Programmer's Guide

Version 2.0, 23rd Jun 2001

BPEL-TUT

目 次

第 1 章

システム構成

1.1 会話型システム解析支援環境 : *SATELLITE*

*SATELLITE*は, 会話環境および C 言語に似た言語体系 (*SATELLITE*言語) を提供する *SATELLITE*シェルと, 信号処理やシミュレーションなど約 200 種類に及ぶ解析コマンドを機能別にまとめたシステム・モジュールによって構成される (図 1.1). ユーザは, *SATE-*

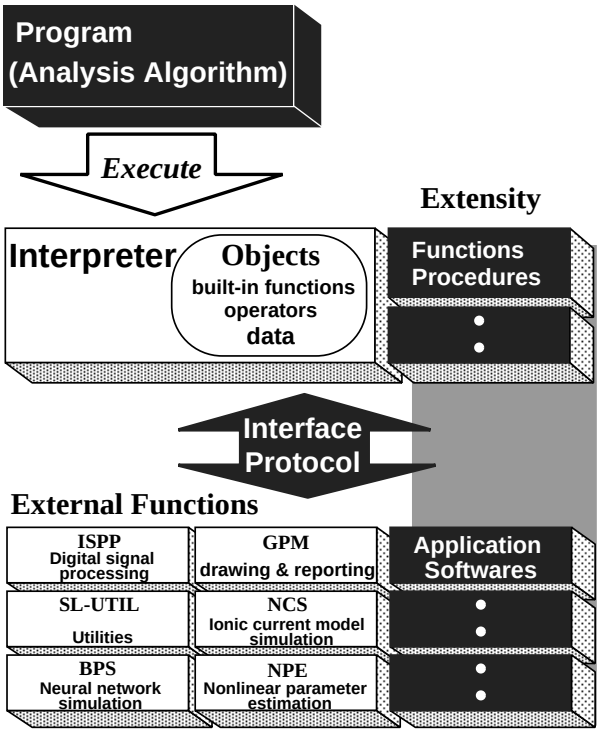

図 1.1 *SATELLITE*言語処理系の概観

*LITE*シェルが提供する一貫した対話環境から, システムモジュール内の任意のコマンド

を組み合わせて解析・シミュレーションを進めることができる. また必要に応じて, ディスプレーはもちろん, レーザープリンタへ解析結果を出力することができる.

1.2 ソフトウェア構成

*SATELLITE*は, すべて C 言語によって記述され, *SATELLITE*シェル, ライブラリ, 各システム・モジュールからなる. そのソフトウェアファイル構成は以下に示す通りである.

*SATELLITE*シェル	会話的環境を提供し, システム環境を管理する言語処理系.
コマンドプログラム	システム・モジュール内の各コマンドに対応する解析・シミュレーションプログラム.
システムファイル	システム・モジュール情報, コマンド登録情報, パラメータ・エラーメッセージを保持し, *SATELLITE*シェルに読み込まれるファイル.
データファイル	ユーザが作成する計測・解析データやモデルデータを記録する, バイナリ形式のファイル.
スクリプトファイル	ユーザが作成する一連のコマンドを記述した一括処理用プログラムファイル.

また, 起動中はシステムワークエリアとして, 以下の領域が確保される.

システムコモンエリア	*SATELLITE*シェルとコマンド間, コマンド相互間で必要な情報を保持している.
システムパラメータエリア	同一システム・モジュール内のコマンド間で必要な情報を保持している.
データバッファエリア	*SATELLITE*シェルおよびコマンドが演算対象とする解析データを格納している.

1.2.1 システムファイル

システムファイルには, システムの環境設定およびシステム・モジュールを定義する setup ファイルと, システム・モジュール内のコマンドを定義するコマンド登録ファイル, パラメータメッセージファイル, エラーメッセージファイルの 4 つのテキストファイルがある. これらのファイル中, setup ファイルを除く 3 つのファイルは, システム・モジュール毎に必要であり, 標準ではそのシステム・モジュールのコマンドプログラムと同一のディレクトリ内に格納される.

以下にこれらのファイルの役割を説明する.

● Setup ファイル

*SATELLITE*起動時に, *SATELLITE*シェルの初期設定を行なうのが, setup ファイルである. setup ファイルには, システム rc ファイルと, ユーザ毎に用意するユーザ setup ファイルの 2 つがあり, 両者ともファイル形式は, スクリプトファイルと同じである. *SATELLITE*

起動時にオプションを指定しなかった場合は, システム rc ファイル, ユーザ setup ファイルの順に読み込まれ実行される.

システム rc ファイルには, *SATELLITE* が提供している標準システム・モジュールの定義, 定数の定義が記述されている. ファイル名は,

/usr/local/satellite/lib/satellite/rc/rc.sl

である.

ユーザ setup ファイルでは, 標準システム・モジュールの登録, システム・モジュールの定義・登録, 別名定義 (エイリアス定義), 手続き, 関数の記述を行なう. ファイル名は, ".setup.sl" で, ユーザのホームディレクトリに置いておく必要がある.

また, setup ファイルには, *SATELLITE* 終了時に読み込まれる clean ファイルがあり, 実行終了時に行ないたい処理を記述しておく. ファイル名は, ".clean.sl" で, これもユーザのホームディレクトリに置いておく必要があるが, 存在しなくとも問題ない.

● コマンド登録ファイル

コマンド登録ファイルは, 登録するコマンドのパラメータ数など, 以下に示す情報を持っている.

- グループ番号 ··· モジュール内での機能分類番号. (現在は使用していない)
- コマンド名 ··· *SATELLITE* シェルがコマンドとして登録する名前.
- パラメータの個数 ··· 処理プログラム実行時に最低限必要とされるパラメータの数.
- パラメータのデフォルト値 ··· パラメータ入力をした時に表示されるパラメータ.
- メッセージ番号 ··· 各パラメータに対して表示するメッセージの登録番号.

● パラメータメッセージファイル

パラメータメッセージファイルは, パラメータ入力を補助するための表示メッセージを登録してある. 各メッセージには, コマンド登録ファイルのメッセージ番号と対応した任意の番号が付けられる.

● エラーメッセージファイル

エラーメッセージファイルは, 処理プログラム実行中にエラーが生じたとき *SATELLITE* シェルによって表示されるエラーメッセージを登録している. 各メッセージには, コマンドプログラムで実行する exit() システムコールの引数に対応した任意の番号が付けられる.

1.2.2 データファイル

SATELLITE ユーザが操作するデータファイルには, 計測・解析データファイルとモデルデータファイルの 2 つがある. 以下, これらのファイルの役割について述べる.

● **計測・解析データファイル**

SATELLITE **では**, **様々な情報を実データとともに保存することが可能である**. **格納するデータは**, **整数型** (2byte), **実数型** (4byte, 8byte) **である**.

● **モデルデータファイル**

モデルデータファイルは, NCS, BPS **などのシミュレータ用モデルデータとそのパラメータ情報を保存するものである**. **モデルデータファイルは**, **モデルの規模や構造**, **種類によって格納情報が大きくことなるため解析データファイルのような統一ファイル形式をとらず**, **各システム・モジュールで任意の形式を採用している**. **詳細は**, **各モジュールのユーザーズマニュアルを参照のこと**.

第 2 章

システム仕様

SATELLITE 言語処理系 (インタプリタ) と外部関数 (アプリケーション・ソフトウェア) は, それぞれ完全に独立したプログラムであり, *SATELLITE* 言語の API (Application Program Interface) 仕様に沿って構成される．この API 仕様は, 外部のアプリケーション・ソフトウェア (以下, コマンド) を外部関数として位置付け, 言語処理系とのデータ通信を関数の引数, 戻り値として実現するプロトコルを規定する．この API 仕様は, パラメータ入力方法とコマンドの起動方法を言語処理系に一任させることにより, 各コマンド内でのパラメータ入力処理を削減し, 開発言語や計算機環境に左右されない一貫したパラメータの引き渡しを可能にする．また, データをすべて *SATELLITE* 言語の変数 (オブジェクト) として扱えるためユーザにかかるデータ管理の負担を軽減できる．

ここでは, Series, Snapshot, File, String, Scalar オブジェクトのデータ領域, および API (Application Program Interface) 仕様の実装について説明する.

2.1 言語処理系内部でのオブジェクト表現

SATELLITE 言語で扱えるオブジェクトクラスは Series, Snapshot, File, Scalar, String がある．オブジェクトは表 2.1 に示す構造体として実装され, データ領域と処理方法 (メソッド) を持つ．

表 2.1 において, リンクカウンタ (link) はオブジェクトが参照されている回数を表し, 変数やスタックから参照されている時にインクリメントされ, 不要になった時にデクリメントされる．リンクカウンタがゼロのオブジェクトはガーベジコレクタにより回収される．次元数 (dim) やインデックス (index) はデータの構造を表現し, メソッドがデータを処理する時の情報となる．val はメモリ, ファイルなどに確保されたデータ領域のポインタを持つ．このデータ領域は公開されておらず, その操作は全てオブジェクトごとに定義されたメソッドにより行なわれる．メソッドの実体は各オブジェクトクラスに一つしかなく, オブジェクトはそのポインタを持つ．

表 2.1 オブジェクトの構造

構造体				説明
typedef	*struct*	_Object	{	
	unsigned	*char*	link;	リンクカウンタ
	unsigned	*char*	dim;	データの次元数
	int		*index;	データのインデックス
	void		*val;	データ領域へのポインタ
	Method		*method;	メソッドへのポインタ
}	Object;			

2.1.1 Series／Snapshot オブジェクト

Series は多次元時系列を操作するためのオブジェクト・クラスである．Series オブジェクトはインターナルバッファ(以下, バッファ) の識別子 (ID) を持ち, メッセージを受けるとバッファ内のデータを操作する．バッファはテンポラリディレクトリ上に確保されたファイルとして実装されており, 多次元のデータを扱うことができる．その構造は配列構造 (図 2.1) になっており, データの次元数やデータ数に応じて任意に変長する．バッファの先頭にはデータ次元数とそのインデックスが記録される．インデックスは次元数分確保され, それぞれの次元の大きさを示す．データは次元数に関係なく 1 列にならべられ, 最右端 (最後尾) のインデックスから変化する．また, Snapshot オブジェクトはメソッドが異なるだけで同じデータ構造を持つ．

2.1.2 File オブジェクト

File オブジェクトは, データ領域としてデータファイルを扱い, そのメソッドは, ほぼ Series オブジェクトと同じである．データファイルは, バッファが ID で参照されるのと異なりユーザが自由に名前を付けることのできるファイルである．また, データに関する情報はヘッダに格納される．ヘッダは構造体で定義され 256 バイトの領域を持つ．表 2.2 にデータファイルヘッダの構造を示す．データはこのヘッダの後に, バッファと同じデータ格納方式で記録される．データファイルには整数 (2byte), 単精度実数 (4byte), 倍精度実数 (8byte) の 3 種類のデータサイズに加え, BigEndian, LittleEndian のバイト順位形式を持つため, データ形式として 6 種類の組み合わせが存在する．そのため実際の演算においては, File オブジェクト評価時に Series オブジェクト (倍精度実数) に強制変換し, 演算時の型変換の煩雑さを回避している．

dim = 3
index [1024][10][20]

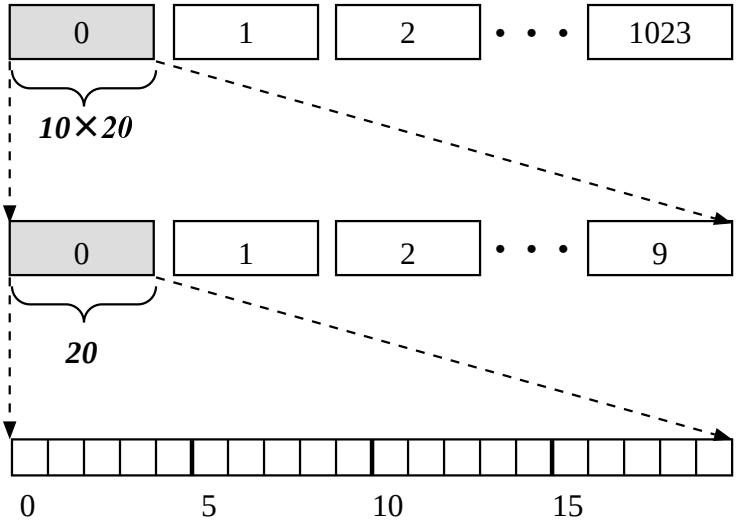

図 2.1 バッファの構造

表 2.2 データファイルヘッダ (型)

		構造体宣言	バイト数	説明	備考
typedef	*struct* {				
	char	type_flag;	1	**ファイル形式**	Big/Little Endian
	char	data_size;	1	**データ形式 (バイト数)**	2,4,8[byte]
	char	opr_name[30];	30	**作成者**	
	char	comment[128];	128	**コメント**	
	char	date[3];	3	**作成日**	
	char	dim;	1	**次元数**	
	int	index[10];	40	**インデックス**	**最大 10 次元**
	float	samf;	4	**サンプリング周波数**	
	char	free[48];	48	**整合領域 (未使用)**	**使用不可**
}	Header;		256		

2.1.3 Scalar オブジェクト

Scalar オブジェクトは, データとして倍精度実数型の値をメモリ上に確保し, そのポインタを持つ.

2.1.4 String オブジェクト

String オブジェクトは, メモリ上に確保された文字列領域のポインタを持つ. 実際には多次元の文字列配列を扱うため, このポインタは文字列のポインタ配列である.

2.2 API仕様

コマンドを外部関数として使用するためには, 通信プロトコルを実装したシステムライブラリ (表 2.3, 2.4) を利用し, 言語処理系とデータ通信を行なう必要がある. これらのシステムライブラリは, 言語処理系とコマンドが共有するシステムコモンエリア (表 2.5) の情報を操作する. システムコモンエリアは, 言語処理系とコマンド間の情報交換領域として, 入力パラメータ列や外部関数の戻り値, サンプリング周波数などの情報を保持している.

表 2.3 システムコモンエリア操作関数

	システム情報操作	説明
int	free_syscom()	システムコモンエリアの解放
double	get_sampling()	サンプリング周波数の取得
char*	get_tempdir()	テンポラリディレクトリ名の取得
int	init_syscom()	システムコモン情報の初期化
int	make_syscom()	システムコモンエリアを作る
int	read_syscom()	システムコモンエリア情報の読み込み
int	set_sampling(double freq)	サンプリング周波数の設定
int	set_tmpdir(char *dir)	テンポラリディレクトリ名の設定
int	write_syscom()	システムコモンエリアへ情報の書き込み

表 2.4 通信プロトコル実装関数

	n番目の引数 (オブジェクト) の取得	オブジェクト
char *	GetArgType(int n)	—
int	GetBufferID(int n)	—
double	GetScalar(int n)	Scalar
char *	GetString(int n)	String
Buffer *	GetSeries(int n, int dim, int *index)	Series
Buffer *	GetSnapshot(int n, int dim, int *index)	Snapshot
	オブジェクトのリターン	
int	ReturnScalar(double val)	Scalar
int	ReturnString(char *str)	String
int	ReturnSeries(Buffer *buf, int dim, int *index)	Series
int	ReturnSnapshot(Buffer *buf, int dim, int *index)	Snapshot

表 2.5 システムコモンエリア構造体 (型)

構造体宣言			説明	備考
typedef	*struct*	sys_common {		
	char	temp_dir[FILE_LENGTH];	テンポラリディレクトリ	
	int	buff_leng;	バッファ・データ点数	旧システム互換
	double	sam_freq;	サンプリング周波数	
	int	err_flag;	エラーフラグ	未使用
	int	narg;	引数の数	
	int	parm_num;	同上	旧システム互換
	char	types[NARG][10];	引数の型	最大NARG個
	char	strings[NARG][ONE_LINE];	引数 (文字列型)	
	double	values[NARG];	引数 (数値型)	
	int	buffer_number;	バッファカウンタ	
	char	return_type[20];	リターン値の型	
	char	return_strings[ONE_LINE];	リターン値 (文字列型)	
	double	return_values;	リターン値 (数値型)	
}	SystemCommon;		システムコモンエリア	

```
#define   FILE_LENGTH   512
#define   NARG          16
#define   ONE_LINE      512
    SystemCommon syscom;
```

言語処理系と外部関数のデータ通信は, 必ずこれらのシステムライブラリを用いて行なわれる．簡単な外部関数のプログラミング例を図2.2 に示す．

```
#include <stdio.h>
#include "SL_macro.h"          /* マクロ宣言 */
#include "SL_cmd.h"            /* システムコモンエリア構造体の宣言・領域確保 */

main()
{
    register int     i;
    int              dim, index[5], length;
    double           vmax;
    Buffer          *data;              /* 計算のためのバッファ領域 */

    read_syscom();                       /* システムコモンエリアの読み込み */
    data = GetSeries(0, &dim, index);    /* 第1番目の引数の取得 */
    if(data == NULL)
        exit(17);                        /* 外部関数のエラー終了 */
    vmax = data[0];
    length = IndexSize(dim, index);      /* Series Objectの全要素数の算出 */
    for (i = 0; i < length; i++) {       /* 最大値の検出 */
        if(vmax < data[i])
             vmax = data[i];
    }
    FreeBuffer(data);                    /* バッファの解放 */
    ReturnScalar(vmax);                  /* 最大値のリターン */
    return 0;                            /* 外部関数の正常終了 */
}
```

図 2.2 外部関数プログラミング例(ISPP,max**関数**)

*SATELLITE*言語のオブジェクトは, データ領域を指し示すポインタしか持たず, 実際のデータ領域はSeries, Snapshot ではバッファ, File ではデータファイル, Scalar, String はメモリ上に確保される．通常, 信号解析やシミュレーションなどではデータ点数が多く, また2次元, 3次元のデータを扱う場合には数メガバイトのメモリ領域を必要とする．それらを全てメモリに常駐させていたのでは, すぐにメモリ領域を使い果たしてしまい効率的な解析を行なうことは難しい．また, 大容量データによるメモリの圧迫はUNIX カーネルによる頻繁なスワップを引き起こし, 計算機全体の使用効率を下げることになりやすい．従って, ファイルに格納されているデータをアクセスすることは, 速度面において一見不利に思えるが, 計算機全体の使用効率を考えれば妥当な方法であるといえる．

また, API仕様はこれらのデータをプロセス間でやりとりするための通信プロトコルを規定する．実際には通信プロトコルを実装したC 言語のシステムライブラリが提供され, そのライブラリを使用すれば容易に外部関数を作成することができる．

2.3 システムライブラリ

2.3.1 基本ライブラリ

システムコモンエリア, 通信プロトコルに関するライブラリ関数の他, API 仕様に沿ったプログラム作成を支援する関数群が用意されている. 以下に, その用途別に一覧を示す.

表 2.6 インデックス計算関係

関数名	機能
CopyIndex	インデックスのコピー
EqualIndex	二つのインデックスが等しいか比較する
Index	指定したデータ位置のデータ先頭からの要素数を得る
IndexSize	インデックスで表される全体の要素数を得る
MaxIndex	二つのインデックスを比較して最大となるインデックスを得る
MinIndex	二つのインデックスを比較して最小となるインデックスを得る
RegularIndex	インデックス 1 に対しインデックス 2 が大きくないか調べる
rIndex	データ先頭からの位置から, インデックスを計算する
SubIndex	サブインデックスを求める

表 2.7 データファイル入出力関係

関数名	機能
ChangeDataType	ファイル形式の変更
ChangeDataSize	データ形式の変更
InitHeader	データファイルヘッダの初期化
LoadData	データファイルからブロック単位でデータを読み込む
LoadHeader	データファイルヘッダを読み込む
PrintHeader	データファイルヘッダの情報を表示する
ReadFile	データファイルから全てのデータを読み込む
StoreData	データファイルにブロック単位でデータを書き込む
StoreHeader	ヘッダをデータファイルに書き込む
WriteFile	データをデータファイルに書き込む

表 2.8 データバッファ操作関数

関数名	機能
AppendBuffer	バッファの最後にデータを追加する
AllocBuffer	メモリを確保する
BufferErrorNo	バッファ操作に失敗したときのエラー番号を得る
BufferErrorMessage	バッファ操作に失敗したときのエラーメッセージを得る
CallocBuffer	メモリを確保し, 初期化する
DestroyBuffer	バッファを破壊する
FreeBuffer	メモリ領域を解放する
GetBufferInfo	バッファのインデックス情報を得る
InitBuffer	バッファを初期化する
ReadBuffer	バッファのデータ全体をメモリに読み込む
ReadSubBuffer	サブバッファの内容をメモリに読み込む
ReadTimeSeries	バッファの時系列データをメモリに読み込む
WriteBuffer	バッファにデータを書き込む
WriteSubBuffer	サブバッファにデータを書き込む
WriteTimeSeries	バッファに時系列データを書き込む

表 2.9 その他の汎用関数

関数名	機能
get_date	現在の日付を得る
gethomedir	ホームディレクトリ名を得る
get_time	現在の時刻を得る
getusername	ユーザのアカウント名を得る
sort	データをソートする

第 3 章

システムライブラリ（C言語）

ここで, Series／Snapshot／File オブジェクトに関するシステムライブラリの説明のためにいくつかの用語を定義する.

1. バッファとは Series／Snapshot オブジェクトが扱うデータ領域であり, 通常テンポラリディレクトリに作成されるファイルである.
2. ここでいうデータファイルとは, ファイルオブジェクトが扱うデータ領域である.
3. 最左端のインデックスを時刻と呼ぶ. また, 残りのインデックスをサブインデックスと呼ぶ.
4. サブインデックスで表現できるデータをサブバッファ, もしくは空間データと呼ぶ. サブバッファは必ず連続した領域をとる.
5. サブインデックスで表現できるデータの大きさをブロックサイズと呼ぶ.
6. 同じサブインデックスで指定できる点の集合は, 時刻変化をインデックスとする 1 次元配列であり, 単純時系列と呼ぶ.
7. サブバッファの個数, 単純時系列の長さ, 最左端インデックスの最大値はすべて同じ値をとり, バッファの長さと呼ぶ.

3.1 基本ライブラリ

3.1.1 システムコモン関係

int free_syscom()

システムコモンエリアを解放する. 正常終了時は 0, 失敗時は -1 が返される.

double get_sampling()

サンプリング周波数の設定値が戻り値である. **事前に** read_syscom() **を行なっておく必要がある**.

char∗ get_tempdir()

作業ディレクトリ名の文字列のポインタが戻り値である. **単に**, syscom.temp_dir **の値を返すだけであるため**, **戻り値に対して** free() **などは行なってはならない**. **失敗時は**, NULL **が返される**.

int init_syscom()

syscom **構造体を初期化する**. **システムコモンエリアには**, **影響を与えない**. **正常終了時は** 0, **失敗時は** -1 **が返される**.

int make_syscom()

システムコモンエリアを作る. set_tmpdir() **によって**, **事前に作業ディレクトリを設定しておかなければならない**. **正常終了時は** 0, **失敗時は** -1 **が返される**.

int read_syscom()

システムコモンエリアの情報を syscom **構造体に読み込む**. **正常終了時は** 0, **失敗時は** -1 **が返される**.

int set_sampling(double freq)

サンプリング周波数を設定する. **正常終了時は** 0, **失敗時は** -1 **が返される**.

int set_tempdir(char ∗dir)

作業ディレクトリのパスを設定する. **引数には**, **ディレクトリパス文字列の先頭アドレスを与える**. **正常終了時は** 0, **失敗時は** -1 **が返される**.

int write_syscom()

システムコモン情報をシステムコモンエリアに書き込む. **この関数を実行しないと**, **ユーザコマンドで変更したシステムパラメータ**, **コマンドの戻り値などが**, **反映されない**.

3.1.2 通信プロトコル関係

char∗ GetArgType(int n)

n 番目のコマンド引数の, オブジェクトのクラスを得る. n 番目の引数が与えられなかった場合には, \0 (ヌル文字) が返され, その他の場合には, “scalar”, “series”, “snapshot”, “string” なる文字列が返される.

int GetBufferID(int n)

n 番目のコマンド引数の, オブジェクトのバッファID を得る. n 番目のコマンド引数が, Scalar, String クラスのオブジェクトであった場合には, 0 が返される.

double GetScalar(int n)

n 番目のコマンド引数の, オブジェクトの値を得る. n 番目のコマンド引数が, String クラスのオブジェクトの場合には 0.0 が返される. また, Series, Snapshot クラスの場合には, バッファID が double 型に変換した値を返す. これは, Scalar オブジェクトの値が 0.0 である場合, およびバッファID が渡された場合に問題があり, 本関数をバッファの ID を得るために使用するのは, プログラムを分かり難くするだけでなく保守の面でも問題があるので避けなければならない.

Buffer∗ GetSeries(int n, int ∗dim, int ∗index)

n 番目のコマンド引数の, オブジェクトの値を得る. n 番目のコマンド引数が, Series, Snapshot クラス以外のオブジェクトの場合には, 戻り値には, NULL が返される. Series, Snapshot クラスの場合には, データ領域の先頭アドレスが返され, dim, index には オブジェクトの次元数, インデックスが渡される. 次項の GetSnapshot() 関数と本関数は, 機能的に全く同じものである.

Buffer∗ GetSnapshot(int n, int ∗dim, int ∗index)

n 番目のコマンド引数の, オブジェクトの値を得る. n 番目のコマンド引数が, Series, Snapshot クラス以外のオブジェクトの場合には, 戻り値には, NULL が返される. Series, Snapshot クラスの場合には, データ領域の先頭アドレスが返され, dim, index には オブジェクトの次元数, インデックスが渡される. 前項の GetSeries() 関数と本関数は, 機能的に全く同じものである.

char∗ GetString(int n)

n 番目のコマンド引数の, オブジェクトの値を得る. n 番目のコマンド引数が, String クラス以外のオブジェクトの場合には, 戻り値には, コマンド引数として記述された文字列が返される. *SATELLITE* シェルでは, String オブジェクトは, 配列として扱えたが, 通信プロトコルにはその機能は提供されていない.

void ReturnScalar(double val)

コマンドの戻り値に Scalar クラスの値 val を返す.

void ReturnSeries(Buffer *buf, int dim, int *index)

コマンドの戻り値に Series クラスのデータ buf (dim 次元, インデックス: index) を返す.

void ReturnSnapshot(Buffer *buf, int dim, int *index)

コマンドの戻り値に Snapshot クラスのデータ buf (dim 次元, インデックス: index) を返す.

void ReturnString(char *str)

コマンドの戻り値に String クラスの値 str (文字列) を返す.

3.1.3 インデックス計算関係

int CopyIndex(int *index2, int *index, int n)

index の内容を, index2 にコピーする. n は index の要素数である. 戻り値には, コピーした要素数が返される. すなわち, 成功時は n が返される.

int EqualIndex(int *index2, int *index, int n)

index2 と index が同じインデックスならば 1, 違うならば 0 を返す. n は index の要素数.

int Index(int *index2, int dim, int *index)

dim 次元でインデックスが index のデータバッファの, index2 のデータ位置がデータ領域の先頭から何番目の要素かを返す.

int IndexSize(int dim, int *index)

dim 次元でインデックスが index で与えられるときの全データ数を返す. 返される値は, バッファの長さ×ブロックサイズにあたる.

int MaxIndex(int *a, int *b, int *index, int dim)

インデックス a と b の各要素を比較し, 大きい値のみをとったインデックスを index に返す. dim は a, b の要素数. 戻り値は要素数 dim.

int MinIndex(int *a, int *b, int *index, int dim)

インデックス a と b の各要素を比較し, 小さい値のみをとったインデックスを index に返す. dim は a, b の要素数. 戻り値は要素数 dim.

int RegularIndex(int *index2, int *index, int n)

n は, index の要素数, index2 が index と同じか小さいインデックスならば 1 を違うならば, 0 を返す. index2 が NULL もしくは, 要素が 0 以下である場合にも 0 を返す.

int* rIndex(int n, int *index2, int dim, int *index)

dim 次元でインデックスが index のデータバッファのデータ領域の先頭から n 番目のデータのインデックスを index2 に返す.

int* SubIndex(int *index)

index のサブインデックスを返す. ただし, 行なっている処理は, index+1 であるから, サブインデックスを直接書き換えた場合は, index の値も書き換えられる.

3.1.4 データファイル入出力関係

int ChangeDataType(int type)

ファイル形式の変更. type には, BigEndian, LittleEndian のいずれかを指定する. BigEndian は sun, luna, titan などのワークステーション, LittleEndian は NEC, IBM 等のパーソナルコンピュータ及び DEC 社の VAX, α チップを搭載したワークステーション. この関数で変更したファイル形式は, データファイルへの書き込み時に参照される. デフォルトは BigEndian.

int ChangeDataSize(int siz)

データ形式の変更．データ形式は倍精度実数(8byte), 単精度実数(4byte), 整数(2byte)があり, sizにバイト数を指定する．この関数で変更したファイル形式は, データファイルへの書き込み時に参照される．デフォルトは単精度実数(4byte)データ.

void InitHeader(Header *head)

データファイルヘッダの初期化を行なう．

char* LoadData(char *fname, int numb, Header *head)

データファイルからブロック単位でデータ(data)を読み込む．nbはブロック番号であり, 0から始まる．自動的にメモリ領域を確保し, そこにデータを転送したのちメモリ領域の先頭アドレスを返す．

int LoadHeader(char *fname, Header *head)

データファイルヘッダをHeader型の構造体に読み込む．

void PrintHeader(Header *head)

データファイルヘッダの情報を標準出力に表示する.

char* ReadFile(char *f_name, Header *head)

データファイル(fname)からすべてのデータを一括にメモリ上に読み込む．データを転送するためのメモリ領域を自動的に確保し, その先頭アドレスを返す．リターン値を使用する場合には, 2byteデータならばshort, 4byteデータならばfloat, 8byteデータならばdoubleのポインタにキャストする必要がある．headには, データファイルのヘッダ情報が設定される．

int StoreData(char *f_name, int numb, int dim, int *index, char *data)

データファイルにブロック単位でデータ(data)を書き込む．dimはブロックの次元数, indexはブロックのインデックス. 一つのファイルに異なるブロックサイズのデータを書き込むことはできない．

int StoreHeader(char *f_name, Header *head)

ヘッダをデータファイルに書き込む.

int WriteFile(char *f_name, int dim, int *index, char *data)

メモリ上のデータ(data)をデータファイル(fname)に一括して書き込む．dimはデータの次元数, indexは各次元の大きさである．

3.1.5 バッファ操作関数

int AppendBuffer(int id, int sub_dim, int *sub_index, Buffer *area)

バッファの最後尾にサブバッファを追加する．成功時はバッファの長さ (index[0]), **失敗時は**-1 **を返す．**

Buffer *AllocBuffer(unsigned buf_siz)

バッファを操作するためのメモリ領域を確保する． buf_siz **は要素数．メモリ確保に成功した場合は確保したメモリの先頭アドレスを返し**, **失敗した場合は** NULL **ポインタを返す**.

int BufferErrorNo()

上記の関数でバッファ操作に失敗したとき, **この関数を呼べば失敗理由をエラー番号で得ることができる．**

1. out of memory
2. failed to open internal buffer
3. failed to read data
4. failed to write data
5. failed to read header
6. failed to write header
7. index, out of range
8. illegal index
9. illegal dimension
10. dimension mismatch
11. not ready internal buffer
12. block size mismatch
13. failed to destroy internal buffer

char *BufferErrorMessage()

BufferErrorNo() **がエラー番号であるのに対し**, **この関数を用いれば失敗理由をエラーメッセージで取得することができる．戻り値は**, **エラーメッセージが格納された文字列のポインタである．**

Buffer *CAllocBuffer(unsigned buf_siz)

バッファを操作するためのメモリ領域確保と初期化．確保した領域がゼロクリアされる以外は, AllocBuffer **と同じ．**

int DestroyBuffer(int id)

バッファを破壊する．id **はバッファの識別子**(ID)**．成功時は**0, **失敗時は**-1 **を返す．**

int FreeBuffer(Buffer *buf)

バッファ操作用のメモリ領域の解放. buf **が**NULL **の場合は何も起こらない．**

int GetBufferInfo(int id, int *index)

バッファ(id) **の次元数**, **インデックス情報を取得する．インデックスは**index **に書き込まれる．次元数は戻り値として返される．失敗時は**-1 **を返す．**

int InitBuffer(int id, int dim, int *index)

バッファを新たに作成し, **その領域を初期化する．バッファの次元数**, **インデックスを設定し**, **その領域分をすべてゼロクリアする．成功時は**0, **失敗時は**-1 **を返す．**

Buffer *ReadBuffer(int id, int *dim, int *index)

バッファの内容をすべてメモリ領域に読み込む．id **はバッファの**ID, dim **は読み込んだバッファの次元数**, index **はバッファのインデックス**(MAX_INDEX **だけ予め確保する必要がある**)**．成功した時は**, **バッファの内容を転送したメモリ領域**(**動的に確保**) **の先頭アドレスを返し**, **失敗時は**NULL **を返す**.

Buffer *ReadSubBuffer(int id, int time, int *sub_dim, int *sub_index)

サブバッファ(**空間データ**) **を切り出す．**time **で切り出す時刻を指定する．**

sub_dim **は切り出したサブバッファの次元数**, sub_index **はそのインデックスが書き込まれる．成功時はサブバッファを格納したメモリ領域の先頭アドレス**, **失敗時は**NULL **を返す．**

Buffer *ReadTimeSeries(int id, int sub_dim, int *sub_index, int *length)

バッファ(id) **から**sub_index **で指定した単純時系列を切り出す．**sub_dim **は空間データの次元数**, **及び**sub_index **の要素数を表す．**length **に切り出した単純時系列の長さが設定される．成功時の戻り値は**, **単純時系列がコピーされたメモリ領域の先頭アドレス**, **失敗時は**NULL **を返す．**

int WriteBuffer(int id, int dim, int *index, Buffer *area)

メモリ上のデータ(area)**を一括してバッファに書き込む．**area**はメモリ上に確保されたデータ領域の先頭アドレス**.

int WriteSubBuffer(int id, int time, int sub_dim, int *sub_index, Buffer *area)

sub_dim, sub_index**の次元，インデックスを持つサブバッファ**(area)**をバッファ**(id)**に書き込む**. sub_dim**と**sub_index**が書き込むバッファに整合しないときはエラー．成功時はバッファの長さ**(index[0]), **失敗時は**-1**を返す．**

int WriteTimeSeries(int id, int sub_dim, int *sub_index, Buffer *area, int length)

バッファ(id)**に**sub_index**で指定した位置に単純時系列**(area)**を書き込む．**length**は単純時系列の長さ**. **成功時は**0, **失敗時は**-1**を返す．**

3.1.6 その他

int get_date(int *year, int *month, int *day)

現在の日付けを返す. year **には年**, month **には月**, day **には日が返される**. int **型で宣言しているが**, **戻り値はない**. (**修正の必要がある**)

int gethomedir(char *dir, int siz)

ホームディレクトリのパスを dir **に返す**. siz **は** dir **の領域の大きさを与える**. int **型で宣言しているが**, **戻り値はない**. (**修正の必要がある**)

int get_time(int *hour, int *minute, int *second)

現在の時刻を返す. hour **には時**, minute **には分**, second **には秒が返される**. int **型で宣言しているが**, **戻り値はない**. (**修正の必要がある**)

int getusername(char *name, int siz)

ユーザのアカウント名を name **に返す**. siz **は** name **の領域の大きさを与える**.

void sort(float *a, float*index, int n)

クイックソートを float **型の配列** a **に対して行なう**. index **には**, a **の要素数** n **と同じ大きさの** float **型の配列を渡す**. **ソート終了後は**, a **には**, **ソートの結果**, index **には** a **のデータの各要素の以前のデータ位置が返される**.

第 4 章

ユーザコマンドの作成方法

ここでは, ユーザコマンドのコンパイル方法, コマンド登録の方法について説明する.

4.1 コンパイルの方法

前章で説明した関数群は,

/usr/local/satellite/lib/satellite/libsatellite.a

にアーカイブされている. また, ライブラリ関数の宣言などは, ヘッダファイル

/usr/local/satellite/include/satellite/SL_macro.h
/usr/local/satellite/include/satellite/SL_cmd.h

でなされているので, ユーザコマンドを作成する際には, プログラムの先頭に,

```
#include "SL_macro.h"
#include "SL_cmd.h"
```

を記述しなければならない. また, コンパイルの際には,

```
% cc prog.c -o prog -I/usr/local/satellite/include/satellite \
        -L/usr/local/satellite/lib/satellite -lsatellite -lm
```

とする.

4.2 コマンド登録の方法

ユーザコマンドの登録は, 単一もしくは複数のコマンド群をコマンドモジュールとして登録する方法をとらなければならない. 以下, コマンド登録に必要な, 3 つのシステムファイル (コマンド登録ファイル, パラメータメッセージファイル, エラーメッセージファイル) の記述方法について, および コマンド登録の方法について説明する.

4.2.1 コマンド登録ファイルの記述

コマンド登録ファイルの記述フォーマットは次のような, 仕様になっている.

コマンド登録ファイル仕様

m_number#com#program#p_number#default#p_message

m_number モジュール番号 (現在は使用していない)

com コマンド名

program コマンドプログラム名

p_number パラメータ数

default パラメータのデフォルト値

p_message パラメータメッセージ番号

コマンド登録ファイルの記述例を以下に示す.

```
2#wopen#wopen#4#1,"A4",0,0#20,79,67,68
2#wclose#wclose#1#1#20
2#frame#frame#0#0#0
2#draw#drawl#2#"Y",0#21,22
2#graph#Graph#7#x,"T",0,0,0,0,0#23,24,25,26,27,28,29
2#axis#axis#10#1,1,"XY","XY",3.5,0,0,0,0,0#30,31,32,33,34,74,75,76,77,78
```

パラメータ数と, デフォルト値の個数は一致していなければならない. また, メッセージ番号は, パラメータメッセージファイルに記述された番号に対応させなければならない.

4.2.2 パラメータメッセージファイルの記述

パラメータメッセージファイルの記述フォーマットは, 次のような仕様になっている.

パラメータメッセージファイルの仕様

number　message

number パラメータメッセージ番号

message パラメータメッセージ

パラメータメッセージファイルの記述例を以下に示す.

```
30 TITLE NUMBER
31 AXIS TYPE(1:LL 2:LU 3:RL 4:RU)
32 DISPLAY AXIS(X,Y,XY,other)
33 DISPLAY SCALE(X,Y,XY,other)
34 SIZE[mm]
```

```
35 GRADIENT
36 REMOVE HIDDEN LINES (0:YES 1:NO)
37 GRID (0:XY 1:X 2:Y)
38 DRAW STYLE (0:ALL 1:UPPER 2:LOWER)
```

4.2.3 エラーメッセージファイルの記述

エラーメッセージファイルの記述フォーマットは, 次のような仕様になっている.

エラーメッセージファイルの仕様

number message

number エラーメッセージ番号

message エラーメッセージ

エラーメッセージファイルの記述例を以下に示す.

```
1 Command Not Found
2 Illegal Parameter
3 Buffer Write Error
4 Buffer Read Error
5 No Such Directory
6 ALIAS Set Error
```

エラーメッセージ番号は, コマンドプログラムの終了コード (exit, **もしくは** return **によって返される値**) **に対応する. この終了コードが** 0 **であれば,** *SATELLITE* **シェルは, コマンド実行が正常に終了したものと判断し, 次の処理に移る. その他の値である場合には, その値に対応するメッセージを表示し, 処理を中断する.**

4.2.4 コマンド登録の方法

コマンド登録ファイル, パラメータメッセージファイル, エラーメッセージファイルをそれぞれ記述したら, 次は, コマンドとして実際に登録する. これは, 次のようにしておこなう.

例えば, ユーザコマンドの実行プログラム, コマンド登録ファイル (COMMAND.USR), **パラメータメッセージファイル** (MESSAGE.USR), **エラーメッセージファイル** (ERROR.USR) **が** /home/tom/sl-command **にあるとすると,**

```
define USRCOM {
set Command_Path "/home/tom/sl-command"
set Command_File "COMMAND.USR"
set Message_File "MESSAGE.USR"
set Error_File   "ERROR.USR"
```

```
}
module(USRCOM)
```

と入力すれば, ユーザコマンドが登録され, その場で他のコマンドと同様に利用することが可能になる. これと同じ記述を, Setup ファイルに記述しておけば, 次回の起動からは, 自動的にユーザコマンドが登録される.

● 複数の機種でユーザコマンドを使えるようにする

アーキテクチャのことなる計算機では, ユーザコマンドはそれぞれの計算機でコンパイルしたものを用意しておく必要がある. それぞれでコンパイルしたコマンドを Setup ファイルでそれぞれの計算機にあわせて登録する方法を以下に示す.

2 つのアーキテクチャについて登録する場合について説明する. コマンド登録ファイル, パラメータメッセージファイル, エラーメッセージファイルが /home/tom/sl-com にあり, ホスト名 titan2 のアーキテクチャ用のコマンドが, /home/tom/sl-com/alpha に, もう一方のアーキテクチャ用のコマンドが, /home/tom/sl-com/sun にあるとする.

```
if ( ‘hostname‘ == "titan2" ) {
  define USRCOM{
    set Command_Path "/home/tom/sl-com/alpha";
    set Command_File "/home/tom/sl-com/COMMAND.USR";
    set Message_File "/home/tom/sl-com/MESSAGE.USR";
    set Error_File   "/home/tom/sl-com/ERROR.USR";
  }
} else {
  define USRCOM{
    set Command_Path "/home/tom/sl-com/sun";
    set Command_File "/home/tom/sl-com/COMMAND.USR";
    set Message_File "/home/tom/sl-com/MESSAGE.USR";
    set Error_File   "/home/tom/sl-com/ERROR.USR";
  }
}

module(USRCOM);
```

このように記述すれば, 二つのアーキテクチャについて, ユーザコマンドを自動的に登録することができる. Setup ファイルに記述すれば, 起動時に自動的に登録される.

付 録 A

関数ライブラリ一覧

A.1 *SATELLITE*関数ライブラリ

A.1.1 システムコモン関係

int	free_syscom()
double	get_sampling()
char∗	get_tempdir()
int	init_syscom()
int	make_syscom()
int	NextBuffer()
int	read_syscom()
int	set_mypid(int pid)
int	set_sampling(double freq)
int	set_tempdir(char ∗dir)
int	write_syscom()

A.1.2 通信プロトコル関係

char∗	GetArgType(int n)
int	GetBufferID(int n)
double	GetScalar(int n)
Buffer∗	GetSeries(int n, int ∗dim, int ∗index)
Buffer∗	GetSnapshot(int n, int ∗dim, int ∗index)
char∗	GetString(int n)
void	ReturnScalar(double val)
void	ReturnSeries(Buffer ∗buf, int dim, int ∗index)
void	ReturnSnapshot(Buffer ∗buf, int dim, int ∗index)
void	ReturnString(char ∗str)

A.1.3 メモリ・バッファ入出力関係

int AppendBuffer(int n, int sub_dim, int *sub_index, Buffer *area)
Buffer* AllocBuffer(unsigned buf_size)
int BufferErrorNo()
char* BufferErrorMessage()
Buffer* CallocBuffer(unsigned buf_size)
int DestroyBuffer(int n)
void FreeBuffer(Buffer *area)
int GetBufferInfo(int n, int *index)
int InitBuffer(int n, int dim, int *index)
Buffer* ReadBuffer(int n, int *dim, int *index)
Buffer* ReadSubBuffer(int n, int time, int *sub_dim, int *sub_index)
Buffer* ReadTimeSeries(int n, int sub_dim, int *sub_index, int *length)
int WriteBuffer(int n, int dim, int *index, Buffer *area)
int WriteSubBuffer(int n, int time, int sub_dim, int *sub_index, Buffer *area)
int WriteTimeSeries(int n, int sub_dim, int *sub_index, Buffer *area, int length)

A.1.4 データファイル入出力関係

int ChangeDataType(int type)
int ChangeDataSize(int siz)
void InitHeader(Header *head)
char* LoadData(char *fname, int numb, Header *head)
int LoadHeader(char *fname, Header *head)
void PrintHeader(Header *head)
char* ReadFile(char *f_name, Header *head)
int StoreData(char *f_name, int numb, int dim, int *index, char *data)
int StoreHeader(char *f_name, Header *head)
int WriteFile(char *f_name, int dim, int *index, char *data)

A.1.5 インデックス計算関係

int RegularIndex(int *index2, int *index, int n)
int CopyIndex(int *index2, int *index, int n)
int EqualIndex(int *index2, int *index, int n)
int IndexSize(int dim, int *index)
int* SubIndex(int *index)
int Index(int *index2, int dim, int *index)
int* rIndex(int n, int *index2, int dim, int *index)
int MaxIndex(int *a, int *b, int *index, int dim)
int MinIndex(int *a, int *b, int *index, int dim)

A.1.6 その他

int get_date(int *year, int *month, int *day)
int gethomedir(char *dir, int siz)
int get_time(int *hour, int *minute, int *second)
int getusername(char *name, int siz)
void sort(float *a, float*index, int n)

A.2 SATELITE 互換関数

A.2.1 通信プロトコル関係

int get_p(int nprm, int str_size, char *str, int *integer, float *real)

A.2.2 バッファ関係

float* buff_alloc()
int buff_append(int n, float *area, int length)
void buff_free(float *p)
int buff_info(int n, int *index)
void buff_init(int n, int dim, int *index)
int buff_read(int n, float *area, int *index)
int buff_write(int n, float *area, int length, int *index)

A.2.3 データファイル関係

int load_data(float *data, char *f_name, int numb)
int load_header(char *f_name, h_type *head)
void m_file(char *f_name)
int r_file(int s_blk, int num_blk, char *f_name, char *data)
int stor_data(float *data, int d_point, char *f_name, int numb)
void stor_header(char *f_name, h_type *head)
void stor_init(char *f_name, int type)
int w_file(int s_blk, int num_blk, char *f_name, char *data)