    switch (message) {
188       case WM_INITDIALOG  :
189          help_file = fopen (HELP_FILE_NAME, "r");
190          if (help_file == (FILE *) NULL) {
191             return FALSE;
192          } /* end if */
193          break;
194       case WM_PAINT :
195          BeginPaint (hDlg, (LPPAINTSTRUCT) &ps);
196          EndPaint   (hDlg, (LPPAINTSTRUCT) &ps);
```

```
197          hDC = GetDC (hDlg);
198          y = tmStuff.tmHeight;
199          for (i = 0; i < 6; i++){
200             if (help_file != (FILE*) NULL) {
201                if (fgets (txt, 80, help_file) == (FILE*) NULL) {
202                   fclose (help_file);
203                   help_file = (FILE*) NULL;
204                   strcpy (txt, "\n\0");
205                } /* end if */
206             } /* end if */
207             else {
208                strcpy (txt, "\n\0");
209             } /* end else */
210             for (j = strlen (txt)-1; j < 79; j ++) {
211                txt [j] = ' ';
212             } /* end for */
213             txt [79] = '\0';
214             TextOut (hDC, 1, y, txt, strlen (txt));
215             y += tmStuff.tmHeight;
216          } /* end for */
217          ReleaseDC (hDlg, hDC);
218          break;
219       case WM_COMMAND :
220          if (wParam == IDOK) {
221             fclose (help_file);
222             EndDialog (hDlg, TRUE);
223             return (TRUE);
224          }
225          else if (wParam == ID_WEITER) {
226             PostMessage (hDlg, WM_PAINT, 0, 0L);
227          } /* end if */
228          else {
229             return (FALSE);
230          } /* end else */
231          break;
232       default:
233          return (FALSE);
234          break;
235    } /* end switch */
236    return TRUE;
237 } /* end HelpDlg */
238
239 BOOL FAR PASCAL LegendeDlg (hDlg, message, wParam, lParam)
240 HWND hDlg;
241 unsigned message;
242 WORD wParam;
243 LONG lParam;
244 {
245     HDC           hDC;
246     RECT          winsize;
247     int           y1,
248                   y2,
249                   y_dist,
250                   y_len,
251                   y_txt_start,
252                   x1,
253                   x2,
254                   x3;
255     PAINTSTRUCT   ps;
256
257     switch (message) {
258
259     case WM_INITDIALOG  :
260       PostMessage (hDlg, WM_PAINT, 0, 0L);
261       break;
262     case WM_PAINT :
```

```
263       BeginPaint (hDlg, (LPPAINTSTRUCT) &ps);
264       EndPaint   (hDlg, (LPPAINTSTRUCT) &ps);
265       hDC = GetDC (hDlg);
266       GetClipBox (hDC, (LPRECT) &winsize);
267       if (winsize.bottom == winsize.top) {
268          ReleaseDC (hDlg, hDC);
269          break;
270       };
271
272       y_dist = (winsize.bottom - winsize.top - 2*tmStuff.tmHeight) / 12;
273       y_len  = y_dist - y_dist / 8;
274       x1     = winsize.left + tmStuff.tmMaxCharWidth;
275       x2     = x1 + (winsize.right - winsize.left) / 8;
276       x3     = x2 + tmStuff.tmMaxCharWidth;
277       y1     = winsize.top + 2*tmStuff.tmHeight + y_dist - y_len;
278       y_txt_start = (y_len - tmStuff.tmHeight) / 2;
279
280       y2 = y1 + y_len;
281       SelectObject (hDC, hBrNormal);
282       SelectObject (hDC, hPenNormal);
283       Rectangle (hDC, x1, y1, x2, y2);
284       SelectObject (hDC, hPenAktuell);
285       SelectObject (hDC, hBrAktuell);
286       SetROP2 (hDC, RopAktuell);
287       Rectangle (hDC, x1, y1, x2, y2);
288       SetROP2 (hDC, R2_COPYPEN);
289       SelectObject (hDC, hPenNormal);
290       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "ausgewaehlter AG", 16);
291       y1 += y_dist;
292
293       SelectObject (hDC, hBrAngebot);
294       y2 = y1 + y_len;
295       Rectangle (hDC, x1, y1, x2, y2);
296       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "Angebotstermin", 14);
297       y1 += y_dist;
298
299       SelectObject (hDC, hBrNormal);
300       y2 = y1 + y_len;
301       Rectangle (hDC, x1, y1, x2, y2);
302       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "grob terminierter AG", 20);
303       y1 += y_dist;
304
305       SelectObject (hDC, hBrGrobInkons);
306       y2 = y1 + y_len;
307       Rectangle (hDC, x1, y1, x2, y2);
308       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "Grobplanung inkonsistent", 24);
309       y1 += y_dist;
310
311       SelectObject (hDC, hBrFeingeplant);
312       y2 = y1 + y_len;
313       Rectangle (hDC, x1, y1, x2, y2);
314       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "feingeplanter AG", 16);
315       y1 += y_dist;
316
317       SelectObject (hDC, hBrFeinInkons);
318       y2 = y1 + y_len;
319       Rectangle (hDC, x1, y1, x2, y2);
320       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "Feinplanung inkonsistent", 24);
321       y1 += y_dist;
322
323       SelectObject (hDC, hBrVerspaetet);
324       y2 = y1 + y_len;
325       Rectangle (hDC, x1, y1, x2, y2);
326       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "verspaeteter AG", 15);
327       y1 += y_dist;
328
```

```
329       SelectObject (hDC, hBrFreigegeben);
330       y2 = y1 + y_len;
331       Rectangle (hDC, x1, y1, x2, y2);
332       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "freigegebener AG", 16);
333       y1 += y_dist;
334
335       SelectObject (hDC, hBrBegonnen);
336       y2 = y1 + y_len;
337       Rectangle (hDC, x1, y1, x2, y2);
338       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "begonnener AG", 13);
339       y1 += y_dist;
340
341       SelectObject (hDC, hBrTeilRueck);
342       y2 = y1 + y_len;
343       Rectangle (hDC, x1, y1, x2, y2);
344       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "teilw. rueckgem. AG", 19);
345       y1 += y_dist;
346
347       SelectObject (hDC, hBrGestoert);
348       y2 = y1 + y_len;
349       Rectangle (hDC, x1, y1, x2, y2);
350       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "gestoerter AG", 13);
351       y1 += y_dist;
352
353       SelectObject (hDC, hBrWartung);
354       y2 = y1 + y_len;
355       Rectangle (hDC, x1, y1, x2, y2);
356       TextOut (hDC, x3, y1 + y_txt_start, (LPSTR) "Wartung", 7);
357       y1 += y_dist;
358
359       ReleaseDC (hDlg, hDC);
360       break;
361     case WM_COMMAND :
362       if (wParam == IDOK) {
363          EndDialog (hDlg, TRUE);
364          return (TRUE);
365       }
366       else
367          return (FALSE);
368       break;
369     default:
370         return (FALSE);
371         break;
372     };
373     return (TRUE);
374 } /* end LegendeDlg */
375
376 void BMAuswahlCmd (hDlg, wParam, lParam)
377 HWND  hDlg;
378 WORD  wParam;
379 LONG  lParam;
380 {
381    char txt [15];
382
383    switch (wParam) {
384       case IDOK :
385          GetDlgItemText (hDlg, IDBMA, (LPSTR) hKapDat->kd_bmnr, 14);
386          EndDialog (hDlg, TRUE);
387          break;
388       case IDBMACANCEL :
389          EndDialog (hDlg, FALSE);
390          break;
391       case IDBMALIST :
392          SendDlgItemMessage (hDlg,
393                              IDBMALIST,
394                              LB_GETTEXT,
```

```
395                             (WORD) SendDlgItemMessage (hDlg,
396                                                        IDBMALIST,
397                                                        LB_GETCURSEL,
398                                                        0, /* not used */
399                                                        0L /* not used */),
400                             (LONG) txt);
401          SetDlgItemText (hDlg, IDBMA, (LPSTR) txt);
402          break;
403       default :
404          break;
405    } /* end switch */
406 } /* end BMAuswahlCmd */
407
408 BOOL FAR PASCAL BMAuswahlDlg (hDlg, message, wParam, lParam)
409 HWND hDlg;
410 unsigned message;
411 WORD wParam;
412 LONG lParam;
413 {
414     BMNR    bmnr;
415     char    txt [ERRTXTLEN],
416             errtxt [ERRTXTLEN];
417     int     error;
418
419     switch (message) {
420     case WM_INITDIALOG  :
421       if (!InitBMGList (&error)){
422          if (ErrorText (error, errtxt)) {
423             sprintf (txt, "ERROR %d : %s", error, errtxt);
424          }
425          else {
426             sprintf (txt, "ERROR %d, Fehler unbekannt");
427          }
428          SetCursor (hArrowCursor);
429          ClearMsgQueue (hDlg);
430          MessageBeep (MB_OK);
431          MessageBox (hDlg, (LPSTR) txt, (LPSTR) "DB - ERROR", MB_OK);
432          EndDialog (hDlg, FALSE);
433          return TRUE;
434       } /* end if */
435       while (NextBMGList (bmnr, &error)) {
436          SendDlgItemMessage (hDlg,
437                              IDBMALIST,
438                              LB_ADDSTRING,
439                              0, /* not used */
440                              (LONG) bmnr);
441       } /* end while */;
442       if (!EndBMGList (&error)){
443          if (ErrorText (error, errtxt)) {
444             sprintf (txt, "ERROR %d : %s", error, errtxt);
445          }
446          else {
447             sprintf (txt, "ERROR %d, Fehler unbekannt");
448          }
449          SetCursor (hArrowCursor);
450          ClearMsgQueue (hDlg);
451          MessageBeep (MB_OK);
452          MessageBox (hDlg, (LPSTR) txt, (LPSTR) "DB - ERROR", MB_OK);
453          EndDialog (hDlg, FALSE);
454       } /* end if */
455       SetCursor (hArrowCursor);
456       ClearMsgQueue (hDlg);
457       break;
458     case WM_COMMAND :
459       BMAuswahlCmd (hDlg, wParam, lParam);
460       break;
```

```
461     default:
462         return (FALSE);
463         break;
464     };
465     return (TRUE);
466 } /* end BMAuswahlDlg */
467
468 BOOL FAR PASCAL DatenDlg (hDlg, message, wParam, lParam)
469 HWND        hDlg;
470 unsigned    message;
471 WORD        wParam;
472 LONG        lParam;
473 {
474    HDC            hDC;
475    FA_DATEN       fad;
476    char           txt [ERRTXTLEN],
477                   errtxt [ERRTXTLEN];
478    int            y,
479                   y_abstand,
480                   error;
481    PAINTSTRUCT    ps;
482
483    switch (message) {
484    case WM_INITDIALOG :
485       /* PostMessage (hDlg, WM_PAINT, 0, 0L); */
486       /*  WM_PAINT wird automatisch vom System gesetzt */
487       return (TRUE);
488       break;
489    case WM_PAINT :
490       BeginPaint (hDlg, (LPPAINTSTRUCT) &ps);
491       EndPaint   (hDlg, (LPPAINTSTRUCT) &ps);
492       hDC = GetDC (hDlg);
493       y_abstand = tmStuff.tmHeight;
494       y = 1;
495       strcpy (fad.fad_fanr, hAktFAG->fag_fanr);
496       fad.fad_fagnr = hAktFAG->fag_nr;
497       if (FADaten (&fad, &error)) {
498          sprintf (txt, "Teilenummer            :%s", fad.fad_tnr);
499          TextOut (hDC, 1, y, txt, strlen (txt));
500          y += y_abstand;
501          sprintf (txt, "Variante               :%s",fad.fad_tvr);
502          TextOut (hDC, 1, y, txt, strlen (txt));
503          y += y_abstand;
504          sprintf (txt, "Kundenauftragsnummer   :%s",fad.fad_kanr);
505          TextOut (hDC, 1, y, txt, strlen (txt));
506          y += y_abstand;
507          sprintf (txt, "  Position             :%d",fad.fad_kapos);
508          TextOut (hDC, 1, y, txt, strlen (txt));
509          y += y_abstand;
510          sprintf (txt, "AG - Bezeichnung       :%s",fad.fad_agbez);
511          TextOut (hDC, 1, y, txt, strlen (txt));
512          y += y_abstand;
513          sprintf (txt, "AG - Status            :%d",fad.fad_agstat);
514          TextOut (hDC, 1, y, txt, strlen (txt));
515          y += y_abstand;
516          sprintf (txt, "AG - Prioritaet        :%d",fad.fad_prio);
517          TextOut (hDC, 1, y, txt, strlen (txt));
518          y += y_abstand;
519          sprintf (txt, "bestellte Menge        :%.2lf %s",fad.fad_bmenge,
fad.fad_me);
520          TextOut (hDC, 1, y, txt, strlen (txt));
521          y += y_abstand;
522          sprintf (txt, "ges. Bearbeitungsdauer :%ld Std %ld Min",
523                   fad.fad_gbzeit / 60,
524                   fad.fad_gbzeit % 60);
525          TextOut (hDC, 1, y, txt, strlen (txt));
```

```
526       } /* end if */
527       else { /* DB-Anfrage gibt fehlerhaftes Ergebnis */
528          y += 2 * y_abstand;
529          TextOut (hDC, 1, y, "Fehler bei Datenbankzugriff :", 29);
530          y += y_abstand;
531          if (ErrorText (error, errtxt)) {
532             sprintf (txt, "ERROR %d : %s", error, errtxt);
533          } /* end if */
534          else {
535             sprintf (txt, "ERROR %d : unbekannter Fehler");
536          } /* end else */
537          TextOut (hDC, 20, y, txt, strlen (txt));
538       } /* end else */
539       ReleaseDC (hDlg, hDC);
540       SetCursor (hArrowCursor);
541       ClearMsgQueue (hDlg);
542       break;
543    case WM_COMMAND :
544       EndDialog (hDlg, TRUE);
545       break;
546    default :
547       return FALSE;
548       break;
549    } /* end switch */
550    return TRUE;
551 } /* end DatenDlg */
552
553 BOOL FAR PASCAL LoginDlg (hDlg, message, wParam, lParam)
554 HWND        hDlg;
555 unsigned    message;
556 WORD        wParam;
557 LONG        lParam;
558 {
559    char  username [USERNAME_SIZE+1],
560          password [PASSWORD_SIZE+1],
561          errtxt   [ERRTXTLEN],
562          txt      [ERRTXTLEN];
563    int   error;
564
565    switch (message) {
566    case WM_INITDIALOG :
567       break;
568    case WM_COMMAND :
569       switch (wParam) {
570       case IDOK :
571          GetDlgItemText (hDlg, IDLUSER, (LPSTR) username, USERNAME_SIZE);
572          GetDlgItemText (hDlg, IDLPW, (LPSTR) password, PASSWORD_SIZE);
573          if (!Login (username, password, hKapDat, &error)) {
574             /* Fehler beim LOGIN */
575 #ifdef DEBUG
576             fprintf (stddbg, "  LOGIN failed\n");
577             fflush (stddbg);
578 #endif
579             if (ErrorText (error, errtxt)) {
580                sprintf (txt, "ERROR %d : %s", error, errtxt);
581             }
582             else {
583                sprintf (txt, "ERROR %d : nicht bekannt", error);
584             } /* end else */
585             MessageBeep (MB_OK);
586             MessageBox (hDlg, (LPSTR) txt, (LPSTR) "ERROR", MB_OK);
587             EndDialog (hDlg, FALSE);
588          } /* end if */
589          else {
590 #ifdef DEBUG
591             fprintf (stddbg, "  LOGIN accepted\n");
```

```
592             fflush (stddbg);
593 #endif
594             EndDialog (hDlg, TRUE);
595          } /* end else */
596          break;
597       case IDCANCEL :
598          EndDialog (hDlg, FALSE);
599          break;
600       default :
601          break;
602       } /* end switch */
603       break;
604    default :
605       return FALSE;
606       break;
607    } /* end switch */
608    return TRUE;
609 } /* end LoginDlg */
610
611 /*--------------------------------------- KapDat bearbeiten */
612
613 void InitKapDat (hKapDat)
614 KAPDAT *hKapDat;
615 {
616    hKapDat->kd_zeit       = MENU_Z_TAGE;
617    hKapDat->kd_semode     = MENU_FSLE;
618    hKapDat->max_y         = STD_PRO_TAG;
619    hKapDat->kd_eintraege  = (int) MAKE_SLOTS ;
620    hKapDat->kd_daten_vorhanden = FALSE;
621    hKapDat->fag_liste     = (FAG *) NULL;
622    hKapDat->kd_bmnr [0]   = '\0';
623 } /* end InitKapDat */
624
625 /*--------------------------------------- Prozeduren zum Zeichnen */
626 #include "paint.inc"
627
628 /* Procedure called when the application is loaded for the first time */
629 BOOL KapPlanInit( hInstance )
630 HANDLE hInstance;
631 {
632     NPWNDCLASS  pKapPlanClass;
633     long FAR PASCAL KapPlanWndProc ();
634
635     /* Load strings from resource */
636     LoadString( hInstance, IDSNAME, (LPSTR)szAppName, 8 );
637     LoadString( hInstance, IDSABOUT, (LPSTR)szAbout, 30 );
638     MessageLength = LoadString( hInstance, IDSTITLE, (LPSTR)szMessage, 40 );
639
640     pKapPlanClass = (NPWNDCLASS)LocalAlloc( LPTR, sizeof(WNDCLASS) );
641
642     /* register MAIN Window */
643     pKapPlanClass->hCursor        = LoadCursor( NULL, IDC_ARROW );
644     pKapPlanClass->hIcon          = LoadIcon( hInstance, (LPSTR)szAppName );
645     pKapPlanClass->lpszMenuName   = (LPSTR)szAppName;
646     pKapPlanClass->lpszClassName  = (LPSTR)szAppName;
647     pKapPlanClass->hbrBackground  = (HBRUSH)GetStockObject( WHITE_BRUSH );
648     pKapPlanClass->hInstance      = hInstance;
649     pKapPlanClass->style          = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
650     pKapPlanClass->lpfnWndProc    = KapPlanWndProc;
651
652     if (!RegisterClass( (LPWNDCLASS)pKapPlanClass ) )
653         /* Initialization failed.
654          * Windows will automatically deallocate all allocated memory.
655          */
656         return FALSE;
657
```

```
658     LocalFree( (HANDLE)pKapPlanClass );
659     return TRUE;        /* Initialization succeeded */
660 }
661
662
663 int PASCAL WinMain( hInstance, hPrevInstance, lpszCmdLine, cmdShow )
664 HANDLE hInstance, hPrevInstance;
665 LPSTR lpszCmdLine;
666 int cmdShow;
667 {
668     MSG      msg;
669     HWND     hWnd;
670
671 #ifdef DEBUGFILE
672     if ((FILE*) NULL == (stddbg = fopen (STDDBG, "w"))) {
673       return (FALSE);
674     };
675     fprintf (stddbg, "Debug - Datei fuer KapPlan\n");
676     fprintf (stddbg, " sizeof KAPDAT = %d\n", sizeof (KAPDAT));
677     fflush  (stddbg);
678 #endif
679
680     /*    hKapDat = (KAPDAT *) malloc (sizeof (KAPDAT)); */
681     hKapDat = & KapDat;
682     InitKapDat (hKapDat);
683
684 #ifdef DEBUG
685     fprintf (stddbg,
686              "  nach INIT : kd_zeit %d, kd_startzeit %ld, kd_endzeit %ld\n",
687              hKapDat->kd_zeit,
688              hKapDat->kd_startzeit,
689              hKapDat->kd_endzeit);
690     fflush (stddbg);
691 #endif
692
693     if (!hPrevInstance) {
694         /* Call initialization procedure if this is the first instance */
695         if (!KapPlanInit( hInstance ))
696             return FALSE;
697         }
698     else {
699         /* Copy data from previous instance */
700 #ifdef DEBUG
701           fprintf (stddbg, "  copy data from previous instance\n");
702           fflush (stddbg);
703 #endif
704         GetInstanceData( hPrevInstance, (NPSTR)szAppName,  8 );
705         GetInstanceData( hPrevInstance, (NPSTR)szAbout, 30 );
706         GetInstanceData( hPrevInstance, (NPSTR)szMessage, 40 );
707         GetInstanceData( hPrevInstance, (NPSTR)&MessageLength, sizeof(int) );
708         }
709
710     hWnd = CreateWindow((LPSTR)szAppName,
711                         (LPSTR)szMessage,
712                         WS_TILEDWINDOW | WS_VISIBLE | WS_HSCROLL | WS_MAXIMIZE,
713                         CW_USEDEFAULT,     /*  x Position */
714                         CW_USEDEFAULT,     /*  y - ignored for default x */
715                         CW_USEDEFAULT,     /* x-width */
716                         CW_USEDEFAULT,     /* y-width - ignored for default x*/
717                         (HWND)NULL,        /* no parent */
718                         (HMENU)NULL,       /* use class menu */
719                         (HANDLE)hInstance, /* handle to window instance */
720                         (LPSTR) hKapDat    /* handle to KapDat as lParam */
721                         );
722
723     /* Create Pens and Brushes */
```

```
724     if (FALSE) { /* normal steht hier Abfrage auf Farbbildschirm */
725       /* s/w Bildschirm */
726        hBrNormal     = CreateSolidBrush (WEISS);
727        hBrAktuell    = CreateSolidBrush (GRAU);
728        hBrFeingeplant= CreateHatchBrush (HS_HORIZONTAL, SCHWARZ);
729        hBrFreigegeben= CreateHatchBrush (HS_VERTICAL  , SCHWARZ);
730        hBrBegonnen   = CreateHatchBrush (HS_FDIAGONAL , SCHWARZ);
731        hBrVerspaetet = CreateHatchBrush (HS_BDIAGONAL , SCHWARZ);
732        hBrGestoert   = CreateHatchBrush (HS_DIAGCROSS , SCHWARZ);
733
734        hPenAktuell   = CreatePen (0 /*solid*/, 1, SCHWARZ);
735        hPenNormal    = CreatePen (0 /*solid*/, 1, SCHWARZ);
736        hLinePen      = CreatePen (2 /*dots */, 1, SCHWARZ);
737        hScalePen     = CreatePen (0 /*solid*/, 1, SCHWARZ);
738
739        RopAktuell    = R2_XORPEN;
740     } /* end if */
741     else {
742       /* Farbbildschirm */
743        hBrNormal     = CreateSolidBrush (WEISS);
744        hBrAktuell    = CreateHatchBrush (HS_DIAGCROSS , SCHWARZ);
745        hBrFeingeplant= CreateSolidBrush (GELB);
746        hBrFreigegeben= CreateSolidBrush (GRUEN);
747        hBrBegonnen   = CreateSolidBrush (BLAU);
748        hBrVerspaetet = CreateSolidBrush (PINK);
749        hBrTeilRueck  = CreateSolidBrush (MAGENTA);
750        hBrGestoert   = CreateSolidBrush (ROT);
751        hBrWartung    = CreateSolidBrush (SCHWARZ);
752        hBrAngebot    = CreateSolidBrush (BRAUN);
753        hBrFeinInkons = CreateSolidBrush (TIEFROT);
754        hBrGrobInkons = CreateSolidBrush (GRAU);
755
756        hPenAktuell   = CreatePen (0 /*solid*/, 1, MAGENTA);
757        hPenNormal    = CreatePen (0 /*solid*/, 1, SCHWARZ);
758        hLinePen      = CreatePen (0 /*solid*/, 1, ROT);
759        hScalePen     = CreatePen (0 /*solid*/, 1, SCHWARZ);
760
761        RopAktuell    = R2_NOTXORPEN;
762     } /* end else */
763
764 #ifdef DEBUG
765     fprintf (stddbg, "  pens and brushes created\n");
766     fflush (stddbg);
767 #endif
768
769     /* Load Cursors */
770     hWaitCursor = LoadCursor (NULL, IDC_WAIT);
771     hArrowCursor = LoadCursor (NULL, IDC_ARROW);
772
773 #ifdef DEBUG
774     fprintf (stddbg, "  cursors loaded\n");
775     fflush (stddbg);
776 #endif
777
778     /* initialize global values */
779     hAktFAG             = (FAG *) NULL;
780
781 #ifdef DEBUG
782     fprintf (stddbg, "  global values initialized\n");
783     fflush (stddbg);
784 #endif
785
786     /* Save instance handle for DialogBox */
787     hInst = hInstance;
788
789 #ifdef DEBUG
```

```
790     fprintf (stddbg, "  instance handle for dialog box saved\n");
791     fflush (stddbg);
792 #endif
793
794     /* Bind callback function with module instance */
795     lpprocAbout     = MakeProcInstance ((FARPROC) AboutKapPlan, hInstance );
796     lpprocDebug     = MakeProcInstance ((FARPROC) DebugInfo,    hInstance);
797     lpprocLegende   = MakeProcInstance ((FARPROC) LegendeDlg,   hInstance);
798     lpprocBMAuswahl = MakeProcInstance ((FARPROC) BMAuswahlDlg, hInstance);
799     lpprocDatenDlg  = MakeProcInstance ((FARPROC) DatenDlg,     hInstance);
800     lpprocLoginDlg  = MakeProcInstance ((FARPROC) LoginDlg,     hInstance);
801     lpprocHelpDlg   = MakeProcInstance ((FARPROC) HelpDlg,      hInstance);
802     lpprocMarkDlg   = MakeProcInstance ((FARPROC) MarkDlg,      hInstance);
803
804 #ifdef DEBUG
805     fprintf (stddbg, "  callback functions bound with module instance\n");
806     fflush (stddbg);
807 #endif
808
809     /* Make window visible according to the way the app is activated */
810     ShowWindow  (hWnd, cmdShow);
811     UpdateWindow (hWnd);
812     hActWnd = hWnd;
813
814 #ifdef DEBUG
815     fprintf (stddbg, "  made window visible like %d (magic %d)\n", cmdShow, 4711);
816     fflush (stddbg);
817 #endif
818
819     /* Polling messages from event queue */
820     while (GetMessage( (LPMSG)&msg, NULL, 0, 0 )) {
821         TranslateMessage( (LPMSG)&msg );
822 #ifdef DEBUG_WINMSG
823         fprintf (stddbg, "**** MESSAGE %d %d %ld dispatching\n",
824                          msg.message,
825                          msg.wParam,
826                          msg.lParam);
827         fflush (stddbg);
828 #endif
829         DispatchMessage( (LPMSG)&msg );
830         }
831
832 #ifdef DEBUGFILE
833     fprintf (stddbg, "DEBUG - Datei: Programm normal beendet\n");
834     fclose (stddbg);
835 #endif
836     return (int)msg.wParam;
837 } /* end WinMain */
838
839 BOOL process_scroll_cmds (cmd, pos, hKapDat)
840 WORD     cmd;
841 LONG     pos;
842 KAPDAT   *hKapDat;
843 {
844 #ifdef DEBUG
845    fprintf (stddbg,
846             "HSCROLL : alte Zeit von : %ld bis %ld \n",
847             hKapDat->kd_startzeit,
848             hKapDat->kd_endzeit);
849    fflush (stddbg);
850 #endif
851
852    switch (cmd) {
853    case SB_LINEUP :
854       hKapDat->kd_startzeit -= MAKE_STD ;
855       break;
```

```
856    case SB_PAGEUP :
857       hKapDat->kd_startzeit -= MAKE_WIDTH ;
858       break;
859    case SB_LINEDOWN :
860       hKapDat->kd_startzeit += MAKE_STD ;
861       break;
862    case SB_PAGEDOWN :
863       hKapDat->kd_startzeit += MAKE_WIDTH ;
864       break;
865    case SB_TOP :
866       hKapDat->kd_startzeit = hKapDat->kd_minzeit;
867       break;
868    case SB_BOTTOM :
869       hKapDat->kd_startzeit = hKapDat->kd_maxzeit
870                               - MAKE_STD ;
871       break;
872    case SB_THUMBPOSITION :
873       hKapDat->kd_startzeit = hKapDat->kd_minzeit + LOWORD (pos);
874       break;
875    default :
876       return FALSE;
877       break;
878    } /* end switch */
879    if (hKapDat->kd_startzeit < hKapDat->kd_minzeit) {
880       hKapDat->kd_startzeit = hKapDat->kd_minzeit;
881    } /* end if */
882    if (hKapDat->kd_startzeit + MAKE_STD
883        > hKapDat->kd_maxzeit) {
884       hKapDat->kd_startzeit = hKapDat->kd_maxzeit
885                               - MAKE_STD ;
886    } /* end if */
887    hKapDat->kd_startzeit -= hKapDat->kd_startzeit
888                             % MAKE_STD ;
889    hKapDat->kd_endzeit = hKapDat->kd_startzeit
890                          + MAKE_WIDTH ;
891
892 #ifdef DEBUG
893    fprintf (stddbg,
894             "HSCROLL : neue Zeit von : %ld bis %ld \n",
895             hKapDat->kd_startzeit,
896             hKapDat->kd_endzeit);
897    fflush (stddbg);
898 #endif
899    return TRUE;
900 } /* end process scroll commands */
901
902 /* Procedures which make up the window class. */
903 long FAR PASCAL KapPlanWndProc( hWnd, message, wParam, lParam )
904 HWND hWnd;
905 unsigned message;
906 WORD wParam;
907 LONG lParam;
908 {
909     HDC           hDC;
910     PAINTSTRUCT   ps;
911     RECT          rect;
912     POINT         org;
913     long          l_org;
914     HMENU         hMenu;
915     int           error;
916
917     switch (message) {
918     case WM_CREATE :
919         /* hKapDat = (KAPDAT *) lParam; */   /* Information is not used !!*/
920          /* immediate login */
921       if (Login ("PPS", "PPS", hKapDat, &error)) {
```

```
922          hMenu = GetMenu (hWnd);
923          EnableMenuItem (hMenu, MENU_ZEIT_POS, MF_ENABLED | MF_BYPOSITION);
924          EnableMenuItem (hMenu, MENU_BM,   MF_ENABLED);
925          EnableMenuItem (hMenu, MENU_UMPLAN,   MF_ENABLED);
926          EnableMenuItem (hMenu, MENU_ENGPASS,   MF_ENABLED);
927          EnableMenuItem (hMenu, MENU_ERLAEUTERN,   MF_ENABLED);
928          EnableMenuItem (hMenu, MENU_DATENANZEIGE,   MF_ENABLED);
929          ChangeMenu (hMenu, MENU_LOGIN, (LPSTR) "New &Login", MENU_LOGOUT,
930                      MF_CHANGE | MF_STRING);
931          DrawMenuBar (hWnd);
932          hKapDat->kd_startzeit  = hKapDat->kd_sysdate * STD_PRO_TAG;
933          hKapDat->kd_endzeit    = hKapDat->kd_startzeit
934                                   + MAKE_WIDTH ;
935 #ifdef DEBUG
936          fprintf (stddbg, "  Zeiten : %ld bis %ld. \n",
937                   hKapDat->kd_startzeit,
938                   hKapDat->kd_endzeit);
939          fflush (stddbg);
940 #endif
941       } /* end if */
942         break;
943     case WM_SYSCOMMAND:
944         switch (wParam) {
945         case IDSABOUT:
946             DialogBox (hInst, MAKEINTRESOURCE(ABOUTBOX), hWnd, lpprocAbout);
947             break;
948         default:
949             return DefWindowProc( hWnd, message, wParam, lParam );
950         }
951         break;
952
953     case WM_DESTROY:
954         PostQuitMessage( 0 );
955         break;
956
957     case WM_COMMAND:
958         if (LOWORD(lParam) == NULL)
959             Menu(wParam,hWnd, hKapDat);
960         break;
961
962     case WM_MOUSEMOVE:
963 #ifdef MOUSEDEBUG
964         fprintf (stddbg, "MOUSEMOVE verschieben = %d\n", verschieben);
965         fflush (stddbg);
966 #endif
967         if (verschieben) {
968             BlockSchieben (hWnd, lParam, hKapDat);
969 #ifdef MOUSEDEBUG
970         fprintf (stddbg, "MOUSEMOVE BlockSchieben\n");
971         fflush (stddbg);
972 #endif
973         } /* end if */
974         else {
975             TrackMousePosition (hWnd, lParam, hKapDat);
976 #ifdef MOUSEDEBUG
977         fprintf (stddbg, "MOUSEMOVE TrackMousePosition\n");
978         fflush (stddbg);
979 #endif
980         } /* end else */
981         break;
982     case WM_RBUTTONDOWN:
983         if (hAktFAG != (FAG *) NULL) {
984            /* start einer grafischen umplanung */
985            verschieben = TRUE;
986 #ifdef DEBUG
987            fprintf (stddbg, "START verschieben, %d := %d\n", verschieben, TRUE);
```

```
 988            fflush (stddbg);
 989 #endif
 990            hDC = GetDC (hWnd);
 991            l_org = GetDCOrg (hDC);
 992            org = MAKEPOINT (l_org);
 993 #ifdef DEBUG
 994            fprintf (stddbg, "  origin (x/y) : %d/%d\n", org.x, org.y);
 995            fflush (stddbg);
 996 #endif
 997            ReleaseDC (hWnd, hDC);
 998            rect.top    = hKapDat->y_ende + org.y;
 999            rect.bottom = rect.top;
1000            rect.left   = hKapDat->x_start + org.x;
1001            rect.right  = hKapDat->x_ende  + org.x;
1002            ClipCursor ((LPRECT) &rect);
1003            GetCursorPos ((LPPOINT) &org);
1004            SetCursorPos (org.x, rect.top);
1005 #ifdef DEBUG
1006            fprintf (stddbg, "  cursor (x/y; alt-neu) : %d/%d-%d/%d\n",
1007                             org.x, org.y, org.x, rect.top);
1008            fflush (stddbg);
1009 #endif
1010         } /* end if */
1011         break;
1012     case WM_RBUTTONUP:
1013         if (verschieben) {
1014            /* ende einer grafischen umplanung, grafik aktualisieren */
1015            verschieben = FALSE;
1016            ClipCursor ((LPRECT) NULL);
1017            SetCursor (hWaitCursor);
1018            BlockVerschieben (lParam, hKapDat, hWnd);
1019            darstellung_berechnen (hKapDat);
1020            SetCursor (hArrowCursor);
1021            ClearMsgQueue (hWnd);
1022            PostMessage (hWnd, WM_PAINT, 0, 0L);
1023         } /* end if */
1024         break;
1025     case WM_LBUTTONUP:
1026         break;
1027     case WM_LBUTTONDBLCLK:
1028         PostMessage (hWnd, WM_LBUTTONDOWN, MK_SHIFT, lParam);
1029         PostMessage (hWnd, WM_COMMAND, (WORD) MENU_DATENANZEIGE, (long) 0);
1030         break; /* end WM_LBUTTONDBLCLK */
1031     case WM_RBUTTONDBLCLK:
1032         /* printscreen (hWnd); */
1033         break;
1034
1035     case WM_LBUTTONDOWN:
1036         IdentifyBlock (hWnd, hKapDat, lParam, &Block);
1037         if (hAktFAG == (FAG*) NULL) {
1038            hDC = GetDC (hWnd);
1039            SetROP2 (hDC, RopAktuell);
1040            SelectObject (hDC, hBrAktuell);
1041            SelectObject (hDC, hPenAktuell);
1042            hAktFAG = hKapDat->kap_bed [Block.bl_slot][Block.bl_auftrag].kb_fag;
1043            ZeichneFAG (hDC, hKapDat, hAktFAG);
1044            SetROP2 (hDC, R2_COPYPEN);
1045            KapBestZeichnen (hWnd, hDC, hKapDat);
1046            ReleaseDC (hWnd, hDC);
1047          } /* end if */
1048          else if ((hAktFAG ==
1049                     hKapDat->kap_bed[Block.bl_slot][Block.bl_auftrag].kb_fag )
1050                   & (wParam != MK_SHIFT)) {
1051            hDC = GetDC (hWnd);
1052            SelectBlockObjects (hDC, hKapDat->kap_bed
[Block.bl_slot][Block.bl_auftrag].kb_status);
```

```
1053            ZeichneFAG (hDC, hKapDat, hAktFAG);
1054            KapBestZeichnen (hWnd, hDC, hKapDat);
1055            hAktFAG = (FAG *) NULL;
1056            ReleaseDC (hWnd, hDC);
1057          } /* end if */
1058          else if (Block.bl_auftrag != 0) {
1059            hDC = GetDC (hWnd);
1060            SelectBlockObjects (hDC, hAktFAG->fag_status);
1061            ZeichneFAG (hDC, hKapDat, hAktFAG);
1062            SelectObject (hDC, hBrAktuell);
1063            SelectObject (hDC, hPenAktuell);
1064            SetROP2 (hDC, RopAktuell);
1065            hAktFAG = hKapDat->kap_bed [Block.bl_slot][Block.bl_auftrag].kb_fag;
1066            ZeichneFAG (hDC, hKapDat, hAktFAG);
1067            SetROP2 (hDC, R2_COPYPEN);
1068            StatRepZeichnen (hDC, hKapDat, hAktFAG);
1069            KapBestZeichnen (hWnd, hDC, hKapDat);
1070            ReleaseDC (hWnd, hDC);
1071          } /* end else */
1072         break;
1073
1074     case WM_HSCROLL :
1075       if (process_scroll_cmds (wParam, lParam, hKapDat)) {
1076          SetScrollPos  (hWnd,
1077                         SB_HORZ,
1078                         (int) (hKapDat->kd_startzeit - hKapDat->kd_minzeit),
1079                         TRUE);
1080          SetCursor (hWaitCursor);
1081          darstellung_berechnen (hKapDat);
1082          SetCursor (hArrowCursor);
1083          ClearMsgQueue (hWnd);
1084          PostMessage (hWnd, WM_PAINT, 0, 0L);
1085       } /* end if */
1086       else {
1087          return (DefWindowProc (hWnd, message, wParam, lParam));
1088       } /* end else */
1089       break;
1090
1091     case WM_PAINT:
1092         SetCursor (hWaitCursor);
1093         BeginPaint (hWnd, (LPPAINTSTRUCT) &ps);
1094         EndPaint   (hWnd, (LPPAINTSTRUCT) &ps);
1095         hDC = GetDC (hWnd);
1096         KapPlanPaint (hWnd, hDC, hKapDat);
1097         if (hKapDat->kd_daten_vorhanden) {
1098 #ifdef DEBUG
1099            fprintf (stddbg, "StatRepZeichnen ->, \n");
1100            fflush (stddbg);
1101 #endif
1102            StatRepZeichnen (hDC, hKapDat, hAktFAG);
1103 #ifdef DEBUG
1104            fprintf (stddbg, "evtl. AktBlock zeichnen. (%d)\n",
1105                     hAktFAG != (FAG*) NULL);
1106            fflush (stddbg);
1107 #endif
1108            if (hAktFAG != (FAG*) NULL) {
1109               /* aktuellen Block sichtbar  machen */
1110               SetROP2 (hDC, RopAktuell);
1111               SelectObject (hDC, hBrAktuell);
1112               SelectObject (hDC, hPenAktuell);
1113               ZeichneFAG (hDC, hKapDat, hAktFAG);
1114            } /* end if */
1115         } /* end if */
1116         ReleaseDC (hWnd, hDC);
1117         SetCursor (hArrowCursor);
1118         ClearMsgQueue (hWnd);
```

```
1119         break;
1120
1121     default:
1122         return DefWindowProc( hWnd, message, wParam, lParam );
1123         break;
1124     }
1125     return( 0L );
1126 }
1127
1128 BOOL TestEnde (hWnd)
1129 HWND hWnd;
1130 {
1131    return (IDYES == MessageBox (hWnd,
1132                        (LPSTR) "Wollen Sie das Programm wirklich beenden ?",
1133                        (LPSTR) "E N D E",
1134                        MB_YESNO));
1135 }
1136
1137 void PASCAL
1138 Menu (cmd, hWnd, hKapDat)
1139 int cmd;
1140 HWND hWnd;
1141 KAPDAT *hKapDat;
1142 {
1143    HMENU hMenu;
1144    int   error;
1145    char  txt [ERRTXTLEN],
1146          errtxt [ERRTXTLEN];
1147
1148     hMenu = GetMenu (hWnd);
1149     switch (cmd) {
1150     case MENU_LOGIN :
1151          if (DialogBox (hInst,
1152                         MAKEINTRESOURCE (LOGINBOX),
1153                         hWnd,
1154                         lpprocLoginDlg)) {
1155             EnableMenuItem (hMenu, MENU_ZEIT_POS, MF_ENABLED | MF_BYPOSITION);
1156             EnableMenuItem (hMenu, MENU_BM,   MF_ENABLED);
1157             EnableMenuItem (hMenu, MENU_UMPLAN,   MF_ENABLED);
1158             EnableMenuItem (hMenu, MENU_ENGPASS,   MF_ENABLED);
1159             EnableMenuItem (hMenu, MENU_ERLAEUTERN,   MF_ENABLED);
1160             EnableMenuItem (hMenu, MENU_DATENANZEIGE,   MF_ENABLED);
1161             ChangeMenu (hMenu, MENU_LOGIN, (LPSTR) "New &Login", MENU_LOGOUT,
1162                         MF_CHANGE | MF_STRING);
1163             DrawMenuBar (hWnd);
1164             hKapDat->kd_startzeit  = hKapDat->kd_sysdate * STD_PRO_TAG;
1165             hKapDat->kd_endzeit    = hKapDat->kd_startzeit
1166                                      + MAKE_WIDTH ;
1167 #ifdef DEBUG
1168             fprintf (stddbg, "  Zeiten : %ld bis %ld. \n",
1169                      hKapDat->kd_startzeit,
1170                      hKapDat->kd_endzeit);
1171             fflush (stddbg);
1172 #endif
1173          } /* end if */
1174          else { /* Programm beenden */
1175             PostMessage (hWnd, WM_DESTROY, (short) 0, (long) 0);
1176          } /* end else */
1177          break;
1178     case MENU_LOGOUT :
1179          hKapDat->kd_daten_vorhanden = FALSE;
1180          if (Logout (&error)) {
1181             if (!DialogBox (hInst,
1182                             MAKEINTRESOURCE (LOGINBOX),
1183                             hWnd,
1184                             lpprocLoginDlg)) {
```

```
1185                EnableMenuItem (hMenu,
1186                                MENU_ZEIT_POS,
1187                                MF_DISABLED | MF_BYPOSITION);
1188                EnableMenuItem (hMenu, MENU_BM, MF_DISABLED);
1189                EnableMenuItem (hMenu, MENU_DATENANZEIGE, MF_DISABLED);
1190                ChangeMenu (hMenu, MENU_LOGOUT, (LPSTR) "&Login", MENU_LOGIN,
1191                            MF_CHANGE | MF_STRING);
1192                DrawMenuBar (hWnd);
1193             } /* end if */
1194          } /* end if */
1195          else {
1196             if (ErrorText (error, errtxt)) {
1197                sprintf (txt, "ERROR %d : %s", error, errtxt);
1198             } /* end if */
1199             else {
1200                sprintf (txt, "ERROR %d : unbekannt", error);
1201             } /* end else */
1202             MessageBeep (MB_OK);
1203             MessageBox (hWnd, (LPSTR) txt, (LPSTR) "ERROR", MB_OK);
1204          } /* end else */
1205          PostMessage (hWnd, WM_PAINT, 0, 0L);
1206          break;
1207     case MENU_Z_TAGE :
1208          if (hKapDat->kd_zeit != MENU_Z_TAGE){
1209             SetCursor (hWaitCursor);
1210             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_UNCHECKED);
1211             hKapDat->kd_zeit = MENU_Z_TAGE;
1212             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_CHECKED);
1213             hKapDat->kd_endzeit   = hKapDat->kd_startzeit
1214                                     + MAKE_WIDTH ;
1215             hKapDat->kd_eintraege = (int) MAKE_SLOTS ;
1216             if (hKapDat->kd_daten_vorhanden) {
1217                darstellung_berechnen (hKapDat);
1218                PostMessage (hWnd, WM_PAINT, 0, 0L);
1219             } /* end if */
1220             SetCursor (hArrowCursor);
1221             ClearMsgQueue (hWnd);
1222          } /* end if */
1223          break;
1224     case MENU_Z_WOCHEN :
1225          if (hKapDat->kd_zeit != MENU_Z_WOCHEN) {
1226             SetCursor (hWaitCursor);
1227             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_UNCHECKED);
1228             hKapDat->kd_zeit = MENU_Z_WOCHEN;
1229             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_CHECKED);
1230             hKapDat->kd_endzeit   = hKapDat->kd_startzeit
1231                                     + MAKE_WIDTH ;
1232             hKapDat->kd_eintraege = (int) MAKE_SLOTS ;
1233             if (hKapDat->kd_daten_vorhanden) {
1234                darstellung_berechnen (hKapDat);
1235                PostMessage (hWnd, WM_PAINT, 0, 0L);
1236             } /* end if */
1237             SetCursor (hArrowCursor);
1238             ClearMsgQueue (hWnd);
1239          } /* end if */
1240          break;
1241     case MENU_Z_MONATE :
1242          if (hKapDat->kd_zeit != MENU_Z_MONATE) {
1243             SetCursor (hWaitCursor);
1244             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_UNCHECKED);
1245             hKapDat->kd_zeit = MENU_Z_MONATE;
1246             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_CHECKED);
1247             hKapDat->kd_endzeit   = hKapDat->kd_startzeit
1248                                     + MAKE_WIDTH ;
1249             hKapDat->kd_eintraege = (int) MAKE_SLOTS ;
1250             if (hKapDat->kd_daten_vorhanden) {
```

```
1251                darstellung_berechnen (hKapDat);
1252                PostMessage (hWnd, WM_PAINT, 0, 0L);
1253             } /* end if */
1254             SetCursor (hArrowCursor);
1255             ClearMsgQueue (hWnd);
1256          } /* end if */
1257          break;
1258     case MENU_Z_QUARTALE :
1259          if (hKapDat->kd_zeit != MENU_Z_QUARTALE) {
1260             SetCursor (hWaitCursor);
1261             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_UNCHECKED);
1262             hKapDat->kd_zeit = MENU_Z_QUARTALE;
1263             CheckMenuItem (hMenu, (WORD) hKapDat->kd_zeit, MF_CHECKED);
1264             hKapDat->kd_endzeit   = hKapDat->kd_startzeit
1265                                   + MAKE_WIDTH ;
1266             hKapDat->kd_eintraege = (int) MAKE_SLOTS ;
1267             if (hKapDat->kd_daten_vorhanden) {
1268                darstellung_berechnen (hKapDat);
1269                PostMessage (hWnd, WM_PAINT, 0, 0L);
1270             } /* end if */
1271             SetCursor (hArrowCursor);
1272             ClearMsgQueue (hWnd);
1273          } /* end if */
1274          break;
1275     case MENU_FSLE :
1276          if (hKapDat->kd_semode != MENU_FSLE) {
1277             SetCursor (hWaitCursor);
1278             CheckMenuItem (hMenu, (WORD) hKapDat->kd_semode, MF_UNCHECKED);
1279             hKapDat->kd_semode = MENU_FSLE;
1280             CheckMenuItem (hMenu, (WORD) hKapDat->kd_semode, MF_CHECKED);
1281             if (hKapDat->kd_daten_vorhanden) {
1282                darstellung_berechnen (hKapDat);
1283                PostMessage (hWnd, WM_PAINT, 0, 0L);
1284             } /* end if */
1285             SetCursor (hArrowCursor);
1286             ClearMsgQueue (hWnd);
1287          } /* end if */
1288          break;
1289     case MENU_FSFE :
1290          if (hKapDat->kd_semode != MENU_FSFE) {
1291             SetCursor (hWaitCursor);
1292             CheckMenuItem (hMenu, (WORD) hKapDat->kd_semode, MF_UNCHECKED);
1293             hKapDat->kd_semode = MENU_FSFE;
1294             CheckMenuItem (hMenu, (WORD) hKapDat->kd_semode, MF_CHECKED);
1295             if (hKapDat->kd_daten_vorhanden) {
1296                darstellung_berechnen (hKapDat);
1297                PostMessage (hWnd, WM_PAINT, 0, 0L);
1298             } /* end if */
1299             SetCursor (hArrowCursor);
1300             ClearMsgQueue (hWnd);
1301          } /* end if */
1302          break;
1303     case MENU_LSLE :
1304          if (hKapDat->kd_semode != MENU_LSLE) {
1305             SetCursor (hWaitCursor);
1306             CheckMenuItem (hMenu, (WORD) hKapDat->kd_semode, MF_UNCHECKED);
1307             hKapDat->kd_semode = MENU_LSLE;
1308             CheckMenuItem (hMenu, (WORD) hKapDat->kd_semode, MF_CHECKED);
1309             if (hKapDat->kd_daten_vorhanden) {
1310                darstellung_berechnen (hKapDat);
1311                PostMessage (hWnd, WM_PAINT, 0, 0L);
1312             } /* end if */
1313             SetCursor (hArrowCursor);
1314             ClearMsgQueue (hWnd);
1315          } /* end if */
1316          break;
```

```
1317     case MENU_BM :
1318          SetCursor (hWaitCursor);
1319          if (DialogBox (hInst,
1320                         MAKEINTRESOURCE (BMAUSWAHL),
1321                         hWnd,
1322                         lpprocBMAuswahl)) {
1323             SetCursor (hWaitCursor);
1324             hAktFAG = (FAG*) NULL;   /* aktuellen Block "vergessen" */
1325             hKapDat->kd_daten_vorhanden = TRUE;
1326                /* wird wieder auf FALSE gesetzt, wenn Fehler auftritt */
1327             if (read_fags (hKapDat)){
1328                SetScrollRange (hWnd,
1329                                SB_HORZ,
1330                                0,
1331                                (int) (hKapDat->kd_maxzeit - hKapDat->kd_minzeit),
1332                                FALSE);
1333                SetScrollPos  (hWnd,
1334                               SB_HORZ,
1335                               (int) (hKapDat->kd_startzeit - hKapDat->kd_minzeit),
1336                               TRUE);
1337                read_kap_best (hKapDat);
1338                darstellung_berechnen (hKapDat);
1339             } /* end if */
1340             else {
1341                hKapDat->kd_daten_vorhanden = FALSE;
1342             } /* end else */
1343             SetCursor (hArrowCursor);
1344             ClearMsgQueue (hWnd);
1345             PostMessage (hWnd, WM_PAINT, 0, 0L);
1346          } /* end if */
1347          break;
1348     case MENU_UMPLAN :
1349     case MENU_ENGPASS :
1350          SetCursor (hWaitCursor);
1351          umplanen (hKapDat, cmd);
1352          darstellung_berechnen (hKapDat);
1353          SetCursor (hArrowCursor);
1354          ClearMsgQueue (hWnd);
1355          PostMessage (hWnd, WM_PAINT, 0, 0L);
1356          break;
1357     case MENU_MARK :
1358          SetCursor (hWaitCursor);
1359          if (DialogBox (hInst, MAKEINTRESOURCE (MARKBOX), hWnd, lpprocMarkDlg)){
1360             PostMessage (hWnd, WM_PAINT, 0, 0L);
1361          } /* end if */
1362          SetCursor (hArrowCursor);
1363          break; /* end MENU_MARK */
1364     case MENU_DATENANZEIGE :
1365          SetCursor (hWaitCursor);
1366          DialogBox (hInst,
1367                     MAKEINTRESOURCE (DATENBOX),
1368                     hWnd,
1369                     lpprocDatenDlg);
1370          break;
1371     case MENU_ABOUT :
1372          DialogBox (hInst, MAKEINTRESOURCE(ABOUTBOX), hWnd, lpprocAbout);
1373          break;
1374     case MENU_LEGENDE :
1375          DialogBox (hInst,
1376                     MAKEINTRESOURCE (LEGENDEBOX),
1377                     hWnd,
1378                     lpprocLegende);
1379          break;
1380     case MENU_ERLAEUTERN :
1381          DialogBox (hInst, MAKEINTRESOURCE(HELPBOX), hWnd, lpprocHelpDlg);
1382          break;
```

```
1383     case MENU_DEBUG :
1384          DialogBox (hInst, MAKEINTRESOURCE(DEBUGBOX), hWnd, lpprocDebug);
1385          break;
1386     case MENU_ENDE :
1387          if (TestEnde (hWnd)) {
1388             Logout (&error);
1389             hKapDat->kd_daten_vorhanden = FALSE;
1390             PostMessage (hWnd, WM_DESTROY, (short) 0, (long) 0);
1391          } /* end if */
1392          break;
1393     default:
1394         break;
1395     }
1396 }
1397
```

8.2.5 PAINT.INC

```
 1 void SelectBlockObjects (hDC, Status)
 2 HDC      hDC;
 3 int      Status;
 4 {
 5       switch (Status) {
 6          case FAS_GV :
 7          case FAS_GR :
 8             SelectObject (hDC, hBrNormal);
 9             SelectObject (hDC, hPenNormal);
10             break;
11          case FAS_GI :
12             SelectObject (hDC, hBrGrobInkons);
13             SelectObject (hDC, hPenNormal);
14             break;
15          case FAS_AT :
16             SelectObject (hDC, hBrAngebot);
17             SelectObject (hDC, hPenNormal);
18             break;
19          case FAS_FT :                              /* Feinterminiert */
20             SelectObject (hDC, hBrFeingeplant);
21             SelectObject (hDC, hPenNormal);
22             break;
23          case FAS_FI :
24             SelectObject (hDC, hBrFeinInkons);
25             SelectObject (hDC, hPenNormal);
26             break;
27          case FAS_AV :
28             SelectObject (hDC, hBrVerspaetet);
29             SelectObject (hDC, hPenNormal);
30             break;
31          case FAS_AF :
32             SelectObject (hDC, hBrFreigegeben);
33             SelectObject (hDC, hPenNormal);
34             break;
35          case FAS_AB :
36             SelectObject (hDC, hBrBegonnen);
37             SelectObject (hDC, hPenNormal);
38             break;
39          case FAS_TR :
40             SelectObject (hDC, hBrTeilRueck);
41             SelectObject (hDC, hPenNormal);
42             break;
43          case FAS_AG :
44             SelectObject (hDC, hBrGestoert);
45             SelectObject (hDC, hPenNormal);
46             break;
47          case FAS_W1 :
48          case FAS_W2 :
49             SelectObject (hDC, hBrWartung);
50             SelectObject (hDC, hPenNormal);
51             break;
52          default :
53             SelectObject (hDC, hBrGestoert);
54             SelectObject (hDC, hPenNormal);
55             break;
56       } /* end switch */
57 } /* end SelectBlockObjects */
58
59 void SkalaZeichnen (hWnd, hDC, hKapDat)
60 HWND     hWnd;
61 HDC      hDC;
62 KAPDAT   * hKapDat;
63 {
64    int   x,
```

```
 65          y,
 66          skal_abstand,
 67          h,
 68          beschriftung,
 69          error,
 70          std, std1, std2,
 71          min, min1, min2,
 72          tt, tt1, tt2,
 73          mm, mm1, mm2,
 74          jj, jj1, jj2,
 75          woche;
 76    char  txt [20],
 77          txt2 [20],
 78          txt3 [20];
 79
 80 #ifdef DEBUG
 81    fprintf (stddbg, "beginn skala_zeichnen\n");
 82    fflush (stddbg);
 83 #endif
 84
 85    SelectObject (hDC, hScalePen);
 86    /* Koordinatenkreuz */
 87    MoveTo (hDC, hKapDat->x_start, hKapDat->y_start - 15);
 88    LineTo (hDC, hKapDat->x_start, hKapDat->y_ende);
 89    LineTo (hDC, hKapDat->x_ende + 20, hKapDat->y_ende);
 90
 91    /* Pfeil der Y - Achse */
 92    MoveTo (hDC, hKapDat->x_start-5, hKapDat->y_start -10);
 93    LineTo (hDC, hKapDat->x_start, hKapDat->y_start -15);
 94    LineTo (hDC, hKapDat->x_start+5, hKapDat->y_start -10);
 95    /* Beschriftung */
 96    TextOut (hDC,
 97             hKapDat->x_start - 11 * tmStuff.tmMaxCharWidth,
 98             hKapDat->y_start - 15,
 99             (LPSTR) "Kapazitaet",
100             10);
101    TextOut (hDC,
102             hKapDat->x_start - 11 * tmStuff.tmMaxCharWidth,
103             hKapDat->y_start - 15 + tmStuff.tmHeight,
104             (LPSTR) "  [Std]",
105             7);
106
107    /* Pfeil der X - Achse */
108    MoveTo (hDC, hKapDat->x_ende+10, hKapDat->y_ende -3);
109    LineTo (hDC, hKapDat->x_ende+20, hKapDat->y_ende);
110    LineTo (hDC, hKapDat->x_ende+10, hKapDat->y_ende +3);
111    /* Beschriftung */
112    stdtodate (hKapDat->kd_startzeit, &tt1, &mm1, &jj1, &std1, &min1, &error);
113    stdtodate (hKapDat->kd_endzeit, &tt2, &mm2, &jj2, &std2, &min2, &error);
114    sprintf (txt2, "%02d/%02d", mm1, jj1);
115    sprintf (txt3, "%02d/%02d", mm2, jj2);
116    switch (hKapDat->kd_zeit) {
117       case MENU_Z_TAGE :
118          strcpy (txt, "Tag");
119          break;
120       case MENU_Z_WOCHEN :
121          strcpy (txt, "Woche");
122          break;
123       case MENU_Z_MONATE :
124          strcpy (txt, "Monat");
125          break;
126       case MENU_Z_QUARTALE :
127          strcpy (txt, "Quartal");
128          break;
129       default:
130          strcpy (txt, "Zeit");
```

```
131         break;
132    } /* end switch */
133    TextOut (hDC,
134             hKapDat->x_start - 9 * tmStuff.tmMaxCharWidth,
135             hKapDat->y_ende + 3 - 3*tmStuff.tmHeight,
136             (LPSTR) txt2,
137             strlen (txt2));
138    TextOut (hDC,
139             hKapDat->x_start - 9 * tmStuff.tmMaxCharWidth,
140             hKapDat->y_ende + 3 - 2*tmStuff.tmHeight,
141             (LPSTR) txt3,
142             strlen (txt3));
143    TextOut (hDC,
144             hKapDat->x_start - 9 * tmStuff.tmMaxCharWidth,
145             hKapDat->y_ende + 3,
146             (LPSTR) txt,
147             strlen (txt));
148
149    /* X-Achse skalieren : */
150    for (x = hKapDat->x_start + hKapDat->x_einheit, beschriftung = 1;
151         beschriftung <= (int) MAKE_SLOTS ;
152         x += hKapDat->x_einheit, beschriftung += 1)
153    {
154       MoveTo (hDC, x, hKapDat->y_ende-3);
155       LineTo (hDC, x, hKapDat->y_ende+3);
156       /*  beschriften */
157       switch (hKapDat->kd_zeit) {
158       case MENU_Z_TAGE :
159          stdtodate (hKapDat->kd_startzeit + (beschriftung-1) * STD_PRO_TAG,
160                     &tt, &mm, &jj, &std, &min, &error);
161          if ((tt == 1) | (tt % 5 == 0)) {
162             sprintf (txt, "%d", tt);
163             TextOut (hDC,
164                      x - hKapDat->x_einheit / 2
165                        - tmStuff.tmMaxCharWidth * strlen (txt) / 2,
166                      hKapDat->y_ende+3,
167                      (LPSTR) txt,
168                      strlen (txt));
169          } /* end if */
170          break;
171       case MENU_Z_WOCHEN :
172          stdtowoche (hKapDat->kd_startzeit + (beschriftung-1) * STD_PRO_WOCHE,
173                     &woche, &error);
174          if ((woche == 1) | (woche % 5 == 0)) {
175             sprintf (txt, "%d", woche);
176             TextOut (hDC,
177                      x - hKapDat->x_einheit / 2
178                        - tmStuff.tmMaxCharWidth * strlen (txt) / 2,
179                      hKapDat->y_ende+3,
180                      (LPSTR) txt,
181                      strlen (txt));
182          } /* end if */
183          break;
184       case MENU_Z_MONATE :
185          stdtodate (hKapDat->kd_startzeit + (beschriftung-1) * STD_PRO_MONAT,
186                     &tt, &mm, &jj, &std, &min, &error);
187          if ((mm == 1) | (mm % 5 == 0)) {
188             sprintf (txt, "%d", mm);
189             TextOut (hDC,
190                      x - hKapDat->x_einheit / 2
191                        - tmStuff.tmMaxCharWidth * strlen (txt) / 2,
192                      hKapDat->y_ende+3,
193                      (LPSTR) txt,
194                      strlen (txt));
195          } /* end if */
196          break;
```

```
197       case MENU_Z_QUARTALE :
198          stdtodate (hKapDat->kd_startzeit + (beschriftung-1) * STD_PRO_QUARTAL,
199                     &tt, &mm, &jj, &std, &min, &error);
200          sprintf (txt, "%d", (mm-1)/3 + 1);
201          TextOut (hDC,
202                   x - hKapDat->x_einheit / 2
203                     - tmStuff.tmMaxCharWidth * strlen (txt) / 2,
204                   hKapDat->y_ende+3,
205                   (LPSTR) txt,
206                   strlen (txt));
207          break;
208       default :
209          if ((beschriftung == 1) | (beschriftung % 5 == 0)) {
210             sprintf (txt, "%d", beschriftung);
211             TextOut (hDC,
212                      x - hKapDat->x_einheit / 2
213                        - tmStuff.tmMaxCharWidth * strlen (txt) / 2,
214                      hKapDat->y_ende+3,
215                      (LPSTR) txt,
216                      strlen (txt));
217          } /* end if */
218          break;
219       } /* end switch */
220    }
221
222    /* Y-Achse skalieren : */
223    h = (int) hKapDat->max_y;
224    skal_abstand =
h>9999?5000:h>4999?1000:h>999?500:h>499?100:h>99?50:h>49?10:h>9?5:1;
225
226 #ifdef DEBUG
227    fprintf (stddbg, "  skal_abstand = %d, y_einheit %f, kd_max_y %d\n",
228             skal_abstand,
229             hKapDat->y_einheit,
230             hKapDat->max_y);
231    fflush (stddbg);
232 #endif
233
234    for (y = hKapDat->y_ende - (int) (hKapDat->y_einheit * (float) skal_abstand),
235            beschriftung = skal_abstand;
236         (beschriftung <= (int) hKapDat->max_y) & (y >= hKapDat->y_start);
237         y -= (int) (hKapDat->y_einheit * (float) skal_abstand),
238            beschriftung += skal_abstand)
239    {
240       MoveTo (hDC, hKapDat->x_start-5, y);
241       LineTo (hDC, hKapDat->x_start+5, y);
242       /*  beschriften */
243       sprintf (txt, "%d", beschriftung);
244       TextOut (hDC, (10 - strlen (txt)) * tmStuff.tmMaxCharWidth ,
245                y - tmStuff.tmHeight / 2,(LPSTR) txt, strlen (txt));
246    }
247 #ifdef DEBUG
248    fprintf (stddbg, "ende skala_zeichnen\n");
249    fflush (stddbg);
250 #endif
251 } /* end SkalaZeichnen */
252
253 void KapBestZeichnen (hWnd, hDC, hKapDat)
254 HWND     hWnd;
255 HDC      hDC;
256 KAPDAT   * hKapDat;
257 {
258    int   x,
259          y,
260          tag,
261          best_nr;
```

```
262    long  bestand;
263
264    SelectObject (hDC, hLinePen);
265
266    x = hKapDat->x_start;
267    y = hKapDat->y_ende
268       - (int) ((float) hKapDat->kap_best [0] * hKapDat->y_einheit);
269    MoveTo (hDC, x, y);
270
271    for (best_nr = 0; best_nr < hKapDat->kd_eintraege; best_nr ++)
272    {
273       /* bestand berechnen */
274       bestand = (long) 0;
275       for ( tag = (best_nr * TAGE_PRO_SLOT )
276                   - (int) hKapDat->kd_beststart
277                   + (int) (hKapDat->kd_startzeit / STD_PRO_TAG);
278             tag < ((best_nr + 1) * TAGE_PRO_SLOT )
279                   - (int) hKapDat->kd_beststart
280                   + (int) (hKapDat->kd_startzeit / STD_PRO_TAG);
281             tag ++) {
282          bestand += hKapDat->kap_best [tag];
283       } /* end for */
284
285       y = hKapDat->y_ende - (int) ((float) bestand * hKapDat->y_einheit);
286       LineTo (hDC, x, y);
287       x += hKapDat->x_einheit;
288       LineTo (hDC, x, y);
289    }
290 }
291
292 void KapBedZeichnen (hWnd, hDC, hKapDat)
293 HWND     hWnd;
294 HDC      hDC;
295 KAPDAT   * hKapDat;
296 {
297    int   slot,
298          auftrag,
299          x1,
300          x2,
301          y1,
302          y2;
303
304    for (slot = 0; slot < hKapDat->kd_eintraege; slot ++) {
305       x1 = hKapDat->x_start + (slot * hKapDat->x_einheit)
306            + hKapDat->x_einheit * (10 - /*hKapDat->blockbreite*/ 8) / 20;
307       x2 = x1 + hKapDat->x_einheit * /*hKapDat->blockbreite*/ 8 / 10;
308       y2 = hKapDat->y_ende;
309       for (auftrag = 1; auftrag <= hKapDat->kap_bed [slot][0].kb_bed; auftrag ++){
310          y1 = y2 + 1;
311          y2 -= (int) ((float) hKapDat->kap_bed [slot][auftrag].kb_bed
312                               * hKapDat->y_einheit);
313          SelectBlockObjects (hDC, hKapDat->kap_bed [slot][auftrag].kb_status);
314          Rectangle (hDC, x1, y1, x2, y2);
315          } /* end for */
316       } /* end for */
317 }
318
319 void StatFormZeichnen (hWnd, hDC, hKapDat)
320 HWND     hWnd;
321 HDC      hDC;
322 KAPDAT   *hKapDat;
323 {
324    SelectObject (hDC, hPenNormal);
325
326    hKapDat->statuszeile.y_header = hKapDat->winsize.bottom - 2 * tmStuff.tmHeight;
327    hKapDat->statuszeile.y_werte  = hKapDat->winsize.bottom - tmStuff.tmHeight;
```

```
 328    hKapDat->statuszeile.x_fanr   = hKapDat->winsize.left
 329                                   + tmStuff.tmMaxCharWidth / 2;
 330    hKapDat->statuszeile.x_fagnr  = hKapDat->statuszeile.x_fanr
 331                                   + 17 * tmStuff.tmMaxCharWidth;
 332    hKapDat->statuszeile.x_fstart = hKapDat->statuszeile.x_fagnr
 333                                   +  7 * tmStuff.tmMaxCharWidth ;
 334    hKapDat->statuszeile.x_lend   = hKapDat->statuszeile.x_fstart
 335                                   + 19 * tmStuff.tmMaxCharWidth ;
 336    hKapDat->statuszeile.x_bmnr   = hKapDat->statuszeile.x_lend
 337                                   + 19 * tmStuff.tmMaxCharWidth ;
 338
 339    MoveTo (hDC, hKapDat->winsize.left, hKapDat->statuszeile.y_header - 1);
 340    LineTo (hDC, hKapDat->winsize.right,hKapDat->statuszeile.y_header - 1);
 341    MoveTo (hDC, hKapDat->statuszeile.x_fagnr - 2, hKapDat->statuszeile.y_header -
1);
 342    LineTo (hDC, hKapDat->statuszeile.x_fagnr - 2, hKapDat->winsize.bottom);
 343    MoveTo (hDC, hKapDat->statuszeile.x_fstart - 2, hKapDat->statuszeile.y_header -
1);
 344    LineTo (hDC, hKapDat->statuszeile.x_fstart - 2, hKapDat->winsize.bottom);
 345    MoveTo (hDC, hKapDat->statuszeile.x_lend - 2, hKapDat->statuszeile.y_header -
1);
 346    LineTo (hDC, hKapDat->statuszeile.x_lend - 2, hKapDat->winsize.bottom);
 347    MoveTo (hDC, hKapDat->statuszeile.x_bmnr - 2, hKapDat->statuszeile.y_header -
1);
 348    LineTo (hDC, hKapDat->statuszeile.x_bmnr - 2, hKapDat->winsize.bottom);
 349
 350    TextOut (hDC, hKapDat->statuszeile.x_fanr, hKapDat->statuszeile.y_header,
 351             (LPSTR) "FA-NR :", 7);
 352    TextOut (hDC, hKapDat->statuszeile.x_fagnr, hKapDat->statuszeile.y_header,
 353             (LPSTR) "AG-NR:", 6);
 354    TextOut (hDC, hKapDat->statuszeile.x_fstart, hKapDat->statuszeile.y_header,
 355             (LPSTR) "START :", 7);
 356    TextOut (hDC, hKapDat->statuszeile.x_lend, hKapDat->statuszeile.y_header,
 357             (LPSTR) "ENDE :", 6);
 358    TextOut (hDC, hKapDat->statuszeile.x_bmnr, hKapDat->statuszeile.y_header,
 359             (LPSTR) "BM-NR :", 7);
 360
 361 } /* end StatFormZeichnen */
 362
 363 void StatRepZeichnen (hDC, hKapDat, hFAG)
 364 HDC      hDC;
 365 KAPDAT   *hKapDat;
 366 FAG      *hFAG;
 367 {
 368    char txt [20];
 369    int   tt,
 370          mm,
 371          jj,
 372          std,
 373          min,
 374          error;
 375
 376 #define START     ((hKapDat->kd_semode == MENU_LSLE)              \
 377                    ? hFAG->fag_lstart : hFAG->fag_fstart)
 378
 379 #define END       ((hKapDat->kd_semode == MENU_FSFE)              \
 380                    ? hFAG->fag_fend : hFAG->fag_lend)
 381
 382    if (hFAG != (FAG *) NULL) {
 383       sprintf (txt, "%-15s", hFAG->fag_fanr);
 384       TextOut (hDC, hKapDat->statuszeile.x_fanr, hKapDat->statuszeile.y_werte,
 385                (LPSTR) txt, strlen (txt));
 386       sprintf (txt, "%4d", hFAG->fag_nr);
 387       TextOut (hDC, hKapDat->statuszeile.x_fagnr, hKapDat->statuszeile.y_werte,
 388                (LPSTR) txt, strlen (txt));
 389       stdtodate (START, &tt, &mm, &jj, &std, &min, &error);
```

```
390       sprintf (txt, "%2d.%2d.19%2d %2d:%02d", tt, mm, jj, std, min);
391       TextOut (hDC, hKapDat->statuszeile.x_fstart, hKapDat->statuszeile.y_werte,
392                (LPSTR) txt, strlen (txt));
393       stdtodate (END, &tt, &mm, &jj, &std, &min, &error);
394       sprintf (txt, "%2d.%2d.19%2d %2d:%02d", tt, mm, jj, std, min);
395       TextOut (hDC, hKapDat->statuszeile.x_lend, hKapDat->statuszeile.y_werte,
396                (LPSTR) txt, strlen (txt));
397    } /* end if */
398    else {
399       TextOut (hDC, hKapDat->statuszeile.x_fanr, hKapDat->statuszeile.y_werte,
400                (LPSTR) "               ", 15);
401       TextOut (hDC, hKapDat->statuszeile.x_fagnr, hKapDat->statuszeile.y_werte,
402                (LPSTR) "    ", 4);
403       TextOut (hDC, hKapDat->statuszeile.x_fstart, hKapDat->statuszeile.y_werte,
404                (LPSTR) "                  ", 18);
405       TextOut (hDC, hKapDat->statuszeile.x_lend, hKapDat->statuszeile.y_werte,
406                (LPSTR) "                  ", 18);
407    } /* end else */
408 #undef START
409 #undef END
410 } /* end StatRepZeichnen */
411
412 void StatBMNRZeichnen (hDC, hKapDat)
413 HDC         hDC;
414 KAPDAT      *hKapDat;
415 {
416    char  txt [20];
417
418    if (hKapDat->kd_bmnr [0] != '\0') {
419       sprintf (txt, "%-15s", hKapDat->kd_bmnr);
420       TextOut (hDC, hKapDat->statuszeile.x_bmnr, hKapDat->statuszeile.y_werte,
421             (LPSTR) txt, strlen (txt));
422    } /* end if */
423    else {
424       TextOut (hDC, hKapDat->statuszeile.x_bmnr, hKapDat->statuszeile.y_werte,
425                (LPSTR) "               ", 15);
426    } /* end else */
427 } /* end StatBMNRZeichnen */
428
429 void KapPlanPaint (hWnd, hDC, hKapDat)
430 HWND hWnd;
431 HDC hDC;
432 KAPDAT * hKapDat;
433 {
434    HBRUSH   hBr,
435             hBrOld;
436
437 #ifdef DEBUG
438    fprintf (stddbg, "Beginn KapPlanPaint\n");
439    fflush (stddbg);
440 #endif
441
442    GetClipBox (hDC, (LPRECT) &(hKapDat->winsize));
443 #ifdef DEBUG
444    fprintf (stddbg, "  GetClipBox OK.\n");
445    fflush (stddbg);
446 #endif
447    GetTextMetrics (hDC, (LPTEXTMETRIC) &tmStuff);
448 #ifdef DEBUG
449    fprintf (stddbg, "  GetTextMetrics O.K.\n");
450    fflush (stddbg);
451 #endif
452
453    /* loesche Zeichenflaeche */
454    hBr = CreateSolidBrush (GetBkColor (hDC));
455    hBrOld = SelectObject (hDC, hBr);
```

```
456    FillRect (hDC, (LPRECT) &(hKapDat->winsize), hBr);
457    SelectObject (hDC, hBrOld);
458    DeleteObject (hBr);
459
460 #ifdef DEBUG
461    fprintf (stddbg, "  Zeichenflaeche geloescht.\n");
462    fflush (stddbg);
463 #endif
464
465    hKapDat->x_start = 11 * tmStuff.tmMaxCharWidth;
466    hKapDat->x_ende  = hKapDat->winsize.right - 20;
467    hKapDat->x_size  = hKapDat->x_ende - hKapDat->x_start;
468    hKapDat->y_start = hKapDat->winsize.top + 15;
469    hKapDat->y_ende  = hKapDat->winsize.bottom - (3 * tmStuff.tmHeight + 5);
470    hKapDat->y_size  = hKapDat->y_ende - hKapDat->y_start;
471    StatFormZeichnen     (hWnd, hDC, hKapDat);
472 #ifdef DEBUG
473    fprintf (stddbg, "  StatusForm gezeichnet.\n");
474    fflush (stddbg);
475 #endif
476    StatBMNRZeichnen     (hDC, hKapDat);
477 #ifdef DEBUG
478    fprintf (stddbg, "  BMNR gezeichnet.\n");
479    fflush (stddbg);
480 #endif
481
482    if (hKapDat->kd_daten_vorhanden) {
483 #ifdef DEBUG
484       fprintf (stddbg, "  if angefangen\n");
485       fflush (stddbg);
486 #endif
487       hKapDat->x_einheit = hKapDat->x_size / hKapDat->kd_eintraege;
488       hKapDat->y_einheit = (float) hKapDat->y_size
489                            / (float) (hKapDat->max_y == 0L
490                                       ? 1L : hKapDat->max_y);
491
492 #ifdef DEBUG
493       fprintf (stddbg,
494                "  Einheit X/Y = %d/%d\n",
495                hKapDat->x_einheit,
496                hKapDat->y_einheit);
497       fflush (stddbg);
498 #endif
499
500       SkalaZeichnen        (hWnd, hDC, hKapDat);
501 #ifdef DEBUG
502       fprintf (stddbg, "  Skala gezeichnet.\n");
503       fflush (stddbg);
504 #endif
505       KapBedZeichnen       (hWnd, hDC, hKapDat);
506 #ifdef DEBUG
507       fprintf (stddbg, "  Bedarf gezeichnet.\n");
508       fflush (stddbg);
509 #endif
510       KapBestZeichnen      (hWnd, hDC, hKapDat);
511 #ifdef DEBUG
512       fprintf (stddbg, "  Bestand gezeichnet.\n");
513       fflush (stddbg);
514 #endif
515    }; /* end if */
516
517 #ifdef DEBUG
518    fprintf (stddbg, "Ende KapPlanPaint\n");
519    fflush (stddbg);
520 #endif
521 }  /* end KapPlanPaint */
```

```
522
523 void IdentifyBlock (hWnd, hKapDat, lPos, hBlock)
524 HWND     hWnd;
525 KAPDAT   * hKapDat;
526 long     lPos;
527 BLOCK    *hBlock;
528 {
529    int      y1,
530             y2,
531             slot,
532             auftrag;
533    POINT    pos;
534
535    pos = MAKEPOINT (lPos);
536
537    if (hKapDat->kd_daten_vorhanden) {
538       slot = (pos.x - hKapDat->x_start) / hKapDat->x_einheit;
539       if ((slot < hKapDat->kd_eintraege)
540          & (pos.y < hKapDat->y_ende)
541          & (pos.x > hKapDat->x_start)) {
542          y1 = hKapDat->y_ende;
543          for (auftrag = 1;
544               (auftrag <= hKapDat->kap_bed [slot][0].kb_bed)
545                & (pos.y < y1);
546               auftrag ++) {
547             y2 = y1 + 1;
548             y1 -= (int) ((float) hKapDat->kap_bed [slot][auftrag].kb_bed
549                                  * hKapDat->y_einheit);
550          }; /* end for */
551
552          if ((y2 > pos.y) & (y1 <= pos.y)) {
553             /* Box gefunden */
554             hBlock->bl_auftrag = auftrag - 1;
555             hBlock->bl_slot    = slot;
556             hBlock->bl_basis   = y2;
557          } /* end if */
558          else {
559             /* Box nicht gefunden */
560             hBlock->bl_auftrag = 0;
561          } /* end else */
562       } /* end then */
563       else {
564          /* Box nicht gefunden */
565          hBlock->bl_auftrag = 0;
566       } /* end else */
567    } /* end if */
568    else {
569       /* keine Daten vorhanden */
570       hBlock->bl_auftrag = 0;
571    } /* end else */
572 } /* end IdentifyBlock */
573
574 void ZeichneBlock (hDC, hKapDat, hBlock)
575 HDC      hDC;
576 KAPDAT   * hKapDat;
577 BLOCK    * hBlock;
578 {
579    int      x1,
580             x2,
581             y1,
582             y2;
583
584    if (hBlock->bl_auftrag == 0)     /* Block ist nicht gueltig */
585       return;
586
587    y1 = hBlock->bl_basis + 1;
```

```
588    y2 = hBlock->bl_basis
589         - (int) ((float) hKapDat->kap_bed [hBlock->bl_slot]
590                                           [hBlock->bl_auftrag].kb_bed
591                         * hKapDat->y_einheit);
592
593    x1 = hKapDat->x_start + (hBlock->bl_slot * hKapDat->x_einheit)
594         + hKapDat->x_einheit * (100 - /*hKapDat->blockbreite*/ 80) / 200;
595    x2 = x1 + hKapDat->x_einheit * /*hKapDat->blockbreite*/ 80 / 100;
596    Rectangle (hDC, x1, y1, x2, y2);
597
598 } /* end ZeichneBlock */
599
600 void ZeichneFAG (hDC, hKapDat, hFAG)
601 HDC      hDC;
602 KAPDAT   * hKapDat;
603 FAG      * hFAG;
604 {
605    BLOCK b;
606
607 #ifdef DEBUG
608    fprintf (stddbg, "Start ZeichneFAG (%p)\n", hFAG);
609    fflush (stddbg);
610 #endif
611
612    for (b.bl_slot = 0; b.bl_slot  < MAX_SLOTS; b.bl_slot ++) {
613       /* suche block */
614       for (b.bl_auftrag = 1, b.bl_basis = hKapDat->y_ende;
615            b.bl_auftrag <= hKapDat->kap_bed[b.bl_slot][0].kb_bed
616         & hFAG != hKapDat->kap_bed[b.bl_slot][b.bl_auftrag].kb_fag;
617            b.bl_auftrag ++) {
618         b.bl_basis -= (int) ((float) hKapDat->kap_bed[b.bl_slot]
619                                                      [b.bl_auftrag].kb_bed
620                                    * hKapDat->y_einheit);
621       } /* end for */
622       if (b.bl_auftrag <= hKapDat->kap_bed[b.bl_slot][0].kb_bed) {
623          if (hFAG == hKapDat->kap_bed [b.bl_slot][b.bl_auftrag].kb_fag) {
624             ZeichneBlock (hDC, hKapDat, &b);
625          } /* end if */
626       } /* end if */
627    } /* end for */
628 #ifdef DEBUG
629    fprintf (stddbg, "Ende ZeichneFAG\n");
630    fflush (stddbg);
631 #endif
632 } /* end ZeichneFAG */
633
634 void BlockVerschieben (lParam, hKapDat, hWnd)
635 long     lParam;
636 KAPDAT   *hKapDat;
637 HWND     hWnd;
638 {
639    POINT       aktpos;
640    long        spiel,
641                fdauer,
642                ldauer,
643                aktzeit;
644
645    /* Plausibilitaet ueberpruefen */
646    if ((hAktFAG->fag_status == FAS_AF)
647       |(hAktFAG->fag_status == FAS_AB)
648       |(hAktFAG->fag_status == FAS_AG)
649       |(hAktFAG->fag_status == FAS_TR)) {
650       /* Warnung ausgeben */
651       MessageBox (hWnd,
652                   (LPSTR) "Auftrag kann nicht umgeplant werden !",
653                   (LPSTR) "Umplanen",
```

```
654                  MB_OK | MB_ICONEXCLAMATION);
655      return;
656    } /* end if */
657
658    /* Zeiten berechnen */
659    spiel  = hAktFAG->fag_lstart - hAktFAG->fag_fstart;
660    ldauer = hAktFAG->fag_lend - hAktFAG->fag_lstart;
661    fdauer = hAktFAG->fag_fend - hAktFAG->fag_fstart;
662
663    aktpos = MAKEPOINT (lParam);
664    aktzeit =   (long) (aktpos.x - hKapDat->x_start)
665              * (hKapDat->kd_endzeit - hKapDat->kd_startzeit)
666              / (long) (hKapDat->x_ende - hKapDat->x_start)
667              + hKapDat->kd_startzeit;
668
669    /* FAG aendern */
670    switch (hKapDat->kd_semode) {
671    case MENU_FSFE :
672       hAktFAG->fag_fstart = aktzeit;
673       hAktFAG->fag_fend   = aktzeit + fdauer;
674       if (hAktFAG->fag_fstart > hAktFAG->fag_lstart) {
675          hAktFAG->fag_lstart = hAktFAG->fag_fstart;
676          hAktFAG->fag_lend = hAktFAG->fag_lstart + ldauer;
677       } /* end if */
678       break;
679    case MENU_FSLE :
680       hAktFAG->fag_fstart = aktzeit;
681       hAktFAG->fag_fend   = aktzeit + fdauer;
682       hAktFAG->fag_lstart = aktzeit + spiel;
683       hAktFAG->fag_lend   = hAktFAG->fag_lstart + ldauer;
684       break;
685    case MENU_LSLE :
686       hAktFAG->fag_lstart = aktzeit;
687       hAktFAG->fag_lend   = aktzeit + ldauer;
688       if (hAktFAG->fag_fstart > hAktFAG->fag_lstart) {
689          hAktFAG->fag_fstart = hAktFAG->fag_lstart;
690          hAktFAG->fag_fend = hAktFAG->fag_fstart + fdauer;
691       } /* end if */
692       break;
693    } /* end switch */
694
695    hAktFAG->fag_veraendert = TRUE;
696 } /* end BlockVerschieben */
697
698 void BlockSchieben (hWnd, lParam, hKapDat)
699 HWND  hWnd;
700 long  lParam;
701 KAPDAT *hKapDat;
702 {
703    POINT       aktpos;
704    int         tt,
705                mm,
706                jj,
707                std,
708                min,
709                error;
710    long        aktzeit,
711                differenz;
712    char        txt [20];
713    HDC         hDC;
714
715    /* Zeit bestimmen */
716    differenz = (hKapDat->kd_semode == MENU_FSFE
717                   ? hAktFAG->fag_fend : hAktFAG->fag_lend)
718              - (hKapDat->kd_semode == MENU_LSLE
719                   ? hAktFAG->fag_lstart : hAktFAG->fag_fstart);
```

```
720    aktpos = MAKEPOINT (lParam);
721    aktzeit = (((long) (aktpos.x - hKapDat->x_start)
722                 * (hKapDat->kd_endzeit - hKapDat->kd_startzeit))
723                / (long) (hKapDat->x_size))
724               + hKapDat->kd_startzeit;
725
726    /* Zeit anzeigen  */
727    hDC = GetDC (hWnd);
728    stdtodate (aktzeit, &tt, &mm, &jj, &std, &min, &error);
729    sprintf (txt, "%2d.%2d.19%2d %2d:%02d", tt, mm, jj, std, min);
730    TextOut (hDC, hKapDat->statuszeile.x_fstart, hKapDat->statuszeile.y_werte,
731             (LPSTR) txt, strlen (txt));
732
733    aktzeit += differenz;
734    stdtodate (aktzeit, &tt, &mm, &jj, &std, &min, &error);
735    sprintf (txt, "%2d.%2d.19%2d %2d:%02d", tt, mm, jj, std, min);
736    TextOut (hDC, hKapDat->statuszeile.x_lend, hKapDat->statuszeile.y_werte,
737             (LPSTR) txt, strlen (txt));
738    ReleaseDC (hWnd, hDC);
739
740 } /* end BlockSchieben */
741
742 void TrackMousePosition (hWnd, lParam, hKapDat)
743 HWND  hWnd;
744 long  lParam;
745 KAPDAT *hKapDat;
746 {
747    HDC      hDC;
748    BLOCK    block;
749    char     txt [10];
750
751    hDC = GetDC (hWnd);
752    if ((hAktFAG == (FAG *) NULL) & (hKapDat->kd_daten_vorhanden)) {
753       IdentifyBlock (hWnd, hKapDat, lParam, &block);
754
755       if (block.bl_auftrag != 0) {
756          StatRepZeichnen (hDC, hKapDat,
757                                hKapDat->kap_bed [block.bl_slot][block.bl_auftrag]
758                                .kb_fag);
759       } /* end if */
760       else {
761          StatRepZeichnen (hDC, hKapDat, (FAG *) NULL);
762       } /* end else */
763    } /* end if */
764    ReleaseDC (hWnd, hDC);
765 } /* end track mouse position */
766
```

8.2.6 DATEN.H

```
1 extern int  umplanen              (KAPDAT*, int);
2 extern BOOL read_fags             (KAPDAT*);
3 extern int  read_kap_best         (KAPDAT*);
4 extern int  darstellung_berechnen (KAPDAT*);
```

8.2.7 DATEN.C

```
 1 /*-----------------------------------------------
 2 **
 3 **          Modul  Datenbankzugriffe und
 4 **                 Datenaufbereitung
 5 **
 6 **          Autor : A.Heidemann
 7 **
 8 **-----------------------------------------------*/
 9
10 #define  NOMINMAX
11 #include "windows.h"
12 #include "Kap_Plan.h"
13 #include "stdio.h"
14 #include "math.h"
15 #include "malloc.h"
16 #include "stdlib.h"
17 #include "dos.h"
18 #include "string.h"
19 #include "process.h"
20 #include "daten.dcl"       /* Prozedurkoepfe fuer Daten */
21
22 #ifdef DEBUG
23    extern FILE *stddbg;
24 #endif
25
26 /*-----------------------------------------------------------*/
27 /* externe datenbank - prozeduren                            */
28 /*    Funktionswert immer fehlerfrei = TRUE, sonst FALSE  */
29 /*    letzter Parameter immer Fehlernummer im Fehlerfall  */
30 /*-----------------------------------------------------------*/
31
32 extern BOOL InitBMGList (int *);
33 extern BOOL EndBMGList (int *);
34 extern BOOL NextBMGList (char [], int *);
35 extern BOOL InitFAGList (char[], long int, long int, int, int *);
36 extern BOOL EndFAGList (int *);
37 extern BOOL NextFAGList (FAG *, int *);
38 extern BOOL InitBestList (BMNR, long int, long int, int *);
39 extern BOOL EndBestList (int *);
40 extern BOOL NextBestList (long int *, int *);
41 extern BOOL Login (char[], char[], KAPDAT *, int *);
42 extern BOOL Logout (int *);
43 extern BOOL Rollback (int *);
44 extern BOOL Commit (int *);
45 extern BOOL FADaten (FA_DATEN *, int *);
46 extern BOOL ErrorText (int, char []);
47 extern BOOL datetostd (long int *, int, int, int, int, int, int *);
48 extern BOOL stdtodate (long int, int *, int *, int *, int *, int *, int *);
49 extern BOOL stdtowoche (long int, int *, int *);
50
51 extern AGUMTERM (char [], int, char [], char [], char [], char [],
52                  int *, int, int *);
53
54 extern ENGTERM  (char [], int, char [], char [], char [], char [],
55                  int *, int, int *);
56
57 /*--------------------------------------- externe Fehlerbehandlung */
58 extern int  ErrorMsg (char*, char*);
59
60 BOOL UpdateFAG (hFag, cmd, error)
61 FAG   *hFag;
62 int   cmd,
63       *error;
64 {
```

```
 65    int      tag,
 66             mon,
 67             jahr,
 68             std,
 69             min,
 70             date_error;
 71    char     fstart   [15],
 72             fend     [15],
 73             lstart   [15],
 74             lend     [15];
 75
 76 #ifdef DEBUG
 77    fprintf (stddbg, "UPDATE FAG %s/%d, FS %ld, LS %ld, FE %ld, LE %ld\n",
 78                     hFag->fag_fanr,
 79                     hFag->fag_nr,
 80                     hFag->fag_fstart,
 81                     hFag->fag_lstart,
 82                     hFag->fag_fend,
 83                     hFag->fag_lend);
 84    fflush (stddbg);
 85 #endif
 86    stdtodate (hFag->fag_fstart, &tag, &mon, &jahr, &std, &min, error);
 87    sprintf (fstart, "%02d.%02d.%02d %02d:%02d", tag, mon, jahr, std, min);
 88
 89    stdtodate (hFag->fag_fend  , &tag, &mon, &jahr, &std, &min, error);
 90    sprintf (fend, "%02d.%02d.%02d %02d:%02d", tag, mon, jahr, std, min);
 91
 92    stdtodate (hFag->fag_lstart, &tag, &mon, &jahr, &std, &min, error);
 93    sprintf (lstart, "%02d.%02d.%02d %02d:%02d", tag, mon, jahr, std, min);
 94
 95    stdtodate (hFag->fag_lend  , &tag, &mon, &jahr, &std, &min, error);
 96    sprintf (lend, "%02d.%02d.%02d %02d:%02d", tag, mon, jahr, std, min);
 97
 98    switch (cmd) {
 99    case MENU_UMPLAN :
100       AGUMTERM   (hFag->fag_fanr, hFag->fag_nr,
101                   fstart, fend, lstart, lend, (int *) &(hFag->fag_status),
102                   0, error);
103       break;
104    case MENU_ENGPASS :
105       ENGTERM    (hFag->fag_fanr, hFag->fag_nr,
106                   fstart, fend, lstart, lend, (int *) &(hFag->fag_status),
107                   0, error);
108       break;
109    } /* end switch */
110
111    sscanf (fstart, "%02d.%02d.%02d %02d:%02d", &tag, &mon, &jahr, &std, &min);
112    datetostd (&(hFag->fag_fstart), tag, mon, jahr, std, min, & date_error);
113
114    sscanf (fend, "%02d.%02d.%02d %02d:%02d", &tag, &mon, &jahr, &std, &min);
115    datetostd (&(hFag->fag_fend), tag, mon, jahr, std, min, & date_error);
116
117    sscanf (lstart, "%02d.%02d.%02d %02d:%02d", &tag, &mon, &jahr, &std, &min);
118    datetostd (&(hFag->fag_lstart), tag, mon, jahr, std, min, & date_error);
119
120    sscanf (lend, "%02d.%02d.%02d %02d:%02d", &tag, &mon, &jahr, &std, &min);
121    datetostd (&(hFag->fag_lend), tag, mon, jahr, std, min, & date_error);
122
123    return (*error == 0);
124 } /* end UpdateFAG */
125
126 void umplanen (hKapDat, cmd)
127 KAPDAT   *hKapDat;
128 int      cmd;
129 {
130    FAG   * pfag;
```

```
131    int   error;
132    char  txt [80];
133
134 #ifdef DEBUG
135    fprintf (stddbg, "beginn umplanen (%d)\n", cmd);
136    fflush (stddbg);
137 #endif
138
139    for (pfag = hKapDat->fag_liste; pfag != (FAG*) NULL; pfag = pfag->fag_next) {
140       if (pfag->fag_veraendert) {
141          /* umterminieren anstossen */
142          error = 0;
143          if (! UpdateFAG (pfag, cmd, &error)) {
144             switch (error) {
145             case ERR_TERMIN :
146                sprintf (txt, "Ecktermin nicht haltbar bei AG-Nr.%s/%d",
147                         pfag->fag_fanr, pfag->fag_nr);
148                break;
149             case MEMORY_OUT :
150                sprintf (txt, "Zu wenig Speicher fuer AG-Nr.%s/%d",
151                         pfag->fag_fanr, pfag->fag_nr);
152                break;
153             default :
154                sprintf (txt, "Fehler Nr. %d bei AG-Nr.%s/%d",
155                         error, pfag->fag_fanr, pfag->fag_nr);
156                break;
157             } /* end switch */
158             ErrorMsg (txt, "FAG Umterminieren :");
159             Rollback (&error);
160          } /* end if */
161          else {
162             if (! Commit (&error)) {
163                sprintf (txt, "Commit ERROR (%d)", error);
164                ErrorMsg (txt, "FAG Umterminieren :");
165             } /* end if */
166          } /* end else */
167          pfag->fag_veraendert = FALSE;
168       } /* end if */
169    } /* end for */
170
171 #ifdef DEBUG
172    fprintf (stddbg, "ende umplanen\n");
173    fflush (stddbg);
174 #endif
175 } /* end umplanen */
176
177 BOOL
178 read_fags (hKapDat)
179 KAPDAT   *hKapDat;
180 {
181    FAG   fag,
182          **fag_pptr,
183          *fag_ptr1,
184          *fag_ptr2;
185    char  errtxt [ERRTXTLEN],
186          txt [ERRTXTLEN];
187    int   error;
188    HANDLE hMem;
189
190 #ifdef DEBUG
191    fprintf (stddbg, "Begin read_fags :\n");
192    fflush (stddbg);
193 #endif
194
195    if (! InitFAGList (hKapDat->kd_bmnr,
196                       hKapDat->kd_startzeit,
```

```
197                      hKapDat->kd_endzeit,
198                      hKapDat->kd_semode,
199                      &error)) {
200       /* Fehler beim Initialisieren */
201       if (ErrorText (error, errtxt)) {
202          sprintf (txt, "ERROR %d : %s", error, errtxt);
203       }
204       else {
205          sprintf (txt, "ERROR %d, Fehler unbekannt");
206       }
207       ErrorMsg (txt, "DB - ERROR");
208       return FALSE;
209    } /* end if */
210    fag_pptr = & (hKapDat->fag_liste);
211    hKapDat->kd_maxzeit = 0;
212    hKapDat->kd_minzeit = MAX_LONG;
213
214    while (NextFAGList (&fag, &error)){
215 #ifdef DEBUG
216       fprintf (stddbg, "  Insert into list : %d/%s/%ld/%ld/%ld/%ld/%ld/%d\n",
217                        fag.fag_nr,
218                        fag.fag_fanr,
219                        fag.fag_fstart,
220                        fag.fag_lstart,
221                        fag.fag_fend,
222                        fag.fag_lend,
223                        fag.fag_bedarf,
224                        fag.fag_status);
225       fflush (stddbg);
226 #endif
227       if (*fag_pptr == (FAG*) NULL) {
228          /* neuen Speicherplatz anfordern */
229 #ifdef DEBUG
230          fprintf (stddbg, "  allocate new memory, size : %ld\n",
231                           (unsigned long) sizeof (FAG));
232          fflush (stddbg);
233 #endif
234       /* *fag_pptr = (FAG *) malloc (sizeof (FAG)); */
235          hMem = GlobalAlloc (GMEM_FIXED, (DWORD) sizeof (FAG));
236          if (hMem == (HANDLE) NULL) {
237 #ifdef DEBUG
238             fprintf (stddbg, "  Memory allocation error\n");
239             fflush (stddbg);
240 #endif
241             ErrorMsg ("Memory allocation error", "GlobalAlloc");
242             return FALSE;
243          } /* end if */;
244          *fag_pptr = (FAG*) GlobalWire (hMem);
245 #ifdef DEBUG
246          fprintf (stddbg, "  new memory allocated at %p\n", *fag_pptr);
247          fflush (stddbg);
248 #endif
249          (*fag_pptr)->fag_next = (FAG *) NULL;
250       }
251 #ifdef DEBUG
252       fprintf (stddbg, "  begin copy values\n");
253       fflush (stddbg);
254 #endif
255       (*fag_pptr)->fag_nr = fag.fag_nr;
256       strcpy ((*fag_pptr)->fag_fanr, fag.fag_fanr);
257       (*fag_pptr)->fag_fstart  = fag.fag_fstart;
258       (*fag_pptr)->fag_lstart  = fag.fag_lstart;
259       (*fag_pptr)->fag_fend    = fag.fag_fend;
260       (*fag_pptr)->fag_lend    = fag.fag_lend;
261       (*fag_pptr)->fag_bedarf  = fag.fag_bedarf;
262       (*fag_pptr)->fag_status  = fag.fag_status;
```

```
263       (*fag_pptr)->fag_veraendert = FALSE;
264       hKapDat->kd_minzeit = min (hKapDat->kd_minzeit, fag.fag_fstart);
265       hKapDat->kd_maxzeit = max (hKapDat->kd_maxzeit, fag.fag_lend);
266 #ifdef DEBUG
267       fprintf (stddbg, "  end copy values\n");
268       fflush (stddbg);
269 #endif
270       fag_pptr = &((*fag_pptr)->fag_next);
271 #ifdef DEBUG
272       fprintf (stddbg, "  next pointer : %p\n", fag_pptr);
273       fflush (stddbg);
274 #endif
275    } /* end while */
276
277    fag_ptr1 = *fag_pptr;
278    *fag_pptr = (FAG*) NULL;
279    /* ueberfluessigen Speicher freigeben */
280    while (fag_ptr1 != (FAG*) NULL) {
281       /* gib Platz frei */
282       fag_ptr2 = fag_ptr1->fag_next;
283       if ((HANDLE) NULL != GlobalFree ((HANDLE) GlobalHandle (fag_ptr1))) {
284 #ifdef DEBUG
285          fprintf (stddbg, "  Memory free error\n");
286          fflush (stddbg);
287 #endif
288          ErrorMsg ("Memory free error", "GlobalFree");
289          return (FALSE);
290       } /* end if */
291       fag_ptr1 = fag_ptr2;
292    } /* end while */
293
294    if (!EndFAGList (&error)) {
295       if (ErrorText (error, errtxt)) {
296          sprintf (txt, "ERROR %d : %s", error, errtxt);
297       }
298       else {
299          sprintf (txt, "ERROR %d, Fehler unbekannt");
300       }
301       ErrorMsg (txt, "DB - ERROR");
302       return FALSE;
303    } /* end if */
304
305 #ifdef DEBUG
306    fprintf (stddbg, "End read_fags :\n");
307    fflush (stddbg);
308 #endif
309    return (TRUE);
310 } /* end read_fags */
311
312 read_kap_best (hKapDat)
313 KAPDAT   *hKapDat;
314 {
315    int   i,
316          error;
317    long  bestand;
318    char  errtxt [ERRTXTLEN],
319          txt [ERRTXTLEN];
320
321    if (!InitBestList (hKapDat->kd_bmnr,
322                       hKapDat->kd_minzeit - 23 * STD_PRO_MONAT,
323                       hKapDat->kd_maxzeit + 23 * STD_PRO_MONAT,
324                       &error)) {
325       if (ErrorText (error, errtxt)) {
326          sprintf (txt, "ERROR %d : %s", error, errtxt);
327       }
328       else {
```

```
329          sprintf (txt, "ERROR %d, Fehler unbekannt");
330       }
331       ErrorMsg (txt, "DB - ERROR");
332       return FALSE;
333    } /* end if */
334
335    hKapDat->kd_beststart = (hKapDat->kd_minzeit - 23 * STD_PRO_MONAT)
336                            / STD_PRO_TAG;
337
338    i = 0;
339    while (NextBestList (&bestand, &error)) {
340       if (i < MAX_BESTS) {
341          hKapDat->kap_best [i] = bestand;
342       };
343       i ++;
344    }
345    if (!EndBestList (&error)) {
346       if (ErrorText (error, errtxt)) {
347          sprintf (txt, "ERROR %d : %s", error, errtxt);
348       }
349       else {
350          sprintf (txt, "ERROR %d, Fehler unbekannt");
351       }
352       ErrorMsg (txt, "DB - ERROR");
353       return FALSE;
354    } /* end if */
355    return TRUE;
356 } /* end read_kap_best */
357
358 void darstellung_berechnen (hKapDat)
359 KAPDAT *hKapDat;
360 {
361    int         slot,
362                tage_pro_slot,
363                auftrag,
364                startslot,
365                endslot,
366                tag;
367    long        ges_bed,
368                ges_best,
369                slot_best;
370    float       rel_start_anteil,
371                rel_end_anteil,
372                rel_best_anteil;
373    FAG         *fag_ptr;
374
375 #define START     ((hKapDat->kd_semode == MENU_LSLE)               \
376                    ? fag_ptr->fag_lstart : fag_ptr->fag_fstart)
377
378 #define END       ((hKapDat->kd_semode == MENU_FSFE)               \
379                    ? fag_ptr->fag_fend : fag_ptr->fag_lend)
380
381
382 #ifdef DEBUG
383    fprintf (
384       stddbg,
385       "dar.-ber. :\n  kd_zeit :%d\n  kd_startzeit :%ld\n  kd_endzeit :%ld\n",
386       hKapDat->kd_zeit, hKapDat->kd_startzeit, hKapDat->kd_endzeit
387    );
388    fflush (stddbg);
389 #endif
390
391    /* initialisieren */
392    hKapDat->max_y = 0;
393
394    for (slot = 0; slot < MAX_SLOTS; slot ++) {
```

```
395       hKapDat->kap_bed[slot][0].kb_bed = 0;
396     } /* end for */
397
398     /* alle FAGs bearbeiten */
399     for (fag_ptr = hKapDat->fag_liste;
400          fag_ptr != (FAG*) NULL;
401          fag_ptr = fag_ptr->fag_next) {
402       /* einen FAG bearbeiten */
403 #ifdef DEBUG
404       fprintf (stddbg, "  FAG bearbeiten : START = %ld, END = %ld, BEDARF = %ld\n",
405                START,
406                END,
407                fag_ptr->fag_bedarf);
408       fflush (stddbg);
409 #endif
410
411       if ((START < hKapDat->kd_endzeit)
412           & (END > hKapDat->kd_startzeit)) {
413 #ifdef DEBUG
414         fprintf (stddbg, "  bloecke erzeugen\n");
415         fflush (stddbg);
416 #endif
417
418         startslot = (int) floor ((double) (START - hKapDat->kd_startzeit)
419                                  / (double) MAKE_STD );
420         endslot   = (int) floor ((double) (END - hKapDat->kd_startzeit)
421                                  / (double) MAKE_STD );
422
423         rel_start_anteil = (float) 1.0
424                          - (float)((START - hKapDat->kd_startzeit)
425                                    % MAKE_STD )
426                            / (float) MAKE_STD ;
427 #ifdef DEBUG
428         fprintf (stddbg, "  r_start : %d, %ld, %ld, %ld, %d, %ld, %ld \n",
429                  startslot,
430                  hKapDat->kd_startzeit,
431                  hKapDat->kd_endzeit,
432                  START,
433                  hKapDat->kd_zeit,
434                  MAKE_STD   ,
435                  MAKE_SLOTS );
436         fflush (stddbg);
437 #endif
438
439         rel_end_anteil = (float) ((END - hKapDat->kd_startzeit)
440                                   % MAKE_STD )
441                          / (float) MAKE_STD ;
442
443 #ifdef DEBUG
444         fprintf (stddbg, "  r_end : %d, %ld, %ld, %ld, %d, %ld, %ld \n",
445                  endslot,
446                  hKapDat->kd_startzeit,
447                  hKapDat->kd_endzeit,
448                  END,
449                  hKapDat->kd_zeit,
450                  MAKE_STD   ,
451                  MAKE_SLOTS );
452         fflush (stddbg);
453 #endif
454
455         /* bestand berechnen */
456         if ((START / STD_PRO_TAG) == (END / STD_PRO_TAG)) {
457            /* starttag == endtag */
458            ges_best = hKapDat->kap_best [START / STD_PRO_TAG
459                                          - hKapDat->kd_beststart];
460         } /* end if */
```

```
461          else {
462             ges_best = (long) (rel_start_anteil * (float) hKapDat->kap_best
463                                [START / STD_PRO_TAG - hKapDat->kd_beststart]
464                             + rel_end_anteil * (float) hKapDat->kap_best
465                                [END / STD_PRO_TAG - hKapDat->kd_beststart]);
466
467             for (slot = (int)(START / STD_PRO_TAG - hKapDat->kd_beststart) +1;
468                  slot < (int)(END / STD_PRO_TAG - hKapDat->kd_beststart);
469                  slot ++) {
470 #ifdef DEBUG_ALL
471                fprintf (stddbg, "  tag %d, best %ld, zw-erg %ld\n",
472                                 slot,
473                                 hKapDat->kap_best [slot],
474                                 ges_best);
475                fflush (stddbg);
476 #endif
477                ges_best += hKapDat->kap_best [slot];
478             } /* end for */
479          } /* end else */
480 #ifdef DEBUG
481          fprintf (stddbg, "  ges_best : %ld\n", ges_best);
482          fflush (stddbg);
483 #endif
484
485          if (ges_best > 0) {
486             rel_best_anteil = fag_ptr->fag_bedarf / (float) ges_best;
487          } /* end if */
488          else {
489             rel_best_anteil = (float) 0.0;
490          } /* end else */
491
492 #ifdef DEBUG
493          fprintf (stddbg,
494                   "  Startslot : %d, Anteil : %e\n",
495                   startslot,
496                   rel_start_anteil);
497          fprintf (stddbg,
498                   "  Endslot : %d, Anteil : %e\n",
499                   endslot,
500                   rel_end_anteil);
501          fflush (stddbg);
502 #endif
503
504          for (slot = max (startslot, 0);
505               slot <= min (endslot, (int) MAKE_SLOTS  - 1);
506               slot ++) {
507             /* einen Block erzeugen */
508             auftrag = ++ (hKapDat->kap_bed [slot][0].kb_bed);
509             if (auftrag >= MAX_AUFTRAEGE) {
510                /* zuviele Auftraege ! */
511                auftrag = -- (hKapDat->kap_bed [slot][0].kb_bed);
512             } /* end if */
513
514             /* slot - bestand berechnen */
515             slot_best = (long) 0;
516             for (tag = slot * TAGE_PRO_SLOT
517                        + (int) (hKapDat->kd_startzeit / STD_PRO_TAG
518                                 - hKapDat->kd_beststart);
519                  tag < (slot + 1) * TAGE_PRO_SLOT
520                        + (int) (hKapDat->kd_startzeit / STD_PRO_TAG
521                                 - hKapDat->kd_beststart);
522                  tag ++) {
523                slot_best += hKapDat->kap_best [tag];
524             } /* end for */
525
526             if (rel_best_anteil == (float) 0.0) {
```

```
527               hKapDat->kap_bed[slot][auftrag].kb_bed = 50; /* bel. Wert */
528            } /* end if */
529            else if (startslot == endslot) {
530               hKapDat->kap_bed[slot][auftrag].kb_bed =
531                  (int)(fag_ptr->fag_bedarf);
532            } /* end if */
533            else if (slot == startslot) {
534               hKapDat->kap_bed[slot][auftrag].kb_bed =
535                  (int)(slot_best * rel_best_anteil * rel_start_anteil);
536            } /* end if */
537            else if (slot == endslot) {
538               hKapDat->kap_bed[slot][auftrag].kb_bed =
539                  (int)(slot_best * rel_best_anteil * rel_end_anteil);
540            } /* end if */
541            else {
542               hKapDat->kap_bed[slot][auftrag].kb_bed =
543                  (int)(slot_best * rel_best_anteil);
544            } /* end else */
545            hKapDat->kap_bed [slot][auftrag].kb_status = fag_ptr->fag_status;
546            hKapDat->kap_bed [slot][auftrag].kb_fag = fag_ptr;
547 #ifdef DEBUG
548            fprintf (stddbg, "  Block %d/%d, Wert : %d, Status :%d\n",
549                     slot,
550                     auftrag,
551                     hKapDat->kap_bed[slot][auftrag].kb_bed,
552                     hKapDat->kap_bed [slot][auftrag].kb_status);
553            fflush (stddbg);
554 #endif
555         } /* end for */
556      } /* end if */
557   } /* end for */
558
559   /* max Y feststellen */
560   hKapDat->max_y = 0;
561   for (slot = 0; slot < (int) MAKE_SLOTS ; slot ++) {
562      /* bedarf berechnen */
563      ges_bed = 0;
564      for (auftrag = 1; auftrag <= hKapDat->kap_bed [slot][0].kb_bed; auftrag ++) {
565         ges_bed += hKapDat->kap_bed [slot][auftrag].kb_bed;
566      } /* end for */
567      /* bestand berechnen */
568      ges_best = 0L;
569      tage_pro_slot = TAGE_PRO_SLOT ;
570      for (tag = (slot * tage_pro_slot)
571                  - (int) hKapDat->kd_beststart
572                  + (int) (hKapDat->kd_startzeit / STD_PRO_TAG);
573           tag < ((slot+1) * tage_pro_slot)
574                   - (int) hKapDat->kd_beststart
575                   + (int) (hKapDat->kd_startzeit / STD_PRO_TAG);
576           tag ++) {
577 #ifdef DEBUG
578         fprintf (stddbg, "  tag %d, best %ld, zw-erg %ld\n",
579                          tag,
580                          hKapDat->kap_best [tag],
581                          ges_best);
582         fflush (stddbg);
583 #endif
584         ges_best += (long) hKapDat->kap_best [tag];
585      } /* end for */
586 #ifdef DEBUG
587      fprintf (stddbg, "  ges_best : %ld\n", ges_best);
588      fflush (stddbg);
589 #endif
590      hKapDat->max_y = max (hKapDat->max_y, max (ges_best, ges_bed));
591   } /* end for */
592 #ifdef DEBUG
```

```
593    fprintf (stddbg, "----- end dar.-ber.\n");
594    fflush (stddbg);
595 #endif
596 } /* end darstellung_berechnen */
597
598 #undef START
599 #undef END
```

8.2.8 DB_SQL.PC

```
/*

      Datenbank - Modul    Version 1.3

      (c) 1988 by A.Heidemann
                                 */
#include <time.h>
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include <process.h>
#define  NOMINMAX
#include "windows.h"
#include "Kap_Plan.h"
#include "db_sql.dcl"         /* Prozedurkoepfe fuer DB_SQL */

/* Variablenvereinbarung DB-Modul */
static struct tm     beg_time = {0,0,0,1,0,88,0,0,0};

/* Variablenvereinbarung ORACLE */
EXEC SQL BEGIN DECLARE SECTION;
   VARCHAR     sql_bmnr [15];  /* BMNR_SIZE     = 15 */
   VARCHAR     sql_fanr [15];  /* FANR_SIZE     = 15 */
   VARCHAR     sql_teilenr [15];
   VARCHAR     sql_var [15];
   VARCHAR     sql_kanr [15];
   VARCHAR     sql_agbez [30];
   VARCHAR     sql_me [5];
   VARCHAR     sql_bmengestr [10];
   VARCHAR     sql_kalname [10];
   VARCHAR     sql_fstart [16],
               sql_lstart [16],
               sql_fend [16],
               sql_lend [16];
   VARCHAR     sql_uid [20];   /* USERNAME_SIZE = 20 */
   VARCHAR     sql_pwd [20];   /* PASSWORD_SIZE = 20 */
   long int    sql_starttag,
               sql_endtag,
               sql_akttag,
               sql_fabtag,
               sql_guelvon,
               sql_guelbis,
               sql_gbzeit,
               sql_bedarf,
               sql_sysdate;
   int         sql_agstat,
               sql_posnr,
               sql_prio,
               sql_fagnr,
               sql_minprotag,
               sql_bmcount,
               sql_smodnr;
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

EXEC ORACLE OPTION (AREASIZE = 16);

#ifdef DEBUGFILE
   extern FILE *stddbg;
#endif

```

```
 65 BOOL GetUID (uid, pwd)
 66 char  *uid,
 67       *pwd;
 68 {
 69    int   i;
 70
 71    for (i=0;i<sql_uid.len; i++) {
 72       *uid = sql_uid.arr [i];
 73       uid ++;
 74    } /* end for */
 75    *uid = '\0';
 76
 77    for (i=0;i<sql_pwd.len; i++) {
 78       *pwd = sql_pwd.arr [i];
 79       pwd ++;
 80    } /* end for */
 81    *pwd = '\0';
 82
 83    return TRUE;
 84 } /* end GetUID */
 85
 86 BOOL InitBMGList (error)
 87 int * error;
 88 {
 89    EXEC SQL WHENEVER SQLERROR GOTO NOINIT;
 90    EXEC SQL WHENEVER NOT FOUND GOTO NOINIT;
 91    EXEC SQL DECLARE C_BMGR CURSOR FOR
 92       SELECT BMNR FROM BM
 93       WHERE GRPFLG = 1;
 94    EXEC SQL OPEN C_BMGR;
 95    return (TRUE);
 96
 97    NOINIT :
 98       *error = (int) sqlca.sqlcode;
 99       return (FALSE);
100 } /* end InitBMGList */
101
102 BOOL EndBMGList (error)
103 int * error;
104 {
105    EXEC SQL WHENEVER SQLERROR GOTO NOEND;
106    EXEC SQL CLOSE C_BMGR;
107    EXEC SQL COMMIT WORK;
108    return (TRUE);
109
110    NOEND:
111       *error = (int) sqlca.sqlcode;
112       return FALSE;
113 } /* end EndBMGList */
114
115 BOOL NextBMGList (bmnr, error)
116 BMNR  bmnr;
117 int   *error;
118 {
119    EXEC SQL WHENEVER SQLERROR GOTO NONEXT;
120    EXEC SQL WHENEVER NOT FOUND GOTO NONEXT;
121    EXEC SQL FETCH C_BMGR INTO :sql_bmnr;
122    strncpy (bmnr, sql_bmnr.arr, sql_bmnr.len);
123    bmnr [sql_bmnr.len] = '\0';
124    return (TRUE);
125
126    NONEXT:
127       *error = (int) sqlca.sqlcode;
128       return (FALSE);
129 } /* end NextBMGList */
130
```

```
131 BOOL InitFAGList (bmnr, startzeit, endzeit, semode, error)
132 char     bmnr [];
133 long int startzeit,
134          endzeit;
135 int      semode,
136          *error;
137 {
138    strncpy (sql_bmnr.arr, bmnr, BMNR_SIZE);
139    sql_bmnr.len = strlen (bmnr);
140
141    EXEC SQL WHENEVER SQLERROR GOTO NOINIT;
142    EXEC SQL WHENEVER NOT FOUND GOTO NOINIT;
143
144    EXEC SQL DECLARE C_FAG CURSOR FOR
145       SELECT TO_CHAR (FAG.FAGNR),
146              FAG.FANR,
147              FAG.AGSTAT,
148              ((FAG.GBZEIT+30)/60),
149              TO_CHAR (FAG.FSTART, 'J HH24'),
150              TO_CHAR (FAG.LSTART, 'J HH24'),
151              TO_CHAR (FAG.FEND,   'J HH24'),
152              TO_CHAR (FAG.LEND,   'J HH24')
153       FROM FAG, FAGBEL, FA
154       WHERE FAGBEL.BMNR IN (SELECT BMNR
155                             FROM BM
156                             CONNECT BY PRIOR BMNR = BMGRP
157                             START WITH BMNR = :sql_bmnr)
158         AND FAG.FANR  = FAGBEL.FANR
159         AND FAG.FAGNR = FAGBEL.FAGNR
160         AND FAG.AGSTAT BETWEEN 5 AND 50
161         AND FA.FANR   = FAG.FANR
162       ORDER BY FA.PRIO, FAG.AGSTAT DESC
163    ;
164
165    EXEC SQL OPEN C_FAG;
166
167 #ifdef DEBUG
168    fprintf (stddbg, "Ende InitFAGList normal.\n");
169    fflush (stddbg);
170 #endif
171
172    return (TRUE);
173
174    NOINIT :
175       *error = (int) sqlca.sqlcode;
176 #ifdef DEBUG
177       fprintf (stddbg, "Ende InitFAGList' FEHLERHAFT %d.\n", *error);
178       fflush (stddbg);
179 #endif
180       return (FALSE);
181 } /* end INITFAGList */
182
183 BOOL EndFAGList (error)
184 int   *error;
185 {
186    EXEC SQL WHENEVER SQLERROR GOTO NOEND;
187    EXEC SQL CLOSE C_FAG;
188    EXEC SQL COMMIT WORK;
189    return TRUE;
190
191    NOEND :
192       *error = (int) sqlca.sqlcode;
193       return (FALSE);
194 } /* end EndFAGList */
195
196 BOOL NextFAGList (hFag, error)
```

```
197 FAG *hFag;
198 int   *error;
199 {
200    long int    tage,
201                stunden;
202
203 #ifdef DEBUG
204    fprintf (stddbg, "Beginn NextFAGList.\n");
205    fflush (stddbg);
206 #endif
207    EXEC SQL WHENEVER SQLERROR GOTO NONEXT;
208    EXEC SQL WHENEVER NOT FOUND GOTO NONEXT;
209    EXEC SQL FETCH C_FAG INTO :sql_fagnr,
210                              :sql_fanr,
211                              :sql_agstat,
212                              :sql_bedarf,
213                              :sql_fstart,
214                              :sql_lstart,
215                              :sql_fend,
216                              :sql_lend;
217 #ifdef DEBUG
218    fprintf (stddbg, "  Fetch ausgefuehrt.\n");
219    fflush (stddbg);
220 #endif
221
222    hFag->fag_nr = sql_fagnr;
223    strncpy (hFag->fag_fanr, sql_fanr.arr, sql_fanr.len);
224    hFag->fag_fanr [sql_fanr.len] = '\0';
225    sscanf (sql_fstart.arr, "%ld%ld", &tage, &stunden);
226    hFag->fag_fstart = (24L * (tage - sql_starttag)) + stunden ;
227    sscanf (sql_lstart.arr, "%ld%ld", &tage, &stunden);
228    hFag->fag_lstart = (24L * (tage - sql_starttag)) + stunden ;
229    sscanf (sql_fend.arr  , "%ld%ld", &tage, &stunden);
230    hFag->fag_fend   = (24L * (tage - sql_starttag)) + stunden ;
231    sscanf (sql_lend.arr  , "%ld%ld", &tage, &stunden);
232    hFag->fag_lend   = (24L * (tage - sql_starttag)) + stunden ;
233    hFag->fag_bedarf = sql_bedarf;
234    hFag->fag_status = sql_agstat;
235
236 #ifdef DEBUG
237    fprintf (stddbg, "Ende NextFAGList normal.\n");
238    fflush (stddbg);
239 #endif
240
241    return (TRUE);
242
243    NONEXT :
244       *error = (int) sqlca.sqlcode;
245 #ifdef DEBUG
246       fprintf (stddbg, "Ende NextFAGList FEHLER %d\n", *error);
247       fflush (stddbg);
248 #endif
249       return (FALSE);
250 } /* end NextFAGList */
251
252 BOOL InitBestList (bmnr, startzeit, endzeit, error)
253 char     bmnr[];
254 long int startzeit,
255          endzeit;
256 int      *error;
257 {
258 #ifdef DEBUG
259    fprintf (stddbg,
260             "Beginn InitBestList from %ld to %ld on %s\n",
261             startzeit,
262             endzeit,
```

```
263             bmnr);
264    fflush (stddbg);
265 #endif
266
267    EXEC SQL WHENEVER SQLERROR GOTO NOINIT;
268    EXEC SQL WHENEVER NOT FOUND GOTO NOINIT;
269
270    sql_akttag   = sql_starttag + startzeit / 24;
271    sql_endtag   = sql_starttag + endzeit / 24;
272    strcpy (sql_bmnr.arr, bmnr);
273    sql_bmnr.len = strlen (bmnr);
274    sql_guelvon  = 0L;
275    sql_guelbis  = -1L;
276
277    EXEC SQL
278       SELECT   COUNT (BMNR)
279       INTO     :sql_bmcount
280       FROM     BM
281       WHERE    GRPFLG IS NULL
282       CONNECT BY PRIOR BMNR = BMGRP
283       START WITH BMNR = :sql_bmnr
284    ;
285    EXEC SQL
286       SELECT   KALNAME
287       INTO     :sql_kalname
288       FROM     BM
289       WHERE    BMNR = :sql_bmnr
290    ;
291    EXEC SQL DECLARE C_FKT CURSOR FOR
292       SELECT TO_NUMBER(TO_CHAR(DATUM,'J'))
293       FROM  FABKAL
294       WHERE DATUM BETWEEN TO_DATE(TO_CHAR(:sql_akttag),'J')
295                       AND TO_DATE(TO_CHAR(:sql_endtag),'J')
296         AND FABTAG IS NULL
297         AND NAME = :sql_kalname
298       ORDER BY DATUM
299    ;
300    EXEC SQL OPEN C_FKT;
301    EXEC SQL FETCH C_FKT INTO :sql_fabtag;
302 #ifdef DEBUG
303       fprintf (stddbg, "  next fabtag : %ld\n", sql_fabtag);
304       fflush (stddbg);
305 #endif
306
307    EXEC SQL DECLARE C_SMOD CURSOR FOR
308       SELECT   SMODNR,
309                TO_NUMBER(TO_CHAR(VON,'J')),
310                TO_NUMBER(TO_CHAR(BIS,'J'))
311       FROM     BMBETR
312       WHERE    :sql_bmnr = BMNR
313          AND   BIS >= TO_DATE(TO_CHAR(:sql_akttag),'J')
314       ORDER BY VON
315    ;
316    EXEC SQL OPEN C_SMOD;
317
318 #ifdef DEBUG
319    fprintf (stddbg,
320             "Ende InitBestList, bm's: %d, von: %ld, bis: %ld\n",
321             sql_bmcount,
322             sql_akttag,
323             sql_endtag);
324    fflush (stddbg);
325 #endif
326
327    return (TRUE);
328
```

```
329    NOINIT :
330       *error = (int) sqlca.sqlcode;
331 #ifdef DEBUG
332       fprintf (stddbg,
333                "Ende InitBestList, FEHLER : %d\n",
334                *error);
335       fflush (stddbg);
336 #endif
337       return (FALSE);
338 } /* end InitBestList */
339
340 BOOL EndBestList (error)
341 int      *error;
342 {
343 #ifdef DEBUG
344    fprintf (stddbg, "EndBestList ()\n");
345    fflush (stddbg);
346 #endif
347    EXEC SQL WHENEVER SQLERROR GOTO NOEND;
348    EXEC SQL CLOSE C_FKT;
349    EXEC SQL CLOSE C_SMOD;
350    EXEC SQL COMMIT WORK;
351    return TRUE;
352
353    NOEND :
354       *error = (int) sqlca.sqlcode;
355       return (FALSE);
356 } /* end EndBestList */
357
358 BOOL NextBestList (hBestand, error)
359 long int *hBestand;
360 int      *error;
361 {
362    int   minprotag;
363
364 #ifdef DEBUG
365    fprintf (stddbg,
366             "Beginn NextBestList, bmcount = %d, bmnr = '%-15s'\n",
367             sql_bmcount,
368             sql_bmnr.arr);
369    fflush (stddbg);
370 #endif
371
372    EXEC SQL WHENEVER SQLERROR GOTO NONEXT;
373    EXEC SQL WHENEVER NOT FOUND GOTO NONEXT;
374
375    *hBestand = 0L;
376    if (sql_akttag > sql_endtag) {
377       /* alle Daten gelesen */
378       *error = 0;
379       return (FALSE);
380    } /* end if */
381
382    if (sql_akttag > sql_guelbis) {
383       /* Daten des Schichtmodells sind nicht mehr aktuell */
384       EXEC SQL
385          FETCH C_SMOD
386          INTO     :sql_smodnr,
387                   :sql_guelvon,
388                   :sql_guelbis
389       ;
390       EXEC SQL
391          SELECT   SUM (60 * (TO_NUMBER(SUBSTR(ABEND,1,2))
392                              - TO_NUMBER(SUBSTR(ABBEG,1,2)))
393                        + TO_NUMBER(SUBSTR(ABEND,4,2))
394                        - TO_NUMBER(SUBSTR(ABBEG,4,2)))
```

```
395          INTO     :sql_minprotag
396          FROM     SCHICHT
397          WHERE    SMODNR = :sql_smodnr
398       ;
399    } /* end if */
400    if (sql_akttag < sql_guelvon) {
401       /* es existiert kein Schichtmodell */
402       minprotag = 0;
403    } /* end if */
404    else {
405       /* Schichtmodell ist gueltig */
406       minprotag = sql_minprotag;
407    } /* end else */
408
409    if (sql_akttag < sql_fabtag) {
410       *hBestand += ((long) minprotag * (long) sql_bmcount + 59L) / 60L;
411    } /* end if */
412    else {
413       *hBestand = (long) 0;
414       EXEC SQL FETCH C_FKT INTO :sql_fabtag;
415 #ifdef DEBUG
416       fprintf (stddbg, "  next fabtag : %ld\n", sql_fabtag);
417       fflush (stddbg);
418 #endif
419    } /* end else */
420
421    sql_akttag ++;
422
423 #ifdef DEBUG
424    fprintf (stddbg,
425             "Ende NextBestList, bestand %ld\n",
426             *hBestand);
427    fflush (stddbg);
428 #endif
429    return (TRUE);
430
431    NONEXT:
432       *error = (int) sqlca.sqlcode;
433 #ifdef DEBUG
434       fprintf (stddbg,
435                "NextBestList, FEHLER : %d\n",
436                *error);
437       fflush (stddbg);
438 #endif
439       return (FALSE);
440 } /* end NextBestList */
441
442 BOOL Login (name, passwd, hKapDat, error)
443 char     name [],
444          passwd [];
445 KAPDAT   *hKapDat;
446 int      *error;
447 {
448 #ifdef DEBUG
449    fprintf (stddbg, "Login start\n");
450    fflush (stddbg);
451 #endif
452
453    strncpy (sql_uid.arr, name, USERNAME_SIZE);
454    strncpy (sql_pwd.arr, passwd, PASSWORD_SIZE);
455    sql_uid.len = strlen (name);
456    sql_pwd.len = strlen (passwd);
457    EXEC SQL WHENEVER SQLERROR GOTO NOLOGIN;
458    EXEC SQL WHENEVER NOT FOUND GOTO NOLOGIN;
459
460 #ifdef DEBUG
```

```
461    fprintf (stddbg, "User %s identified by %s shall be connected to Database\n",
462             name,
463             passwd);
464    fflush (stddbg);
465 #endif
466
467    EXEC SQL CONNECT :sql_uid IDENTIFIED BY :sql_pwd;
468 #ifdef DEBUG
469    fprintf (stddbg, "  connection ok\n");
470    fflush (stddbg);
471 #endif
472    EXEC SQL
473       SELECT   TO_NUMBER (TO_CHAR (TO_DATE ('01-JAN-88'),'J'))
474       INTO     :sql_starttag
475       FROM     DUAL
476    ;
477    EXEC SQL SELECT TO_NUMBER(TO_CHAR(SYSDATE,'J'))
478                    -TO_NUMBER(TO_CHAR(TO_DATE('01-JAN-88'),'J'))
479             INTO :sql_sysdate FROM DUAL;
480    hKapDat->kd_sysdate = sql_sysdate;
481    return (TRUE);
482
483    /* SQL Fehlerbehandlung */
484    NOLOGIN:
485       *error = (int) sqlca.sqlcode;
486 #ifdef DEBUG
487       fprintf (stddbg, "  connection rejected\n");
488       fflush (stddbg);
489 #endif
490       return (FALSE);
491 } /* end Login */
492
493 BOOL Logout (error)
494 int   *error;
495 {
496 #ifdef DEBUG
497    fprintf (stddbg, "Database is now getting offline\n");
498    fflush (stddbg);
499 #endif
500
501    EXEC SQL WHENEVER SQLERROR GOTO NOLOGOUT;
502    EXEC SQL COMMIT WORK RELEASE;
503 #ifdef DEBUG
504    fprintf (stddbg, "  connection cut normally\n");
505    fflush (stddbg);
506 #endif
507    return (TRUE);
508
509    /* SQL Fehlerbehandlung */
510    NOLOGOUT:
511       *error = (int) sqlca.sqlcode;
512 #ifdef DEBUG
513       fprintf (stddbg, " connection cut with error\n");
514       fflush (stddbg);
515 #endif
516       return (FALSE);
517 } /* end Logout */
518
519 BOOL Rollback (error)
520 int   *error;
521 {
522 #ifdef DEBUG
523    fprintf (stddbg, "changes are now rolled back\n");
524    fflush (stddbg);
525 #endif
526
```

```
527    EXEC SQL WHENEVER SQLERROR GOTO NOROLLBACK;
528    EXEC SQL ROLLBACK WORK ;
529 #ifdef DEBUG
530    fprintf (stddbg, "  rollback o.k.\n");
531    fflush (stddbg);
532 #endif
533    return (TRUE);
534
535    /* SQL Fehlerbehandlung */
536    NOROLLBACK:
537       *error = (int) sqlca.sqlcode;
538 #ifdef DEBUG
539       fprintf (stddbg, "  rollback not o.k.\n");
540       fflush (stddbg);
541 #endif
542       return (FALSE);
543 } /* end Rollback */
544
545 BOOL Commit (error)
546 int   *error;
547 {
548 #ifdef DEBUG
549    fprintf (stddbg, "changes are now committed\n");
550    fflush (stddbg);
551 #endif
552
553    EXEC SQL WHENEVER SQLERROR GOTO NOCOMMIT;
554    EXEC SQL COMMIT WORK ;
555 #ifdef DEBUG
556    fprintf (stddbg, "  commit o.k.\n");
557    fflush (stddbg);
558 #endif
559    return (TRUE);
560
561    /* SQL Fehlerbehandlung */
562    NOCOMMIT:
563       *error = (int) sqlca.sqlcode;
564 #ifdef DEBUG
565       fprintf (stddbg, "  commit not o.k.\n");
566       fflush (stddbg);
567 #endif
568       return (FALSE);
569 } /* end Commit */
570
571 BOOL FADaten (fad, error)
572 FA_DATEN *fad;
573 int      *error;
574 {
575 #ifdef DEBUG
576    fprintf (stddbg, "begin FADaten %s/%d\n",fad->fad_fanr, fad->fad_fagnr);
577    fflush (stddbg);
578 #endif
579
580    sql_fagnr = fad->fad_fagnr;
581    strncpy (sql_fanr.arr, fad->fad_fanr, FANR_SIZE);
582    sql_fanr.len = strlen (fad->fad_fanr);
583
584    EXEC SQL WHENEVER SQLERROR GOTO NODATEN;
585    EXEC SQL WHENEVER NOT FOUND GOTO NODATEN;
586
587 #ifdef DEBUG
588    fprintf (stddbg, "  declare cursor C_DAT\n");
589    fflush (stddbg);
590 #endif
591
592    EXEC SQL DECLARE C_DAT CURSOR FOR
```

```
593       SELECT      FAG.AGBEZ,
594                   FAG.AGSTAT,
595                   FAG.GBZEIT,
596                   FKABEZ.KANR,
597                   FKABEZ.POSNR,
598                   FA.TEILENR,
599                   FA.VAR,
600                   FA.ME,
601                   FA.PRIO,
602                   TO_CHAR (FA.BMENGE)
603       FROM        FAG,
604                   FKABEZ,
605                   FA
606       WHERE       FAG.FANR   = :sql_fanr
607          AND      FAG.FAGNR  = :sql_fagnr
608          AND      FKABEZ.FANR = FAG.FANR
609          AND      FA.FANR    = FAG.FANR
610    ;
611 #ifdef DEBUG
612    fprintf (stddbg, "  open cursor C_DAT\n");
613    fflush (stddbg);
614 #endif
615
616    EXEC SQL OPEN C_DAT;
617
618 #ifdef DEBUG
619    fprintf (stddbg, "  fetch from cursor C_DAT\n");
620    fflush (stddbg);
621 #endif
622
623    EXEC SQL FETCH C_DAT
624       INTO        :sql_agbez,
625                   :sql_agstat,
626                   :sql_gbzeit,
627                   :sql_kanr,
628                   :sql_posnr,
629                   :sql_teilenr,
630                   :sql_var,
631                   :sql_me,
632                   :sql_prio,
633                   :sql_bmengestr
634                   ;
635
636    EXEC SQL CLOSE C_DAT;
637    EXEC SQL COMMIT WORK;
638
639    strncpy (fad->fad_tnr, sql_teilenr.arr, sql_teilenr.len);
640    fad->fad_tnr [sql_teilenr.len] = '\0';
641    strncpy (fad->fad_tvr, sql_var.arr, sql_var.len);
642    fad->fad_tvr [sql_var.len] = '\0';
643    strncpy (fad->fad_me, sql_me.arr , sql_me.len);
644    fad->fad_me [sql_me.len] = '\0';
645    fad->fad_prio = (short) sql_prio;
646    sql_bmengestr.arr[sql_bmengestr.len] = '\0';
647    sscanf (sql_bmengestr.arr, "%lf", &(fad->fad_bmenge));
648    fad->fad_agstat = sql_agstat;
649    fad->fad_gbzeit = sql_gbzeit;
650    strncpy (fad->fad_agbez, sql_agbez.arr, sql_agbez.len);
651    fad->fad_agbez [sql_agbez.len] = '\0';
652
653    strncpy (fad->fad_kanr, sql_kanr.arr, sql_kanr.len);
654    fad->fad_kanr [sql_kanr.len] = '\0';
655    fad->fad_kapos = sql_posnr;
656
657 #ifdef DEBUG
658    fprintf (stddbg, "normales ende FADaten %s/%d ORA%d\n",
```

```
659             fad->fad_fanr,
660             fad->fad_fagnr,
661             (int) sqlca.sqlcode);
662    fflush (stddbg);
663 #endif
664
665    return (TRUE);
666
667    NODATEN:
668       if ((int) sqlca.sqlcode == -1003) {
669          /* no statement parsed */
670 #ifdef DEBUG
671          fprintf (stddbg, "  ORA-1003  Fehler übergangen\n");
672          fprintf (stddbg, "normales ende FADaten %s/%d ORA%d\n",
673                   fad->fad_fanr,
674                   fad->fad_fagnr,
675                   (int) sqlca.sqlcode);
676          fflush (stddbg);
677 #endif
678          return (TRUE);               /* Fehler wird übergangen */
679       }
680       *error = (int) sqlca.sqlcode;
681 #ifdef DEBUG
682       fprintf (stddbg, "fehlerhaftes ende FADaten %s/%d ORA%d\n",
683                fad->fad_fanr,
684                fad->fad_fagnr,
685                (int) sqlca.sqlcode);
686       fflush (stddbg);
687 #endif
688       return (FALSE);
689 }  /* end Daten */
690
691 BOOL datetostd (std, tt, mm, jj, ss, min, err)
692 long  *std;
693 int   tt,
694       mm,
695       jj,
696       ss,
697       min,
698       *err;
699 {
700    struct tm   datum;
701    time_t      zeit;
702
703 #ifdef   DEBUG_DATE
704    fprintf (stddbg, "-> DATETOSTD (std, %d, %d, %d, %d, %d, err)\n",
705             tt, mm, jj, ss, min);
706    fflush (stddbg);
707 #endif
708    *err = 0;
709    datum.tm_mday = tt;
710    datum.tm_mon = mm - 1;
711    datum.tm_year = jj;
712    datum.tm_hour = ss;
713    datum.tm_min = min;
714    datum.tm_sec = 0;
715    datum.tm_isdst = 0;
716
717    zeit = mktime (&datum);
718    if (zeit == (time_t) -1) {
719       *err = WRONG_DATE;
720 #ifdef   DEBUG_DATE
721       fprintf (stddbg, "DATETOSTD -> (std, %d, %d, %d, %d, %d, %d)\n",
722                tt, mm, jj, ss, min, *err);
723       fflush (stddbg);
724 #endif
```

```
725       return FALSE;
726    } /* end if */
727
728    *std = (zeit - mktime (&beg_time)) / 3600L ;
729
730 #ifdef   DEBUG_DATE
731    fprintf (stddbg, "DATETOSTD -> (%ld, %d, %d, %d, %d, %d, err)\n",
732             *std, tt, mm, jj, ss, min);
733    fflush (stddbg);
734 #endif
735    return TRUE;
736 } /* end datetostd */
737
738 BOOL ErrorText (error, errortxt)
739 int      error;
740 char     errortxt [];
741 {
742    if (error == (int) sqlca.sqlcode) {
743       strcpy (errortxt, (char *) sqlca.sqlerrm.sqlerrmc+10);
744       return (TRUE);
745    }
746    else {
747       strcpy (errortxt, "unbekannter Fehler");
748       return (FALSE);
749    } /* end else */
750 } /* end ErrorText */
751
752 BOOL stdtodate (std, tt, mm, jj, ss, min, err)
753 long  std;
754 int   *tt,
755       *mm,
756       *jj,
757       *ss,
758       *min,
759       *err;
760 {
761    struct tm   *datum;
762    time_t      zeit;
763
764 #ifdef   DEBUG_DATE
765    fprintf (stddbg, "-> STDTODATE (%ld, tt, mm, jj, ss, min, err)\n", std);
766    fflush (stddbg);
767 #endif
768    *err = 0;
769    zeit = std * 3600L + mktime (&beg_time);
770    datum = localtime (&zeit);
771    if (datum == (struct tm *) NULL) {
772       *err = WRONG_DATE;
773 #ifdef   DEBUG_DATE
774       fprintf (stddbg, "STDTODATE -> (%ld, tt, mm, jj, ss, min, %d)\n",
775                        std, *err);
776       fflush (stddbg);
777 #endif
778       return FALSE;
779    } /* end if */
780
781    *tt = datum->tm_mday;
782    *mm = datum->tm_mon + 1;
783    *jj = datum->tm_year;
784    *ss = datum->tm_hour;
785    *min = datum->tm_min;
786
787 #ifdef   DEBUG_DATE
788    fprintf (stddbg, "STDTODATE -> (%ld, %d, %d, %d, %d, %d, err)\n",
789             std, *tt, *mm, *jj, *ss, *min);
790    fflush (stddbg);
```

```
791 #endif
792
793    return TRUE;
794 } /* end stdtodate */
795
796 BOOL stdtowoche (std, woche, err)
797 long  std;
798 int   *woche,
799       *err;
800 {
801    struct tm   *datum;
802    time_t      zeit;
803
804 #ifdef   DEBUG_FEIN
805    fprintf (stddbg, "-> STDTOWOCHE (%ld, woche, err)\n", std);
806    fflush (stddbg);
807 #endif
808    *err = 0;
809    zeit = std * 3600L + mktime (&beg_time);
810    datum = localtime (&zeit);
811    if (datum == (struct tm *) NULL) {
812       *err = WRONG_DATE;
813 #ifdef   DEBUG_FEIN
814       fprintf (stddbg, "STDTOWOCHE -> (%ld, woche, %d)\n", std, *err);
815       fflush (stddbg);
816 #endif
817       return FALSE;
818    } /* end if */
819
820    *woche = ((datum->tm_yday - datum->tm_wday) / 7) + 1;
821
822 #ifdef   DEBUG_FEIN
823    fprintf (stddbg, "STDTOWOCHE -> (%ld, %d, err)\n", std, *woche);
824    fflush (stddbg);
825 #endif
826
827    return TRUE;
828 } /* end stdtowoche */
```

8.3 Literaturverzeichnis

Balzert, Helmut : Die Entwicklung von Software - Systemen, Bibliographisches Institut; Mannheim-Wien-Zürich ; 1982

Duncan, Ray : Mit Microsoft Windows /386 in die Zukunft in Microsoft System Journal, Jan/Febr. 1988, Microsoft GmbH, pp. 5 - 12

Gancarz, M. : UWM:a user interface for X-Windows in Summer Conference Proceedings, Atlanta, Ga. June 10-13; USENIX Association, 1986, pp. 429 - 440

Hackstein, R. : Produktionsplanung und -steuerung (PPS): ein Handbuch für die Betriebspraxis Düsseldorf 1984

Kurbel, Karl : Software Engineering im Produktionsbereich, Wiesbaden 1983

Lantz, K.A., Nowicki, W.I. : Structured Graphics for distributed systems, in ACM Transactions on Graphics, Vol. 3, No. 1, Jan. 1984, pp. 23 - 51

Lipkie, D.E., Evans, S.R., Newlin, J.K. Weissman, R.L.: Star Graphics: An object-oriented implementation in Comput. Graph. Vol 16, No. 3, July 1982, pp. 115 - 124

MEF Environmental : No Limit Version 3.0 Documentation, MEF Environmental Inc., Austin, Texas, 1986

Microsoft Corporation : Microsoft Windows : Programmer's Guide, Microsoft Corporation, Redmont, Washington, 1985

Nye, Adrian : Xlib Programming Manual for Version 11,
Newton, Massachussetts, 1988

Scheifler, Robert W., Gettys, Jim : The X Window System
in ACM Transactions on Graphics, Vol. 5, No. 2,
April 1986, pp. 79 - 109

Stern, Hal L. : Comparison of Window Systems
in Byte, Nov 1987, pp. 265 - 272