〔非公開〕

ＴＲ－Ｃ－００５４

知的電話機設計ドキュメント

林　潔　KIYOSHI HAYASHI
島　健一　KENICHI SHIMA
佐藤　隆　TAKASHI SATO
横田　政憲　MASANORI YOKOTA
内田　修市　SHUICHI UCHIDA
芝本　尚樹　NAOKI SHIBAMOTO
中島　俊介　SHUNSUKE NAKAJIMA

１９９０．８．８

ＡＴＲ通信システム研究所

Intelliphone™ ソフトウェア要求仕様書

（第０．２版）

１９８７年　７月　１日

１．概要

２．システム構成

2．1　スピーカ部，　2．2　マイク部，

2．3　フックスイッチ部，　2．4　記憶部，

2．5　符号化／復号化部，　2．6　制御部

３．機能

3．1　名前によるダイヤリング，　3．2　再ダイヤル，

3．3　番号案内，　3．4　発信者・着信者の識別，

3．5　話中処理

４．プロトコル

4．1　正常通話，　4．2　話中

５．データ定義

5．1　記憶部データ，　5．2　プロトコルメッセージデータ

６．ハードウェアインタフェース

変更履歴

1987.06.15　第０．０版　リリース

1987.06.15　図２．１　システム構成図　フックスイッチ部追加
　　マイク部－符号化／復号化部間スイッチ追加

1987.06.15　２．２　マイク部　符号化／復号化部との接続スイッチの記述追加

1987.06.15　第２章　「２．３　フックスイッチ部」追加

1987.06.15　２．６　制御部　フックスイッチ部記述追加

1987.06.15　３．２　再ダイヤル　通話相手確認機能追加

1987.06.15　３．３　番号案内機能追加　・選択条件の再指定
　　・案内メッセージ付きの全候補紹介

1987.06.15　第３章　利用例追加

1987.06.15　第４章　プロトコル図　形式変更

1987.06.15　５．１　記憶部データ　項目追加

1987.06.15　５．２　プロトコルメッセージデータ　データ種別追加

1987.06.16　第０．１版　リリース

1987.06.18　図４．１　正常通信プロトコル図
　　確認メッセージ，呼出応答メッセージ　追加

1987.06.18　図４．２　話中プロトコル図　確認メッセージ追加

1987.06.18　第３章　「３．６　オンフックリセット」追加

1987.06.18　３．１　名前によるダイヤリング
　　・名前が登録されてないときのメッセージ　追加
　　・名前がユニークに決まらないときのメッセージ　削除
　　・名前の重複に関する記述　追加
　　・ダイヤル先確認メッセージ　追加

1987.06.18　３．３　番号案内　利用手順の「全部案内する場合」及び利用例３
　　確認メッセージ追加

1987.06.18　５．１　記憶部データ　データ内容一部変更（検索は全て属性をキーとする）
　　データに関する規定　追加

1987.07.01　第０．２版　リリース

第１章　概要

Ｉｎｔｅｌｌｉｐｈｏｎｅ（仮称）は，①番号をダイヤルせずに電話を簡単にかけることができ，②留守にしている場合も確実に電話をつないでくれて，しかも③その多機能さにもかかわらず操作する部分が少なく使いやすい，という特徴を持った知的電話機である．

第０．０版は，これらのうち①の一部分だけを実現するプロトタイプである．この中に含まれる利用者サービス機能は，音声によるダイヤリング・再ダイヤル・番号案内がある．

第２章　システム構成

システム全体の構成は図２．１のようになっている．

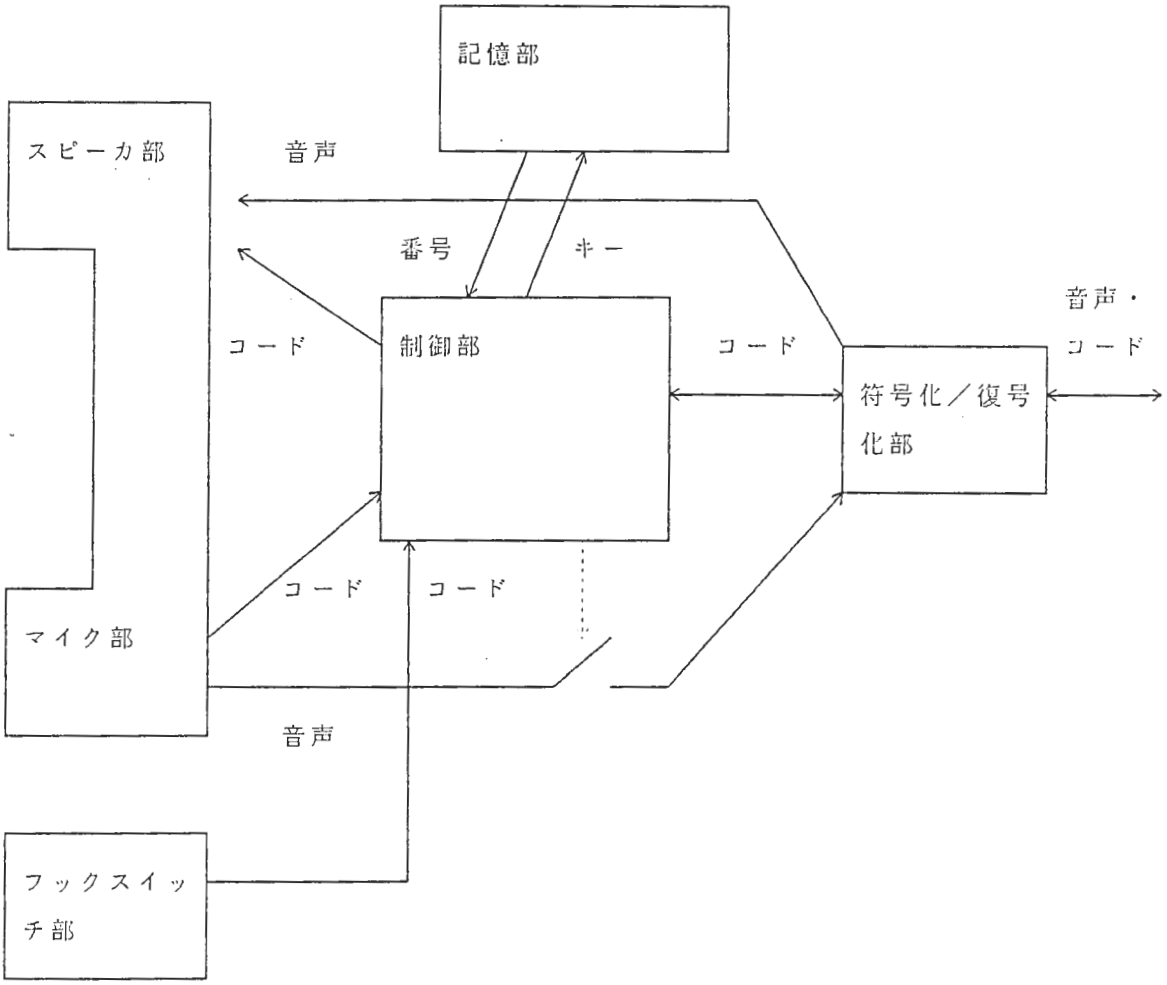


図２．１　システム構成

２．１　スピーカ部

送られてきた音声信号はそのままスピーカから流す．送られてきたコード信号は音声信

号に変換してスピーカから流す．

２．２　マイク部

マイクで話された声をそのまま音声信号として符号化／復号化部に送り，またコード信号に変換して制御部にも送る．但し，マイク部と符号化／復号化部との接続は，制御部の制御によりスイッチが開閉される．

２．３　フックスイッチ部

フックスイッチの状態（ＯｎまたはＯｆｆ）が遷移した時に，制御部に対してコードを送る．また，このフックスイッチによって，電話機と交換機との接続が開閉される．

２．４　記憶部

通話相手を指定するキーを受け取り，該当する電話番号を返す．

２．５　符号化／復号化部

マイク部からの音声信号と制御部からのコード信号を混合して送り出す．また逆に，送られて来る音声とコードの混合信号を音声信号とコード信号に分離して，それぞれスピーカ部・制御部に渡す．

２．６　制御部

上記のスピーカ部・マイク部・フックスイッチ部・記憶部・符号化／復号化部を制御し，知的サービスを利用者に提供する．

第３章　機能

３．１　名前によるダイヤリング

（機能内容）

ｉ）通話相手の名前をマイクに向かって言うと，自動的にダイヤリングを行う．

ii）指名した名前が記憶部に登録されてなくてみつからない場合は，スピーカから「該当する所がありません．」と答える．

（方法）

i）あらかじめ記憶部に名前と電話番号が登録されている．

ii）記憶部は連想記憶であり，キーにより電話番号を引き出す．

iii）名前は個人名，団体名，またはその２つの組合せを使える．

iv）番号記憶は個人用の記憶と全ての人に共通の記憶がある．

v）重複した名前は登録されてないものとする．

（利用手順）

ｉ）オフフックする．
ii）電話機「どこにおかけになりますか？」
iii）利用者「

（個人名）　さん．
（団体名）．
（団体名）　の　（個人名）　さん．

」
iv）電話機「○○ですね．少しお待ち下さい．」
v）電話機が自動的にダイヤルする．

（利用例）
ｉ）オフフックする．
ii）電話機「どこにおかけになりますか？」
iii）利用者「三洋電機の横田さん．」
iv）電話機「三洋電機の横田さんですね．少しお待ち下さい．」
v）電話機が自動的にダイヤルする．

3．2　再ダイヤル

（機能内容）
ｉ）直前にかけた電話番号を自動的にダイヤルする．
ii）各個人が直前にかけた電話番号をダイヤルすることができる．
iii）再ダイヤルする相手の名前を利用者に確認する．
iv）相手が違っている場合は，利用者がオンフックする．

（方法）
ｉ）発信時に発信者・相手電話番号を最新履歴として記憶部に記録し，再ダイヤル時にその履歴を参照する．
ii）発信時に話中等で通話が成立しなかった場合にも履歴を残す．

（利用手順）
ｉ）オフフックする．
ii）電話機「どこにおかけになりますか？」
iii）利用者「さっきの所．」
iv）電話機「

（個人名）　さん （団体名） （団体名）　の　（個人名）　さん	ですね．少しお待ち下さい．」

v）電話機が自動的にダイヤルする．

（利用例　1）

i）オフフックする．

ii）電話機「どこにおかけになりますか？」

iii）利用者「三洋電機の横田さん．」

iv）電話機が自動的にダイヤルする．

v）電話機「お話中です．」

vi）オンフックする．

⋮

⋮

vii）オフフックする．

viii）電話機「どこにおかけになりますか？」

ix）利用者「さっきの所．」

x）電話機「三洋電機の横田さんですね．少しお待ち下さい．」

xi）電話機が自動的にダイヤルする．

（利用例　2）

i）～vi）　利用例1と同じ

⋮

⋮

vii）オフフックする．

viii）電話機「どこにおかけになりますか？」

ix）利用者「さっきの所．」

x）電話機「三洋電機の横田さんですね．少しお待ち下さい．」

（利用者がかけたいところが横田さんではなかった場合）

xi）　オンフックする．

3．3　番号案内

（機能内容）

i）通話相手の名前を指定しなくても，通話相手の条件を言うと，その条件に適う相手に対して自動的にダイヤルする．

ii）条件に適う通話相手が記憶部でみつからない場合，電話機は「該当する所はありません．」と答えて，最初から通話相手の指定をやりなおす．

iii）条件に適う通話相手が1個だけみつかった場合，電話機は「○○に電話します．少しお待ち下さい．」と利用者に確認して，自動的にダイヤルする．

iv）通話相手が複数個みつかった場合，通話相手の候補を絞るために，さらに利用者に条件を聞く．

v）利用者が条件を全部言い切っても複数個の候補が残っている場合，それらを全部案内付きで紹介する．

（方法）

ｉ）記憶部の番号記憶の中に，指定され得る条件を各電話番号の属性として，案内内容を案内用メッセージとして，持っている．

ii）指定された条件にあう属性を持つ電話番号を探し，ユニークに決まるとダイヤルする．

iii）利用者に固有の番号記憶と電話機に固有の番号記憶がある．

（利用手順）

ｉ）オフフックする．

ii）電話機「どこにおかけになりますか？」

iii）利用者「（属性）．」

（通話相手がみつからない場合）

iv）電話機「該当する所はありません．」

ii）に戻る．

（通話相手が１個だけみつかった場合）

iv）電話機「○○に電話します．少しお待ち下さい．」

v）電話機が自動的にダイヤルする．

（通話相手が１個だけみつかったが，利用者が気にいらなかった場合）

iv）電話機「○○に電話します．少しお待ち下さい．」

v）オンフックする．

（通話相手が複数みつかり，利用者がさらに条件を言う場合）

iv）電話機「他に条件はありますか？」

iii）に戻る．

（通話相手が複数みつかり，それらを全部案内する場合）

iv）電話機「他に条件はありますか？」

v）利用者「ありません．」

vi）電話機「ご希望の条件にあう所はｎ個あります．

（名称１）　（案内用メッセージ１）

（名称２）　（案内用メッセージ２）

……

（名称ｎ）　（案内用メッセージｎ）

どこにおかけになりますか？」

vii）利用者「（名称ｉ）．」　　（１≦ｉ≦ｎ）

viii）電話機「（名称ｉ）ですね．少しお待ち下さい．」

ix）電話機が自動的にダイヤルする．

（利用例　1）

i）オフフックする．

ii）電話機「どこにおかけになりますか？」

iii）利用者「ＤＤハウスの寿司屋．」

iv）電話機「すしバー「遊」に電話します．」

v）電話機が自動的にダイヤルする．

（利用例　2）

i）オフフックする．

ii）電話機「どこにおかけになりますか？」

iii）利用者「ツイン２１の食べ物屋．」

iv）電話機「他に条件はありますか？」

v）利用者「安くて旨い．」

vi）電話機「該当する所にありません．」

vii）電話機「どこにおかけになりますか？」

⋮

⋮

（利用例　3）

i）オフフックする．

ii）電話機「どこにおかけになりますか？」

iii）利用者「ＤＤハウスの中の店．」

iv）電話機「他に条件はありますか？」

v）利用者「ありません．」

vi）電話機「ご希望の条件にあう所は１０個あります．

あいらんど　：　火山の噴火でお馴染みの……

マハラジャ　：　当店にふさわしい服装で……

……

どこにおかけになりますか？」

vii）利用者「あいらんど．」

viii）電話機「あいらんどですね．少しお待ち下さい．」

ix）電話機が自動的にダイヤルする．

３．４　発信者・着信者の識別

（機能内容）

i）記憶部の中には，利用者個人別に電話番号を記憶している部分がある．発信時に誰のための個人用電話番号記憶を使うかを決めるために発信者を識別する．

ii）電話機が「○○さんに電話です．」，「△△さんから電話です．」ということを利用者に告げるために，発信者・着信者を識別する．

（方法）

i）発信者はソフトウェアの外で決定され，ソフトウェアに対してはパラメタとして与えられる．

ii）着信者は，発信時の指名に使われた名前で，記憶部においてユニークに特定できたものを使う．

3．5　話中処理

（機能内容）

i）電話をかけた相手が話中だった場合は，発信側電話機は発信者に「お話し中です．」と告げる．

（方法）

i）着信時，既にその電話機が話中ならば，発信側に“話中”のコードを返す．

ii）発信時，“話中”のコードが返ってくれば，スピーカから「お話中です．」と言う．

3．6　オンフックリセット

（機能内容）

i）通話中であろうとなかろうと，オンフックによりプログラムは初期状態にリセットされる．また，もし通話中ならばその通話は終了する．

（方法）

i）オンフックされたというメッセージがフックスイッチ部から制御部にくると，制御部は全ての変数・オブジェクト等を初期化する．

ii）通話中の回線はフックスイッチによりハードウェア的に切断される．

第4章　プロトコル

この章では，正常通話の場合と話中の場合の必須プロトコルを記述する．第0．0版では基本的にこの2つの処理シーケンスだけを実現すればよい．

4．1　正常通話

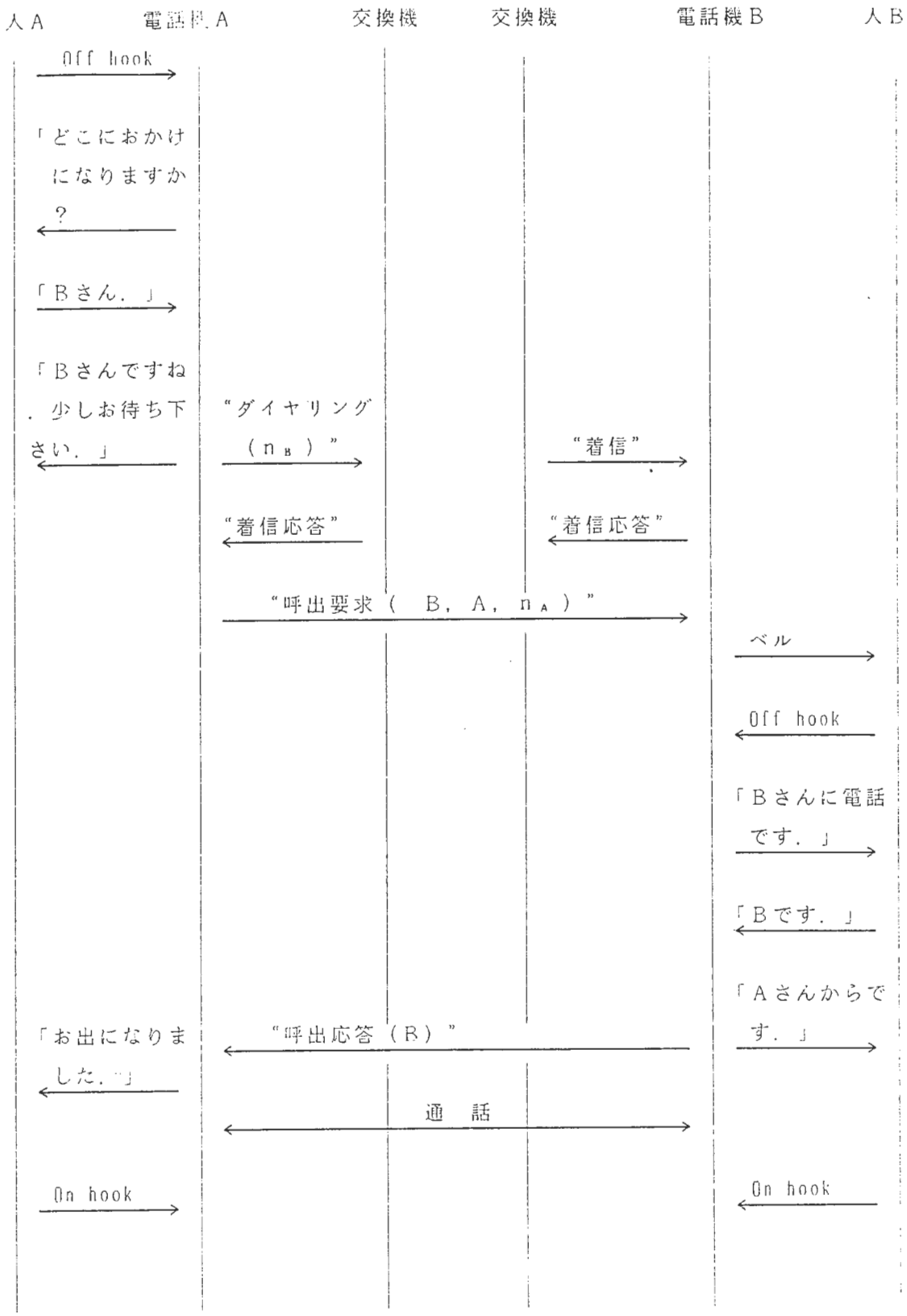


図4．1　正常通話プロトコル

（「……」は音声，“……”はコードを表す，

n_A：電話機Aの番号，n_B：電話機Bの番号）

4．2　話中

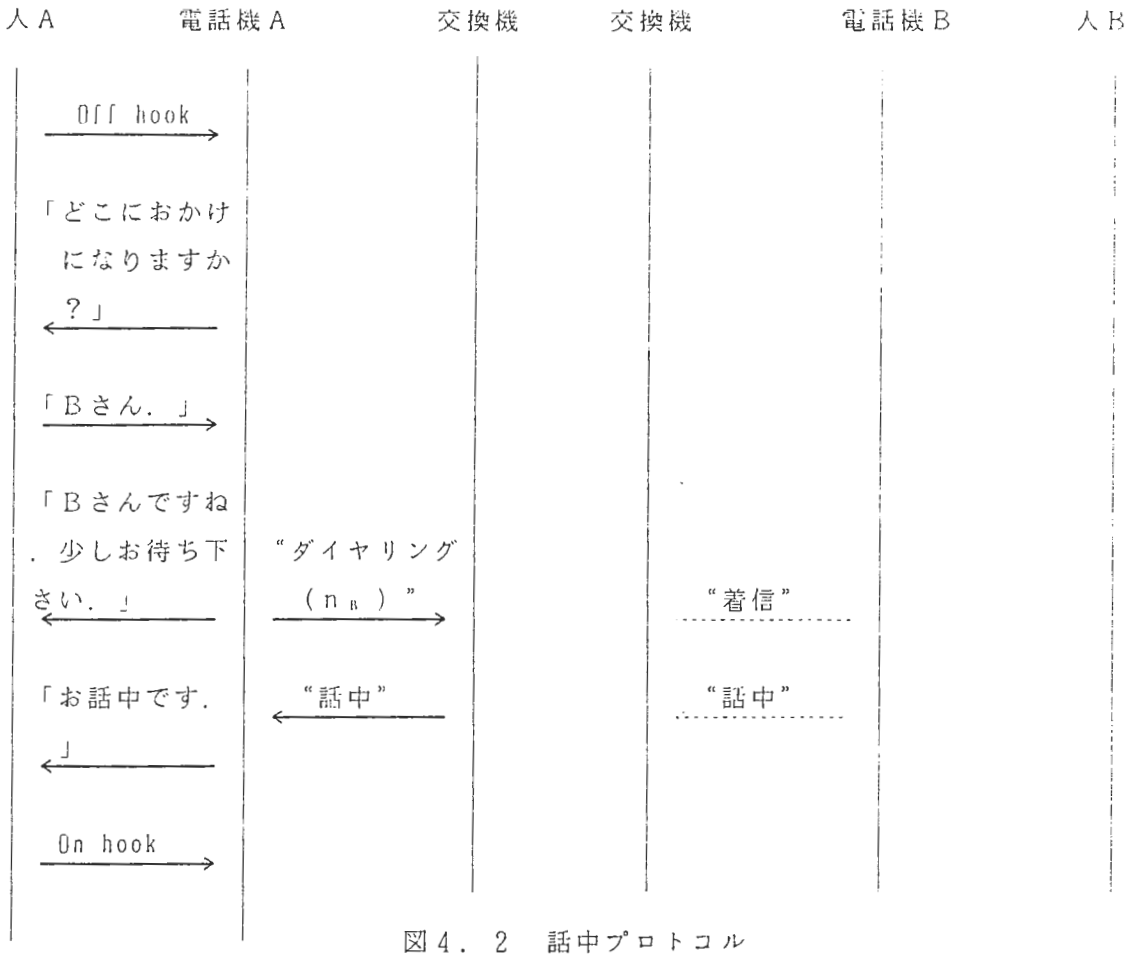


図4．2　話中プロトコル

第5章　データ定義

この章では，記憶部上及びプロトコルメッセージ上の必須データを記述する．

5．1　記憶部データ

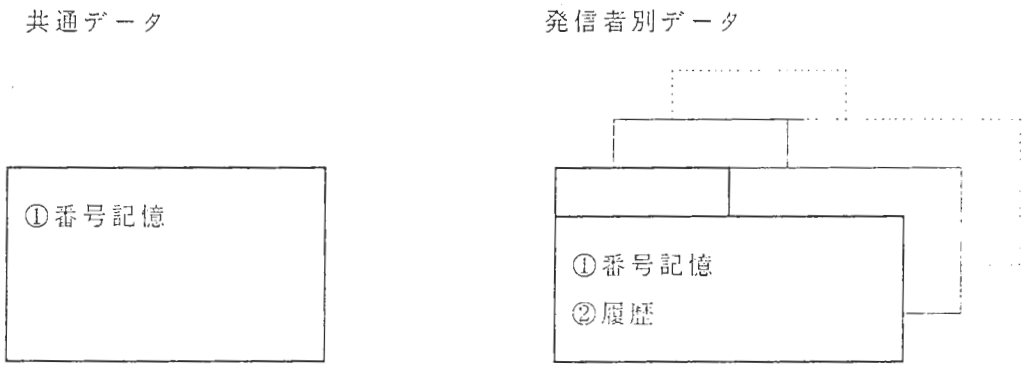


図5．1　記憶部データ

①番号記憶

電話をかける相手の名前と電話番号を記憶する．共通データと発信者別データがある．

（必須データ項目）

個人／団体別	名　　称	電話番号	属性（複数値）	案内用メッセージ

（データに関する規定）

i）「属性」の欄には，必ず自分の「名称」を値として含む．

ii）その「名称」は記憶部の番号記憶内において「属性」としてユニークでなければならない．即ち，ある「名称」をキーとして「属性」を検索した場合，その名称を含むレコードが１個だけ求まる．

（データ例）

個人	中島　俊介	06-123-4567	（中島　俊介）	－
個人	芝本　尚樹	06-987-6543	（芝本　尚樹）	－
団体	三洋電機	06-000-0000	（三洋電機）	－
団体	ＡＴＲ	06-949-1825	（ＡＴＲ）	－
団体	ユニバック	06-999-9999	（ユニバック）	－
団体	呑喜帆亭	06-351-8638	（呑喜帆亭	「待て！この味わい……」
			お好み焼き	
			鉄板焼き	
			片町の食べ物屋）	

②履歴

各発信者が最後にダイヤルした相手を記憶する．常に最新の状態に更新される．

（必須データ項目）

名　　称	電話番号

（データ例）

横田　政憲　　06-000-0000

5．2　プロトコルメッセージデータ

① ダイヤリングメッセージ

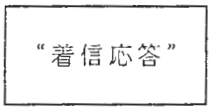
（必須データ項目）

② 着信メッセージ

（必須データ項目）

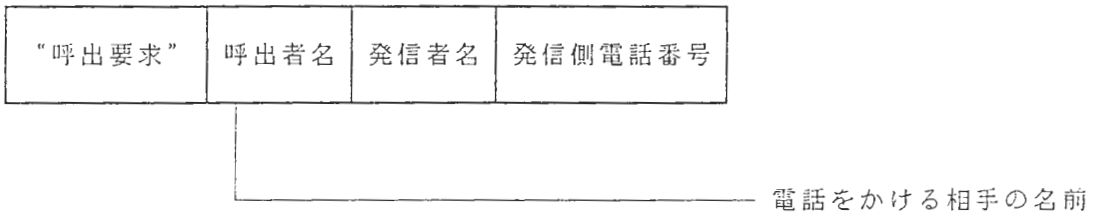
③ 着信応答メッセージ

（必須データ項目）



④ 呼出要求メッセージ

（必須データ項目）



⑤ 呼出応答メッセージ

（必須データ項目）

⑥　話中メッセージ

（必須データ項目）

“話中”

第６章　ハードウェアインタフェース

制御部のソフトウェアはコード化されたデータのみを扱う．ハードウェアとのやりとりもコードだけを用いて行う．

スピーカ部，マイク部，符号化／復号化部，記憶部に対するコードの入出力は，入出力すべきコードをそのままパラメタとして書けるＬｉｓｐ関数を用いる．

Intelliphone™ ソフトウェア設計仕様書

（第　1.2　版）

1987年 6 月26日

目　　次

1.　まえがき

2.　状態遷移図

3.　データ定義

4.　機能オブジェクト構成図
　　及び　関係図

5.　オブジェクト機能定義

変　更　履　歴

第1.0版　昭和62年6月23日　作成
第1.1版　昭和62年6月24日　改定
第1.2版　昭和62年6月26日　改定

〔1.0版から1.1版への主な変更箇所〕

- ”機能ユニット”から”機能オブジェクト”へ名称変更
- オブジェクト間の階層構成を追加定義
- 外部からのメッセージを一括して受ける”知的電話機制御”オブジェクトの定義
- 共通データの”着信者名”を削除、これに伴って名称等一部変更
- データ及びオブジェクトに英語名付与
- 一部メソッド名およびパラメータの変更

〔1.1版から1.2版への主な変更箇所〕

- 状態遷移図において電話機使用中の着信イベント受信時の処理ルートに間違いがあったための修正（要求仕様書と不一致であった）

1. まえがき

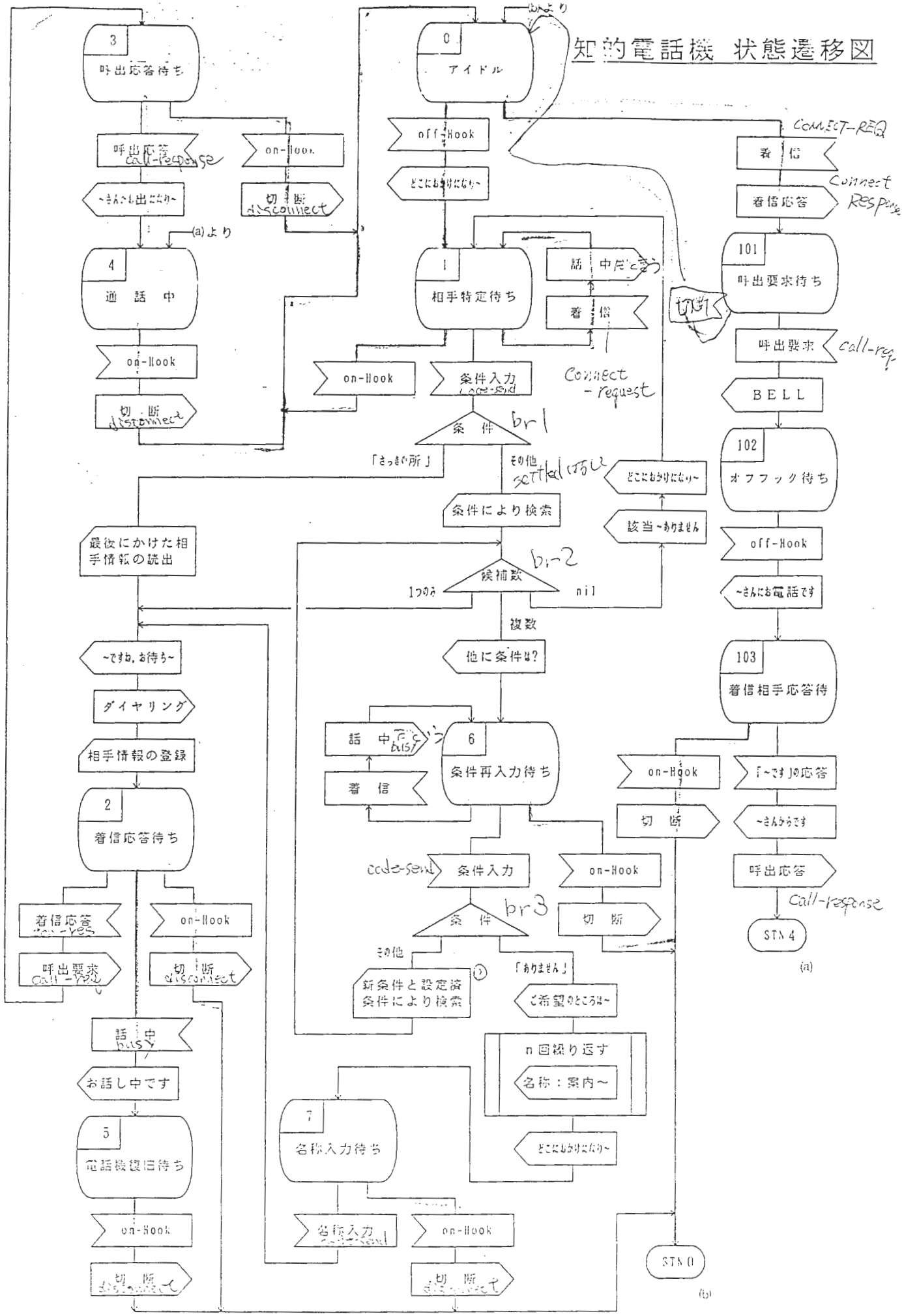
本設計仕様書は Intelliphone ™ の制御部のソフトウェアをオブジェクト指向で記述する事を前提として書かれています。

また、実際にソフトウェアを製造する際には、要求仕様書と合わせてご利用になられることをお勧めします。

基本的に状態遷移図があればソフトウェアはインプリメント可能と考えらるので、詳細な設計条件はインプリメンタデザインとします。

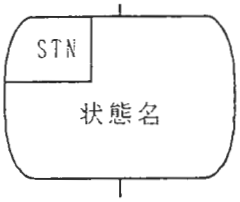
2. 状態遷移図

知的電話機 状態遷移図


0
アイドル
(b)より
off-Hook
どこにおかけになり~
1
相手特定待ち
話 中だってさ
着 信
on-Hook
切 断
disconnect
条件入力
code-send
条件
br1
「さっきの所」
その他
settled はい
どこにおかけになり~
該当~ありません
条件により検索
候補数
br2
1つのみ
nil
複数
他に条件は?
最後にかけた相手情報の読出
~ですね, お待ち~
ダイヤリング
相手情報の登録
2
着信応答待ち
着信応答
call-response
呼出要求
call-send
on-Hook
切 断
disconnect
話 中
busy
お話し中です
5
電話機復旧待ち
on-Hook
切 断
disconnect
話 中だってさ
busy
着 信
6
条件再入力待ち
code-send
条件入力
条件
br3
on-Hook
切 断
その他
新条件と設定済条件により検索
「ありません」
ご希望のところは~
n回繰り返す
名称:案内~
どこにおかけになり~
7
名称入力待ち
名称入力
code-send
on-Hook
切 断
disconnect
STN 0
(b)
CONNECT-REQ
着 信
Connect
RESPONSE
着信応答
101
呼出要求待ち
切断
呼出要求
call-req
BELL
102
オフフック待ち
off-Hook
~さんにお電話です
103
着信相手応答待
on-Hook
切 断
「~です」の応答
~さんからです
呼出応答
Call-response
STN 4
(a)
(a)より
3
呼出応答待ち
呼出応答
call-response
~さんから出になり~
on-Hook
切 断
disconnect
4
通 話 中
on-Hook
切 断
disconnect
Connect
-request

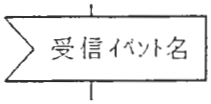
表記法の説明

〔記　号〕　　〔意　味〕

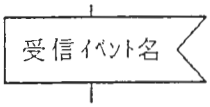


電話機の安定状態（イベントを待っている状態を意味する）

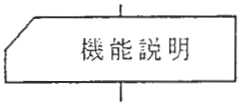
注）STN : STatus Number （状態番号）



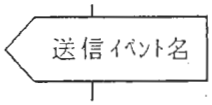
人間側(Hook-Switch 又は Mic)からのイベントの受信



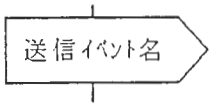
交換機側(Code-Detector)からのイベントの受信



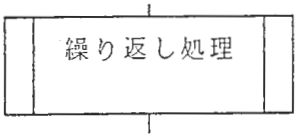
記憶部(Memory)への操作



人間側(Speaker)へのメッセージ送出



交換機側(Code-Generator)へのメッセージ送出



繰り返し処理を定義する

3. データ定義

≪　共通データ　≫

No.	名　　称	英　名	意　　味
①	本人名	< self-name >	電話をかける人または、受ける人の名前
②	本人側電話番号	< self-side-phone-id >	本人側電話機の番号
③	相手名称	< other-party's-name >	電話相手の名称
④	相手側電話番号	< other-side-phone-id >	相手側電話機の番号
⑤	検索条件	< retrieval-condition >	相手検索のための条件
⑥	設定済条件	< settled-condition >	すでに検索対象となっている条件
⑦	検索結果リスト	< retrieval-response-list >	検索結果のリスト

≪　個別データ　≫

〔起動タスク分析〕

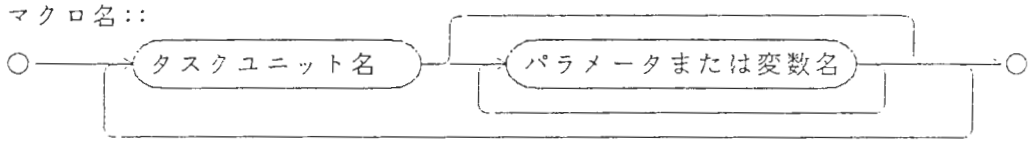
No.	名　　称	英　名	意　　味
①	状態番号	< ~~status-number~~ current-status >	状態を管理するための番号
②	起動ﾀｽｸ分析ﾃｰﾌﾞﾙ	< wake-up-task-analysis-table >	起動するﾀｽｸを決定するための分析ﾃｰﾌﾞﾙ（起動ｲﾍﾞﾝﾄ毎に状態番号からﾀｽｸﾏｸﾛ名を分析できる構造とする。）

〔タスク実行制御〕

No.	名　　称	英　名	意　　味
①	ﾀｽｸﾏｸﾛﾃｰﾌﾞﾙ	< task-macro-table >	実行すべきﾀｽｸﾏｸﾛのﾃｰﾌﾞﾙ

「タスクマクロの定義」

::タスクマクロ名::

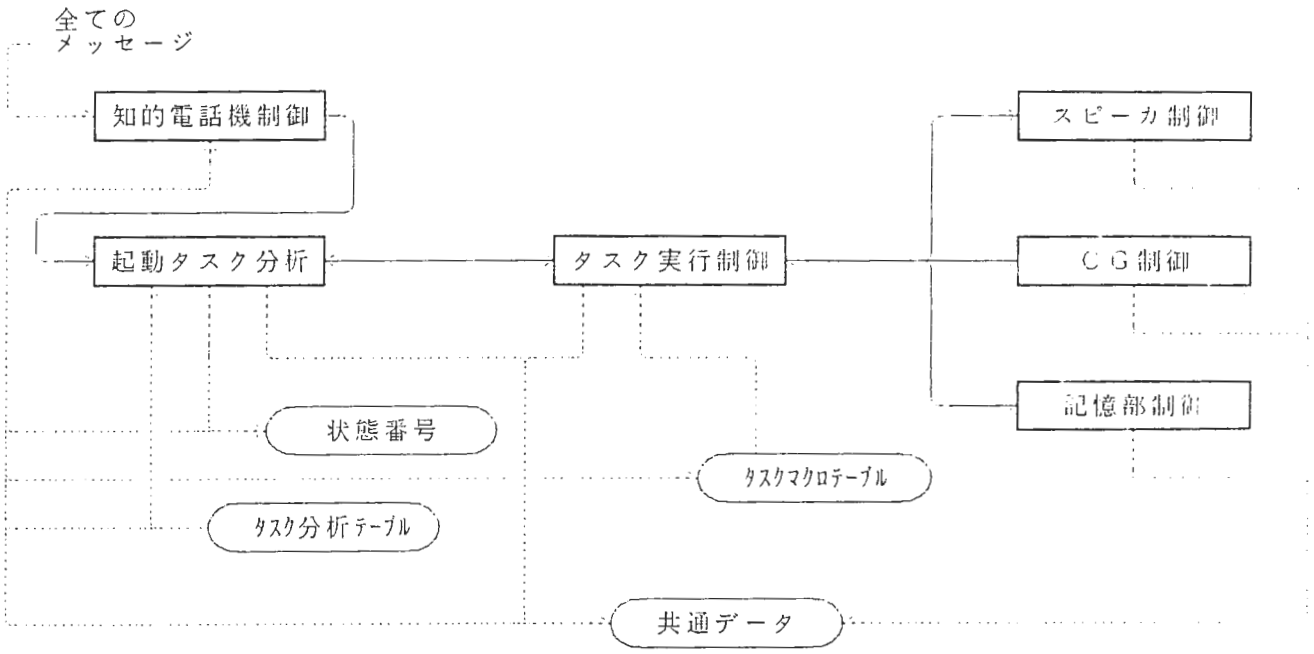


タスクマクロの終わりには遷移後の状態番号をいれておく事とする。

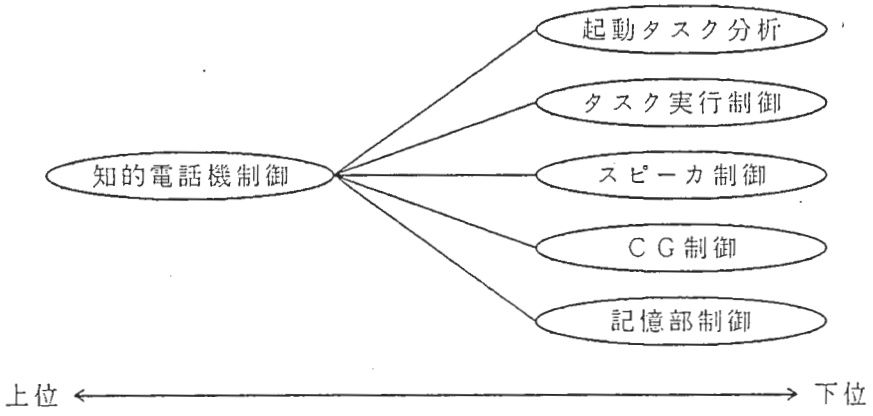
例〕（（ＳＰ－ＴＡＳＫ－１　”どこにおかけになりますか？”）

（ＴＡＳＫ－ＥＮＤ　次状態番号））

4. 機能オブジェクト構成図及び関係図



機能オブジェクト構成図



機能オブジェクト関係図

5. オブジェクト機能定義

〔知的電話機制御 < intelliphone-control > 〕

① メソッド定義

メソッド名	パラメータ	返値
・INITIALIZE	—	nil
・OFF	—	nil
・ON	—	〃
・CONNECT-REQUEST	—	〃
・CONNECT-RESPONSE	—	〃
・CALL-REQUEST	呼出者名　発信者名　発信側電話番号	〃
・CALL-RESPONSE	着信者名	〃
・BUSY	—	〃
・CODE-SEND	文字列／エラーコード	〃

② 機　能

INITIALIZE：　共通データ及び個別データを初期設定する。
　　下位のインスタンスを生成する。

OFF ～

CODE-SEND：　受けたパラメータを共通データ等へ設定して、
　　下位のインスタンスへ該当メッセージを送出する。

〔起動タスク分析 < wake-up-task-analysis > 〕

① メソッド定義

メソッド名	パラメータ	返値
・OFF	—	nil
・ON	—	〃
・CONNECT-REQUEST	—	〃
・CONNECT-RESPONSE	—	〃
・CALL-REQUEST	—	〃
・CALL-RESPONSE	—	〃
・BUSY	—	〃
・CODE-SEND	—	〃

② 機 能 （共 通）

- 各メソッドにより起動され、”タスク分析テーブル”と”状態番号”により起動すべきタスクを決定し、タスク実行制御を起動する。
- メッセージのパラメータは”共通データ”に格納して、各タスクユニットで使用可能な状態にしておく。
- タスクの実行が終了したら、状態番号を更新する。

〔タスク実行制御 < task-execution-control > 〕

① メソッド定義

メソッド名	パラメータ	返値
・WAKE-UP	タスクマクロ名	次状態番号

② 機 能

- タスクマクロ名より”タスクマクロテーブル”を参照し、各タスクユニットを順次起動する。
- タスクマクロに変数名が記述されている場合、共通データの実際の値に変換して、各タスクユニットを起動する。
- 各タスクユニットの処理が終了したら、返値として次状態番号を返す。

〔スピーカ制御 < speaker-control > 〕

① メソッド定義

メソッド名	パラメータ	返値
・SP-TASK-1	文字列1　文字列2　文字列3	nil
・SP-TASK-2	−	〃
・SP-TASK-3	×検索結果リスト	〃

② 機　能

SP-TASK-1 :　”文字列1～3”を連結したメッセージをスピーカに送出する。

SP-TASK-2 :　ＢＥＬＬメッセージをスピーカに送出する。

SP-TASK-3 :　以下のｉ）～ⅲ）の処理を要素が無くなるまで繰り返す。

ｉ）”検索結果リスト”から要素を取り出す。

ⅱ）該要素から「名称」と「案内用メッセージ」を取り出し、
「名称＋案内用メッセージ」としてメッセージを編集する。

ⅲ）ⅱ）で編集したメッセージをスピーカに送出する。

〔ＣＧ制御 < code-generator-control > 〕

① メソッド定義

メソッド名	パラメータ	返値
・CG-TASK-1	−	nil
・CG-TASK-2	−	〃
・CG-TASK-3	−	〃
・CG-TASK-4	×電話番号	〃
・CG-TASK-5	×呼出者名　発信者名　発側電話番号	〃
・CG-TASK-6	×着信者名	〃

② 機　能

CG-TASK-1 :　切断メッセージをＣＧに送出する。

CG-TASK-2 :　着信応答メッセージをＣＧに送出する。

CG-TASK-3 :　話中メッセージをＣＧに送出する。

CG-TASK-4 :　”電話番号”にダイヤルする。

CG-TASK-5 :　パラメータにより呼出要求メッセージをＣＧに送出する。

CG-TASK-6 :　”着信者名”により呼出応答メッセージをＣＧに送出する。

〔記憶部制御 < memory-control > 〕

① メソッド定義

メソッド名	パラメータ	返値
・MEM-TASK-1	－	nil
・MEM-TASK-2	＊相手名称　相手電話番号	〃
・MEM-TASK-3	＊条件１　条件２	〃

② 機　能

MEM-TASK-1：　最後にかけた相手の情報を取り出し、共通データに設定する。

MEM-TASK-2：　”相手名称”と”相手電話番号”を記憶部に登録する。

MEM-TASK-3：　”条件１”と”条件２”のＡＮＤ条件として記憶部を検索し、結果として得られる検索結果リストを共通データに設定する。また、検索時の条件は共通データの設定済条件に設定しておく。

for CLM13:>kshima>intelliphone>intelliphone-main-program.lisp.68

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 9/04/89 13:15:01
Queued at: 8/07/90 18:53:32

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;; Intelliphone の 立ちあげかた (デモの方法)
;;
;; すべてのfile は "CLM06:>INTELLIPHONE>" のdirectory にあります。
;; もし、変更したひとは、 そのfile を compile してから、
;;  "clm06:>intelliphone>" の directory にcopy してください。

;; main-programe からよんでいる各file の構成
;; For header print
;; (load "clm06:>intelliphone>intelliphone-header-print")  ;; made by Kshima.
;; For speaker unit
;; (load "clm06:>intelliphone>symbolics-to-ntt-voice")  ;; made by kshima.
;; For mic unit
;; (load "clm06:>intelliphone>dp3000v7")  ;; Made by uchida.
;; For hook unit
;; (load "clm06:>intelliphone>hook-switch-unit")  ;; made by kshima.
;; For memory unit
;; (load "clm06:>intelliphone>memory-unit")  ;; made by Nakajima.
;; For control unit
;; (load "clm06:>intelliphone>ip-control")  ;; made by Sato, Hayashi.
;; For code-generator and code-detector unit
;; (load "clm06:>intelliphone>cdcg")  ;; made by Shibamoto, Yokota.


;; まず、"clm06:>intelliphone>intelliphone-main-program" を load します。
;; つまり、(LOAD "CLM06:>INTELLIPHONE>INTELLIPHONE-MAIN-PROGRAM")
;; 補足, もし Mic or Speaker を使用しないときは
;;  (load "clm06:>intelliphone>test-intelliphone") を実行します。

;; 次に, (INTELLIPHONE-MAIN) を実行します。
;; 暫くすると、各ユニットの window が現われます。
;; その後は皆様ご承知の Intelliphone ですので 御自由にお使い下さい。

;; もし、おかしくなった時は,
;; おかしくなったユニットの window を reset してみる。
;; または、 control unit を initialize する。
;; つまり、main window(lisp listener) をselect して, (initialize *control-unit-var*) を実行する。
;; 其れでもだめなら、 初期のlisp listener に戻って(select-button L を2回)
;; 次に, (intelliphone-main) を実行します。


;; 終了は, hook-switch window で end-button をおす。

;; 共通データとしては、以下のものがあります
;; 各フレーバ (インスタンス) の変数の宣言
;; 共通データ
;; 各フレーバ (インスタンス) の変数の宣言
(defvar *speaker-unit-var*)
(defvar *mic-unit-var*)
(defvar *hook-switch-unit-var*)
(defvar *memory-unit-var*)
(defvar *control-unit-var*)
(defvar *cd-cg-unit-var*)

;; 各画面 (インスタンス) の変数の宣言
(defvar *intelliphone-main-unit-window-var*)
(defvar *speaker-unit-window-var*)
(defvar *mic-unit-window-var*)
(defvar *hook-switch-unit-window-var*)
(defvar *memory-unit-window-var*)
(defvar *control-unit-window-var*)
(defvar *cd-cg-unit-window-var*)

;; もし, 最初に宣伝をいれたいなら。。。 この switch を T にする。
;; header print switch for demo.
(defvar header-print-switch nil)


;; それぞれのwindow は以下のように定義されています。
;; speaker unit flavor
;; (defflavor speaker-unit () ())
;; see "clm06:>intelliphone>symbolics-to-ntt-voice" made by kshima.
```

```
;; mic-unit flavor
;; (defflavor mic-unit (voice-switch) ())
;; see "clm06:>intelliphone>dp3000v7" made by Uchida.

;; hook-switch-unit flavor
;; (defflavor hook-switch-unit ((hook-switch-status 'on)) () )
;; see "clm06:>intelliphone>hook-switch-unit" made by kshima.

;; memory unit flavor
;; (defflavor memory-unit (ROM-Name Card-Name Your-NO Your-Name Data-Base Last-Call) ())
;; see "clm06:>intelliphone>memory-unit.lisp" made by Nakajima.

;; control unit flavor
;; (defflavor intelliphone-control () ())
;; see "clm06:>intelliphone>ip-control.lisp" made by Takashi sato.

;; code detector / code generator unit flavor
;;(defflavor INTELLIPHONE-CDCG-PART (control-part (present-status :S00) port (hook-switch :on
)
;;                                   responding-person)
;;          (BASIC-PROTOCOL-HANDLER SERIAL-PORT-INTERFACE CG-CONNECT-REQUEST CG-CALL-REQUEST
;;           CG-CONNECT-RESPONSE CG-CALL-RESPONSE CG-BUSY CG-DISCONNECT CD-RNC
;;           CD-RSC CD-RDR CD-RFC CD-RNN CD-RSN CD-RFN CD-RNF CD-RNI CD-RSF CD-RSI)
;;  (:initable-instance-variables) ;
;;  (:readable-instance-variables) ; For test, declaring "writable"
;;  (:writable-instance-variables) ;
;;  (:required-methods serial-port-output))
;; see "clm06:>intelliphone>cdcg.lisp" made by Shibamoto.
```

```
;; window flavor declareration

;; intelliphone main window
(defflavor intelliphone-main-unit-window ()
           (dw:dynamic-window tv:lisp-listener)
  (:default-init-plist
   :expose-p t
   :edges-from '(64. 608. 1131. 791.)
   ;;:more-p nil
   ;;   :name "Intelliphone window"
   ;;:save-bits t
   ;;:border-margin-width 2
   ;;   :process '(intelliphone-header-window)
   ))

(defmethod (:initialize intelliphone-main-unit-window) ()
  (if (boundp '*intelliphone-main-unit-window-var*)
      (send *intelliphone-main-unit-window-var* :kill) )
  (setf *intelliphone-main-unit-window-var*
        (tv:make-window 'intelliphone-main-unit-window :expose-p t) ))


(defflavor speaker-unit-window ()
           (;;tv:process-mixin
            tv:window)
  (:default-init-plist
   :expose-p t
   :edges-from '(64. 43. 437. 266.)
   :more-p nil
   ;;   :name "Intelliphone window"
   :save-bits t
   :border-margin-width 2
;;   :process '(intelliphone-header-window)
   ))

(defmethod (:initialize speaker-unit-window) ()
  (if (boundp '*speaker-unit-window-var*)
      (send *speaker-unit-window-var* :kill) )
  (setf *speaker-unit-window-var* (tv:make-window 'speaker-unit-window :expose-p t )))


(defflavor mic-unit-window ()
           (tv:process-mixin tv:window)
  (:default-init-plist
   :expose-p t
   :edges-from '(64. 267. 437. 506.)
   :more-p nil
   ;;   :name "Intelliphone window"
   :save-bits t
   :border-margin-width 2
   :process '(recog-from-dp-3000-loop)
   ))

(defmethod (:initialize mic-unit-window) ()
  (if (boundp '*mic-unit-window-var*)
      (send *mic-unit-window-var* :kill) )
  (setf *mic-unit-window-var* (tv:make-window 'mic-unit-window :expose-p t)))

(defun recog-from-dp-3000-loop (window)
  (let ((*terminal-io* window)
        (recog-result))
    (loop (send *terminal-io* :set-reverse-video-p t)
          (setq recog-result (recog-from-dp-3000))
          (send *terminal-io* :set-reverse-video-p nil)
          (print recog-result window)
          (code-send *control-unit-var* recog-result) )))


(defflavor hook-switch-unit-window ()
           (tv:process-mixin tv:window)
  (:default-init-plist
   :expose-p t
   :edges-from '(64. 507. 437. 607.)
   :more-p nil
   ;;   :name "Intelliphone window"
   :save-bits t
   :border-margin-width 2
```

```
   :process '(hook-on-off-catch-loop)
   ))

(defmethod (:initialize hook-switch-unit-window) ()
  (if (boundp '*hook-switch-unit-window-var*)
      (send *hook-switch-unit-window-var* :kill) )
  (setf *hook-switch-unit-window-var*
        (tv:make-window 'hook-switch-unit-window :expose-p t)))

(defun hook-on-off-catch-loop (window)
  (loop (sleep 2.)
        (let ((*terminal-io* window))
          (if (char= (read-char) '#\end)
              (return (progn (all-kill-window-for-intelliphone)
                             (close-ntt-voice)
                             (close-dp-3000)
                             (send *cd-cg-unit-var* :kill)
                             (get-card *memory-unit-var* "clm14:>intelliphone>my-phone.jap")
                             'end-of-intelliphone ))
              (progn (set-hook-switch *hook-switch-unit-var*) ;; on  --> off or off --> on
                     (print (list 'on-off-hook
                                  (status? *hook-switch-unit-var*)))
                     (cond ((eq (status? *hook-switch-unit-var*) 'on)
                            (on-hook *control-unit-var*) )
                           ((eq (status? *hook-switch-unit-var*) 'off)
                            ;; refresh the window
                            (send *intelliphone-main-unit-window-var* :clear-window)
                            ;;(send *intelliphone-main-unit-window-var* :home-cursor)
                            (send *speaker-unit-window-var* :clear-window)
                            ;;(send *speaker-unit-window-var* :home-cursor)
                            (send *mic-unit-window-var* :clear-window)
                            ;;(send *mic-unit-window-var* :home-cursor)
                            (send *hook-switch-unit-window-var* :clear-window)
                            ;;(send *hook-switch-unit-window-var* :home-cursor)
                            (send *memory-unit-window-var* :clear-window)
                            ;;(send *memory-unit-window-var* :home-cursor)
                            (send *control-unit-window-var* :clear-window)
                            ;;(send *control-unit-window-var* :home-cursor)
                            (send *cd-cg-unit-window-var* :clear-window)
                            ;;(send *cd-cg-unit-window-var* :home-cursor)
                            ;; to control off-hook send
                            (off-hook *control-unit-var*) )
                           (t nil) ))))))


(defflavor memory-unit-window ()
           (;;tv:process-mixin
            tv:window)
  (:default-init-plist
   :expose-p t
   :edges-from '(438. 43. 834. 184.)
   :more-p nil
   ;;   :name "Intelliphone window"
   :save-bits t
   :border-margin-width 2
;;   :process '(intelliphone-header-window)
   ))

(defmethod (:initialize memory-unit-window) ()
  (if (boundp '*memory-unit-window-var*)
      (send *memory-unit-window-var* :kill) )
  (setf *memory-unit-window-var* (tv:make-window 'memory-unit-window :expose-p t)))


(defflavor intelliphone-control-window ()
           (;;tv:process-mixin
            tv:window)
  (:default-init-plist
   :expose-p t
   :edges-from '(438. 185. 834. 607.)
   :more-p nil
   ;;   :name "Intelliphone window"
   :save-bits t
   :border-margin-width 2
;;   :process '(intelliphone-header-window)
   ))

(defmethod (:initialize intelliphone-control-window) ()
```

```
  (if (boundp '*control-unit-window-var*)
      (send *control-unit-window-var* :kill) )
  (setf *control-unit-window-var*
        (tv:make-window 'intelliphone-control-window :expose-p t)))


(defflavor cd-cg-unit-window ()
           (tv:process-mixin tv:window)
  (:default-init-plist
   :expose-p t
   :edges-from '(835. 43. 1131. 607.)
   :more-p nil
   ;;   :name "Intelliphone window"
   :save-bits t
   :border-margin-width 2
   :process '(cd-cg-polling-loop)
   ))

(defmethod (:initialize cd-cg-unit-window) ()
  (if (boundp '*cd-cg-unit-window-var*)
      (send *cd-cg-unit-window-var* :kill) )
  (setf *cd-cg-unit-window-var* (tv:make-window 'cd-cg-unit-window :expose-p t)))

(defun cd-cg-polling-loop (window)
  (let ((*terminal-io* window)
        polling-result)
    (loop (sleep 2.)
          (send *terminal-io* :set-reverse-video-p t)
          (setq polling-result (polling *cd-cg-unit-var*))
          (if polling-result
              (progn (send *terminal-io* :set-reverse-video-p nil)
                     (terpri)
                     (print polling-result)
                     ))
          )))
```

```
;; main program
(defun intelliphone-main ()
  ;; serial port open
  ;; NTT voice port open (port no. 1)
  (if (boundp '*ntt-voice-port*)
      (close *ntt-voice-port*) )
  (open-ntt-voice)
  ;; NEC voice recognition port open (port no. 2)
  (if (boundp '*dp-3000-port*)
      (close *dp-3000-port*) )
  (open-dp-3000)
  (unwind-protect
      (progn
        ;; 初期画面の設定
        (all-kill-window-for-intelliphone) ;; kill all window

        ;; デバック用のシステム構成初期画面のオブジェクトを生成する
        ;; main window create
        ;;(if (boundp '*intelliphone-main-unit-window-var*)
        ;;    (send *intelliphone-main-unit-window-var* :kill) )
        (setf *intelliphone-main-unit-window-var*
              (tv:make-window 'intelliphone-main-unit-window :expose-p t ))
        ;; speaker window init
        ;;(if (boundp '*speaker-unit-window-var*)
        ;;    (send *speaker-unit-window-var* :kill) )
        (setf *speaker-unit-window-var* (tv:make-window 'speaker-unit-window :expose-p t ))
        (send *speaker-unit-window-var* :set-reverse-video-p t)
        ;; mic window init
        ;;(if (boundp '*mic-unit-window-var*)
        ;;    (send *mic-unit-window-var* :kill) )
        (setf *mic-unit-window-var* (tv:make-window 'mic-unit-window :expose-p t))
        (send *mic-unit-window-var* :set-reverse-video-p t)
        ;; hook window init
        ;;(if (boundp '*hook-switch-unit-window-var*)
        ;;    (send *hook-switch-unit-window-var* :kill) )
        (setf *hook-switch-unit-window-var*
              (tv:make-window 'hook-switch-unit-window :expose-p t))
        ;; memory window init
        ;;(if (boundp '*memory-unit-window-var*)
        ;;    (send *memory-unit-window-var* :kill) )
        (setf *memory-unit-window-var* (tv:make-window 'memory-unit-window :expose-p t))
        (send *memory-unit-window-var* :set-reverse-video-p t)
        ;; control window init
        ;;(if (boundp '*control-unit-window-var*)
        ;;    (send *control-unit-window-var* :kill) )
        (setf *control-unit-window-var*
              (tv:make-window 'intelliphone-control-window :expose-p t))

        ;; システム構成オブジェクトを生成する
        ;; speaker init
        ;;  (if (boundp '*speaker-unit-var*)
        ;;      (send *speaker-unit-var* :kill) )
        (setf *speaker-unit-var* (make-instance 'speaker-unit))
        ;; mic init
        ;;  (if (boundp '*mic-unit-var*)
        ;;      (send *mic-unit-var* :kill) )
        (setf *mic-unit-var* (make-instance 'mic-unit))
        ;; hook init
        ;;  (if (boundp '*hook-switch-unit-var*)
        ;;      (send *hook-switch-unit-var* :kill) )
        (setf *hook-switch-unit-var* (make-instance 'hook-switch-unit))
        ;; memory init
        ;;  (if (boundp '*memory-unit-var*)
        ;;  (send *memory-unit-var* :kill) )
        (setf *memory-unit-var* (make-instance 'memory-unit))
        ;; control init
        ;;  (if (boundp '*control-unit-var*)
        ;;  (send '*control-unit-var* :kill) )
        (setf *control-unit-var* (make-instance 'intelliphone-control))
        ;; cd cg init
        ;;  (if (boundp '*cd-cg-unit-var*)
        ;;      (send *cd-cg-unit-var* :kill) )
        ;;(setf *cd-cg-unit-var* (make-instance 'cd-cg-unit))
        (if (and (boundp '*cd-cg-unit-var*)
                 (intelliphone-cdcg-part-port *cd-cg-unit-var*) )
            (send *cd-cg-unit-var* :kill) )
```

```
    (setf *cd-cg-unit-var*
          (make-instance 'intelliphone-cdcg-part
                         :control-part *control-unit-var*)) ;; made by shibamo.
    (send *cd-cg-unit-var* :create)
    ;; cd cg window init
    ;;(if (boundp '*cd-cg-unit-window-var*)
    ;;    (send *cd-cg-unit-window-var* :kill) )
    (setf *cd-cg-unit-window-var* (tv:make-window 'cd-cg-unit-window :expose-p t))
    (send *cd-cg-unit-window-var* :set-reverse-video-p t)
    ;; this window create is postpond because of polling.
    ;; if you make window first, polling start, polling require serial port.
    ;; Its port is made by :create method of cdcg-part.

    ;; intelliphone の説明(画面, 合成音)
    ;; see "clm06:>intelliphone>intelliphone-header-print"
    (and header-print-switch
         (intelliphone-attention) )

    ;; all window expose.
    (all-expose-window-for-intelliphone)

    ;; rom (IC card) loading...
    (code-send *speaker-unit-var* "ただいま、ICカードを読み込んでいます。暫くおまちください。")
    (set-rom *memory-unit-var* "clm14:>intelliphone>phone.jap")
    (set-card *memory-unit-var* "clm14:>intelliphone>my-phone.jap")

    ;; initalize controler
    (initialize *control-unit-var*)
    (code-send *speaker-unit-var* "準備 完了いたしました。どうぞ お使いください。")
    (send *hook-switch-unit-window-var* :select)
    )
))
```

```
;; すべてのwindowを表示する
(defun all-expose-window-for-intelliphone ()
  ;; main window create
  (if (boundp '*intelliphone-main-unit-window-var*)
      (send *intelliphone-main-unit-window-var* :expose) )
  ;; speaker window init
  (if (boundp '*speaker-unit-window-var*)
      (send *speaker-unit-window-var* :expose) )
  ;; mic window init
  (if (boundp '*mic-unit-window-var*)
      (send *mic-unit-window-var* :expose) )
  ;; hook window init
  (if (boundp '*hook-switch-unit-window-var*)
      (send *hook-switch-unit-window-var* :expose) )
  ;; memory window init
  (if (boundp '*memory-unit-window-var*)
      (send *memory-unit-window-var* :expose) )
  ;; control window init
  (if (boundp '*control-unit-window-var*)
      (send *control-unit-window-var* :expose) )
  ;; cd cg window init
  (if (boundp '*cd-cg-unit-window-var*)
      (send *cd-cg-unit-window-var* :expose) )
  )

;; すべてのwindowをkillする
(defun all-kill-window-for-intelliphone ()
  ;; cd cg window init
  (if (boundp '*cd-cg-unit-window-var*)
      (progn (send *cd-cg-unit-window-var* :kill)
             (makunbound '*cd-cg-unit-window-var*) ))
  ;; memory window init
  (if (boundp '*memory-unit-window-var*)
      (progn (send *memory-unit-window-var* :kill)
             (makunbound '*memory-unit-window-var*) ))
  ;; speaker window init
  (if (boundp '*speaker-unit-window-var*)
      (progn (send *speaker-unit-window-var* :kill)
             (makunbound '*speaker-unit-window-var*) ))
  ;; mic window init
  (if (boundp '*mic-unit-window-var*)
      (progn (send *mic-unit-window-var* :kill)
             (makunbound '*mic-unit-window-var*) ))
  ;; control window init
  (if (boundp '*control-unit-window-var*)
      (progn (send *control-unit-window-var* :kill)
             (makunbound '*control-unit-window-var*) ))
  ;; main window create
  (if (boundp '*intelliphone-main-unit-window-var*)
      (progn (send *intelliphone-main-unit-window-var* :kill)
             (makunbound '*intelliphone-main-unit-window-var*) ) )
  ;; hook window init
  (if (boundp '*hook-switch-unit-window-var*)
      (progn (send *hook-switch-unit-window-var* :kill)
             (makunbound '*hook-switch-unit-window-var*) ))
  )
```

```
;; For other progmam loading

(defun all-other-program-load-for-intelliphone ()
  ;; see "clm06:>intelliphone>intelliphone-header-print" made by Kshima.
  (load "clm14:>intelliphone>intelliphone-header-print")
  ;; For speaker unit
  (load "clm14:>intelliphone>symbolics-to-ntt-voice")  ;; made by kshima
  ;; For mic unit
  (load "clm14:>intelliphone>dp3000v7")    ;; Made by uchida
  ;; For hook unit
  (load "clm14:>intelliphone>hook-switch-unit")  ;; made by kshima
  ;; For memory unit  see "clm06:>intelliphone>memory-unit.lisp" made by Nakajima.
  (load "clm14:>intelliphone>memory-unit")
  ;; For control unit                              ;; ask for sato.
  (load "clm14:>intelliphone>ip-control")
  ;; For code-generator and code-detector unit    ;; ask for Shibamoto.
  (load "clm14:>intelliphone>cdcg")
  )


;; see "clm06:>intelliphone>intelliphone-header-print" made by Kshima.
(or (functionp 'intelliphone-attention)
    (load "clm14:>intelliphone>intelliphone-header-print") )

;; For speaker unit
(or (functionp 'open-ntt-voice)
    (load "clm14:>intelliphone>symbolics-to-ntt-voice") )  ;; made by kshima

;; For mic unit
(or (functionp 'open-dp-3000)
    (load "clm14:>intelliphone>dp3000v7") )    ;; Made by uchida

;; For hook unit
(or (functionp 'status?)
    (load "clm14:>intelliphone>hook-switch-unit") )  ;; made by kshima

;; For memory unit
;;    see "clm02:>intelliphone>memory-unit.lisp" made by Nakajima.
(or (functionp '$select)
    (load "clm14:>intelliphone>memory-unit") )

;; For control unit                              ;; ask for sato.
(or (functionp 'wake-up-task-analysis-aux)
    (load "clm14:>intelliphone>ip-control") )

;; For code-generator and code-detector unit    ;; ask for Shibamoto.
(or (functionp 'delimit )
    (load "clm14:>intelliphone>cdcg") )


;; Utility program

;; すべてのwindowを表示する
;; (all-expose-window-for-intelliphone)

;; すべてのwindowをkillする
;; (all-kill-window-for-intelliphone)

;; 各window の初期化
;; それぞれの window にたいして、 :initialize を送る。
;; たとえば、(send *mic-unit-window-var* :initialize)


;; cd cg unit の close
;; (send *cd-cg-unit-var* :kill)
;; cd cg unit の open
;; (send *cd-cg-unit-var* :create)

;; ntt voice no close
;; (close-ntt-voice)
;; ntt voice no open
;; (open-ntt-voice)

;; dp-3000 voice no close
;; (close-dp-3000)
;; dp-3000 voice no open
;; (open-dp-3000)
```

```
;; For other progmam loading
;; (all-other-program-load-for-intelliphone)

;; End of file.
```

CLM13:>kshima>intelliphone>intelliphone-main-program-warning.lisp.1

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 9/04/89 13:16:57
Queued at: 8/07/90 18:54:56

```
For Function INTELLIPHONE-MAIN
  While compiling *NTT-VOICE-PORT*:
    The variable *NTT-VOICE-PORT* is unknown and has been declared SPECIAL
  While compiling *DP-3000-PORT*:
    The variable *DP-3000-PORT* is unknown and has been declared SPECIAL
The following functions were referenced but don't seem defined:
 OPEN-DP-3000 referenced by INTELLIPHONE-MAIN
 INTELLIPHONE-CDCG-PART-PORT referenced by INTELLIPHONE-MAIN
 INTELLIPHONE-ATTENTION referenced by INTELLIPHONE-MAIN
 INITIALIZE referenced by INTELLIPHONE-MAIN
 OPEN-NTT-VOICE referenced by INTELLIPHONE-MAIN
 SET-ROM referenced by INTELLIPHONE-MAIN
 RECOG-FROM-DP-3000 referenced by RECOG-FROM-DP-3000-LOOP
 CODE-SEND referenced by INTELLIPHONE-MAIN, RECOG-FROM-DP-3000-LOOP
 CLOSE-NTT-VOICE referenced by HOOK-ON-OFF-CATCH-LOOP
 CLOSE-DP-3000 referenced by HOOK-ON-OFF-CATCH-LOOP
 GET-CARD referenced by HOOK-ON-OFF-CATCH-LOOP
 SET-HOOK-SWITCH referenced by HOOK-ON-OFF-CATCH-LOOP
 STATUS? referenced by HOOK-ON-OFF-CATCH-LOOP
 ON-HOOK referenced by HOOK-ON-OFF-CATCH-LOOP
 OFF-HOOK referenced by HOOK-ON-OFF-CATCH-LOOP
 POLLING referenced by CD-CG-POLLING-LOOP
 SET-CARD referenced by INTELLIPHONE-MAIN

  SPEAKER-UNIT is not a known flavor name.
  MIC-UNIT is not a known flavor name.
  HOOK-SWITCH-UNIT is not a known flavor name.
  MEMORY-UNIT is not a known flavor name.
  INTELLIPHONE-CONTROL is not a known flavor name.
  INTELLIPHONE-CDCG-PART is not a known flavor name.
>Breakpoint ZMACS.  Press   to continue or   to quit.

You are being asked to enter a command or form.

These are the possible command names starting with "Copy ":
  Copy File         Copy Output History
  Copy Flod Files   Copy World
  Copy Microcode

Command: Copy Output History (a destination) Buffer (an editor buffer) aaa
```

CLM13:>kshima>intelliphone>ip-control.lisp.25

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/06/87 14:46:35
Queued at: 8/07/90 18:57:16

```
;;; -*- Mode: LISP; Syntax: Common-lisp -*-

;;;
;;; flavor 名の定義
;;;
(defvar *control-unit-var*)                     ; 自分自身の flavor
(defvar *control-unit-window-var*)              ; 虫とり用の window

(defvar *wake-up-task-analysis*  nil)           ; 起動 task 分析 flavor
(defvar *task-execution-control* nil)           ; task 実行制御 flavor
(defvar *speaker-control*        nil)           ; speaker 制御 flavor
(defvar *code-generator-control* nil)           ; CG 制御 flavor
(defvar *memory-control*         nil)           ; 記憶部制御 flavor

;;;
;;; 共通 data の定義
;;;
(defvar *self-name*                nil)         ; 本人名 : 電話をかける人、または受ける人の名前
(defvar *self-side-phone-id*       nil)         ; 本人側電話番号 : 本人側電話機の番号
(defvar *other-partys-name*        nil)         ; 相手名称 : 電話（通話）相手の名称
(defvar *other-side-phone-id*      nil)         ; 相手側電話番号 : 相手側電話機の番号
(defvar *retrieval-condition*      nil)         ; 検索条件 : 相手検索のための条件
(defvar *settled-condition*        nil)         ; 設定済条件 : すでに検索対象となっている条件
(defvar *retrieval-response-list* nil)          ; 検索結果 list : 検索結果の list

;;;
;;; 個別 data の定義
;;;
(defvar *flagment-of-in-arc*          nil)      ; 遷移中 flag
(defvar *status-number*               nil)      ; 状態番号 : 状態を管理するための番号
(defvar *wake-up-task-analysis-table* nil)      ; 起動 task 分析表 : 起動する task を決定する分析表
(defvar *task-macro-table*            nil)      ; task macro 表 : 実行すべき task macro の表

;;;
;;; message 用文字列の定義
;;;
(defvar *string-last-call*       "")            ; さっきの所
(defvar *string-no-cond*         "")            ; ありません
(defvar *string-where-dial*      "")            ; 何処におかけになりますか?
(defvar *string-hold-on*         "")            ; ~aに電話します。少しお待ち下さい。
(defvar *string-no-place*        "")            ; 該当する所はありません。~a
(defvar *string-more-cond*       "")            ; 他に条件はありますか?
(defvar *string-busy*            "")            ; お話し中です。
(defvar *string-he-speak*        "")            ; ~aがお出になりました。どうぞ、お話し下さい。
(defvar *string-say-list*        "")            ; ご希望の条件にあう所は~a箇所あります。
(defvar *string-to-call*         "")            ; ~aに~aからお電話です。受話器をお取り下さい。
(defvar *string-from-call*       "")            ; ~aに~aからお電話すで。~aですね。
(defvar *string-do-talk*         "")            ; どうぞ、お話し下さい。
(defvar *string-on-hook*         "")            ; それでは、電話をお切り致します。
(defvar *string-no-person-name* "")             ; ~%名前(~a)はありません。
(defvar *string-no-macro-name*  "")             ; ~%タスクマクロ名(~a)がありません。

;;;
;;; intelliphone-control == 知的電話機制御 flavor
;;;     initialize() : 共通 data、および個別 data を初期設定する。また、下位の instance を生成する。
;;;     off-hook() : off-hook message を起動 task 分析 instance に送る。
;;;     on-hook() : on-hook message を起動 task 分析 instance に送る。
;;;     disconnect() : 切断 message を起動 task 分析 instance に送る。
;;;     connect-request() : 着信 messageを 起動 task 分析 instance に送る。
;;;     connect-response() : 着信応答 message を起動 task 分析 instance に送る。
;;;     call-request(呼出者名 発信者名 発信側電話番号) : 各 parameter を共通 data に設定し、呼出要求
;;;                                                 message を起動 task 分析 instance に送る。
;;;                                                     *self-name* ..... 呼出者名
;;;                                                     *other-partys-name* ..... 発信者名
;;;                                                     *other-side-phone-id* ... 発信側電話番号
;;;     call-response(着信者名) : "着信者名"を共通 data (*other-partys-name*) に設定し、呼出応答
;;;                               message を起動 task 分析 instance に送る。
;;;     busy() : 話中 message を起動 task 分析 instance に送る。
;;;     code-send(文字列) : "文字列"を共通 data (*retrieval-condition*) に設定し、code-send message
;;;                         を起動 task 分析 instance に送る。
;;;     select-set-other-partys(名称 対象者list) : "対象者list"から、"名称"と同じ名前を持つ個人/団体を
;;;                                                取り出し、その名前と電話番号とを共通dataに設定する。
;;;                                                    *other-partys-name* ..... 名前
;;;                                                    *other-side-phone-id* ... 電話番号
;;;     set-other-partys(属性list) : "属性list"から名称と電話番号とを取り出し、共通 data に設定する。
```

```
;;;                                     *other-partys-name* ..... 名称
;;;                                     *other-side-phone-id* ... 電話番号
;;;
(defflavor intelliphone-control () ())

(defmethod (initialize intelliphone-control) ()
  (format *control-unit-window-var* "~%{initialize}")

  ;; 下位の機能 instance を生成する
  (setf *wake-up-task-analysis*  (make-instance 'wake-up-task-analysis))
  (setf *task-execution-control* (make-instance 'task-execution-control))
  (setf *speaker-control*        (make-instance 'speaker-control))
  (setf *code-generator-control* (make-instance 'code-generator-control))
  (setf *memory-control*         (make-instance 'memory-control))

  ;; 個別 data、共通 data を初期化する
  (setf *flagment-of-in-arc* t)
  (setf *self-side-phone-id* (your-no? *memory-unit-var*))
  (setf *status-number*      0)

  ;; message を設定する
  (setf *string-last-call*       "さっきの所")
  (setf *string-no-cond*         "ありません")
  (setf *string-where-dial*      "何処におかけになりますか?")
  (setf *string-hold-on*         "~aさんに電話します。少しお待ち下さい。")
  (setf *string-no-place*        "該当する所はありません。何処におかけになりますか?")
  (setf *string-more-cond*       "他に条件はありますか?")
  (setf *string-busy*            "お話しちゅうです。受話器を置いて下さい。")
  (setf *string-he-speak*        "~aさんがお出になりました。どうぞ、お話し下さい。")
  (setf *string-say-list*        "ご希望の条件にあう所は~a箇所あります。")
  (setf *string-to-call*         "~aさんに~aさんからお電話です。受話器をお取り下さい。")
  (setf *string-from-call*       "~aさんに~aさんからお電話です。~aさんですね?")
  (setf *string-do-talk*         "どうぞ、お話し下さい。")
  (setf *string-on-hook*         "それでは、電話をお切り致します。")
  (setf *string-no-person-name*  "~%名前(~a)はありません。")
  (setf *string-no-macro-name*   "~%タスクマクロ名(~a)がありません。")

  ;;
  ;; (((状態番号 event) macro名) ... )
  ;;
  (setf *wake-up-task-analysis-table*
        '(
          ;; 発信系の arc
          ((0 off-hook)          tm-0-off-hook)
          ((0 connect-request)   tm-0-connect-request)
          ((1 on-hook)           tm-1-on-hook)
          ((1 code-send)         tm-1-code-send)
          ((1 connect-request)   tm-1-connect-request)
          ((2 connect-response)  tm-2-connect-response)
          ((2 busy)              tm-2-busy)
          ((2 on-hook)           tm-x-on-hook-disconnect)
          ((3 call-response)     tm-3-call-response)
          ((3 on-hook)           tm-x-on-hook-disconnect)
          ((4 on-hook)           tm-x-on-hook-disconnect)
          ((5 on-hook)           tm-x-on-hook-disconnect)
          ((6 connect-request)   tm-6-connect-request)
          ((6 code-send)         tm-6-code-send)
          ((6 on-hook)           tm-x-on-hook-disconnect)
          ((7 code-send)         tm-7-code-send)
          ((7 on-hook)           tm-x-on-hook-disconnect)
          ;; 着信系の arc
          ((101 disconnect)      tm-101-disconnect)
          ((101 call-request)    tm-101-call-request)
          ((102 off-hook)        tm-102-off-hook)
          ((103 on-hook)         tm-x-on-hook-disconnect)
          ((103 code-send)       tm-103-code-send)))
  ;;
  ;; ((macro名
  ;;    (sp-task-1 "どこにおかけになりますか?") ...
  ;;    (cond 変数名
  ;;        (値1 (sp-task-2) ... (task-end 次状態番号))
  ;;        (値2 (sp-task-3) ... (task-end 次状態番号)))) ... )
  ;;
  (setf *task-macro-table*
        '((tm-x-on-hook-disconnect
            ;; 切断 message を CG に送り、idle 状態に戻る。
```

```
        (sp-task-1 *speaker-control* *string-on-hook*)
        (cg-task-1 *code-generator-control*)
        (task-end 0))

      (tm-0-off-hook
        ;; "何処におかけに~"と応えて、相手特定待ち状態に移る。
        (setf *self-name* (your-name? *memory-unit-var*))
        (sp-task-1 *speaker-control* *string-where-dial*)
        (task-end 1))

      (tm-0-connect-request
        ;; 着信応答 message を CG に送り、呼出要求待ち状態に移る。
        (cg-task-2 *code-generator-control*)
        (task-end 101))

      (tm-1-on-hook
        ;; idle 状態に戻る。
        (task-end 0))

      (tm-1-code-send
        (cond ((string= *retrieval-condition* *string-last-call*)
               ;; 最後にかけた相手に接続する。
               '((mem-task-1 *memory-control*)
                 (sp-task-1 *speaker-control*
                            (format nil *string-hold-on* *other-partys-name*))
                 (cg-task-4 *code-generator-control* *other-side-phone-id*)
                 (task-end 2)))
              (t
               ;; 条件指定による検索を行なう。
               '((mem-task-3 *memory-control* *retrieval-condition* *retrieval-condition*
)
                 (cond ((null *retrieval-response-list*)
                        ;; 該当者がいない。
                        '((sp-task-1 *speaker-control* *string-no-place*)
                          (task-end 1)))
                       ((= 1 (length *retrieval-response-list*))
                        ;; 該当者が一人だけなので、その相手に接続する。
                        '((set-other-partys *control-unit-var*
                                            (car *retrieval-response-list*))
                          (sp-task-1 *speaker-control*
                                     (format nil *string-hold-on* *other-partys-name*))
                          (cg-task-4 *code-generator-control* *other-side-phone-id*)
                          (mem-task-2 *memory-control*
                                      *other-partys-name* *other-side-phone-id*)
                          (task-end 2)))
                       (t
                        ;; 該当者が複数なので、さらに条件を聞く。
                        '((sp-task-1 *speaker-control* *string-more-cond*)
                          (task-end 6)))))))))

      (tm-1-connect-request
        ;; 話中 message を CG に送り、相手特定待ち状態に戻る。
        (cg-task-3 *code-generator-control*)
        (task-end 1))

      (tm-2-connect-response
        ;; 呼出要求 message を CG に送り、呼出応答待ち状態に移る。
        (cg-task-5 *code-generator-control*
                   *other-partys-name* *self-name* *self-side-phone-id*)
        (task-end 3))

      (tm-2-busy
        ;; "お話し中です。"と応え、電話機復旧待ち状態に移る。
        (sp-task-1 *speaker-control* *string-busy*)
        (task-end 5))

      (tm-3-call-response
        ;; "~さんがお出になり~"と応え、通話中状態に移る。
        (sp-task-1 *speaker-control* (format nil *string-he-speak* *other-partys-name*))
        (task-end 4))

      (tm-6-connect-request
        ;; 話中 message を CG に送り、条件入力待ち状態に戻る。
        (cg-task-3 *code-generator-control*)
        (task-end 6))

      (tm-6-code-send
```

```
            (cond ((string= *retrieval-condition* *string-no-cond*)
                   ;; 条件がなければ、該当者の名前を読み上げる。
                   '((sp-task-1 *speaker-control*
                                (format nil *string-say-list*
                                        (length *retrieval-response-list*)))
                     (sp-task-3 *speaker-control* *retrieval-response-list*)
                     (sp-task-1 *speaker-control* *string-where-dial*)
                     (task-end 7)))
                  (t
                   ;; 条件指定による検索を行なう。
                   '((mem-task-3 *memory-control* *settled-condition* *retrieval-condition*)
                     (cond ((null *retrieval-response-list*)
                            ;; 該当者がいない。
                            '((sp-task-1 *speaker-control*
                                         (format nil *string-no-place* *string-where-dial*))
                              (task-end 1)))
                           ((= 1 (length *retrieval-response-list*))
                            ;; 該当者が一人だけなので、その相手に接続する。
                            '((set-other-partys *control-unit-var*
                                                (car *retrieval-response-list*))
                              (sp-task-1 *speaker-control*
                                         (format nil *string-hold-on* *other-partys-name*))
                              (cg-task-4 *code-generator-control* *other-side-phone-id*)
                              (mem-task-2 *memory-control*
                                          *other-partys-name* *other-side-phone-id*)
                              (task-end 2)))
                           (t
                            ;; 該当者が複数なので、さらに条件を聞く。
                            '((sp-task-1 *speaker-control* *string-more-cond*)
                              (task-end 6)))))))))

          (tm-7-code-send
            ;; 通話相手の情報を取り出し、dial し、last call 情報を登録する。着信応答待ち状態に移る。
            (select-set-other-partys *control-unit-var*
                                     *retrieval-condition* *retrieval-response-list*)
            (sp-task-1 *speaker-control* (format nil *string-hold-on* *other-partys-name*))
            (cg-task-4 *code-generator-control* *other-side-phone-id*)
            (mem-task-2 *memory-control* *other-partys-name* *other-side-phone-id*)
            (task-end 2))

          (tm-101-disconnect
            ;; idle 状態に戻る。
            (task-end 0))

          (tm-101-call-request
            ;; 着信応答messageをCGに送る。次に、BELL を鳴らし、"~さんにお電話です。"と応える。
            ;; off-hook待ち状態に移る。
            (sp-task-2 *speaker-control*)
            (sp-task-1 *speaker-control*
                       (format nil *string-to-call* *self-name* *other-partys-name*))
            (task-end 102))

          (tm-102-off-hook
            ;; "~さんからです。"と応えて、着信相手応答待ち状態に移る。
            (sp-task-1 *speaker-control*
                       (format nil *string-from-call*
                               *self-name* *other-partys-name* *self-name*))
            (task-end 103))

          (tm-103-code-send
            ;; "どうぞ~"と応え、呼出応答 message を CG に送り、呼出応答待ち状態に移る。
            (sp-task-1 *speaker-control* *string-do-talk*)
            (cg-task-6 *code-generator-control* *self-name*)
            (task-end 4))
          ))
  )

;;; message を取り扱う method
(defmethod (off-hook intelliphone-control) ()
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
    (format *control-unit-window-var* "~%{off-hook}")
    (off-hook *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (on-hook intelliphone-control) ()
  (when *flagment-of-in-arc*
```

```
    (setf *flagment-of-in-arc* nil)
    (format *control-unit-window-var* "~%{on-hook}")
    (on-hook *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (disconnect intelliphone-control) ()
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
    (format *control-unit-window-var* "~%{disconnect}")
    (disconnect *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (connect-request intelliphone-control) ()
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
    (format *control-unit-window-var* "~%{connect-request}")
    (connect-request *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (connect-response intelliphone-control) ()
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
    (format *control-unit-window-var* "~%{connect-response}")
    (connect-response *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (call-request intelliphone-control) (yobidashi hasshin hatsu-id)
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
    (format *control-unit-window-var* "~%{call-request ~a ~a ~a}" yobidashi hasshin hatsu-id)
    (setf *self-name*           yobidashi)
    (setf *other-partys-name*   hasshin)
    (setf *other-side-phone-id* hatsu-id)
    (call-request *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (call-response intelliphone-control) (chakushin)
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
  (format *control-unit-window-var* "~%{call-response ~a}" chakushin)
  (setf *other-partys-name* chakushin)
  (call-response *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (busy intelliphone-control) ()
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
    (format *control-unit-window-var* "~%{busy}")
    (busy *wake-up-task-analysis*)
    (setf *flagment-of-in-arc* t)))

(defmethod (code-send intelliphone-control) (mojiretsu)
  (when *flagment-of-in-arc*
    (setf *flagment-of-in-arc* nil)
    (when (typep mojiretsu 'string)
      (format *control-unit-window-var* "~%{code-send ~a}" mojiretsu)
      (setf *retrieval-condition* mojiretsu)
      (code-send *wake-up-task-analysis*))
    (setf *flagment-of-in-arc* t)))

;;; utility 機能の method
(defmethod (select-set-other-partys intelliphone-control) (name members-list &aux foo)
  (setf foo (member name members-list :test #'equal :key #'cadr))
  (when (null foo)
    (format *control-unit-window-var* *string-no-person-name* name)
    (setf foo members-list))
  (set-other-partys self (car foo)))

(defmethod (set-other-partys intelliphone-control) (attributes-list)
  (setf *other-partys-name* (nth 1 attributes-list))
  (setf *other-side-phone-id* (nth 2 attributes-list)))

;;;
;;; wake-up-task-analysis == 起動 task 分析 flavor
;;;    off-hook()
;;;    on-hook()
;;;    disconnect()
;;;    connect-request()
```

```
;;;    connect-response()
;;;    call-request()
;;;    call-response()
;;;    busy()
;;;    code-send()
;;;
(defflavor wake-up-task-analysis () ())

(defun wake-up-task-analysis-aux (current-status event &aux tmt foo)
  (setf foo (cond ((setf tmt (member (list current-status event)
                                     *wake-up-task-analysis-table*
                                     :test #'equal :key #'car))
                   (wake-up *task-execution-control* (cadar tmt)))
                  (t current-status)))
  (format *control-unit-window-var* "~%current status ~a" foo)
  foo)

(defmethod (off-hook wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'off-hook)))

(defmethod (on-hook wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'on-hook)))

(defmethod (disconnect wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'disconnect)))

(defmethod (connect-request wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'connect-request)))

(defmethod (connect-response wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'connect-response)))

(defmethod (call-request wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'call-request)))

(defmethod (call-response wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'call-response)))

(defmethod (busy wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'busy)))

(defmethod (code-send wake-up-task-analysis) ()
  (setf *status-number* (wake-up-task-analysis-aux *status-number* 'code-send)))

;;;
;;; task-execution-control == task 実行制御 flavor
;;;    wake-up(task-macro名) : "task-macro名"で指定された macro を実行する。返値は、次状態番号である。
;;;
(defflavor task-execution-control () ())

(defmethod (wake-up task-execution-control) (task-macro-name &aux foo)
  (format *control-unit-window-var* " \"~a\"" task-macro-name)
  (setf foo (member task-macro-name *task-macro-table* :test #'equal :key #'car))
  (if (null foo)
      (format *control-unit-window-var* *string-no-macro-name* task-macro-name)
      (do ((bar (cdar foo) (cdr bar)))
          ((equal (caar bar) 'task-end) (cadar bar))
        (if (equal (caar bar) 'cond)
            (setf bar (append '(nil) (eval (car bar))))
            (eval (car bar))))))

;;;
;;; speaker-control == speaker 制御 flavor
;;;    sp-task-1(文字列) : "文字列" (message) を speaker に送り出す。
;;;    sp-task-2() : BELL message を speaker に送り出す。
;;;    sp-task-3(検索結果list) : "検索結果list"の中にある名称と案内用 message を順に speaker に送り出す。
;;;
(defflavor speaker-control () ())

(defmethod (sp-task-1 speaker-control) (string-message)
  (format *control-unit-window-var* "~%[SP code-send ~a]" string-message)
  (code-send *speaker-unit-var* string-message))

(defmethod (sp-task-2 speaker-control) ()
  (format *control-unit-window-var* "~%[SP code-send ~a]" 'bell)
  (code-send *speaker-unit-var* 'bell))

(defmethod (sp-task-3 speaker-control) (retrieval-list)
```

```
  (dolist (foo retrieval-list)
    (format *control-unit-window-var* "~%[SP code-send ~a]"
            (format nil "~a~a" (nth 1 foo) (nth 4 foo)))
    (code-send *speaker-unit-var* (format nil "~a。~a。" (nth 1 foo) (nth 4 foo)))))

;;;
;;; code-generator-control == code generator 制御 flavor
;;;     cg-task-1() : 切断 message を CG に送り出す。
;;;     cg-task-2() : 着信応答 message を CG に送り出す。
;;;     cg-task-3() : 話中 message を CG に送り出す。
;;;     cg-task-4(電話番号) : "電話番号"に dial する。
;;;     cg-task-5(呼出者名 発信者名 発側電話番号) : 呼出要求 message を CG に送り出す。
;;;     cg-task-6(着信者名) : 呼出応答 message を CG に送り出す。
;;;
(defflavor code-generator-control () ())

(defmethod (cg-task-1 code-generator-control) ()
  (format *control-unit-window-var* "~%[CG disconnect]")
  (disconnect *cd-cg-unit-var*))

(defmethod (cg-task-2 code-generator-control) ()
  (format *control-unit-window-var* "~%[CG connect-response]")
  (connect-response *cd-cg-unit-var*))

(defmethod (cg-task-3 code-generator-control) ()
  (format *control-unit-window-var* "~%[CG busy]")
  (busy *cd-cg-unit-var*))

(defmethod (cg-task-4 code-generator-control) (phone-id)
  (format *control-unit-window-var* "~%[CG connect-request ~a]" phone-id)
  (connect-request *cd-cg-unit-var* phone-id))

(defmethod (cg-task-5 code-generator-control) (yobidashi hasshin hatsu-id)
  (format *control-unit-window-var* "~%[CG call-request ~a ~a ~a]" yobidashi hasshin hatsu-id
)
  (call-request *cd-cg-unit-var* yobidashi hasshin hatsu-id))

(defmethod (cg-task-6 code-generator-control) (chakushin)
  (format *control-unit-window-var* "~%[CG call-response ~a]" chakushin)
  (call-response *cd-cg-unit-var* chakushin))

;;;
;;; memory-control == 記憶部制御 flavor
;;;     mem-task-1() : 最後にかけた相手の情報 (last call 情報) を取り出し、共通 data に設定する。
;;;                       *other-partys-name* ..... last call 名称
;;;                       *other-side-phone-id* ... last call 番号
;;;     mem-task-2(相手名称 相手電話番号) : "相手名称"と"相手電話番号"とを last call 情報として記憶部に登録
;;;                                         する。
;;;     mem-task-3(条件1 条件2) : "条件1"と"条件2"との AND 条件により、記憶部を検索し、結果として得られる検索
;;;                               結果 list を共通 data に設定する。また、検索時の条件を共通 data の設定済
;;;                               条件に設定する。
;;;                                  *retrieval-response-list* ... 検索結果 list
;;;                                  *settled-condition* ......... 検索条件 (AND 条件1 条件2)
;;;
(defflavor memory-control () ())

(defmethod (mem-task-1 memory-control) (&aux name-phone)
  (setf name-phone            (last-call? *memory-unit-var*))
  (format *control-unit-window-var* "~%[MM last-call? ~a]" name-phone)
  (setf *other-partys-name*   (car name-phone))
  (setf *other-side-phone-id* (cadr name-phone)))

(defmethod (mem-task-2 memory-control) (aite-name aite-id)
  (format *control-unit-window-var* "~%[MM set-last-call ~a]" (list aite-name aite-id))
  (set-last-call *memory-unit-var* (list aite-name aite-id)))

(defmethod (mem-task-3 memory-control) (cond-1 cond-2)
  (setf *retrieval-response-list* (select? *memory-unit-var* (list cond-1 cond-2)))
  (format *control-unit-window-var* "~%[MM select? ~a]" *retrieval-response-list*)
  (setf *settled-condition*       (list cond-1 cond-2)))

;;; End Of File
```

CLM13:>kshima>intelliphone>memory-unit.lisp.20

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/04/87 21:21:36
Queued at: 8/07/90 18:59:05

```
;;; -*- Mode: LISP; Base: 10 -*-

;;; flavor :
;;;   memory-unit    ; intelliphone 記憶部
;;;
;;; instance-variable :
;;;   ROM-Name       ; ROM データファイルの名前
;;;   Card-Name      ; Card データファイルの名前
;;;   Your-NO        ; 端末の電話番号
;;;   Your-Name      ; 利用者の名前
;;;   Data-Base      ; データベース
;;;   Last-Call      ; 最後にかけたところ
(defflavor memory-unit (ROM-Name Card-Name Your-NO Your-Name Data-Base Last-Call) ())

;;; method :
;;;   set-ROM        ; ROMをセットする
;;;
;;; argument :
;;;   Name           ; ファイルの名前
(defmethod (set-ROM memory-unit) (Name)
  (let ()
    (setq ROM-Name Name)
    nil))

;;; method :
;;;   set-card       ; cardをセットする
;;;
;;; argument :
;;;   Name           ; ファイルの名前
(defmethod (set-card memory-unit) (Name)
  (let (Stream ROM-Data Card-Data)
    (setq Card-Name Name)
    (setq Stream (open ROM-Name :direction :input))
    (setq ROM-Data (read Stream))
    (close Stream)
    (setq Stream (open Card-Name :direction :input))
    (setq Card-Data (read Stream))
    (close Stream)
    (setq Your-NO   (car ROM-Data))
    (setq Your-Name (car Card-Data))
    (setq Data-Base (append (cadr ROM-Data) (cadr Card-Data)))
    (setq Last-Call (caddr Card-Data))
    nil))

;;; method :
;;;   get-card       ; cardを取り出す
;;;
;;; argument :
;;;   Name           ; ファイルの名前
(defmethod (get-card memory-unit) (&optional Name)
  (let (Stream Data)
    (setq Stream (open Card-Name :direction :input))
    (setq Data (read Stream))
    (close Stream)
    (setq Data (list (car Data) (cadr Data) Last-Call))
    (setq Stream (if (null Name)
                     (open Card-Name :direction :output)
                     (open Name      :direction :output)))
    (print Data Stream)
    (terpri Stream)
    (close Stream)
    nil))

;;; method :
;;;   your-name?     ; 利用者の名前は何か
(defmethod (your-name? memory-unit) ()
  Your-Name)

;;; method :
;;;   your-NO?       ; 端末の電話番号は何番か
(defmethod (your-NO? memory-unit) ()
  Your-NO)

;;; method :
;;;   select?        ; 指定した条件を持つレコードを選択する
```

```
;;;
;;; argument :
;;;   Condition     ; 条件

(defmethod (select? memory-unit) (Condition)
  (let ((select-result ($select Condition Data-Base)))
    (send *memory-unit-window-var* :set-reverse-video-p nil)
    (send *memory-unit-window-var* :clear-window)
    (print select-result *memory-unit-window-var*)
    (sleep 3.5)
    (send *memory-unit-window-var* :set-reverse-video-p t)
    select-result) )

(defun $select (C RL)
  (cond ((null RL) nil)
        (($selectp C (cadddr (car RL))) (cons (car RL) ($select C (cdr RL))))
        (t ($select C (cdr RL)))))

(defun $selectp (C PL)
  (cond ((null C) nil)
        ((eq (car C) ':not) ($selectp-not (cdr C) PL))
        ((eq (car C) ':and) ($selectp-and (cdr C) PL))
        ((eq (car C) ':or ) ($selectp-or  (cdr C) PL))
        (t ($selectp-and C PL))))

(defun $selectp-not (AL PL)
  (cond ((null AL) nil)
        ((listp (car AL)) (if ($selectp (car AL) PL) nil t))
        (($memberp (car AL) PL) nil)
        (t t)))

(defun $selectp-and (AL PL)
  (cond ((null AL) t)
        ((listp (car AL)) (if ($selectp (car AL) PL) ($selectp-and (cdr AL) PL) nil))
        (($memberp (car AL) PL) ($selectp-and (cdr AL) PL))
        (t nil)))

(defun $selectp-or  (AL PL)
  (cond ((null AL) nil)
        ((listp (car AL)) (if ($selectp (car AL) PL) t ($selectp-or (cdr AL) PL)))
        (($memberp (car AL) PL) t)
        (t ($selectp-or (cdr AL) PL))))

;;(defun $memberp (I L)
;;  (cond ((null L) nil)
;;        ((string-equal I (car L)) t)
;;        (t ($memberp I (cdr L)))))

(defun $memberp (I L)
  (member i l :test #'string-equal) )

;;; method :
;;;   last-call?    ; 最後にかけたところはどこか
(defmethod (last-call? memory-unit) ()
  Last-Call)

;;; method :
;;;   set-last-call ; 最後にかけたところを設定する
;;;
;;; argument :
;;;   Call          ; 呼出者の名前と電話番号
(defmethod (set-last-call memory-unit) (Call)
  (setq Last-Call Call))



(defun test-memory-unit (&aux mu a)
  (open-ntt-voice)
  (unwind-protect
      (progn
        (setq mu (make-instance 'memory-unit))
        (set-rom mu "clm06:>intelliphone>phone.jap")
        (set-card mu "clm06:>intelliphone>my-phone.jap")
        (print 'your-name?=)
        (print (your-name? mu))
        (print 'your-no?=)
        (print (your-no? mu))
```

```
        (set-last-call mu '("佐藤" "417"))
        (print 'last-call?=)
        (print (last-call? mu))
        (jpn-princ% "どこにおかけになりますか?。")
        (jpn-princ% "映画")
        (print 'select?=)
        (print (setq a (select? mu '("映画"))))
        (jpn-princ% "通話相手が複数みつかりました, 他に条件はありますか?。")
        (jpn-princ% "ドームシアター")
        (print 'select?=)
        (print (setq a (select? mu '("映画" "ドームシアター"))))
        (jpn-princ% 'bell)
        (jpn-princ% "ご希望の条件にあう所は、")
        (jpn-princ% (string-append (string (code-char (+ 48 (length a))))
                                   "個、あります。"))
        (do  ((i 1 (1+ i))
              (voice-word-list a (cdr voice-word-list)) )
             ((> i (length a)) 'end)
          (jpn-princ% (string-append (string (code-char (+ 48 i)))
                                     "番。"))
          (jpn-princ% (second (car voice-word-list)))
          (jpn-princ% (fifth (car voice-word-list)))
          (jpn-princ% "。") )
        (jpn-princ% "どこにおかけになりますか?。")
        (jpn-princ% "富士通ドームシアター。")
        (jpn-princ% "富士通ドームシアターに、電話します。すこしお待ちください。")
        (jpn-princ% "ピ、ポ、パ")
        )
    (get-card mu "clm06:>intelliphone>my-phone.jap") ;; before ending.
    (close-ntt-voice)
    ))


(or (functionp 'jpn-princ%)
    (load "clm06:>intelliphone>symbolics-to-ntt-voice"))
```

CLM13:>kshima>intelliphone>hook-switch-unit.lisp.3

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 6/30/87 16:19:18
Queued at: 8/07/90 18:51:48

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;; hook switch unit flavor definition.
(defflavor hook-switch-unit ((hook-switch-status 'on)) () )

(defmethod (status? hook-switch-unit) ()
;;  (print hook-switch-status)
  hook-switch-status)

(defmethod (set-hook-switch hook-switch-unit) ()
;;  (print hook-switch-status)
  (cond ((eq hook-switch-status 'on)
         (setf hook-switch-status 'off) )
        ((eq hook-switch-status 'off)
         (setf hook-switch-status 'on) )
        (t (setf hook-switch-status 'on) )))
```

CLM13:>kshima>intelliphone>dp3000v7.lisp.19

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 12/11/87 16:12:20
Queued at: 8/07/90 18:51:12

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;;<KSHIMA>SYMBOLIC-TO-DP-3000.LSP.1, 16-May-84 08:45:00, Edit by KSHIMA


(defvar *dp-3000-port*) ;; global variable for dp-3000 serial port.

(defvar *dp-3000-kanji-code-table*)          ;;漢字変換テーブル

;; flavor definition for mic unit (nec dp-3000)
(defflavor mic-unit (voice-switch) () )

(defmethod (voice-on mic-unit) ()
  (setf voice-switch 'on) )

(defmethod (voice-off mic-unit) ()
  (setf voice-switch 'off) )



(defun open-dp-3000 ()
  (setq *dp-3000-port*
        (si:make-serial-stream ':unit 2
                               ':force-output t
                               ':baud 2400.
                               ':number-of-data-bits 8.
                               ':parity :odd))
  (setq *dp-3000-kanji-code-table* '(
                                     ("SOUMU-NO-YAMAMOTO-SAN"."総務の林さん")
                                     ("KEIRI-NO-SUZUKI-SAN"."経理の佐藤さん")
                                     ("KEIRI"."経理")
                                     ("SUOMU"."総務")
                                     ("YAMAMOTO"."林")
                                     ("SUZUKI"."佐藤")
                                     ("KANTO"."カント")
                                     ("MARUKUSU"."マルクス")
                                     ("HEEGERU"."ヘーゲル")
                                     ("HAI"."はい")
                                     ("KENKYUSHITU"."研究室")
                                     ("EIGAKAN"."映画館")
                                     ("TOOEI"."東映")
                                     ("MEIGAZA"."名画座")
                                     ("DOOMU-SHIATA"."ドームシアター")
                                     ("KYOBASHI-AKADEMI"."京橋アカデミー")
                                     ("KYOBASHI-GEKIJYO"."京橋劇場")
                                     ("IMANANZI"."今何時")
                                     ("ASHITANO-TENKIWA"."明日の天気は")
                                     ("BUCYO"."部長")
                                     ("SHITUCYO"."室長")
                                     ("RICACYAN"."リカチャン")
                                     ("KATUZOU"."カツゾウ")
                                     ("TORAKCHI"."トラキチ")
                                     ("TETU"."テツ")
                                     ("YOKOTA"."横田")
                                     ("HAYASHI"."林")
                                     ("SATOU"."佐藤")
                                     ("NAKAZIMA"."中島")
                                     ("MONDEN"."門田")
                                     ("SHIMA"."島")
                                     ("SHIBAMOTO"."芝本")
                                     ("SHIMAKEN"."島健")
                                     ("TAKASHI"."隆")
                                     ("SYUNCYAN"."俊ちゃん")
                                     ("SANYO-DEN"."三洋電機")
                                     ("TOSYOSHITU"."図書室")
                                     ("171KAIGI-SHITU"."171会議室")
                                     ("SAKKINO-TOKORO"."さっきの所")
                                     ("ARIMASEN"."ありません"))))

;; close dp-3000 line
(defun close-dp-3000 ()
  (send *dp-3000-port* ':close) )


;; get dp-3000 status
(defun gl1 ()
```

```
  (prog (result data)
        (send *dp-3000-port* :force-output)
        (send *dp-3000-port* :clear-input)
     loop
        ;;store data till CR code
        (cond ((= (setq data (tyi%%%)) 13.))     ;; CR code
              (t (setq result (cons (code-char data) result))      ;;v7 ascii->code-char
                 (go loop)))
        (return (reverse result))))

;; off hook to dp-3000
(defun p1()
  (send *dp-3000-port* :force-output)
  (send *dp-3000-port* :clear-input)
  (tyo%%% 13.)
  (gl1))

;; (S T O U)--> "STOU"
(defun p2()
  (apply #'string-append (mapcar (lambda (x)
                                   (string x)) (p1))))


;; "STOU"-->get "stou"."佐藤"--->"佐藤"
(defun recog-from-dp-3000 ()
  (prog ((pp nil))
        (setq pp (cdr(assoc (p2) *dp-3000-kanji-code-table* :test #'equal)))    ;;v7 :test
        (return pp)))


;; from dp-3000
(defun tyi%%% ()
  (send *dp-3000-port* ':tyi))

;; to dp-3000
(defun tyo%%% (ch)
  (send *dp-3000-port* ':tyo ch)
  ch)


;; End of file.
```

CLM13:>kshima>intelliphone>cdcg.lisp.15

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 1/18/89 20:21:23
Queued at: 8/07/90 18:47:12

```
;;; -*- Mode: LISP; Base: 10. -*-

;;; インテリフォン CD/CG 部
;;;
;;; Programmed by 芝本 尚樹 (1987年06月30日 16時50分)
;;;
;;; 1. インスタンスの作り方
;;;    例) (setq *CDCG* (make-instance 'INTELLIPHONE-CDCG-PART
;;;                      :control-part (make-instance 'DUMMY-CONTROL-PART)))
;;;    インスタンス変数 control-part は、制御部のインスタンスで初期化する。
;;;    このインスタンスは、ハードウェア資源(シリアルポート)を持っているので、ゴミとして捨てないこと。
;;;
;;; 2. シリアルポートのオープンのし方
;;;    例) (send *CDCG* :create)
;;;
;;; 3. シリアルポートのクローズのし方
;;;    例) (send *CDCG* :kill)
;;;
;;; 4. シリアルポートの入力バッファのクリアのし方
;;;    例) (send (INTELLIPHONE-CDCG-PART-port *CDCG*) :clear-input)
;;;
;;; 5. インスタンスの初期化のし方
;;;    例) (initialize *CDCG*)
;;;    このジェネリック関数は、以下のことを行います。
;;;       i) 必要なインスタンス変数の初期化。
;;;      ii) シリアルポートの入力バッファのクリア。
;;;     iii) モデム電話機に対する初期化コマンドの送出。
;;;
;;; 6. シリアルポートのポーリングのし方
;;;    例) (polling *CDCG*)
;;;    インテリフォン実行中、約1秒に1回、このメソッドを起動すること。

;;;
;;; ELSE macro definition
;;; written 6/24/87 10:32:35
;;;
;;; Example
;;;  (if (null t)
;;;    (print "no")
;;;    (else (print "yes")))
(defmacro else (form)
  form)

;;;
;;; ASCII Code macro definition
;;; written 6/30/87 17:13:00
;;; but not used now in the program.
;;;
(defmacro ENQ () 5)
(defmacro ACK () 6)
(defmacro CR () 13)
;Command: (defun foo ()(print (list (ENQ) (ACK) (CR))))
;Command: (foo)
;(5 6 13)
;=> (5 6 13)
;Command: (disassemble 'foo)
;  0  ENTRY: 0 REQUIRED, 0 OPTIONAL
;  1  PUSH-IMMED 5
;  2  PUSH-IMMED 6
;  3  PUSH-IMMED 13
;  4  CALL-3-STACK #'SI:%LIST-3
;  5  CALL-1-RETURN #'PRINT

;;;
;;; From-JIS-to-3600
;;; written 7/01/87 11:44:31
;;;
;;; JIS コードのストリングを3600コードのストリングに変換する。
;;;
(defun From-JIS-to-3600 (data)
  (cond
    ((oddp (length data)) nil)
    ((string-equal data "") "")
    (t (let ((c1 (char-code (aref data 0)))
             (c2 (char-code (aref data 1))))
         (string-append
           (string (Japanese:JIS-to-char (+ (* 256 c1) c2) t))
```

```
          (From-JIS-to-3600 (substring data 2)))))))

;;;
;;; From-3600-to-JIS
;;; written 7/01/87 12:01:43
;;;
;;; 3600コードのストリングをJIS コードのストリングに変換する。
;;;
(defun From-3600-to-JIS (data)
  (cond
    ((string-equal data "") "")
    (t (let ((c1 (multiple-value-list
                   (floor (Japanese:char-to-JIS (aref data 0) t) 256) )))
         (string-append
           (string (code-char (car c1)))
           (string (code-char (cadr c1)))
           (From-3600-to-JIS (substring data 1)) )))))

;(From-JIS-to-3600 (From-3600-to-JIS "大阪市"))
;(From-JIS-to-3600 (From-3600-to-JIS "ABC123#$%東京;:<>,."))


;;;
;;; Serial Port
;;; written 6/28/87 18:09:15
;;;   -*- CLM01:>soft>nakajima>port.lisp.3 -*-
;;; updated 6/29/87 10:17:25
;;; updated 6/30/87 10:20:22
;;; updated 6/30/87 10:29:35
;;; updated 6/30/87 10:41:57
;;; updated 6/30/87 13:11:02
;;; updated 6/30/87 13:44:24
;;;
(defflavor SERIAL-PORT (port) ())
(defmethod (:create SERIAL-PORT) ()
  (setq port (si:make-serial-stream :unit 0
                                    :force-output t
                                    :baud 4800
                                    :number-of-data-bits 8
                                    :parity nil
                                    :number-of-stop-bits 1))
  (send port ':setf-data-terminal-ready t) )

(defmethod (:kill SERIAL-PORT) ()
  (close port))

(defmethod (:write-char SERIAL-PORT) (char)
  (send port :tyo (char-code char))
  )

(defmethod (:write-line SERIAL-PORT) (line)
  (cond ((string-equal "" line)
         (send self :write-char (code-char 13.)))  ;;#\return ;;edit by kshima 2-oct-87
        (t (progn
             (send self :write-char (aref line 0))
             (send self :write-line (substring line 1))
             (send self :write-char (code-char 13.) )
         ))))

(defmethod (:read-char SERIAL-PORT) ()
  (read-char port))

(defmethod (:read-char-no-hang SERIAL-PORT) ()
  (read-char-no-hang port))

(defmethod (:read-line SERIAL-PORT) ()
  (let ((char (read-char port)))
    (cond ((null char) nil)
          ((equal 13 char) nil)
          ((equal 5 char) (send self :read-line)) ;;; Ignoring ENQ
          (t (cons (code-char char) (send self :read-line))))))

;(defmethod (:read-line-2 SERIAL-PORT) ()
;  (let ((char (read-char port)))
;    (cond ((equal 13 (char-code char)) nil)
;          (t (cons char (send self :read-line-2))))))

;;;
```

```
;;; Serial Port Interface
;;; written 6/29/87 10:32:41
;;; updated 6/29/87 12:43:24
;;; updated 6/29/87 13:12:31
;;; updated 6/29/87 16:35:42

(defflavor SERIAL-PORT-INTERFACE () (SERIAL-PORT))

;;;
;;; Pass Message
;;; re-written 6/30/87 09:28:41
;;; updated 6/30/87 09:50:08
;;; updated 6/30/87 10:01:50
;;; updated 6/30/87 10:14:20
;;;
;;;  POLLING メソッドによって得たストリングメッセージを解析して、適切なメソッドを起動する。
;;;  メッセージの字句を分析して、コマンドと各パラメタに分ける。
(defmethod (pass-message SERIAL-PORT-INTERFACE) (string-message)
  (cond
    ((string-equal string-message "RNC" :end1 3)
       (let ((parm-list (cd-select-parms (substring string-message 4)
                                         '("TRT"))))
         (rnc self (car parm-list)))
       (polling self))
    ((string-equal string-message "RSC" :end1 3)
       (let ((parm-list (cd-select-parms (substring string-message 4)
                                         '("TRT"))))
         (rsc self (car parm-list)))
       (polling self))
    ((string-equal string-message "RDR" :end1 3)
       (rdr self (From-JIS-to-3600 (substring string-message 11)))
       (polling self))
    ((string-equal string-message "RFC" :end1 3)
       (let ((parm-list (cd-select-parms (substring string-message 4)
                                         '("FRT"))))
         (rfc self (car parm-list)))
       (polling self))
    ((string-equal string-message "RNN" :end1 3) (rnn self) (polling self))
    ((string-equal string-message "RSN" :end1 3) (rsn self) (polling self))
    ((string-equal string-message "RFN" :end1 3) (rfn self) (polling self))
    ((string-equal string-message "RNF" :end1 3) (rnf self) (polling self))
    ((string-equal string-message "RNI" :end1 3) (rni self) (polling self))
    ((string-equal string-message "RSF" :end1 3) (rsf self) (polling self))
    ((string-equal string-message "RSI" :end1 3) (rsi self) (polling self))
    (t ;;(print "Error in Message from Telephone" *cd-cg-unit-window-var*)
     nil)))

;;;
;;; POLLING
;;; re-written 6/30/87 09:34:13

(defmethod (polling SERIAL-PORT-INTERFACE) ()
  (let ((code (send self :read-char-no-hang)))
    (cond
      ((null code) nil)
      ((equal code 5) ;Check ENQ
         (send self :write-char (code-char 6)) ;Send ACK
         (let ((message (concatenate 'string (send self :read-line))))
           ;;(terpri *cd-cg-unit-window-var*)
           ;;(print (string-append "From Serial Port" message) *cd-cg-unit-window-var*)
           (pass-message self message)
           (string-append "From Serial Port" message)
           )))))

;;;
;;; Lexical Analyzer
;;; written 6/23/87 15:12:29
;;;
;;;   第1パラメタの文字列を、第2パラメタのデリミタで区切り、区切られた
;;; 要素をリストとして返す。
;;; Example
;;;  (delimit "pascal,fortran,c,lisp" ",")
;;;  => ("pascal" "fortran" "c" "lisp")
(defun delimit (string delimiter)
  (let ((delimit-pos (string-search  delimiter string)))
    (if (null delimit-pos) (list string)
        (cons (substring string 0 delimit-pos)
```

```
            (delimit (substring string (1+ delimit-pos))
                     delimiter)))))

;;;
;;; CD-select-parms
;;; specification written 6/24/87 10:04:06
;;; inplimentation written 6/24/87 14:16:29
;;; updated 6/28/87 17:16:41
;;; *** test not yet completed 6/24/87 17:03:15 ***
;;; *** test completed 6/28/87 17:29:25 ***
;;;
;;;   電話機からのコードのパラメタの並びを区切り、必要なパラメタだけを
;;; 選択する。第1パラメタでパラメタの並びの文字列を、第2パラメタで選択
;;; するパラメタのキーワードのリストを与える。返値は、指定キーワードと
;;; 同じ順序の実パラメタのリストである。
;;; Examples
;;;  (CD-SELECT-PARMS "B=3,A=1,F=23,D=12,E=6,C=2" '("B" "C" "D"))
;;;  => ("3" "2" "12")
;;;  (CD-SELECT-PARMS "LENGTH=123,AID=INTELLIPHONE,ANSWER=YES,PROGRAMMER=SHIBAMOTO"
;;;                   '("ANSWER" "LENGTH" "AID"))
;;;  => ("YES" "123" "INTELLIPHONE")
;;;  (CD-SELECT-PARMS "LENGTH=123,AID=INTELLIPHONE,ANSWER=YES,PROGRAMMER=SHIBAMOTO"
;;;                   '("PROGRAMMER" "LENGTH" "AID"))
;;;  => ("SHIBAMOTO" "123" "INTELLIPHONE")

(defun cd-select-parms-match-key (list key)
  (if (null list) nil
      (if (string-equal key (car list) :end2 (length key))
          (substring (car list) (length key))
          (else (cd-select-parms-match-key (cdr list) key)))))
(defun cd-select-parms-make-list (parms keywords)
  (if (null keywords) nil
      (cons
        (let ((key (string-append (car keywords) "=")))
          (cd-select-parms-match-key parms key))
        (cd-select-parms-make-list parms (cdr keywords)))))
(defun cd-select-parms (string keywords)
  (let ((parms (delimit string ",")))
    (cd-select-parms-make-list parms keywords)))

;;;
;;; 5-digits
;;; written 6/24/87 09:40:56
;;;
;;;   正整数を文字列に変換して、上位桁にゼロをつめる。
;;; Example
;;;  (5-digits 36)
;;;  => "00036"
(defun 5-digits (integer)
  (let ((intermed (string-append "00000"
                                 (format nil "~D" integer))))
    (substring intermed (- (length intermed) 5))))

;;;
;;; BASIC-PROTOCOL-HANDLER
;;; written 6/23/87 14:06:18
;;;
(defflavor BASIC-PROTOCOL-HANDLER (present-status)
           ()
  :initable-instance-variables)

;;;
;;; CG-CONNECT-REQUEST
;;; written 6/23/87 14:06:53
;;; updated 6/24/87 09:28:32
;;; updated 6/24/87 11:38:24
;;; updated 6/24/87 17:05:06
;;; updated 6/29/87 11:32:14
;;; updated 6/29/87 11:55:53
;;; updated 6/29/87 14:11:53
;;; updated 6/30/87 21:56:12

(defflavor CG-CONNECT-REQUEST (present-status hook-switch)
           ()
  (:required-methods serial-port-output))

(defmethod (connect-request CG-CONNECT-REQUEST) (p1)
```

```
  (case present-status
    ((:S00) (setq hook-switch ':off)
            (setq present-status ':S00)
            (serial-port-output self (string-append "CNR TEL=" p1)))
    (otherwise (print "Status error in CG-CONNECT-REQUEST" *cd-cg-unit-window-var*)
               ;Error message for test
               (describe self))))

;;;
;;; CG-CALL-REQUEST
;;; written 6/23/87 14:55:48
;;; updated 6/24/87 09:30:24
;;; updated 6/24/87 11:38:37
;;; updated 6/24/87 17:07:43
;;; updated 6/29/87 11:32:44
;;; updated 6/29/87 14:12:04
;;; updated 6/30/87 21:57:07
;;; updated 7/01/87 12:59:38
;;;
(defflavor CG-CALL-REQUEST (present-status)
           ()
  (:required-methods serial-port-output))

(defmethod (call-request CG-CALL-REQUEST) (p1 p2 p3)
  (case present-status
    ((:S20) (setq present-status ':S20)
            (serial-port-output self
              (let ((data
                      (From-3600-to-JIS
                        (string-append "CALL-REQUEST " p1 "," p2 "," p3))))
                (string-append "CDT Y" (5-digits (1+ (length data))) ;CR の1文字を含む。
                               ":" data)))) ;;; 長さの制限を無視する.
    (otherwise (print "Status error in CG-CALL-REQUEST" *cd-cg-unit-window-var*)
               ;Error message for test
               (describe self))))

;;;
;;; CG-CONNECT-RESPONSE
;;; written 6/23/87 15:50:14
;;; updated 6/24/87 09:49:50
;;; updated 6/24/87 17:10:27
;;; updated 6/28/87 17:40:39
;;; updated 6/29/87 11:33:14
;;; updated 6/29/87 14:12:19
;;; updated 6/30/87 13:15:15
;;; updated 6/30/87 21:57:51
;;;
(defflavor CG-CONNECT-RESPONSE (present-status)
           ()
  (:required-methods serial-port-output))

(defmethod (connect-response CG-CONNECT-RESPONSE) ()
  (case present-status
    ((:S31) (setq present-status ':S32))
            ; State S32 : Only "CONNECT-RESPONSE" has arrived.
    ((:S33) (setq present-status ':S40)
            (serial-port-output self "CSA SRR=Y,AID=INTELLIPHONE"))
    (otherwise (print "Status error in CG-CONNECT-RESPONSE" *cd-cg-unit-window-var*)
               ;Error message for test
               (describe self))))

;;;
;;; CG-CALL-RESPONSE
;;; written 6/23/87 16:01:19
;;; updated 6/24/87 09:51:20
;;; updated 6/24/87 11:38:52
;;; updated 6/24/87 17:11:18
;;; updated 6/29/87 11:33:38
;;; updated 6/29/87 11:57:00
;;; updated 6/29/87 14:12:35
;;; updated 6/29/87 14:41:21
;;; updated 6/30/87 22:01:22
;;; updated 7/01/87 13:00:42
;;;
(defflavor CG-CALL-RESPONSE (present-status hook-switch)
           ()
  (:required-methods serial-port-output))
```

```
(defmethod (call-response CG-CALL-RESPONSE) (p1)
  (case present-status
    ((:S40) (setq hook-switch ':off)
            (setq present-status ':S40)
            (serial-port-output self
              (let ((data
                      (From-3600-to-JIS
                        (string-append "CALL-RESPONSE " p1))))
                (string-append "CDT Y" (5-digits (1+ (length data)));CR の1文字を含む。
                               ":" data)))) ;;;長さの制限を無視する.
    (otherwise (print "Status error in CG-CALL-RESPONSE" *cd-cg-unit-window-var*) ;Error mess
age for test
               (describe self))))

;;;
;;; CG-BUSY
;;; written 6/23/87 16:29:43
;;; updated 6/24/87 09:53:21
;;; updated 6/24/87 17:16:25
;;; updated 6/28/87 17:42:05
;;; updated 6/29/87 11:34:14
;;; updated 6/29/87 14:13:09
;;; updated 6/29/87 14:41:54
;;; updated 6/30/87 22:02:39
;;;
(defflavor CG-BUSY (present-status)
           ()
  (:required-methods serial-port-output))

(defmethod (busy CG-BUSY) ()
  (case present-status
    ((:S31) (setq present-status ':S34))
            ; State S34 : Only "BUSY" has arrived.
    ((:S33) (setq present-status ':S35)
            ; State S35 : Waiting for "RNI SAB=Y"
            ;             'Control part busy' sequence
            (serial-port-output self "CSA SRR=N,AID=INTELLIPHONE"))
    (otherwise (print "Status error in CG-BUSY" *cd-cg-unit-window-var*) ;Error message for t
est
               (describe self))))

;;;
;;; CG-DISCONNECT
;;; written 6/29/87 11:58:52
;;; updated 6/29/87 14:14:47
;;; updated 6/29/87 14:42:46
;;; updated 6/30/87 22:04:42
;;; updated 7/03/87 14:05:48
;;;
(defflavor CG-DISCONNECT (present-status hook-switch)
           ()
  (:required-methods serial-port-output))

(defmethod (disconnect CG-DISCONNECT) ()
  (case present-status
    ((:S00) (setq hook-switch ':on)
            (setq present-status ':S00)
            ;;(serial-port-output self "CNQ")
            (serial-port-output self "LCL"))  ;; reset telephone ;; OKUNOTE!!
    ((:S10) (setq hook-switch ':on)
            (setq present-status ':S10)
            (serial-port-output self "CNQ"))
    ((:S20) (setq hook-switch ':on)
            (setq present-status ':S20)
            (serial-port-output self "CSQ"))
    ((:S31 :S32 :S33 :S34 :S35)
            (setq hook-switch ':on)
            (serial-port-output self "CNQ"))
    ((:S40) (setq hook-switch ':on)
            (setq present-status ':S40)
            (serial-port-output self "CSQ"))
    ((:S50) (setq hook-switch ':on)
            (setq present-status ':S50)
            (serial-port-output self "CFC")
            (serial-port-output self "CSQ"))
    (otherwise (print "Status error in CG-DISCONNECT" *cd-cg-unit-window-var*)
               ;Error message for test
               (describe self))))
```

```
;;;
;;; CD-RNC
;;; written 6/23/87 16:40:21
;;; updated 6/24/87 09:54:07
;;; updated 6/28/87 17:45:29
;;; updated 6/29/87 11:34:38
;;; updated 6/29/87 14:15:23
;;; updated 6/29/87 14:43:01
;;; updated 6/29/87 16:16:10
;;; updated 6/30/87 22:06:48
;;;
(defflavor CD-RNC (present-status control-part)
           ()
  (:required-methods serial-port-output))

(defmethod (rnc CD-RNC) (trt)
  (case present-status
    ((:S00) (cond
              ((string-equal trt "E")
                 (setq present-status ':S10)
                 (serial-port-output self "CSR AID=INTELLIPHONE"))
              ((or (string-equal trt "B")
                   (string-equal trt "N"))
                 (setq present-status ':S00)
                 (busy control-part)))) ;;; Send "BUSY" to control-part.
            (otherwise (print "Status error in CD-RNC" *cd-cg-unit-window-var*) ;Error messag
e for test
                       (describe self))))

;;;
;;; CD-RSC
;;; written 6/23/87 16:53:32
;;; updated 6/28/87 17:48:58
;;; updated 6/29/87 11:35:12
;;; updated 6/29/87 16:20:38
;;; updated 6/30/87 11:11:35
;;; updated 6/30/87 21:47:52
;;;
(defflavor CD-RSC (present-status control-part)
           ())
(defmethod (rsc CD-RSC) (trt)
  (case present-status
    ((:S10) (cond
              ((string-equal trt "E")
                 (setq present-status ':S20)
                 (connect-response control-part))
                 ;;; Send "CONNECT-RESPONSE" to control-part.
              ((or (string-equal trt "B")
                   (string-equal trt "N"))
                 (setq present-status ':S10))))
    (otherwise (print "Status error in CD-RSC" *cd-cg-unit-window-var*) ;Error message for te
st
               (describe self))))

;;;
;;; CD-RDR
;;; written 6/23/87 17:00:17
;;; updated 6/24/87 10:54:57
;;; updated 6/24/87 11:41:05
;;; updated 6/28/87 17:50:51
;;; updated 6/29/87 11:27:37
;;; updated 6/29/87 11:35:45
;;; updated 6/29/87 14:16:41
;;; updated 6/29/87 14:43:31
;;; updated 6/30/87 10:06:05
;;; updated 6/30/87 22:08:33
;;;
(defflavor CD-RDR (present-status responding-person control-part)
           ()
  (:required-methods serial-port-output))

(defmethod (rdr CD-RDR) (data)
  (let ((appl-command (substring data 0
                                 (string-search " " data)))
        (appl-parms (delimit (substring data (1+ (string-search " " data))) ",")))
    (case present-status
      ((:S40) (if (string-equal appl-command "CALL-REQUEST")
                  (let ((p1 (nth 0 appl-parms))
```

```
                      (p2 (nth 1 appl-parms))
                      (p3 (nth 2 appl-parms)))
                (setq present-status ':S40)
                (call-request control-part p1 p2 p3))
                ;;; Send "CALL-REQUEST" to control-part.
               (else (print "Status error : CALL-REQUEST" *cd-cg-unit-window-var*)))) ;Erro
r message for test
      ((:S20) (if (string-equal appl-command "CALL-RESPONSE")
                  (progn
                    (setq responding-person (car appl-parms))
                    (setq present-status ':S20)
                    (serial-port-output self "CFR"))
                  (else (print "Status error : CALL-RESPONSE" *cd-cg-unit-window-var*)))) ;Er
ror message for test
      (otherwise (print "Status error in CD-RDR" *cd-cg-unit-window-var*) ;Error message for
test
                 (describe self)))))

;;;
;;; CD-RFC
;;; written 6/24/87 11:10:13
;;; updated 6/24/87 13:48:07
;;; updated 6/28/87 17:57:41
;;; updated 6/29/87 11:36:16
;;; updated 6/29/87 16:20:23
;;; updated 6/30/87 22:10:24
;;;
(defflavor CD-RFC (present-status responding-person control-part)
           ())
(defmethod (rfc CD-RFC) (frt)
  (case present-status
    ((:S20) (if (string-equal frt "Y")
                (progn
                  (setq present-status ':S50)
                  (call-response control-part responding-person))
                  ;;; Send "CALL-RESPONSE" to control-part.
                (else (print "Message parameter error : CALL-RESPONSE" *cd-cg-unit-window-var
*))))
    (otherwise (print "Status error in CD-RFC" *cd-cg-unit-window-var*) ;Error message for te
st
               (describe self))))

;;;
;;; CD-RNN
;;; written 6/24/87 13:55:17
;;; updated 6/28/87 17:59:21
;;; updated 6/29/87 11:36:36
;;; updated 6/29/87 12:47:37
;;; updated 6/29/87 14:17:55
;;; updated 6/29/87 14:43:54
;;; updated 6/30/87 22:11:38
;;;
(defflavor CD-RNN (present-status control-part hook-switch)
           ()
  (:required-methods serial-port-output))

(defmethod (rnn CD-RNN) ()
  (case present-status
    ((:S00) (case hook-switch
              ((:on) (setq present-status ':S31)
                     (connect-request control-part))
                     ;;; Send "CONNECT-REQUEST" to control-part.
              ((:off) (serial-port-output self "CNQ"))))
    (otherwise (print "Status error in CD-RNN" *cd-cg-unit-window-var*) ;Error message for te
st
               (describe self))))

;;;
;;; CD-RSN
;;; written 6/24/87 14:00:43
;;; updated 6/24/87 17:17:04
;;; updated 6/28/87 18:05:00
;;; updated 6/29/87 11:37:03
;;; updated 6/29/87 14:18:57
;;; updated 6/29/87 14:44:16
;;; updated 6/30/87 22:12:49
;;;
(defflavor CD-RSN (present-status)
```

```
          ()
  (:required-methods serial-port-output))

(defmethod (rsn CD-RSN) ()
  (case present-status
    ((:S31) (setq present-status ':S33))
            ; Status S33 : Only RSN has arrived.
    ((:S32) (setq present-status ':S40)
            (serial-port-output self "CSA SRR=Y,AID=INTELLIPHONE"))
    ((:S34) (setq present-status ':S35)
            ; Status S35 : Waiting for "RNI SAB=Y"
            ;              'Control part busy' sequence
            (serial-port-output self "CSA SRR=N,AID=INTELLIPHONE"))
    (otherwise (print "Status error in CD-RSN" *cd-cg-unit-window-var*) ;Error message for te
st
               (describe self))))

;;;
;;; CD-RFN
;;; written 6/24/87 14:04:08
;;; updated 6/28/87 18:06:17
;;; updated 6/29/87 11:37:37
;;; updated 6/29/87 14:19:26
;;; updated 6/29/87 14:44:30
;;; updated 6/30/87 22:13:41
;;;
(defflavor CD-RFN (present-status)
          ()
  (:required-methods serial-port-output))
(defmethod (rfn CD-RFN) ()
  (case present-status
    ((:S40) (setq present-status ':S50)
            (serial-port-output self "CFA FRR=Y"))
    (otherwise (print "Status error in CD-RFN" *cd-cg-unit-window-var*) ;Error message for te
st
               (describe self))))

;;;
;;; CD-RNF
;;; written 6/29/87 12:59:16
;;; updated 6/30/87 11:52:09
;;; updated 6/30/87 13:30:26
;;; updated 6/30/87 22:15:00
;;;
(defflavor CD-RNF (present-status hook-switch control-part)
          ())
(defmethod (rnf CD-RNF) ()
  (case hook-switch
    ((:on) (setq present-status ':S00))
    ((:off) (case present-status
              ((:S10 S20)
                 (setq present-status ':S00)
                 (busy control-part)) ;;; Send "BUSY" to control-part.
              ((:S31 :S32 :S33 :S34 :S35 :S40 :S50)
                 (setq present-status ':S00)
                 (disconnect control-part)) ;;; Send "DISCONNECT" to control-part.
              (otherwize
                 (print "Status Error in CD-RNF" *cd-cg-unit-window-var*)
                 (describe self))))))

;;;
;;; CD-RNI
;;; written 6/29/87 13:03:57
;;; updated 6/30/87 11:55:07
;;; updated 6/30/87 13:30:34
;;; updated 6/30/87 22:15:52
;;;
(defflavor CD-RNI (present-status hook-switch control-part)
          ())
(defmethod (rni CD-RNI) ()
  (case hook-switch
    ((:on) (setq present-status ':S00))
    ((:off) (case present-status
              ((:S10 S20)
                 (setq present-status ':S00)
                 (busy control-part)) ;;; Send "BUSY" to control-part.
              ((:S31 :S32 :S33 :S34 :S35 :S40 :S50)
                 (setq present-status ':S00)
```

```
              (disconnect control-part)) ;;; Send "DISCONNECT" to control-part.
            (otherwize
              (print "Status Error in CD-RNI" *cd-cg-unit-window-var*)
              (describe self))))))

;;;
;;; CD-RSF
;;; written 6/29/87 13:05:20
;;; updated 6/30/87 15:08:48
;;; updated 6/30/87 22:21:29
;;;
(defflavor CD-RSF (present-status hook-switch)
           ())
(defmethod (rsf CD-RSF) ()
  (case present-status
    ((:S20) (setq present-status ':S10))
    ((:S40) (setq present-status ':S31))
    ((:S50) (setq present-status ':S50))
    (t (print "Status error in CD-RSF" *cd-cg-unit-window-var*) (describe self))))


;;;
;;; CD-RSI
;;; written 6/29/87 13:10:37
;;; updated 6/30/87 15:10:33
;;; updated 6/30/87 22:21:47
;;;
(defflavor CD-RSI (present-status hook-switch)
           ())
(defmethod (rsi CD-RSI) ()
  (case present-status
    ((:S20) (setq present-status ':S10))
    ((:S40) (setq present-status ':S31))
    ((:S50) (setq present-status ':S50))
    (t (print "Status error in CD-RSI" *cd-cg-unit-window-var*) (describe self))))

;;;
;;;
;;;
(defflavor INTELLIPHONE-CDCG-PART (control-part
                                   (present-status :S00)
                                   port
                                   (hook-switch :on)
                                   responding-person)
           (BASIC-PROTOCOL-HANDLER SERIAL-PORT-INTERFACE
            CG-CONNECT-REQUEST CG-CALL-REQUEST CG-CONNECT-RESPONSE
            CG-CALL-RESPONSE CG-BUSY CG-DISCONNECT
            CD-RNC CD-RSC CD-RDR CD-RFC CD-RNN CD-RSN CD-RFN
            CD-RNF CD-RNI CD-RSF CD-RSI)
;  (:initable-instance-variables control-part port)
  (:initable-instance-variables) ;
  (:readable-instance-variables) ; For test, declaring "writable"
  (:writable-instance-variables) ;
  (:required-methods serial-port-output))

;;;
;;; Initialize CD/CG part
;;; written 6/29/87 11:08:58
;;; updated 6/30/87 15:35:38
;;; updated 6/30/87 16:03:53
;;;
(defmethod (initialize INTELLIPHONE-CDCG-PART) ()
  (setq present-status ':S00)
  (setq hook-switch ':on)
  (setq responding-person "")
  (send port :clear-input)
  (serial-port-output self "LCL"))

;;;
;;; Serial-Output
;;; written 6/24/87 09:25:10
;;; updated 6/29/87 14:21:50
;;; updated 6/29/87 16:26:03
;;; updated 6/30/87 15:32:08
;;;

;;; -*- 単体テスト用 -*-
(defmethod (serial-port-output INTELLIPHONE-CDCG-PART)
```

```
                          (string &optional (destination *standard-output*))
  (with-open-stream (stream destination)
            ; 単体テスト時には *standard-output* を用いる。
            (send stream :string-out string)))

;;; -*- 結合テスト用 -*-
;; 結合テスト時には シリアルポートを用いる。
(defmethod (serial-port-output INTELLIPHONE-CDCG-PART) (string)
  (send self :write-char (code-char 5)) ; Send ENQ
  (do ((baka (send self :read-char) (send self :read-char)) )
      ((equal 6 baka) t)
    (print baka) ) ; Check ACK
  (send self :write-line string)
  (send *cd-cg-unit-window-var* :set-reverse-video-p nil)
  (terpri *cd-cg-unit-window-var*)
  (print (string-append "To Serial Port : " string) *cd-cg-unit-window-var*)
  (sleep 2.)
  (send *cd-cg-unit-window-var* :set-reverse-video-p t)
  (string-append "To Serial Port : " string)
  )


;(defun test-out (string)
;  (print "ENQ")
;  (do ()
;      ((equal (read-char) #\R)))
;  (print string))

;;;
;;; These are Codes for Modular Test (CD/CG part) below;
;;; written 6/29/87 13:45:26
;;; updated 6/30/87 15:12:34
;;;

(defflavor DUMMY-CONTROL-PART () ())
(defmethod (connect-request DUMMY-CONTROL-PART) ()
  (print "CONNECT-REQUEST : to control part" *cd-cg-unit-window-var*))
(defmethod (connect-response DUMMY-CONTROL-PART) ()
  (print "CONNECT-RESPONSE : to control part" *cd-cg-unit-window-var*))
(defmethod (call-request DUMMY-CONTROL-PART) (p1 p2 p3)
  (print "CONNECT-RESPONSE : to control part" *cd-cg-unit-window-var*)
  (print (string-append "   parameter-1 " p1) *cd-cg-unit-window-var*)
  (print (string-append "   parameter-2 " p2) *cd-cg-unit-window-var*)
  (print (string-append "   parameter-3 " p3) *cd-cg-unit-window-var*))
(defmethod (call-response DUMMY-CONTROL-PART) (p1)
  (print "CALL-RESPONSE : to control part" *cd-cg-unit-window-var*)
  (print (string-append "   parameter-1 " p1) *cd-cg-unit-window-var*))
(defmethod (busy DUMMY-CONTROL-PART) ()
  (print "BUSY : to control part" *cd-cg-unit-window-var*))
(defmethod (disconnect DUMMY-CONTROL-PART) ()
  (print "DISCONNECT : to control part" *cd-cg-unit-window-var*))

;(setq *CDCG* (make-instance 'INTELLIPHONE-CDCG-PART
;                            :control-part (make-instance 'DUMMY-CONTROL-PART)))


;(send *CDCG* :create) ; create a serial port
```

CLM13:>kshima>intelliphone>intelliphone-header-print.lisp.13

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 5/13/88 14:53:07
Queued at: 8/07/90 18:52:41

```
;;; -*- Mode: LISP;  base: 10;  syntax: common-lisp -*-

;; [NTT-20]PS:<KSHIMA>FRAME-WNDW.LISP.1,
;; 26-Aug-86 08:55:00 , Edit by Kibota Teruhiko

;;kg2:>kshima>frame-wndw.lisp.1
;;; -*- mode: lisp; package: user ; base: 10; lowercase: yes ; fonts: cptfontb -*-


(defvar *intelliphone-header-window*)

(defun intelliphone-attention ()
  (if (boundp '*intelliphone-header-window*)
      (send *intelliphone-header-window* :kill) )
  (make-intelliphone-header-window)
  ;;(send *intelliphone-header-window* ':expose-near '(:mouse))
  (unwind-protect
      (intelliphone-header-window-print
        (read-kanji-file "clm02:>intelliphone>intelliphone-header.jap") )
    (send *intelliphone-header-window* :kill)
    'ok))


;; make-intelliphone-header-window
(defun make-intelliphone-header-window ()
  (setq *intelliphone-header-window*
        (tv:make-window 'intelliphone-header-window
                        ':edges-from '(250 50 750 250)
                        ':expose-p t
                        ':more-p nil
                        ':name "Intelliphone header window"
                        :save-bits t
                        )))


(defflavor intelliphone-header-window ()
           (tv:process-mixin tv:window)
  (:default-init-plist
   :border-margin-width 2
   :process '(intelliphone-header-window)
   ))


(defun intelliphone-header-window (window)
  (let ((*terminal-io* window)
        (input-file (open "clm02:>intelliphone>intelliphone-header.jap"))
        )
    (unwind-protect
        (jpn-princ% (read-line input-file))
      (close input-file) )
    ;;(type% "clm02:>intelliphone>intelliphone-header.jap")
    ))


(defun read-kanji-file (kanji-file)
  ;; kanji-file= "clm02:>intelliphone>intelliphone-header.jap"
  ;; return= read kanji file, and push it and return
  (let ((kanji-file-port (open kanji-file)))
    (do ((read-kanji (read-line kanji-file-port nil 'eof)
                     (read-line kanji-file-port nil 'eof))
         (read-kanji-list) )
        ((eq read-kanji 'eof)
         (close kanji-file-port)
         (reverse read-kanji-list) )
      (push read-kanji read-kanji-list) )))



(defvar intelliphone-header-reverse-p)

;; intelliphone-header-window-print
(defun intelliphone-header-window-print (kanji-string-list)
  (let ((old-terminal-io *terminal-io*))
    (unwind-protect
        (let ((*terminal-io* *intelliphone-header-window*))
          (send *terminal-io* :select)
          (send *terminal-io* :clear-window)
          ;;(japanese:jis-fonts *terminal-io* 16 2)
```

```
          (setq intelliphone-header-reverse-p nil)
          (do ((i 1 (1+ i)))
              ((send *terminal-io* :listen)
               'ok )
            (send *terminal-io* :refresh)
            (send *terminal-io* :home-cursor)
            (dolist (intelliphone-header-string kanji-string-list)
              (princ intelliphone-header-string)
              (terpri) )
            (random-walk-window-untill-tyi)
            ))
        ;;(send *intelliphone-header-window* :deactivate)
        (setq *terminal-io* old-terminal-io)
        (send *terminal-io* :select) )))


;; (random-walk-window-untill-tyi)
(defun random-walk-window-untill-tyi ()
  (let ((new-x (make-new-position (abs (random 1000.)) 559.))
        (new-y (make-new-position (abs (random 1000.)) 529.)) )
    (sleep 5.)
    (send *terminal-io* :set-reverse-video-p
          (setq intelliphone-header-reverse-p (not intelliphone-header-reverse-p)))
    (send *terminal-io* :set-position new-x new-y) ))

(defun make-new-position (x max-x)
  ;; x= 1000
  ;; max-x= 500
  ;; return= 500 (below max-x)
  (+ (mod x max-x) 64. )) ;; for version 7 dynamic window size
                                          ;; max is (64, 43) --- (1131, 793)


;; End of file.
```

CLM13:>kshima>intelliphone>symbolics-to-ntt-voice.lisp.10

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/04/87 19:54:41
Queued at: 8/07/90 19:01:50

```
;;; -*- Mode: LISP; Base: 10; Syntax: Common-lisp -*-

;;<KSHIMA>SYMBOLIC-TO-NTT-VOICE.LSP.1, 16-May-84 08:45:00, Edit by KSHIMA


(defvar *ntt-voice-port*) ;; global variable for ntt-voice serial port.

;; speaker unit flavor definition
(defflavor speaker-unit () () )

(defmethod (code-send speaker-unit) (voice-code)
  (send *speaker-unit-window-var* :set-reverse-video-p nil)
  (jpn-princ% voice-code *speaker-unit-window-var*)
  (sleep 2.)
  (send *speaker-unit-window-var* :set-reverse-video-p t)
  )


(defun open-ntt-voice ()
  (setq *ntt-voice-port*
        (si:make-serial-stream ':unit 1
                               ':force-output t
                               ':baud 9600.
                               ':number-of-data-bits 7.
                               ':parity :even)))

(defun close-ntt-voice ()
  (send *ntt-voice-port* ':close) )

(defun test-output-for-ntt-voice ()
  (send *ntt-voice-port* :force-output)
  (do ((i 1 (1+ i)))
      ((> i 1000.) (print 'ok))
    (send *ntt-voice-port* ':tyo 101) )) ;; all "e"

(defun one-line-read-from-ntt-voice ()
  (prog (result data)
  (send *ntt-voice-port* :force-output)
  (send *ntt-voice-port* :clear-input)
     loop
        ;;store data till CR code
        (cond ((= (setq data (tyi%)) 13.))     ;; CR code
              (t (setq result (cons (zl:ascii data) result))
                 (go loop)))
        (return (reverse result))))


(defun tyi% ()
  ;;(sleep 0.1)
  ;;(read-char-no-hang *ntt-voice-port*)
;;  (send *ntt-voice-port* ':tyi-no-hang)
  (send *ntt-voice-port* ':tyi)
;; error-code= #x46 ;; 70.(decimal)
  )

(defun tyo% (ch)
;;  (print ch)
  (send *ntt-voice-port* ':tyo ch)
;;  (send *terminal-io* ':tyo (character (zl:ascii ch)))
  ch)

(defun princ% (text)
  (princ text *ntt-voice-port*) )

(defun terpri% ()
  (tyo% 10.)
  (tyo% 13.) )

(defun print% (text)
  (terpri%)
  (princ% text) )

(defun type% (file)
  (let (responce-code)
    (dolist (a (read-lm-kanji-file file))
      (jpn-princ% a)
;;      (setq responce-code (tyi%))
;;      (send *ntt-voice-port* :clear-input)
```

```
;;      (if (= responce-code #x50)  ;; normal end.
;;        (print 'ok)
;;        ;;(tyi%) ;; #x03  ;; normal end.
;;        (print 'error!!)
          )))


(defun translate-lm-char-to-jis-code (lm-char-list)
  ;; lm-char-list= LM-code(lisp machine code) kanji string list.
  ;; return= jis-code kanji list
  ;; for example, char-list= '(32 32 2816 2823 2886 2819 2682 141...)
  ;; blank, blank, KRINE(kanji zenkaku) return-code....
  ;; ==> '(32 32 9035(#x234B= #K(ascii 2 byte) ...)
  (let ((kanji-length (length lm-char-list)))
    (do ((i 0 (1+ i))
         ;; for example, char-list= '(32 32 2816 2823 2886 2819 2682 141...)
         ;; blank, blank, KRINE(kanji zenkaku) return-code....
         (char-code)
         (jis-code-list))
        ((> i (- kanji-length 1)) (reverse jis-code-list))
      (setq char-code (japanese:char-to-jis (character (subseq lm-char-list i (1+ i)))))
      (cond (char-code
             (push char-code jis-code-list) )))))

(defun jpn-princ% (lm-kanji-string &optional (output-port *terminal-io*) &aux responce-code)
  ;; lm-kanji-string= LM-code(lisp machine code) kanji string list.
  ;; for example, "abc..." lm-code
  ;; return= jis-code output to voice
  (send *ntt-voice-port* :clear-input)
  (if (eq lm-kanji-string 'bell)
      (progn (tyo% #x20) (tyo% #x12) (tyo% 3) )
      (progn
        (tyo% #x20) ;; command for ntt-voice, kanji voice output
        (tyo% #x1b) ;; escape-code
        (tyo% #x24) ;; $
        (tyo% #x40) ;; @
        (do ((char-list (translate-lm-char-to-jis-code lm-kanji-string) (cdr char-list))
             ;; char-list= '(32 32 9035(#x234B) 9042 ... 13 ) for example,
             ;; blank, blank, KRINE(jis kanji zenkaku) return-code....
             )
            ((null char-list) lm-kanji-string)
          (cond ((= (car char-list) 13.) ;; return code
                 (terpri%) )
                ((= (car char-list) 32.)  ;; blank code
                 nil )
                ((= (car char-list) 8557.)  ;; double quote code
                 nil )
                (t ;; 9035= #x234B ==> #, K
                 (multiple-value-bind (first-byte second-byte)
                     (floor (car char-list) 256.)
                   (tyo% first-byte)
                   (tyo% second-byte) ))))
        (tyo% 27.) ;; escape-code
        (tyo% 40.) ;; "(" ;; #x28
        (tyo% 72.) ;; H  ;;#x48
        (tyo% #x03) ;; command for ntt-voice, output end (ETX)
        ))
  (print lm-kanji-string output-port)
  (setq responce-code (tyi%))
  (if (= responce-code #x50)  ;; normal end.
      (print 'ok output-port)
      ;;(tyi%) ;; #x03  ;; normal end.
      (print 'error!! output-port)
      ))

(defun read-lm-kanji-file (file-name)
  ;; file-name= "clm01:>kshima>krine-header.jis"
  (let ((nip-file (open file-name :direction :input))
        (lm-kanji-string-list) )
    (unwind-protect
        (do ((lm-kanji-string (read-line nip-file nil 'eof) (read-line nip-file nil 'eof)))
            ((eq lm-kanji-string 'eof) 'ok)
          (push lm-kanji-string lm-kanji-string-list) )
      (close nip-file) )
    (reverse lm-kanji-string-list)))

(defun read-lm-kanji-file-old (file-name)
  ;; file-name= "clm01:>kshima>krine-header.jis"
```

```
  (let ((nip-file (open file-name :direction :input))
        (lm-kanji-string-list) )
    (unwind-protect
        (setq lm-kanji-string-list (read-line nip-file 'eof))
      (close nip-file) )
    lm-kanji-string-list))


;; for example, (jpn-princ% (read-lm-kanji-file "clm01:>kshima3>krine-header-test-lm-code.txt
"))
;;              (jpn-princ% (read-lm-kanji-file "clm01:>kshima3>test-kanji-file.jap"))
```

```
(defvar printer-port)

(defun open-printer ()
  (setq printer-port
        (si:make-serial-stream ':unit 3
                               ':force-output t
                               ':xon-xoff-protocol t
                               ':generate-xon-xoff t
                               ':baud 1200
                               ':number-of-data-bits 7.
                               ':parity nil)))

(defun printer-test ()
  (do ((i 1 (1+ i)))
      ((> i 100.) (print 'ok))
    (send printer-port ':tyo 101))) ;; all "e" print.

(defun file-type-to-printer (file)
  (open-printer)
  (let ((file-port (open file)))
    (do a (read-line file-port '%%end&&) (read-line file-port '%%end&&)
        (eq a '%%end&&)
        (terpri)
        (terpri-print)
        (sleep 1.5)
        (princ a)
        (princ a printer-port)
        )
    (terpri-print)
    (close printer-port)
    ))

(defun terpri-print ()
  (send printer-port :tyo 141. )
  (send printer-port :tyo 138. ))


;; End of file.
```

CLM13:>kshima>intelliphone>test-intelliphone.lisp.8

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/06/87 13:55:41
Queued at: 8/07/90 19:02:43

```
;;; -*- Mode: LISP; Syntax: Common-lisp -*-


;;;知的電話機制御フレーバ
;;(defflavor intelliphone-control () ())

;;(defmethod (code-send intelliphone-control) (mojiretsu)
;;  (jpn-princ% mojiretsu *speaker-unit-window-var*) )

;;(defmethod (off-hook intelliphone-control) ()
;;  (recog-from-dp-3000-loop *hook-switch-unit-window-var*)  )

(defun recog-from-dp-3000 ()
  (read ) )


;; see "clm01:>kshima3>make-window.lisp *** (4)
(defun recog-from-dp-3000 ()
  (read-char)
  (let ((z (tv:make-window 'tv:momentary-menu
                           ;;':borders 3
                           ;;':label '(:string " Select one of kanji list")
                           ))
        ;; z is an instance of a momentary window created
        (kanji-list
          (get-kanji-list-from-dp-3000-kanji-code-table *dp-3000-kanji-code-table*)) )
    (send z ':set-item-list kanji-list)
    ;; this send a message to set up an item list
    (send z ':choose) ))

(defun get-kanji-list-from-dp-3000-kanji-code-table (dp-3000-kanji-code-table)
  (let (kanji-item-list)
    (dolist (kanji-list dp-3000-kanji-code-table)
      ;; kanji-list= '("SOUMU-NO-YAMAMOTO-SAN"."総務の山本さん")
      (push (cdr kanji-list) kanji-item-list) )
    (reverse kanji-item-list)))

;;(setq *dp-3000-kanji-code-table* '(("SOUMU-NO-YAMAMOTO-SAN"."総務の山本さん")
;;                                   ("KEIRI-NO-SUZUKI-SAN"."経理の鈴木さん")
;;                                   ("KEIRI"."経理")
;;                                   ("ARIMASEN"."ありません")))


(defmethod (code-send speaker-unit) (voice-code)
  (send *speaker-unit-window-var* :set-reverse-video-p nil)
  (print voice-code *speaker-unit-window-var*)
  (terpri *speaker-unit-window-var*)
  (sleep 2.)
  (send *speaker-unit-window-var* :set-reverse-video-p t)
  )


;; End of file.
```

CLM13:>kshima>intelliphone>intelliphone-header.jap.13

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 7/04/87 14:46:33
Queued at: 8/07/90 18:52:15

このたびは、インテリフォンを お買い上げいただきまして、誠にありがとうございます。インテリフォンは、あなたが 普段 おかけになる 相手先の電話番号を 記憶していて、相手の名前を 指定するだけで、自動的に 電話口まで 呼び出してくれます。しかも、操作はすべて ことばだけで おこなえますので、どなたにでも 簡単に 御使用いただけることと 存じます。

なお、はじめてご使用になるかたは、付属の マニュアルを お読みください。

インテリフォン(仮称)は，(1)番号を ダイヤルせずに 電話を簡単にかけることができ，(2)留守にしている場合も 確実に 電話をつないでくれて，しかも (3)その多機能さにもかかわらず 操作する部分が少なく,使いやすいという特徴を持った知的電話機である。第0.1版は，これらのうち (1)の一部分だけを実現するプロトタイプである。この中に含まれる 利用者サービス機能は，音声によるダイヤリング，再ダイヤル，番号案内がある。

CLM13:>kshima>intelliphone>my-phone.jap.71

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 9/04/89 13:56:44
Queued at: 8/07/90 18:59:53

```
("経理の佐藤"
 ((団体 "沖電気工業" "03-454-2111" ("沖電気" "株式会社" "メーカ") "かわら板で有名な")
  (個人 "芝本 尚樹" "06-321-4183" ("芝本" "男性" "既婚") "東洋情報システムより出向の")
  (個人 "横田 政憲" "0720-45-2985" ("横田" "男性" "未婚") "三洋電機より出向の"))
 ("経理の佐藤" "492"))
```

CLM13:>kshima>intelliphone>phone.jap.27

For: Kiyoshi Hayashi
Printed on: comprinter
Number of copies: 1
Data created at: 10/10/89 06:55:15
Queued at: 8/07/90 19:00:58

```
;;; -*- Mode: LISP -*-
;;;
;;; ROM data
("0:949*1825*251"
 (
  ("団体" "JIP(ASTER)" "0:::::06-448-6024" ("JIP(ASTER)") "")
  ("団体" "SYMBOLCIS" "0:::::06-347-7155" ("SYMBOLICS") "")
  ("個人" "カツゾウ" "250" ("カツゾウ") "")
  ("個人" "トラキチ" "251" ("トラキチ") "")
  ("個人" "テツ" "494" ("テツ") "")
  ("個人" "総務の林" "251" ("総務の林さん" "林" "総務") "")
  ("個人" "経理の佐藤" "250" ("経理の佐藤さん" "佐藤" "経理") "")
  ("個人" "カント" "251" ("カント" ) "")
  ("個人" "ヘーゲル" "250" ("ヘーゲル" ) "")
  ("個人" "マルクス" "494" ("マルクス" ) "")
  ("個人" "島" "422V" ("島" "島健" "NTT" "研究室") "")
  ("個人" "いまなんじ" "0:117V" ("今何時") "")
  ("個人" "明日の天気は" "0:177V" ("明日の天気は") "")
  ("個人" "リカチャン" "0:768-7711V" ("リカチャン" "研究室") "")
  ("団体" "キッチンメイド" "0:117V" ("弁当" "キッチンメイド")
   "一人前でも配達いたします")
  ("団体 " "ほっかほっか亭" "0:117V" ("弁当" "ほっかほっか亭") "")
  ("団体" "ダンキンドーナツ" "0:117V" ("ドーナツ" "喫茶" "ダンキンドーナツ") "")
  ("団体" "やましげ" "0:117V" ("宴会" "肉料理" "やましげ")
   "肉料理の名門")
  ("団体" "グルック" "0:117V" ("カレー" "グルック")
   "カレーとビール")
  ("団体" "だい北京" "0:117V" ("中華料理" "宴会" "大北京")
   "650円で食べられる、本格北京料理の、ひがわりランチが好評。店内は広く、椅子席の他に、大小和室も設けられており、宴会にも利用できる。700円からの海老料理が人気メニュー。")
  ("団体" "ピザ パテオ" "0:117V" ("ピザ" "パテオ")
   "カナダうまれのてづくりピザ。パイ生地を空中高くほうり投げて作る技術に、おいしさの秘密がある。ピザは9種類、690円から、自由に選べるトッピングも。グラタンやドリアは550円から。")
  ("団体" "三十石" "0:117V" ("三十石" "蕎")
   "店名はその昔、淀川を往来した三十石船に由来する。店のウィンドウからはそばを打つ姿が眺められる。売り物は540円の海苔まき寿司で、ダシにつけて食べる。")
  ("団体" "京阪モール食品館" "0:117V" ("食料品" "酒" "京阪モール" "食品館") "")
  ("団体" "リカーショップ オカノ" "0:117V" ("岡野" "酒")
   "奥さん。ご主人のお酒をきらしちゃいませんか。先に寝ちゃってもかまいませんが、優しい心ずかいだけはお部屋に残しておきましょう。岡野より。")
  ("団体" "津田三協堂" "0:117V" ("津田三協堂" "文具" "津田" "三協堂") "ファンシー文具、事務用品")
  ("団体" "京ばし東映" "0:117V" ("京ばし東映" "映画館" "東映") "東映封切館、土曜日オールナイト")
  ("団体" "京バシ劇場" "0:117V" ("京バシ劇場" "映画館" "京橋劇場") "ポルノ3本立て、土曜日オールナイト")
  ("団体" "京バシ名画座" "0:117V" ("京バシ名画座" "映画館" "名画座") "ポルノ3本立て、連日オールナイト")
  ("団体" "京橋アカデミー" "0:117V" ("映画館" "京橋アカデミー") "にっかつ封切館、土曜日オールナイト")
  ("団体" "富士通ドームシアター" "0:117V" ("富士通ドームシアター" "ドームシアター" "映画館")
   "科学博でも話題を呼んだ、全天周立体CG映画を上映。最先端技術が楽しめる。9月23日まで、ザ 宇宙遊泳と、ザ ユニバースを上映中。")
  ("団体" "大阪城ホール" "0:117V" ("大阪城ホール" "イベント")
   "西日本で最大のホール。コンサート、スポーツ イベント、展示会など、幅広い催しを開催。また、国際会議に対応できる、同時通訳設備もある。")
  ("団体" "MIDシアター" "0:117V" ("MIDシアター" "イベント")
   "ファッションショウ主体。展示会、講演会、コンサートなども。44に分割されたフロアが、20センチ刻みで自在に上下できるのが特徴。")
  ("個人" "山下部長" "400V" ("山下" "部長" "社長" "通信") "")
  ("個人" "門電室長" "410V" ("門田" "室長" "通信" "研究室") "")
  ("個人" "小林室長" "440V" ("小林" "室長" "通信") "")
  ("個人" "秋山" "441V" ("秋山" "通信") "")
  ("個人" "田中" "411V" ("田中" "通信") "")
  ("個人" "あらや課長" "" ("顕谷" "課長" "通信") "")
  ("個人" "浦" "424V" ("浦" "CSK" "通信") "")
  ("個人" "大内" "424V" ("大内" "CSK" "新婚" "通信") "")
  ("個人" "むこうひら" "424V" ("向平" "通信") "")
  ("個人" "西尾" "401V" ("西尾" "事務" "通信") "")
  ("個人" "ましば" "402V" ("間芝" "事務" "通信") "")
  ("個人" "中島" "405V" ("中島" "お父さん" "事務" "通信") "")
  ("個人" "伯田" "442V" ("伯田" "通信") "")
  ("個人" "島" "443V" ("島" "島則" "知能処理" "通信") "")
  ("個人" "木下" "444V" ("木下" "通信") "")
  ("個人" "西村" "445V" ("西村" "通信") "")
  ("個人" "高橋" "446V" ("高橋" "通信") "")
  ("個人" "大村" "447V" ("大村" "通信") "")
  ("個人" "トミン" "448V" ("トミン" "通信") "")
  ("個人" "こえずか" "449V" ("肥塚" "通信") "")
```

```
  ("個人" "西野" "450V" ("西野" "通信") "")
  ("個人" "竹村" "451V" ("竹村" "通信")
   "まもなく結婚する予定。")
  ("個人" "たてひら" "452V" ("立平" "ソニー" "通信") "")
  ("個人" "山口" "453V" ("山口" "通信") "")
  ("個人" "渡辺" "403V" ("渡辺" "マリリン" "通信") "")
  ("個人" "大江" "404V" ("大江" "通信") "")
  ("個人" "全" "412V" ("全" "通信") "")
  ("個人" "佐藤" "413V" ("佐藤" "亮一" "シャープ" "通信") "")
  ("個人" "永瀬" "414V" ("永瀬" "通信") "")
  ("個人" "手塚" "415V" ("手塚" "新婚" "通信" "IBM") "")
  ("個人" "林" "416V" ("林" "通信" "富士通" "研究室") "")
  ("個人" "佐藤" "417V" ("佐藤" "隆" "インテック" "通信" "研究室") "")
  ("個人" "内田" "418V" ("内田" "通信" "研究室") "")
  ("個人" "芝本" "419V" ("芝本" "TIS" "通信" "研究室") "")
  ("個人" "中島" "420V" ("中島" "俊ちゃん" "通信" "研究室") "")
  ("個人" "横田" "421V" ("横田" "三洋電機" "通信" "研究室") "")

  ("団体" "171会議室" "396V" ("会議室" "171") "")
  ("団体" "172会議室" "397V" ("会議室" "172") "")
  ("団体" "173会議室" "398V" ("会議室" "173") "")
  ("団体" "174会議室" "399V" ("会議室" "174") "")
  ("団体" "図書室" "389V" ("図書室") "")
  ))

;; End of file.
```