JAERI-Data/Code
2000-006

並列計算機クラスタ上のツール間通信を支援するライブラリ：Starpc

― Starpc利用及び開発手引書 ―

2000年2月

武宮　博・山岸信寛

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所研究情報部研究情報課（〒319-1195　茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-1195　茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken, 319-1195, Japan.

編集兼発行　　日本原子力研究所

JAERI-Data/Code 2000-006

並列計算機クラスタ上のツール間通信を支援するライブラリ: Starpc
–Starpc利用及び開発手引書–

日本原子力研究所計算科学技術推進センター
武宮　博・山岸 信寛※

(2000年1月24日受理)

並列計算機間の通信を支援する遠隔関数呼び出し型通信ライブラリ Starpc について述べる.

Starpc は, Java Applet–C プログラム間及び C プログラム–C プログラム間の通信を支援する. Starpc は, 以下の3つの特徴を持つ.

(1) 一般に, Java Applet はセキュリティ上の制約から Web サーバとの通信しか許されていないが, Starpc はセキュリティを保持しつつ Java Applet と任意の計算機上の C プログラムとの通信を可能にしている.
(2) Argonne 研究所において開発された Nexus 通信ライブラリを利用することにより, 多様な通信プロトコルに対応している.
(3) 8種の並列計算機, 4種の WS サーバを含む多様な計算機上で利用可能である.

本報告書では, Starpc の利用方法及び Starpc を用いたアプリケーションの構築方法について述べる.

日本原子力研究所：〒153-0061 東京都目黒区中目黒2-2-54
※ 外来研究員：日立東北ソフトウェア（株）

JAERI-Data/Code 2000-006

Starpc: a Library for Communication among Tools on a Parallel Computer Cluster
— User's and Developer's Guide to Starpc—

Hiroshi TAKEMIYA and Nobuhiro YAMAGISHI*

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received January 24, 2000)

We report on a RPC(Remote Procedure Call)-based communication library, Starpc, for a parallel computer cluster. Starpc supports communication between Java Applets and C programs as well as between C programs. Starpc has the following three features.

(1) It enables communication between Java Applets and C programs on an arbitrary computer without security violation, although Java Applets are supposed to communicate only with programs on the specific computer (Web server) in subject to a restriction on security.
(2) Diverse network communication protocols are available on Starpc, because of using Nexus communication library developed at Argonne National Laboratory.
(3) It works on many kinds of computers including eight parallel computers and four WS servers.

In this report, the usage of Starpc and the development of applications using Starpc are described.

Keywords: Library, Network, Communication, Distributed Processing, Java, WWW, Remote Procedure Call

* Visiting Researcher : Hitachi Tohoku Software, Ltd.

目 次

Contents

1 はじめに

Starpcは，STA基本システムにおけるツール間の通信を実現するための通信基盤SCEを構成する通信ライブラリの一つである．Starpcは，RPC (Remote Procedure Call [1]) 型の通信形態に基づくライブラリで，Java Appletを用いて構築されたツールGUIと並列計算機上で動作するCプログラムとの通信，及び異なる並列計算機上で動作するCプログラム間の通信を支援する．

本報告書では，Starpcのインストール方法，Starpcを用いたアプリケーション (STAアプリケーション) の構築手法，及び利用方法について述べるとともに，Starpcに実装されている関数群の仕様について説明する．

本報告書の内容は，STAアプリケーションの開発における一般的な流れ，すなわち，ライブラリのインストール，アプリケーションの開発，登録，利用，に即して構成されている．

STAアプリケーションの開発者は，開発フェーズの進行にあわせて本報告書を利用していただきたい．STAアプリケーションの利用者は，6章のみを利用すれば充分である．各章の概要は以下の通りである．まず，2章において，Starpcが利用される環境となるSCEのシステム構成及び，以降の説明に必要となる各種用語について説明する．3章ではStarpcのインストール手法について説明する．Starpcは，米国Argonne国立研究所で開発された通信ライブラリNexus [2] を利用しているため，Starpcを使用するためにはNexus通信ライブラリがインストールされている必要がある．Nexusライブラリは<ftp://ftp.mcs.anl.gov/pub/nexus/nexus-4.1.1.tar.gz>より入手可能である．インストールに関してはそれに含まれるドキュメントを参照していただきたい．

次に，4章では，Starpcにおいて実装されている関数群の仕様を示す．Starpcには，Java Applet用ライブラリ (アプレット関数) とCプログラム用ライブラリ (アダプタ関数) の2種類が提供されている．それら2種類の関数インタフェースについて説明する．

開発されたSTAアプリケーションを利用するためには，SCEに登録する必要がある．5章では，SCEへのSTAアプリケーション登録方法について述べる．登録されたSTAアプリケーションは，利用者の手元の計算機よりNetscape Navigatorを用いて利用することができる．6章では，開発されたSTAアプリケーションの利用方法について述べる．

2 Starpc概要

2.1 システム構成

1. でも述べたようにStarpcは，通信基盤SCEにおける通信を実現するライブラリである．したがって，3章以降での記述内容の理解を促進するために，1章においてSCE全体のシステム構成及び各サブシステムの説明を行なう．

STA通信基盤システムSCEは、以下の4つのサブシステムから構成される。

(1) ツール管理サブシステム
(2) GUIサブシステム
(3) GUI管理サブシステム
(4) ファイル管理サブシステム

SCE上に構築されたアプリケーション通信を支援するライブラリStarpcは、以下の2つのライブラリから構成される。

(1) C言語用ライブラリ
(2) Java言語用ライブラリ

利用者は、GUIサブシステムを操作し、ツールを利用する。各サブシステムは、GUIサブシステムからの要求により、それぞれの機能を実行する。SCE全体のシステム構成を図1に示す。

STA通信基盤システムを構成する各サブシステムの概略を以下に述べる。

(1) ツール管理サブシステム
ツール管理サブシステムは、ツール管理部、ツール、アダプタから構成される。ツール管理部は、ツールの管理、アダプタの実行を行う。
ツールは、並列計算機で利用できる既存のツールである。
アダプタは、ツール管理サブシステムとツールの間のインタフェースを提供する。ツール毎に専用のアダプタが必要である。
ツール管理サブシステムは、本システムから利用するツールを管理し、GUIサブシステムからの要求に従い、アダプタを介してツールを実行する。STAアプリケーション開発者は，このアダプタを開発する．

(2) GUIサブシステム
GUIサブシステムは、WWWブラウザ、ツール制御モジュール、ファイル管理モジュール、STAアプリケーションGUIプログラムから構成される。
WWWブラウザはGUI管理サブシステムのWWWサーバから各モジュールをロードして実行する。
ツール制御モジュールは、GUI管理サブシステムに登録されているツールを調べ、利用者によって設定されたカスタマイズ情報に従い画面上にツール一覧を表示する。画面に表示されたツールを利用者が選択すると、ツール制御モジュールはWWWブラウザに選択されたツールのSTAアプリケーションGUIプログラムをロードする。
ファイル管理モジュールは、ファイル管理サブシステムと連携し、利用者に対し以下

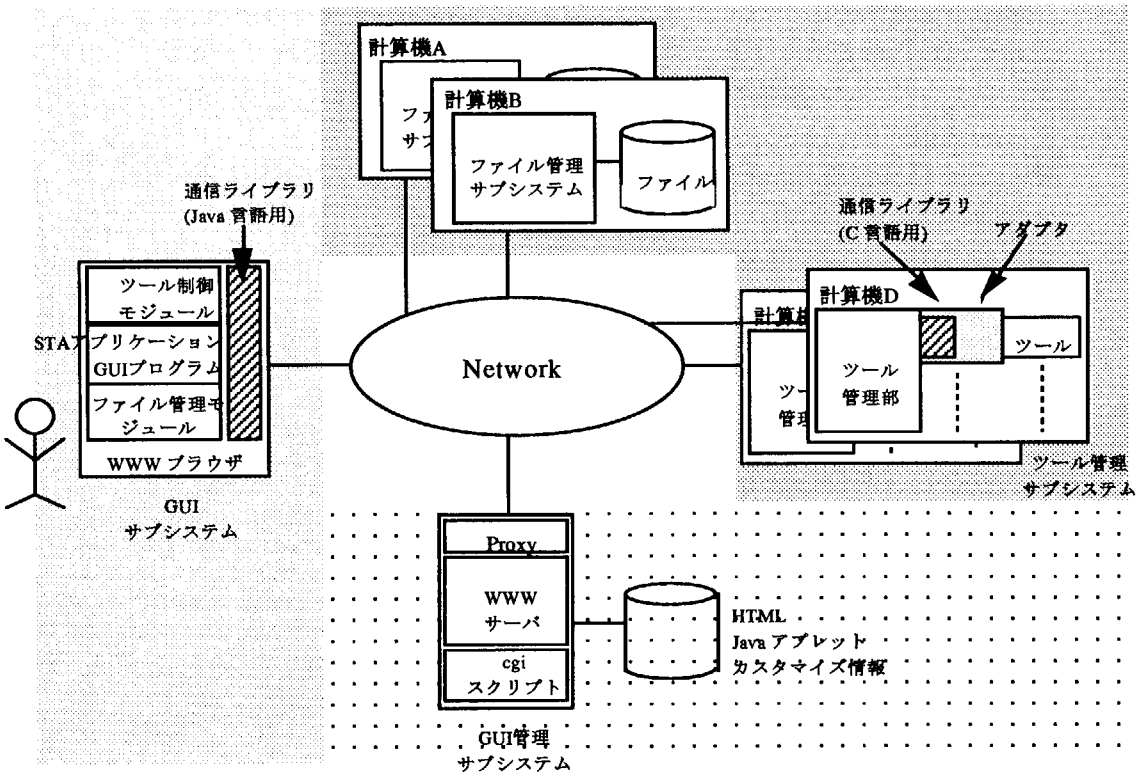

図 1: SCE システム構成

の機能を提供する。

(i) ファイル一覧表示

(ii) ファイルコピー、削除、名称変更、ディレクトリ作成、ディレクトリ削除の操作

(iii) ツールの操作対象となるファイルの選択

STA アプリケーション GUI プログラムは、各ツールのツール実行画面を提供する。ツール毎に専用の STA アプリケーション GUI プログラムが必要である。ツール開発者は，ツール管理サブシステム上に構築したアダプタに対応する GUI として、STA アプリケーション GUI プログラムを構築する必要がある.

(3) GUI 管理サブシステム

GUI 管理サブシステムは、Proxy、WWW サーバ及び cgi スクリプトから構成される。Proxy は利用者の STA 環境への接続を許可し、GUI サブシステムとツール管理サブシステム間の通信、あるいはツール管理サブシステムとツール管理サブシステム間の通信を中継する。

WWW サーバは、GUI サブシステムが使用する HTML、Java アプレット、ユーザカスタマイズ情報の各ファイルを管理し、GUI サブシステムに提供する。

cgi スクリプトは、STA を利用する利用者のカスタマイズ情報保存処理を行う。上記構成要素中、WWW サーバは SCE を利用する際の前提ソフトウェアとする。

GUI 管理サブシステムのインストールされた計算機を STA サーバ機と呼ぶ.

(4) ファイル管理サブシステム

ファイル管理サブシステムは、GUIサブシステムにファイル情報を提供し、GUIサブシステムからの要求に応じ、ファイルコピー、削除、名称変更、ディレクトリ作成、ディレクトリ削除の操作を行う。

(5) Starpc通信ライブラリ

StarpcはC言語用とJava言語用の2種類のライブラリから構成される。

C言語用ライブラリは、ツール管理サブシステムのアダプタに対し、本システムで使用するプロトコルを用いた通信機能を提供する。

Java言語用ライブラリは、GUIサブシステムのSTAアプリケーションGUIプログラムに対し、本システムで使用するプロトコルを用いた通信機能を提供する。

2.2 アダプタ

2.2.1 アダプタの役割

アダプタは、ツールをSCEに接続して、STAアプリケーションGUIプログラムとツールの間の通信を仲介する。

STAアプリケーションGUIプログラムは、スタブ関数からアダプタのスケルトン関数を呼び出すことにより、利用者の操作をアダプタに伝達する。

スケルトン関数は、利用者の操作に対する処理を行うサーバ関数を呼び出す。

2.2.2 アダプタが備えるべき機能

アダプタには以下の機能が必要である。

初期処理

アダプタ設定ファイルから設定情報を読み込む。また、その他ツールの実行に必要な初期処理を行う。

STAアプリケーションGUIプログラムとの通信 (スケルトン関数)

STAアプリケーションGUIプログラムとの通信はスケルトン関数で行う。スケルトン関数は、STAアプリケーションGUIプログラムのスタブ関数から呼び出される関数である。スケルトン関数は引数をデコードして、ツールの実行などを行うサーバ関数を呼び出す。

ツール実行機能 (サーバ関数)

ツールの実行や中断などの処理を行う。

終了処理

アダプタ終了時の終了処理を行う。

2.2.3 アダプタの構成

アダプタの構成を図2に示す。

アダプタは、システムが提供するアダプタ本体、ツール提供者が作成する関数、スケルトン関数テーブルから構成される。

アダプタに共通な主要な機能は、Cライブラリとしてシステムが提供する。

アダプタ本体は、初期処理、スケルトン関数のディスパッチ、終了処理を行う。

システム提供の初期処理、終了処理では、アダプタに共通な処理を行う。アダプタ固有の初期処理、終了処理が必要な場合には、ツール提供者が初期処理関数、終了処理関数を定義する。

アダプタ本体の初期処理、終了処理からツール提供者が用意した初期処理関数、終了処理関数が呼び出されるので、アダプタ固有の処理を行うことができる。

スケルトン関数のディスパッチ処理は、ツール提供者が定義したスケルトン関数テーブルを参照して、スケルトン関数を呼び出す。

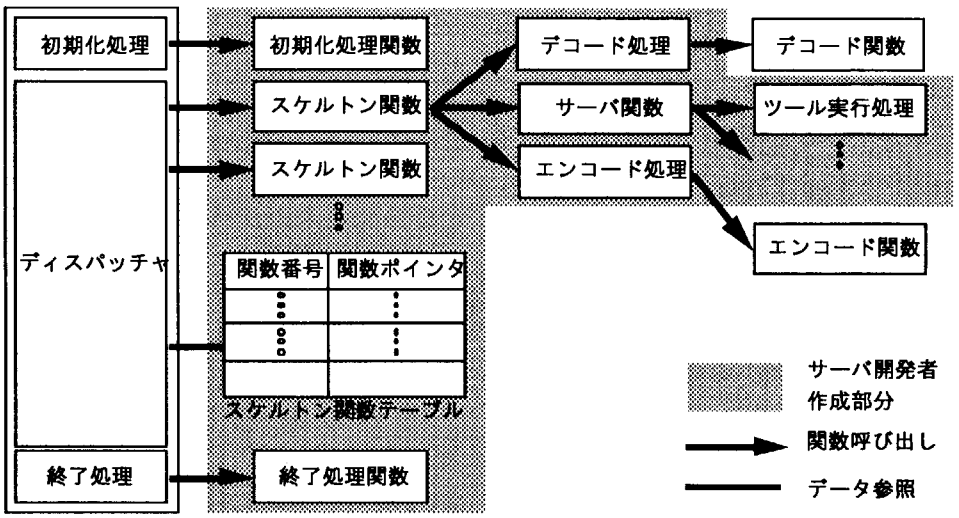

図 2: アダプタの構成

2.2.4 アダプタの起動

アダプタは、ツール管理サブシステムから exec システムコールにより実行される。その際、ツール管理サブシステムは表 1 に示す環境変数を設定する。アダプタは、必要に応じこれらの環境変数を変更してツールを実行する。

表 1: ツール管理サブシステムが設定する環境変数

#	環境変数	説明
1	HOME	利用者のホームディレクトリを設定する
2	PATH	ツール管理サブシステム設定ファイルで指定されたコマンドサーチパスを設定する
3	USER	利用者のユーザ ID を設定する
4	LANG	C を設定する
5	SCE_CONFIG_FILE	アダプタ設定ファイルのパスを設定する。起動されたアダプタのアダプタ設定ファイルがない場合、本環境変数は設定しない

2.2.5 STA アプリケーション GUI プログラムとのインタフェース

アダプタと STA アプリケーション GUI プログラムの間のインタフェースは、Starpc C 言語用ライブラリとして以下の機能を提供する。詳細については 4.1 アダプタ関数仕様 を参照のこと。

スケルトン関数の返信バッファの送信機能

スケルトン関数が作成した返信バッファを STA アプリケーション GUI プログラムに送信する。

他アダプタ接続、切断機能

他のアダプタとの接続及び切断を行う。

Socket 通信機能

ツール固有のソケットを用いた通信の開始、終了、中断を行う。

2.2.6 処理の流れ

アダプタの一般的な処理の概要を図 3 に示す。

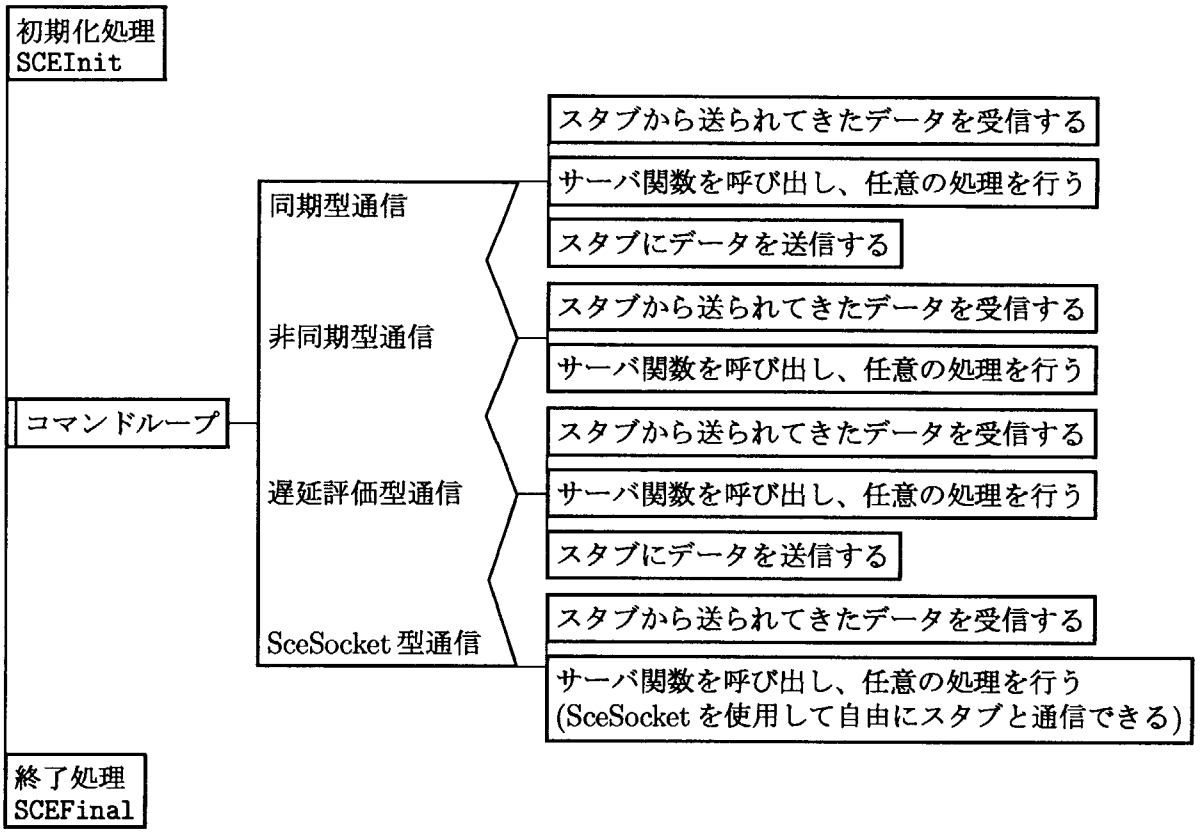

図 3: アダプタの処理概要

2.3 STA アプリケーション GUI プログラム

2.3.1 STA アプリケーション GUI プログラムの役割

STA アプリケーション GUI プログラムは、利用者に各ツールの GUI を提供する。

STA アプリケーション GUI プログラムは、ツール毎の固有の設定や、操作に対応するため、ツール毎に作成する必要がある。利用者は、STA アプリケーション GUI プログラムを操作し、ツールの起動オプションの設定や、ツールの実行、中断等の操作を行う。

STA アプリケーション GUI プログラムは、利用者の画面操作に対応して、アダプタのスケルトン関数を呼び出し、応答を受け取る。アダプタでは、ツール実行要求のスケルトン関数が呼び出されると、ツール実行等の処理を行う。

GUI サブシステム内のデータ交換は、メッセージボードへのメッセージ登録と参照、及びツール制御モジュールを利用したツール連携機能によって行う。

STA アプリケーション GUI プログラムの位置付けを図 4 に示す。

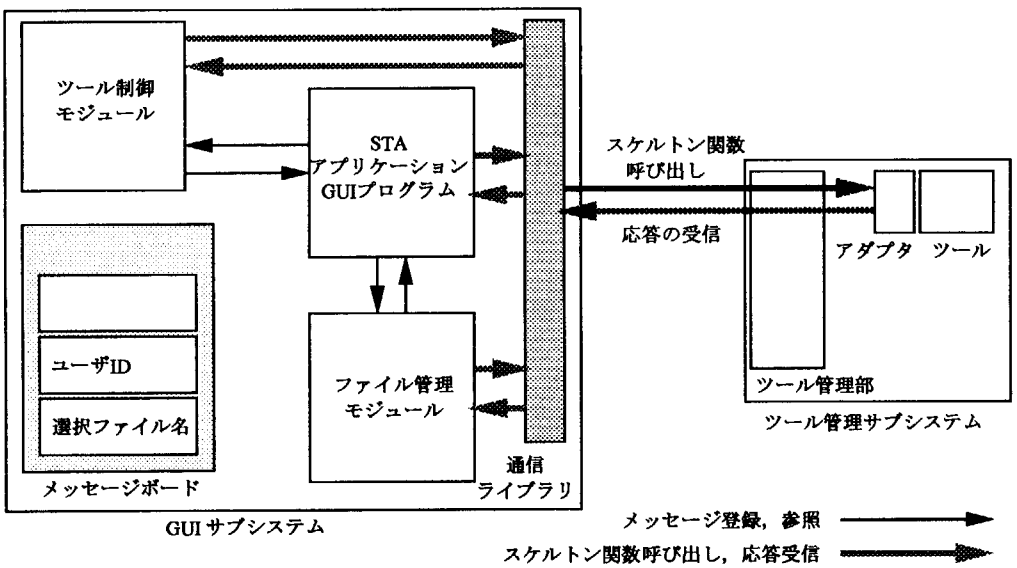

図 4: STA アプリケーション GUI プログラムの位置付け

2.3.2 STA アプリケーション GUI プログラムが備えるべき機能

STA アプリケーション GUI プログラムには、以下の機能が必要である。

ツール操作画面とイベント処理

ツールを実行するために必要な設定や、ツールの実行、操作を行う画面を提供し、画面の操作によって起きるイベントを処理するイベントハンドラを登録し、イベントに対応する処理を行う。

アダプタとの通信が必要な場合は、スタブ関数を呼び出す。

アダプタとの通信 (スタブ関数)

アダプタとの通信はスタブ関数で行う。アダプタには、各スタブ関数に対応するス

ケルトン関数を定義しておき、スタブ関数から呼び出すことができるようにする。

スタブ関数は、必要な情報を引数バッファにエンコードして、対応するスケルトン関数を呼び出す。引数バッファエンコードや、スケルトン関数の呼び出しは、Starpc Java 言語用ライブラリを使用して行う。

ツール実行状況表示

アダプタから受信した、ツール実行状況や、実行結果を解釈し、必要に応じて画面に表示する。

2.3.3 STA アプリケーション GUI プログラムの構成

STA アプリケーション GUI プログラムは, 利用者に GUI を提供するための GUI 部と, ツール管理サブシステムやアダプタとの通信を行うためのサーバスタブ部により構成される.

ツール提供者は, GUI 部を Starpc Java 言語用ライブラリの `SceWindow` クラスの派生クラスとして作成する. `SceWindow` クラスは, `java.awt.Frame` クラスの派生クラスであり, タイトルを持つウインドウを提供する. ツール提供者は, コンストラクタをオーバーライドして画面に必要な部品の生成等の初期化処理を行う.

`SceWindow` クラスで, クラス変数 `sceApplet`, `toolManager`, `fileManager` を提供するので, アプレット, ツール制御モジュール, ファイル管理モジュールの情報にアクセスする際は, これらの変数を参照し, ツール制御モジュールやファイル管理モジュールのメソッドを呼び出して必要な情報を取り出す.

サーバスタブ部は, `SceServerStub` クラスの派生クラスとして作成する. `SceServerStub` クラスには, 通信を行うためのメソッドが用意されており, サーバスタブ部では, それを利用してツール管理サブシステムやアダプタと通信を行う.

2.3.4 STA アプリケーション GUI プログラムの起動

STA アプリケーション GUI プログラムは、ツール制御モジュールから起動される。呼び出しは以下の手順で行う。

1. STA アプリケーション GUI プログラムのインスタンスの生成
2. STA アプリケーション GUI プログラムの `onRequestModule` メソッドの呼び出し
3. STA アプリケーション GUI プログラムの `setVisible()` メソッドの呼び出し

2.3.5 アダプタ及び、GUI サブシステムとのインタフェース

STA アプリケーション GUI プログラムとアダプタ、及び GUI サブシステムとのインタフェースは、Java 言語用ライブラリとして以下の機能を提供する。詳細については 4.2 アプレット関数仕様 を参照のこと。

アダプタとのインタフェース

アダプタの起動、終了、アダプタのスケルトン関数の呼び出しを行う機能。

“メッセージボード” を用いた情報の登録と取得

メッセージボードは、STA アプリケーション GUI プログラム間でのデータ交換のために、ツール実行モジュールが自由に使用できる。STA アプリケーション GUI プログラムは、「ヘッダ」と「データ」で構成される「メッセージ」をメッセージボードに登録する。データを参照する STA アプリケーション GUI プログラムは、「ヘッダ」をキーにして、登録されたメッセージを検索し参照する。ライブラリでは、メッセージの登録、参照、削除の機能を提供する。

メッセージのヘッダの先頭は “ツール名称” とし、それに続き “.” (ピリオド) を必ず付加する。それ以降は、STA アプリケーション GUI プログラムが自由に定義して利用できる。

ツール制御モジュールからの情報の取得

ツール制御モジュールで選択されているサーバ名、利用者のユーザ ID 等を取得する機能。

ファイル管理モジュールからの情報の取得

ファイル管理モジュールで選択されているファイル名を取得する機能。

他 STA アプリケーション GUI プログラム起動

他の STA アプリケーション GUI プログラムの起動を、ツール制御モジュールに依頼する機能。起動した STA アプリケーション GUI プログラムとのデータの交換は、メッセージボードを介するか各ツール毎に実装するツール連携用メソッドの呼び出しによって行う。

2.3.6 処理の流れ

STA アプリケーション GUI プログラムの処理概要を、図 5 に示す。

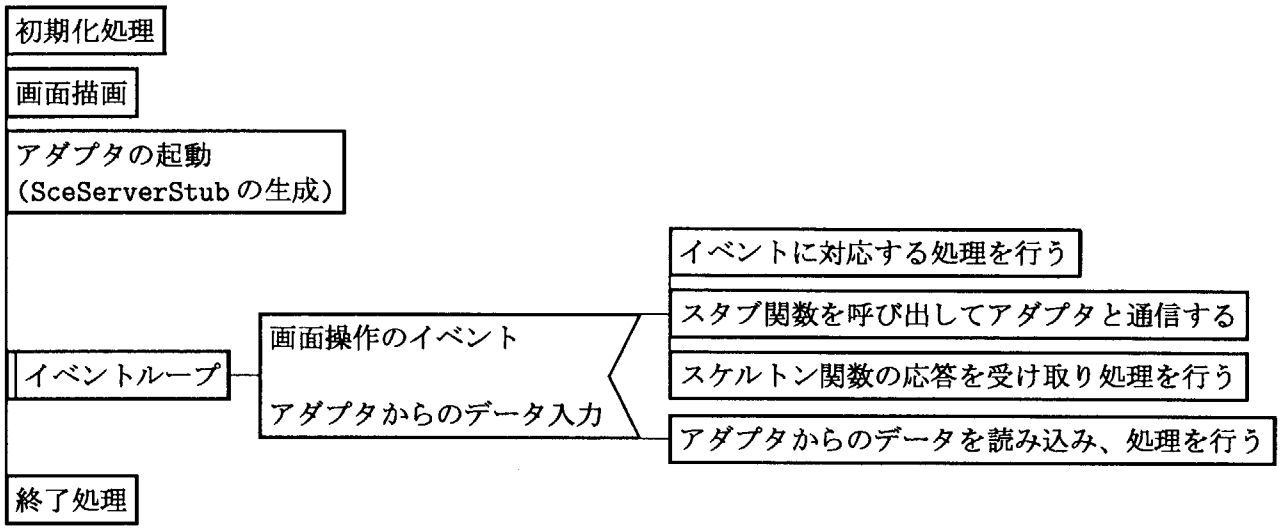

図 5: ツール実行モジュールの処理概要

3 インストール方法

Starpcライブラリを利用するためには，Starpcライブラリを対象計算機にインストールする必要がある．また，Starpcライブラリのインストールは，通信基盤SCE全体のインストール作業に含まれる．

本章では，SCE全体のインストールの流れを説明し，その中でStarpcライブラリのインストール作業についても触れる．

3.1 インストールの準備

SCEシステムのインストールに先立って，以下の項目について準備を行う必要がある．

1. SCEの各サブシステムのインストール対象計算機の決定
2. GUI管理サブシステム–GUIサブシステム間の通信で使用するport番号の取得

3.1.1 各サブシステムのインストール対象計算機の決定

SCEシステムは，GUIサブシステム、GUI管理サブシステム，バックエンドサブシステムから構成されており，全てのサブシステムをインストールする必要がある．

インストールに先立って，どの計算機にどのサブシステムをインストールするかを決定する必要がある．

GUI管理サブシステム
: システム中1台の計算機にインストールする．この計算機では，proyxプロセスとWWWサーバプロセスを常時動作させる必要がある．

ツール管理サブシステム
: SCEアプリケーションを実際に動作する計算機にインストールする．また，SCEアプリケーションが使用するデータファイルが存在する計算機にもインストールする必要がある．

GUIサブシステム
: GUI管理サブシステムから利用時にダウンロードして使用される．したがってGUI管理サブシステムが動作する計算機にインストールする．

3.1.2 port番号の取得

GUI管理サブシステムとGUIサブシステムでは，TCP/IPのsocketを使用した通信を行う．そのため，他のプログラムで使用していないport番号を1つ予約し，SCEシステムの専用portとして使用する．

使用できるport番号は計算機毎に異なるため，どの番号のportを使用できるかは，各計算機の管理者に問い合わせ，確認すること．

3.2 前提となるプログラムのインストール

SCEシステムの動作にはNEXUS通信ライブラリ及びWWWサーバが必要である。

SCEシステムでは、米国Argonne国立研究所で開発された通信ライブラリNEXUS-4.1.1を使用している．NEXUSは<ftp://ftp.mcs.anl.gov/pub/nexus/nexus-4.1.1.tar.gz>から入手できる．NEXUSのコンパイル及びインストールについては、NEXUS付属のドキュメントを参照のこと。

NEXUSは，GUI管理サブシステム及び，ツール管理サブシステムをインストールする全ての計算機にインストールする必要がある．

WWWサーバは，The Apache Software Foundation の Apache server 等が使用可能である．その他のWWWサーバでも，cgiが使用できるものであれば，使用可能である．Apache serverは<http://www.apache.org/dist/>から入手できる．インストールや設定の方法については，<http://www.apache.org/httpd.html>を参照のこと．

WWWサーバは，GUI管理サブシステムをインストールする計算機にインストールする必要がある．

3.3 ファイルの展開

ファイルの展開のための作業用ディレクトリを作成し，そのディレクトリを使用して作業を行う．このディレクトリは任意のディレクトリを使用することができるが，ここでは，例として，*/tmp*ディレクトリに`workdir`という名称のディレクトリを作成し作業用ディレクトリとして使用するものとする．以下，この作業用ディレクトリを*WORKDIR*と表記する．また，DATにSCEのファイルが格納されていることを想定して説明する．

SCEのインストールを行うためには，DATからtarコマンドを使用して必要なファイルをディスクに展開する必要がある．

上記の作業は以下のコマンドによって行うことができる．

```
% cd /tmp
% mkdir workdir
% cd workdir
% tar xvf /dev/rmt0
```

上記の例において，`/dev/rmt0`は，Solarisにおけるテープデバイスファイルであり、システムやその構成により異なったデバイスファイルになる。各システムにおけるデバイスファイルの名称は，各システムの管理者に問い合わせるか，各システムのマニュアルを参照のこと．

上記のコマンドを実行すると、図6に示すディレクトリが*WORKDIR*ディレクトリ下に作成される。

`client`及び`server`ディレクトリ内のサブディレクトリの説明を表2及び表3に示す．

3.4 GUI管理サブシステムのインストール

GUI管理サブシステムのインストール手順を以下に説明する．

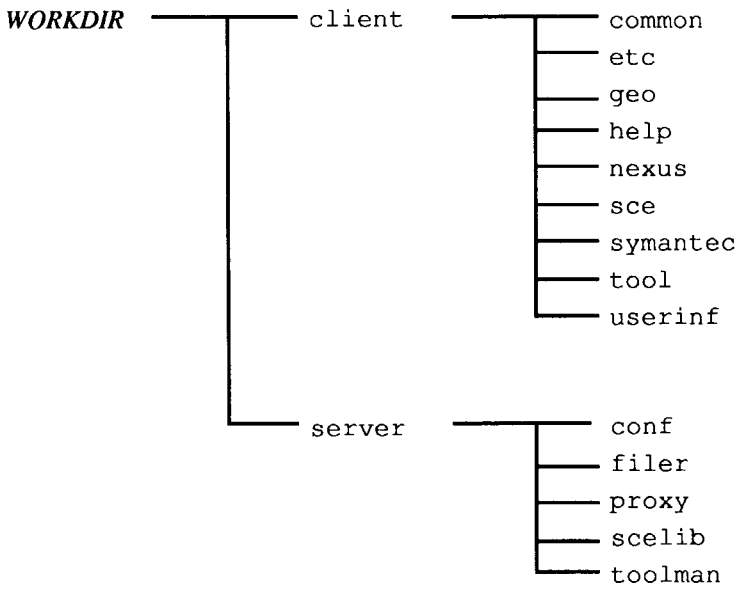

図 6: 展開後のディレクトリ構成

GUI管理サブシステムをインストールするために，まず，GUI管理サブシステムをインストールするディレクトリを用意する．ここでは，GUI管理サブシステムインストールディレクトリとして

/usr/local/SCE/main

を仮定する．

以下，GUI管理サブシステムインストールディレクトリを *SERVERDIR* と記述する．

(1) SCE/server/configure を実行する

```
% configure -i SERVERDIR -n NEXUSインストールディレクトリ
```

(2) SCE/server/conf/startsce ファイルに proxy のポート番号を記述する

```
PROXY_PORT=proxy のポート番号
⋮
```

(3) SCE/server/conf/toolpath に使用するサーバ名と、そのサーバでのツール管理サブシステムの所在を記述する

```
使用するサーバ名 : ツール管理サブシステムの所在
使用するサーバ名 : ツール管理サブシステムの所在
使用するサーバ名 : ツール管理サブシステムの所在
⋮
```

(4) SCE/server ディレクトリで、make clean を実行する

```
% make clean
```

(5) SCE/server ディレクトリで、以下のコマンドを実行しコンパイルを行う

表 2: clientディレクトリ

#	ディレクトリ	説明
1	common	共通部品格納ディレクトリ ツール共通で使用する部品 (CCSEtermなど) はこのディレクトリ下に任意のディレクトリを作成し格納する.
2	etc	SCE システム設定ファイル格納ディレクトリ
3	geo	symantec ライブラリ格納ディレクトリ
4	help	SCE システムヘルプファイル格納ディレクトリ
5	nexus	nexus ライブラリ格納ディレクトリ
6	sce	SCE システムライブラリ格納ディレクトリ
7	symantec	symantec ライブラリ格納ディレクトリ
8	tool	ツール実行モジュール格納ディレクトリ
9	userinf	ユーザ情報格納ディレクトリ

表 3: serverディレクトリ

#	ディレクトリ	説明
1	conf	SCE システム設定ファイル雛型格納ディレクトリ
2	filer	ファイル管理サブシステム　ソースディレクトリ
3	proxy	proxy　ソースディレクトリ
4	scelib	SCE システム C ライブラリ　ソースディレクトリ
5	toolman	ツール管理サブシステム　ソースディレクトリ

```
% make
```

(6) SCE/serverディレクトリで、以下のコマンドを実行しインストールを行う. このコマンドにより, C 言語用 Starpc ライブラリもインストールされる.

```
% make install
```

(7) インストール先ディレクトリ下の startsce をルート権限で実行する

```
% su
# ./startsce
```

GUI 管理サブシステムインストール後のディレクトリ構成を図 7 に示す。また、各ディレクトリの説明を表 4 に示す。

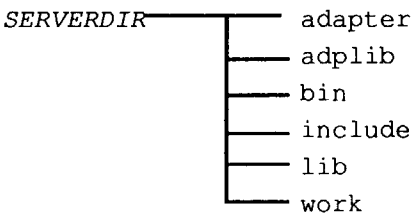

図 7: GUI 管理サブシステムインストール後のディレクトリ構成

表 4: GUI管理サブシステムのディレクトリ

#	ディレクトリ	説明
1	bin	SCEシステム格納ディレクトリ
2	lib	C言語用Starpcライブラリ格納ディレクトリ
3	include	インクルードファイル格納ディレクトリ
4	adapter	アダプタ格納ディレクトリ
5	adplib	アダプタ設定ファイル格納ディレクトリ
6	work	ツール固有ワークファイル格納ディレクトリ

3.5 ツール管理サブシステムのインストール

ツール管理サブシステムのインストール手順を以下に記述する.

ツール管理サブシステムをインストールするために, まず, ツール管理サブシステムをインストールするディレクトリを用意する. ここでは, ツール管理サブシステムインストールディレクトリとして

/usr/local/SCE/main

を仮定する.

以下, ツール管理サブシステムインストールディレクトリを *TOOLDIR* と記述する.

(1) SCE/server/configure を実行する

`% configure -i` *TOOLDIR* `-n` *NEXUS*インストールディレクトリ

(2) SCE/server 直下で、make clean を実行する

```
% make clean
```

(3) SCE/serverディレクトリで、以下のコマンドを実行しコンパイルを行う

```
% make
```

(4) SCE/serverディレクトリで、以下のコマンドを実行しインストールを行う

```
% make install
```

ツール管理サブシステムインストール後のディレクトリ構成を図8に示す。また、各ディレクトリの説明を表5に示す。

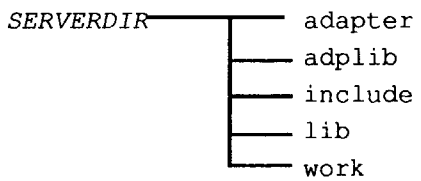

図 8: ツール管理サブシステムインストール後のディレクトリ構成

表 5: ツール管理サブシステムのディレクトリ

#	ディレクトリ	説明
2	lib	Cライブラリ格納ディレクトリ
3	include	インクルードファイル格納ディレクトリ
4	adapter	アダプタ格納ディレクトリ
5	adplib	アダプタ設定ファイル格納ディレクトリ
6	work	ツール固有ワークファイル格納ディレクトリ

3.6 GUIサブシステムのインストール

GUIサブシステムのインストール手順の概略を以下に示す.

(1) Cソースファイルのコンパイル
(2) Javaクラスファイルのインストール
(3) 設定ファイルの編集
(4) ディレクトリの所有者及びパーミッションの変更

3.6.1 Cソースファイルのコンパイル

WORKDIR/client/sceディレクトリにおいて, getinf.cをコンパイルしてgetinf.cgiを作成する.

```
% cc -o getinf.cgi getinf.c
```

3.6.2 Java言語用Starpcライブラリのインストール

GUIサブシステムはJavaアプレットであり, WWWサーバからクライアント計算機で動作するNetscapeCommunicatorにダウンロードされて実行される. そのため, GUIサブシステムの各ファイルは, SCEサーバ計算機のWWWサーバのドキュメントディレクトリ内に格納する必要がある. 以下, WWWサーバのドキュメントディレクトリを*WWWDOCDIR*と表記する. このディレクトリは各計算機毎に異なっているので, ディレクトリ名称はWWWサーバ管理者に問い合わせること.

尚, この作業はroot等WWWサーバのドキュメントディレクトリへの書き込みを行うことのできるユーザIDで行う必要がある.

GUIサブシステムを*WWWDOCDIR*/SCE/にインストールする場合を例とし, 手順を以下に示す. 以下, GUIサブシステムインストールディレクトリを*GUIDIR*と記述する.

```
% su
# cd WWWDOCDIR
# mkdir SCE
# cd WORKDIR
# cp -r client/* WWWDOCDIR/SCE/
```

3.6.3 GUIサブシステム設定ファイルの編集

GUIサブシステムの設定ファイルは *GUIDIR*/etc/server.txt である．このファイルを編集し，

- proxy が使用するポート番号
- ツール管理サブシステムを動作する計算機のホスト名

を設定する．

GUIサブシステム設定ファイルは設定項目の集合体であり書式は

項目名: 設定値

である．表6に項目名の一覧と各項目名の説明を記す．

表 6: GUIサブシステム設定ファイルの項目名一覧

#	項目名	説明
1	ProxyPort	proxy の port 番号
2	Server	ツール管理サブシステムを動作する計算機のホスト名

GUIサブシステム設定ファイルの例を以下に示す．

```
ProxyPort: 0000
Server: desr01.koma.jaeri.go.jp
Server: hi00011.koma.jaeri.go.jp
Server: fu00011.koma.jaeri.go.jp
Server: ne00011.koma.jaeri.go.jp
Server: ibmsp50a.koma.jaeri.go.jp
Server: cr00011.koma.jaeri.go.jp
```

3.6.4 ディレクトリの所有者及びパーミッションの変更

GUIDIR/userinf ディレクトリの所有者を WWW サーバプロセスの所有者の ID に，パーミッションを "755" (rwxr-xr-x) に変更する．WWW サーバプロセスの所有者の ID は nobody であることが多いが，ps コマンドで確認するか，又は WWW サーバ管理者に問い合わせること．

以下に，WWW サーバプロセスの所有者が nobody である場合を例に，上記変更を行うためのコマンドを示す．

```
# cd GUIDIR/SCE
# chown nobody userinf
# chmod 755 userinf
```

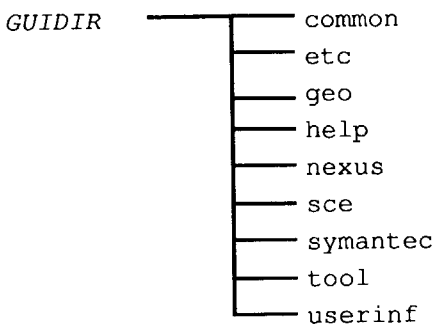

図 9: インストール後の GUI 管理サブシステムのディレクトリ構成

3.6.5 GUI サブシステムインストール後のディレクトリ構成

GUI サブシステムインストール後のディレクトリ構成を図 9 に示す．各ディレクトリの内容は表 3 と同様である．

4 Starpc関数仕様

4.1 アダプタ関数仕様

4.1.1 利用者作成関数

SCEInit

[概要]

アダプタ固有の初期処理を行う。

[インタフェース]

```
int SCEInit(const char *adapter)
```

[説明]

アダプタ期化処理が必要な場合、アダプタの提供者が作成する。アダプタ固有の初期処理としては、グローバル変数の初期化などがある。adapterには、アダプタ名称が設定される。

[返却値]

初期化が正常に終了した場合、0を返す。何らかの理由により、初期化に失敗し、アダプタの処理が続行不可能な場合、非0を返す。アダプタ本体はSCEInitの返却値が非0の場合、処理を終了する。この場合、SCEFinalは呼び出さないので、SCEInitの作成者は注意すること。

[備考]

アダプタ固有の初期処理が特に存在しない場合、アダプタの開発者はSCEInitを省略することができる。この場合、初期処理として、システムが用意しているデフォルトのSCEInitが呼び出される。

SCEFinal

[概要]

アダプタ固有の終了処理を行う。

[インタフェース]

```
int SCEFinal(int isQuit)
```

[説明]

アダプタ固有の終了処理が必要な場合、アダプタの提供者が作成する。アダプタ固有の終了処理としては、動作記録のファイルへの保存などがある。

アダプタ固有終了処理は、QUIT コマンドの処理時、又は、GUI サブシステムとの回線切断時に呼び出される。QUIT コマンド処理時は isQuit に非 0 が、回線切断時には 0 が設定される。

[返却値]

終了処理が正常に終了した場合、0 を返す。何らかの理由により終了処理が失敗した場合、非 0 を返す。アダプタ本体は、SCEFinal の返却値が 0 の場合（終了処理成功）、及び、回線が切断した場合（isQuit==0)、アダプタを終了する。それ以外の場合、アダプタ本体はコマンド処理を再開する。

[備考]

アダプタ固有の終了処理が特に存在しない場合、アダプタ開発者は SCEFinal を省略することができる。この場合、終了処理としてシステムが用意しているデフォルトの SCEFinal が呼び出される。

SCESkeltonTable

[概要]

同期型通信、非同期型通信、遅延評価型通信時の、関数番号とスケルトン関数との対応表

[インタフェース]

```
typedef struct {
        int func_id;
        SCEOutBuffer * (*func)(SCEInBuffer*);
} [] SCESkeltonTable_t SCESkeltonTable;
```

[説明]

SCESkeltonTableは、アダプタが受理する関数番号と、スケルトン関数の対応をとる。アダプタ本体は、同期型通信、非同期型通信、遅延評価型通信の要求があった時に、本テーブルを参照して関数番号を解析し、それに対応するスケルトン関数を呼び出す。

アダプタ作成者は、アダプタ内でSCESkeltonTableの実体を宣言しなければならない。SCESkeltonTableは関数番号とスケルトン関数の組みからなる配列として宣言されており、配列の終端は、関数番号に負の値を指定することによって表わす。

[宣言例]

```
SCEOutBuffer *dirs(SCEInBuffer *); /* ディレクトリ一覧取得 */
SCEOutBuffer *files(SCEInBuffer *); /* ファイル一覧取得 */
SCESkeltonTable_t SCESkeltonTable[] = {
        0,dirs,
        1,files,
       -1,NULL
};
```

SCESocketSkeltonTable

[概要]

SceSocket 通信時の、関数番号とスケルトン関数との対応表

[インタフェース]

```
Typedef struct {
        int func_id;
        void (*func)(int sock, SCEInBuffer *in);
} [] SCESocketSkeltonTable_t SCESocketSkeltonTable;
```

[説明]

SCESocketSkeltonTable は、アダプタが受理する関数番号と、スケルトン関数の対応をとる。アダプタ本体は、SceSocket 通信の要求があった時に、本テーブルを参照して関数番号を解析し、それに対応するスケルトン関数を呼び出す。

アダプタ作成者は、アダプタ内で SCESocketSkeltonTable の実体を宣言しなければならない。SCESocketSkeltonTable は関数番号とスケルトン関数の組みからなる配列として宣言されており、配列の終端は、関数番号に負の値を指定することによって表わす。

[宣言例]

```
void toolexec(int,SCEInBuffer*);      /* ツールの起動 */
SCESocketSkeltonTable_t SCESocketSkeltonTable[] = {
     0, toolexe,
     1, NULL,
};
```

スケルトン関数（同期型通信、非同期型通信、遅延評価型通信用）

[概要]

この関数内で、データのエンコード、デコード処理を行う。

[インタフェース]

```
SCEOutBuffer *skelton_func(SCEInBuffer *in)
```

[説明]

skelton_func はアダプタ開発者が（他の関数の名称と重複しないように）自由に定義してよい。skelton_func は SCESkeltonTable からポイントされ、アダプタ本体から skelton_func に対応する関数番号の処理として呼び出される。

in にはアダプタに送られてきたデータが格納されている。後述するデコード関数を使用して in から任意の型でデータを取り出すことができる。

[返却値]

アダプタからデータを送り出す必要がある場合、後述する出力バッファ生成関数を使用して作成したバッファを返却する。データを送り出す必要がない場合は NULL を返却する。

スケルトン関数（SceSocket 型通信用）

[概要]

この関数内で、クライアントの SceSocket を利用した通信を行う。

[インタフェース]

```
void socket_func(int sock, SCEInBuffer *in)
```

[説明]

socket_func は、アダプタ開発者が（他の関数の名称と重複しないように）自由に定義してよい。

socket_func は SCESocketSkeltonTable からポイントされ、アダプタ本体から socket_func に対応する関数番号の処理として呼び出される。このとき、アダプタは fork され、socket_func は子プロセスとして動作する。

sock には、クライアントと接続が確立されている、ソケットディスクリプタが格納されている。ユーザはこの sock を利用して、任意にクライアントとの通信を行うことができる。in にはアダプタに送られてきたデータが格納されている。後述するデコード関数を使用して in から任意の型でデータを取り出すことができる。

本関数の実行が終了すると、sock で表わされるソケットディスクリプタはクローズされ、子プロセスも終了する。

4.1.2 システム提供関数

SCEGetFloat

[概要]

SCEInBuffer から任意の長さの float データを取り出す。

[インタフェース]

```
int SCEGetFloat(SCEInBuffer *in,float *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの float データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetDouble

[概要]

SCEInBuffer から任意の長さの double データを取り出す。

[インタフェース]

```
int SCEGetDouble(SCEInBuffer *in,double *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの double データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetShort

[概要]

SCEInBuffer から任意の長さの Short データを取り出す。

[インタフェース]

```
int SCEGetShort(SCEInBuffer *in,short *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの short データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetInt

[概要]

SCEInBuffer から任意の長さの int データを取り出す。

[インタフェース]

```
int SCEGetInt(SCEInBuffer *in,int *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの int データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetLong

[概要]

SCEInBuffer から任意の長さの long データを取り出す。

[インタフェース]

```
int SCEGetLong(SCEInBuffer *in,long *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの long データを取り出す。この時、クライアント側の 8 バイト long をアダプタ側のサイズの long として取り出すので、桁あふれが発生する可能性がある。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetChar

[概要]

SCEInBuffer から任意の長さの char データを取り出す。

[インタフェース]

```
int SCEGetChar(SCEInBuffer *in,char *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの char データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetByte

[概要]

SCEInBuffer から任意の長さの byte データを取り出す。

[インタフェース]

```
int SCEGetByte(SCEInBuffer *in,unsigned char *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの byte データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGet2Byte

[概要]

SCEInBuffer から任意の長さの 2byte データを取り出す。

[インタフェース]

```
int SCEGet2Byte(SCEInBuffer *in,unsigned char *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの 2byte データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGet4Byte

[概要]

SCEInBuffer から任意の長さの 4byte データを取り出す。

[インタフェース]

```
int SCEGet4Byte(SCEInBuffer *in,unsigned char *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの 4byte データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGet8Byte

[概要]

SCEInBuffer から任意の長さの 8byte データを取り出す。

[インタフェース]

```
int SCEGet8Byte(SCEInBuffer *in,unsigned char *dest,int count)
```

[説明]

アダプタに送られてきたデータから任意の長さの 8byte データを取り出す。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出したデータの個数を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetString

[概要]

SCEInBuffer から文字列データを取り出す。

[インタフェース]

```
int SCEGetString(SCEInBuffer *in, char *dest)
```

[説明]

アダプタに送られてきたデータから文字列データを取り出す。この時、本関数内では、まず SCEGetInt 関数を発行して文字列長さを取得し、その後 SCEGetChar 関数を発行して文字列データを取り出すという処理を行う。

in にはスケルトン関数の第 1 引数を指定する。

dest にはデータのコピー先を指定する。

[返却値]

正常にデータの取り出しが行われた場合、取り出した文字列の文字数を返却する。入力バッファの終端に達したために取り出しに失敗した場合、SCESTRINGEND を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

SCEGetStringlen

[概要]

SCEInBuffer から文字列データの文字数を取り出す。

[インタフェース]

```
int SCEGetString(SCEInBuffer *in)
```

[説明]

アダプタに送られてきたデータから文字列の文字数を取り出す。本関数を利用した後、SCEGetString 関数を呼び出し文字列データを取り出す事が出来る。SCEGetString 関数を呼び出す際、指定するデータのコピー先の領域を確保する場合本関数の利用で文字数を予め把握できる。

in にはスケルトン関数の第 1 引数を指定する。

[返却値]

正常にデータの取り出しが行われた場合、文字数を返却する。入力バッファの終端に達したために取り出しに失敗した場合、SCESTRINGEND を返却する。また、通信相手側でデータの送信を中断したために取り出しに失敗した場合、SCESTOP を返却する。

[備考]

サンプルプログラムを以下に記す。

```
#include "sce.h"
char getFile(SCEInbuffer *in)
{
    char *file;
    int len;
    int ret;

    len = SCEGetStringlen(in);
    if(len < 0){
        return null; }
    file = (char *)malloc(len+1);
    ret = SCEGetString(in, file);
    if(ret < 0){
        returen null; }
    return file;
}
```

SCEMalloc

[概要]

任意の大きさの領域を確保する。

[インタフェース]

```
void *SCEMalloc(size_t size)
```

[説明]

size で指定された大きさの領域を確保する。なお、本関数で確保された領域は、アダプタ本体のコマンド処理部に戻ったときに、自動的に開放される。

[返却値]

正常に領域の確保が行えたときは、その確保した領域のアドレスを返却する。何らかの理由で領域の確保に失敗した場合は、NULL を返却する。

SCECreateBuffer

[概要]

SCEOutBuffer を生成する。

[インタフェース]

```
SCEOutBuffer * SCECreateBuffer()
```

[説明]

アダプタから送信するデータを格納するための出力バッファを生成する。本関数によって生成された出力バッファに対して、後述するエンコード関数を使用して送信するデータを格納する。

本関数によって生成された出力バッファは、スケルトン関数の返却値として使用された時や、アダプタ連携関数を使用して他アダプタに送信された時に自動的に開放される。もしも出力バッファを送信しないときは、後述する SCEDestroyBuffer 関数を使用して出力バッファを破棄しなければならない。

[返却値]

正常にバッファ生成が行われたときは、SCEOutBuffer を返却する。何らかの理由で生成に失敗した場合は NULL を返却する。

SCEDestroyBuffer

[概要]

SCECreateBuffer 関数で生成した出力バッファを破棄する。

[インタフェース]

```
void SCEDestroyBuffer(SCEOutBuffer *out)
```

[説明]

SCECreateBuffer 関数で生成した出力バッファが不用になったときは、本関数を使用して出力バッファの破棄を行なう。出力バッファは、通信相手に送信された後に自動的に破棄されるが、もしも何らかの理由で出力バッファを通信相手に送信しなかったときは、必ず本関数を使用して出力バッファの破棄を行なわなければならない。

SCEPutFloat

[概要]

SCEOutBuffer に任意の長さの float データを格納する。

[インタフェース]

```
int SCEPutFloat(SCEOutBuffer *out, float *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの float データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutDouble

[概要]

SCEOutBuffer に任意の長さの Double データを格納する。

[インタフェース]

```
int SCEPutDouble(SCEOutBuffer *out, double *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの double データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutShort

[概要]

SCEOutBuffer に任意の長さの short データを格納する。

[インタフェース]

```
int SCEPutShort(SCEOutBuffer *out, short *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの short データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutInt

[概要]

SCEOutBuffer に任意の長さの int データを格納する。

[インタフェース]

```
int SCEPutInt(SCEOutBuffer *out, int *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの int データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutChar

[概要]

SCEOutBuffer に任意の長さの char データを格納する。

[インタフェース]

```
int SCEPutChar(SCEOutBuffer *out, char *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの char データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutLong

[概要]

SCEOutBuffer に任意の長さの long データを格納する。

[インタフェース]

```
int SCEPutLong(SCEOutBuffer *out, long *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの long データを格納する。この時、本関数内では指定されたアダプタ側固有のサイズの long データを 8 バイトの long データに拡張してからバッファに格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutByte

[概要]

SCEOutBuffer に任意の長さの byte データを格納する。

[インタフェース]

```
int SCEPutByte(SCEOutBuffer *out, unsigned char *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの byte データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPut2Byte

[概要]

SCEOutBuffer に任意の長さの 2byte データを格納する。

[インタフェース]

```
int SCEPut2Byte(SCEOutBuffer *out, unsigned char *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの 2byte データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPut4Byte

[概要] SCEOutBuffer に任意の長さの 4byte データを格納する。

[インタフェース]

```
int SCEPut4Byte(SCEOutBuffer *out, unsigned char *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの 4byte データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPut8Byte

[概要]

SCEOutBuffer に任意の長さの 8byte データを格納する。

[インタフェース]

```
int SCEPut8Byte(SCEOutBuffer *out, unsigned char *data, int count)
```

[説明]

アダプタから送信する出力バッファに任意の長さの 8byte データを格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

count にはコピーする個数を指定する。

[返却値] 正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutString

[概要]

SCEOutBuffer に文字列データを格納する。

[インタフェース]

```
int SCEPutString(SCEOutBuffer *out, char *data)
```

[説明]

アダプタから送信する出力バッファに文字列データを格納する。この時、本関数内では、まず SCEPutInt 関数で文字列の長さをバッファに入れ込んだ後に、SCEPutChar 関数で文字列を格納する。

out には SCECreateBuffer で生成したバッファを指定する。

data にはデータのコピー元を指定する。

[返却値]

正常にデータの格納が行われた場合、0 を返却する。格納に失敗した場合、非 0 を返却する。

SCEPutException

[概要]

クライアントに例外発生を通知する。

[インタフェース]

```
int SCEPutException(SCEOutBuffer *out, int error_code)
```

[説明]

アダプタで発生した例外をクライアント側に通知するために、エラーコードを出力バッファに格納する。

out にはSCECreateBufferで生成したバッファを指定する。

error_codeにはクライアント側に通知するエラーコードを指定する。

[返却値]

正常にエラーコードの格納が行われた場合、0を返却する。格納に失敗した場合、非0を返却する。

[備考]

実際にクライアント側に例外が通知されるタイミングは、アダプタ側のスケルトン関数からreturnしたときである。

SCEGetException

[概要]

連携先アダプタで発生した例外を取得する。

[インタフェース]

```
int SCEGetException(SCEInBuffer *in)
```

[説明]

SCEConnectで連携したアダプタ側で発生した例外を取得する。

inにはSCESendRecvやSCELazyRecvで取得したバッファを指定する。

[返却値]

連携先アダプタでSCEPutExceptionによって設定されたエラーコードを返却する。

SCEConfigData

[概要]

アダプタ設定ファイルの参照処理。

[インタフェース]

```
const char *SCEConfigData(const char *key)
```

[説明]

アダプタ設定ファイルに格納されている設定情報の検索を行う。

key に設定ファイル中のキーを指定する。

[返却値]

正常に設定情報を取得できた場合、設定情報を返却する。

key に対応する設定情報が存在しない場合、NULL を返却する。

SCESystemConfigData

[概要]

SCE システム設定ファイルの参照処理。

[インタフェース]

```
const char *SCESystemConfigData(const char *key)
```

[説明]

SCE システム設定ファイルに格納されている設定情報の検索を行う。

key に設定ファイル中のキーを指定する。

[返却値]

正常に設定情報を取得できた場合、設定情報を返却する。

key に対応する設定情報が存在しない場合、NULL を返却する。

[備考]

次の key が指定できる。

#	key	説明
1	Bin-Dir	アダプタのロードモジュールが格納されているディレクトリを取得する。
2	Lib-Dir	アダプタの設定ファイルが格納されているディレクトリを取得する。
3	Work-Dir	アダプタの共通作業用ディレクトリを取得する。
4	Path	アダプタ起動直後の環境変数 PATH の値を取得する。

SCEConnect

[概要]

他アダプタとの接続を行う。

[インタフェース]

```
SCEHandle *SCEConnect(const char *host, const char *adapter)
```

[説明]

他のアダプタを起動し、それと接続する。

host に接続先アダプタがあるサーバの名称を指定する。

adapter に接続先アダプタの名称を指定する。

[返却値]

正常に他アダプタとの接続が確立した場合、そのアダプタと通信するためのハンドルを返却する。

接続に失敗した場合、NULL を返却する。

SCEDisconnect

[概要]

接続済みの他アダプタとの切断を行う。

[インタフェース]

```
int SCEDisconnect(SCEHandle *handle)
```

[説明]

SCEConnect で接続した他アダプタとの切断を行う。

handle に SCEConnect の返却値を指定する。

[返却値] 切断に成功した場合、0 を返却する。何らかの理由により切断に失敗した場合、非 0 を返却する。

SCESendRecv

[概要]

他アダプタに同期通信でデータを送る。

[インタフェース]

```
SCEInBuffer *SCESendRecv(SCEHandle *handle,
                         int func_id,
                         SCEOutBuffer *out)
```

[説明]

SCEConnect で接続した他アダプタに同期通信でデータを送る。

handle に転送先のハンドル（SCEConnect の返却値）を指定する。

func_id に転送先の関数番号を指定する。

out に転送するデータ（SCECreateBuffer の返却値）を指定する。

[返却値]

転送に成功した場合、他アダプタから送られてきたデータを返却する。

転送に失敗した場合、NULL を返却する。

[備考]

本関数は他アダプタからデータが送られてくるまでブロックする。

SCEAsyncSend

[概要]

他アダプタに非同期通信でデータを送る。

[インタフェース]

```
int SCEAsyncSend(SCEHandle *handle, int func_id, SCEOutBuffer *out)
```

[説明]

SCEConnect で接続した他アダプタに非同期通信でデータを送る。

handle に転送先のハンドル（SCEConnect の返却値）を指定する。

func_id に転送先の関数番号を指定する。

out に転送するデータ（SCECreateBuffer の返却値）を指定する。

[返却値]

転送に成功した場合、0 を返却する。

転送に失敗した場合、非 0 を返却する。

[備考]

本関数を使用して他アダプタにデータを転送した場合、他アダプタから本アダプタに対してデータを返送することはできない。

SCELazySend

[概要]

他アダプタに遅延評価型通信でデータを送る。

[インタフェース]

```
int SCELazySend(SCEHandle *handle,
                int func_id,
                SCEOutBuffer *out,
                SCETicket *ticket)
```

[説明]

SCEConnect で接続した他アダプタに遅延評価型通信でデータを送る。

handle に転送先のハンドル（SCEConnect の返却値）を指定する。

func_id に転送先の関数番号を指定する。

out に転送するデータ（SCECreateBuffer の返却値）を指定する。

ticket に他アダプタから送られてきたデータを取得する際に使用する SCETicket の領域のアドレスを指定する。

[返却値]

転送に成功した場合、0 を返却し、ticket に SCETicket が入る。

転送に失敗した場合、非 0 を返却する。

[備考]

本関数を使用して他アダプタにデータを転送した場合、他アダプタから本アダプタに対して送られたデータは、後述する SCELazyRecv を使用して取得する。

SCELazyProbe

[概要]

他アダプタからデータが届いているかを調べる。

[インタフェース]

```
int SCELazyProbe(SCEHandle *handle, SCETicket ticket)
```

[説明]

SCELazySend で他アダプタにデータを送った後に、他アダプタからのデータが本アダプタに届いているかどうかを調べる。

handle に転送先のハンドル（SCEConnect の返却値）を指定する。

ticket にチケット（SCELazySend の返却値）を指定する。

[返却値]

他アダプタからのデータが届いていた場合、非 0 を返却する。

データが届いていなかった場合、0 を返却する

[備考]

本関数はブロックしない。

SCELazyRecv

[概要]

他アダプタから届いたデータを取得する。

[インタフェース]

```
SCEInBuffer *SCELazyRecv(SCEHandle *handle, SCETicket ticket)
```

[説明]

SCELazySend で他アダプタにデータを送った後に、他アダプタから本アダプタに返送されたデータを取得する。

handle に転送先のハンドル（SCEConnect の返却値）を指定する。

ticket にチケット（SCELazySend の返却値）を指定する。

[返却値]

転送に成功した場合、他アダプタから送られてきたデータを返却する。

転送に失敗した場合、NULL を返却する。

[備考]

本関数は他アダプタからデータが送られてくるまでブロックする。

SCECreateSocket

[概要]

他アダプタとのSceSocketを作成する。

[インタフェース]

```
int SCECreateSocket(SCEHandle *handle, int func_id, SCEOutBuffer *out)
```

[説明]

SCEConnectで接続した他アダプタと、通信するためのSceSocketを作成する。

handleにSceSocket接続先のハンドル（SCEConnectの返却値）を指定する。

func_idにSceSocket接続先の関数番号を指定する。

outに転送するデータ（SCECreateBufferの返却値）を指定する。

[返却値]

SceSocketの作成に成功した場合、ソケットディスクリプタを返却する。ユーザは、このソケットディスクリプタを利用して、他アダプタとの通信を行うことができる。

SceSocketの作成に失敗した場合、負の値を返却する。

SCEKeepAlive

[概要]

アダプタ動作モードを設定する。

[インタフェース]

```
int SCEKeepAlive(int mode)
```

[説明]

アダプタ動作モードを設定する。通常動作モードのアダプタはログアウト時に自動的に終了するが、デーモン動作モードのアダプタはログアウト時にも終了せずに、再ログインした際に再び接続できる。

modeにアダプタ動作モードを指定する。通常動作モードにする場合はSCENORMALを、デーモン動作モードにする場合はSCEDAEMONを指定する。

[返却値]

成功時は0を、失敗時は非0を返却する。

SCECheckProxy

[概要]

再ログイン待ち状態であるか否かを取得する。

[インタフェース]

```
int SCECheckProxy()
```

[説明]

アダプタがデーモン動作モードであるときに、再ログイン待ち状態になっているか否かを取得する。

[返却値]

再ログイン待ち状態である場合はSCEDAEMONを返却する。

再ログイン待ち状態でない（つまりログイン中である）場合はSCENORMALを返却する。

失敗時は負を返却する。

4.2 アプレット関数仕様

4.2.1 SceWindow クラス

onRequestModule

[概要]

SceToolManager.requestModule メソッドが呼ばれたときに, SceToolManager から呼出されるメソッド.

[メソッド仕様]

```
public SceHandle onRequestModule(SceWindow caller,
                                 boolean new_request,
                                 Vector params);
```

[説明]

本メソッドは SceToolManager.requestModule メソッドが呼ばれたときに, SceToolManager から呼出される. 本メソッドでは, ツール連携時やツール生成時の初期化処理を行い, ツール連携に必要な SceHandle を作成し返却する.

caller には requestModule や requestCategory メソッドを呼出した STA アプリケーション GUI プログラムが設定される. ツール制御モジュール画面のボタンが押されてツールが起動した場合には, null が設定される.

new_request には起動モードが設定される. new モードのとき true が, renew モードのとき false が設定される.

param には requestModule や requestCategory で指定されたユーザー定義の引数が設定される.

ツール生成時やツール連携時に個別の処理を行う必要のあるツールは本メソッドをオーバーライドし, 必要な処理を行う. また, renew に対応して連携を行うツールは, 本メソッドをオーバーライドして, 連携を要求してきたツールに対しユニークな連携 ID を設定した SceHandle を返す. SceHandle の window には, 連携を行うツールのポインタを設定する.

本メソッドのデフォルト動作では, caller が null の場合, または, new_request が false の場合には null を返し, それ以外の場合には handleId に 0 を, window に this を設定した SceHandle を返す.

連携 ID の使用法は, 各ツール毎に規定するものとし, SCE では規定しない. 例えば, 連携するツールに対し, メソッドの引数として, この ID を指定することで, メソッドを呼ばれた側のツールはどのツールがメソッドを呼んだのかを識別することができるようになり, ツール毎に異なった対応をすることができる.

SceHandle の new_request は, onRequestModule メソッド終了後, ツールマネージャが設定するため,

onRequestModuel メソッドでは設定する必要はない.

[返却値]

ツール連携に成功した場合, 必要な情報を設定した SceHandle を返す. ツール連携に失敗した場合 null を返す.

[使用例]

```
public SceHandle onRequestModule(SceWindow caller, boolean new_request,
                                                          Vector params)
{
  SceHandle handle;
  if(caller == null) {
    // ツール制御モジュールのボタンが押された場合
    return null;
  }
  // ツール連携の場合
  handle = new SceHandle();
  // SceHandle の設定
  if(!new_request) {
    handleID++;
    handle.setHandleId(handleID);
  } else {
    handle.setHandleId(0);
  }
  handle.setWindow(this);
   :
  // ツールの初期化処理を行う
   :
  return handle;
}
```

getParents

[概要]

ツール連携情報 (SceToolRelation) を取得する.

[メソッド仕様]

```
Vector[] getParents();
```

[説明]

ツール連携で起動された際に作成されるツール起動情報オブジェクトはSceWindow内部でVectorの形で管理されている. 配列の先頭から順に最新の起動情報が入っている. 当メソッドではSceWindow内部で管理されているVectorそのものを返却する.

[返却値]

返却値として,SceWindow内部で管理されているVectorを返す. データが登録されていない場合長さが0のVectorを返す.

[使用例]

```
Vactor   vparent;
 :
vparent = sceWindw.getParents();
if (vparent.size() != 0) {
  // 起動情報取得成功
} else {
  // 起動情報が登録されていない
}
```

4.2.2 SceToolRelation クラス

getToolName

[概要]
連携元ツール名称を取得する.

[メソッド仕様]

```
String getToolName();
```

[説明]
ツール連携起動情報オブジェクトから連携元ツール名称を取得する. 上記オブジェクトへの名称等の設定は SceToolManager で設定される.

[返却値]
連携元ツール名称

[使用例]

```
String tool;
SceToolRelation rela;
:
rela = sceWindow.getRelation();
tool = rela.getToolName();
if (tool != null && tool.length() != 0) {
  // 取得に成功した
}else{
  // 取得に失敗した
}
```

getCategoryName

[概要]

連携元カテゴリ名称取得.

[メソッド仕様]

```
String getCategoryName();
```

[説明]

ツール連携起動情報オブジェクトから連携元カテゴリ名称を取得する. 上記オブジェクトへの名称等の設定はSceToolManagerで設定される.

[返却値]

連携元カテゴリ名称

[使用例]

```
String category;
SceToolRelation rela;
:
rela = sceWindow.getRelation();
category = rela.getCategoryName();
if (category != null && category.length() != 0) {
  // 取得に成功した
}else{
  // 取得に失敗した
}
```

getID

[概要]

連携 ID を取得する.

[メソッド仕様]

```
int getID();
```

[説明]

ツール連携起動情報オブジェクトから連携 ID を取得する. 上記オブジェクトへの連携 ID の設定は SceToolManager 内の requestModule を実行する際の引数で設定される.

[返却値]

連携 ID requestModule の際設定されなかった場合デフォルト値の－1が返却される

[使用例]

```
int ID;
SceToolRelation rela;
:
rela = sceWindow.getRelation();
ID = rela.getID();
if (ID > -1) {
  // 連携 ID は設定されており取得に成功した
}else{
  // 連携 ID は設定されていなかった、または取得に失敗した
}
```

4.2.3 SceMessage クラス

write

[概要]

メッセージボードへのメッセージの書き込みを行う.

[メソッド仕様]

```
int write(String header, String data);
```

[説明]

メッセージボードに「ヘッダ」と「データ」の組みを登録する. header には, メッセージボードに登録するためのヘッダを文字列で指定する. data には, メッセージボードに登録するためのデータを文字列で指定する. 指定されたヘッダの項目がすでに存在する場合メッセージボードに登録されているデータは上書きされる.

[返却値]

0 : 登録成功
-1: 登録失敗

[使用例]

```
int rtn;
:
rtn = messageBoard.write("Editor","ABCDEFG");
if (rtn == 0) {
  //登録成功
}else{
  //登録失敗
}
```

read

[概要]

メッセージボードからメッセージの読み込みを行う.

[メソッド仕様]

```
String[] read(String header);
```

[説明]

メッセージボードから, 指定したヘッダを持つデータを読み込む. header には, メッセージボードから読み込むためのヘッダを文字列で指定する.

[返却値]

返却値として, メッセージボードのデータを返す. データが登録されていない場合 null を返す.

[使用例]

```
String   str[];
 :
str = messageBoard.read("Editor");
if (str != null) {
  // 読み込み成功
} else {
  // 読み込み失敗
}
```

delete

[概要]

メッセージボードからメッセージの読み込みを行い, 読み込んだメッセージを削除する

[メソッド仕様]

```
String[] delete(String header);
String[] delete(String header, String data);
```

[説明]

メッセージボードから, 指定したヘッダ, または指定したヘッダとデータを持つデータを読み込み, その後, 読み込んだデータを, メッセージボードから削除する. 読み込んだデータは, 返却値として返される. headerには, メッセージボードから削除するデータに対応するヘッダを, 文字列で指定する. dataには, メッセージボードから削除するデータを, 文字列で指定する.

[返却値]

返却値として, 削除したメッセージボードのデータを返す. データが登録されていない場合, nullを返す.

[使用例]

```
String   data[];
 :
data = messageBoard.delete("Editor","ABC");
if (data != null) {
  // 削除成功
} else {
  // 削除失敗
}
```

4.2.4 SceServerStub クラス

SceServerStub クラスは, ツール管理サブシステムやアダプタと通信を行うためのメソッドを提供する.

コンストラクタ

[概要]

ツール管理サブシステムに接続し, ツールの起動を要求する.

[メソッド仕様]

```
SceServerStub();
SceServerStub(String hostname, String toolname)
              throws ServerNotReadyException,
              ServerNotConnectException;
```

[説明]

ツール管理サブシステムや, アダプタとの通信の準備を行う. 引数が指定された場合は, hostname で指定したホストと接続して toolname で指定したツールを起動する.

[返却値]

hostname で指定したホスト上のツール管理サブシステムがまだ起動されていない場合は, ServerNotReadyException を発行する. また, 通信経路の確立に失敗した場合は,
ServerNotConnectException を発行する. ServerNotConnectException を発行した場合は, 以下のコードを設定する.

-1 : 通信用ソケットの生成に失敗した
-2 : ツール管理サブシステムの起動に失敗した
-3 : IO エラー

[使用例]

```
public class Server_Stub extends SceServerStub
      public Server_Stub() {
            try{
                  super("chobi.koma.jaeri.go.jp","editor");
            }
            catch(ServerNotReadyException e) {
                  // エラー処理
            }
            catch(ServerNotConnectException e) {
                  // エラー処理

                        }
                  }
            }
```

connect

[概要]

ツール管理サブシステムに接続する.

[メソッド仕様]

```
int connect(String hostname);
```

[説明]

GUIサブシステムとツール管理サブシステムとの間の通信路を確立する. hostnameには, 通信先のツール管理サブシステムのホスト名を指定する. ホスト名には, 次の2つの形式がある.

Fully Qualified Domain Name(FQDN)形式例えば, "chobi.koma.jaeri.go.jp"のように, フルドメイン形式でホスト名を指定する. DNSを使ってホスト名からIPアドレスを参照できる必要がある.

Dot Address形式 A.B.C.D 形式のIPアドレス. A, B, C, Dはそれぞれ0〜255の10進値とし, ホストのIPアドレスを表す.

[返却値]

0 : 接続に成功
-1 : 接続に失敗.

[使用例]

```
int rtn;
:
Server_Stub stub = new Server_Stub();
:
rtn = stub.connect("chobi.koma.jaeri.go.jp");
switch(rtn) {
case  0:
  // 接続成功
case  -1:
  // 接続に失敗. }
```

toolExec

[概要]

ツール管理サブシステムにツールの起動を要求する.

[メソッド仕様]

```
int toolExec(String toolname);
```

[説明]

toolnameには, ツール管理サブシステムに登録されているツール名を指定する. 指定したツール名が, ツール管理サブシステムに登録されていない場合, エラーとなる. toolnameで指定したツールは, connectメソッドで指定したユーザの権限でツールを実行する. 本メソッドをコールする前に, あらかじめSceServerStubクラスの, connectメソッドをコールしなければならない.

[返却値]

0 : ツールの起動に成功した
-1 : サーバがまだ使用可能になっていない
-2 : サーバ起動に失敗した
-3 : 引数エラー
-5 : 通信エラー

[使用例]

```
int   rtn;
 :
Server_Stub stub = new Server_Stub();
int rtn;
rtn = stub.connect("chobi.koma.jaeri.go.jp");
 :
}
 :
rtn = stub.toolExec("editor");
switch(rtn) {
case  0:
  // ツールの起動に成功
case  -1:
  // サーバがまだ使用可能になっていない
case  -2:
  // サーバ起動に失敗した
case  -3:
  // 引数エラー
case  -5:
  // 通信エラー
}
```

isToolExec

[概要]

アダプタが起動されているかどうかを問い合わせる.

[メソッド仕様]

```
void isToolExec()
     throws AdapterNotReadyException, AdapterNotConnectException;
```

[説明]

起動要求を行ったアダプタが, 起動されているかどうかを問い合わせる.

[返却値]

本メソッドの返却値はない. ただし, アダプタがまだ起動されていない場合は, AdapterNotReadyException を発行する. また, アダプタの起動に失敗した場合は, AdapterNotConnectException を発行する. AdapterNotConnectException を発行した場合は, 以下のコードを設定する.

-1 : アダプタの起動に失敗した
-2 : アダプタが存在しない
-3 : IO エラー

[使用例]

```
try {
  isToolExec();
}
catch (AdapterNotReadyException e) {
  // リトライ処理
}
catch (AdapterNotConnectException e) {
  // エラー処理
}
```

isToolExecWait

[概要]

アダプタが起動の成否を問い合わせる.

[メソッド仕様]

```
void isToolExecWait() throws AdapterNotConnectException;
```

[説明]

起動要求を行ったアダプタの起動の成否を問い合わせる. まだ起動の成否を受信していない場合は, 処理をブロックする.

[返却値]

本メソッドの返却値はない. ただし, アダプタの起動に失敗した場合は, AdapterNotConnectException を発行する.AdapterNotConnectException を発行した場合は, 以下のコードを設定する.

-1 : アダプタの起動に失敗した
-2 : アダプタが存在しない
-3 : IO エラー

[使用例]

```
try {
  isToolExecWait();
}
catch (AdapterNotConnectException e) {
  // エラー処理
}
```

getToolList

[概要]

ツール管理サブシステムに登録されているツールの一覧を取得する.

[メソッド仕様]

```
String[] getToolList();
```

[説明]

接続したツール管理サブシステムに於いて, システムに登録されているツールの一覧を文字列で返す.

[返却値]

返却値として, ツールの一覧を返す. ツールが 1 つも登録されていない場合, null を返す.

[使用例]

```
String list[];
 :
Server_Stub stub = new Server_Stub();
  // サーバスタブクラスの生成
 :
list = stub.getToolList();
if (list != null) {
  // list にツールの一覧が格納される
} else {
  // ツールが 1 つも登録されていない
}
```

getToolDetails

[概要]

ツール管理サブシステムに登録されている各々のツールの詳細情報を取得する.

[メソッド仕様]

```
String[] getToolDetails(String toolname);
```

[説明]

接続したツール管理サブシステムに於いて, システムに登録されているツールの詳細情報を文字列で返す.

[返却値]

返却値として, ツールの詳細情報を返す. 詳細情報が登録されていない場合, null を返す.

[使用例]

```
String   inf[];
 :
Server_Stub stub = new Server_Stub();
  // サーバスタブクラスの生成
 :
inf = stub.getToolDetails("editor");
if (inf != null) {
  // inf にツールの詳細情報が格納される
} else {
  // 詳細情報が登録されていない
}
```

quit

[概要]

ツール管理サブシステムやアダプタとの通信を終了する.

[メソッド仕様]

メソッドの仕様を以下に示す.

```
int quit();
```

[説明]

ツール管理サブシステムやアダプタに通信の終了を通知する.

[返却値]

0 : 通信終了
-1 : 処理が完了していない等の理由により終了できない
-5 : IO エラー

[使用例]

```
int   rtn;
 :
Server_Stub stub = new Server_Stub();
  // サーバスタブの生成
 :
rtn = stub.quit();
if (rtn == 0) {
  // 通信終了
} else {
  // 終了処理失敗
}
stub.close();
```

close

[概要]

通信用ソケットを切断する.

[メソッド仕様]

```
void close();
```

[説明]

ツール管理サブシステムやアダプタとの通信に使用するソケットをクローズする.

[返却値]

本メソッドの返却値はない.

[使用例]

```
int   rtn;
 :
Server_Stub stub = new Server_Stub();
  // サーバスタブの生成
 :
rtn = stub.quit();
if (rtn == 0) {
  // 通信終了
} else {
  // 終了処理失敗
}
stub.close();
```

getSceSocket

[概要]

SceSocket を使用した通信を行う.

[メソッド仕様]

```
SceSocket getSceSocket(int funcID, SceSocketBuffer buf);
```

[説明]

SceSocket 通信のためのソケットを生成する. アダプタ側には funcID で指定した SceSocket 通信用スケルトン関数の実行を要求する. スケルトン関数への引数は buf で指定する. スケルトン関数の標準入出力は SceSocket のストリームと接続されるため, 利用者は SceSocket からストリームの取得を行う必要がある.

[返却値]

返却値として, SceSocket オブジェクトを返却する. SceSocket の取得に失敗した場合は, null を返却する.

[使用例]

```
SceSocket       socket;
InputStream     is;
OutputStream    os;
int             funcID = 100;
SceSocketBuffer buf = new SceSocketBuffer(this);
int             rtn;
 :
Server_Stub  stub = new Server_Stub();
  // サーバスタブの生成
 :
try {
  buf.putObj("data");
  // 引数のエンコード
}
catch (IOException e) {
  // エラー処理
}
if ((socket = stub.getSceSocket(funcID, buf)) == null) {
  // SceSocket の取得に失敗した
 :
}
```

```
if ((is = socket.getInputStream()) == null) {
  // ストリームの取得に失敗した
   :
}
if ((os = socket.getOutputStream()) == null) {
  // ストリームの取得に失敗した
   :
}
  // Stream を使用した通信を行う.
 :
```

abort

[概要]

SceSocket 通信を使用して実行中の処理を中断する.

[メソッド仕様]

```
int abort(SceSocket socket);
```

[説明]

abort メソッドは, SceSoket 通信を使用して実行中の中断可能な処理を中断する.

[返却値]

0 : 正常終了
-5 : 通信エラー

[使用例]

```
int   rtn;
 :
Server_Stub stub = new Server_Stub();
  // サーバスタブの生成
socket = stub.getSceSocket(funcID, buf)
  // SceSocket の取得
 :
rtn = stub.abort(socket);
switch(rtn) {
case 0:
  // 正常終了
case -5:
  // 通信エラー
}
```

sceSendRecv

[概要]

アダプタと同期型通信を行う.

[メソッド仕様]

```
void sceSendRecv(int funcID, SceOutBuffer outBuf, SceInBufffer inBuf)
                 throws SceUserException, IOException;
```

[説明]

アダプタに対して, outBuf で指定した送信用バッファを送信し, その結果を inBuf で指定した受信用バッファに受信する. 本メソッドは, 結果を受信するまで, クライアントの処理をブロックする.

アダプタ側では, funcID で指定した関数番号を持つスケルトン関数が実行される.

また, データのエンコード時に, 関数番号を設定した場合は, 送信用バッファが一定のサイズに達した時点で, 先行してバッファを送信する.

[返却値]

バッファの送受信に失敗した場合は, IOException を発行する. また, 受信したバッファのエラーコードが 0 以外の場合は, SceUserException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);

int funcID = 100;        // 関数番号

// エンコード処理

try {
        sceSendRecv(funcID, outBuf, inBuf);
}
catch(SceUserException e) {
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}

// デコード処理
```

sceAsyncSend

[概要]

アダプタと非同期型通信を行う.

[メソッド仕様]

```
void sceAsyncSend(int funcID,
                  SceOutBuffer outBuf) throws IOException;
```

[説明]

アダプタに対して, outBuf で指定した送信用バッファを送信する. アダプタ側では, funcID で指定した関数番号を持つスケルトン関数が実行される.

sceAsyncSend メソッドは, バッファの送信のみを行い, 結果の受信は行わないため, スケルトン関数では, 結果を返すことができない.

また, データのエンコード時に, 関数番号を設定した場合は, 送信用バッファが一定のサイズに達した時点で, 先行してバッファを送信する.

[返却値]

本メソッドの返却値はない. ただし, バッファの送信に失敗した場合は, IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);

int funcID = 101;       // 関数番号

// エンコード処理

try {
        sceAsyncSend(funcID, outBuf);
}
catch(IOException e) {
        // エラー処理
}
```

sceLazySend

[概要]

アダプタと遅延評価型通信を行う.

[メソッド仕様]

```
long sceLazySend(int funcID,
                 SceOutBuffer outBuf) throws IOException;
```

[説明]

アダプタに対して, outBuf で指定した送信用バッファを送信する. アダプタ側では, funcID で指定した関数番号を持つスケルトン関数が実行される.

sceLazySend メソッドでは送信のみを行うため, 任意のタイミングで sceLazyRecv メソッドを呼び出して結果を受信しなければならない.

また, データのエンコード時に, 関数番号を設定した場合は, 送信用バッファが一定のサイズに達した時点で, 先行してバッファを送信する.

[返却値]

スケルトン関数が送信した結果を受け取るためのチケット ID を返す.

バッファの送信に失敗した場合は IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);

int funcID = 102;        // 関数番号
long ticketID;

// エンコード処理

try {
        ticketID = sceLazySend(funcID, outBuf);
}
catch(IOException e) {
        // エラー処理
}

:

try {
```

```
        sceLazyRecv(ticketID, inBuf);
}
catch(SceUserException e) {
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}

// デコード処理
```

sceLazyRecv

[概要]

遅延通信型通信の結果を受信する.

[メソッド仕様]

```
void sceLazyRecv(long ticketID, SceInBuffer inBuf)
                 throws SceUserException, IOException;
```

[説明]

sceLazySend メソッドの送信に対する結果を, inBuf で指定した受信用バッファに受信する. 引数 ticketID には, sceLazySend メソッドの返却値であるチケット ID を指定する. 本メソッドは, 結果を受信するまでクライアントの処理をブロックする.

[返却値]

本メソッドの返却値はない. ただし, バッファの受信に失敗した場合は, IOException を発行する. また, 受信したバッファのエラーコードが 0 以外の場合は, SceUserException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);

int funcID = 102;        // 関数番号
long ticketID;

// エンコード処理

try {
        ticketID = sceLazySend(funcID, outBuf);
}
catch(IOException e) {
        // エラー処理
}

:

try {
        sceLazyRecv(ticketID, inBuf);
}
catch(SceUserException e) {
```

```
        // エラー処理
}
catch(IOException e)
        // エラー処理
}

// デコード処理
```

sceLazyProbe

[概要]

遅延評価型通信の結果が着信しているかどうかの問い合わせを行う.

[メソッド仕様]

```
boolean sceLazyProbe(long ticketID) throws IOException;
```

[説明]

sceLazySend メソッドの送信に対する結果が着信しているかどうかを問い合わせる. 引数 ticketID には, sceLazySend メソッドの返却値であるチケット ID を指定する.

[返却値]

結果が着信している場合は true それ以外は false を返す. バッファの受信に失敗した場合は, IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);

int funcID = 102;        // 関数番号
long ticketID;

// エンコード処理

try {
        ticketID = sceLazySend(funcID, outBuf);
}
catch(IOException e) {
        // エラー処理
}

:

try {
       if(sceLazyProbe(ticketID) {
               sceLazyRecv(ticketID, inBuf);
       }
}
catch(SceUserException e) {
```

```
        // エラー処理
}
catch(IOException e)
        // エラー処理
}
```

sceSendCancel

[概要]

バッファのの送信を中断する.

[メソッド仕様]

```
void sceSendCancel(SceOutBuffer outBuf) throws IOException;
```

[説明]

outBuf に対するデータのエンコード時に，先行してバッファを送信している場合、バッファ送信を中断する．送信バッファが無い場合は、バッファをクリアする.

本メソッドを使用する場合は、アダプタ側でデータの送信中断時の処理を行う必要がある.

[返却値]

中断に失敗した場合は, IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);

int funcID = 100;        // 関数番号

putFuncID(funcID);
// エンコード処理

try {
        sceSendCancel(outBuf);
}
catch(IOException e) {
        // エラー処理
}
```

4.2.5 SceOutBuffer クラス

SceOutBuffer クラスは, アダプタとの通信を行う際に, アダプタ側のスケルトン関数に渡すバッファを提供するクラスである.

putFuncID

[概要]

バッファに関数番号を格納する.

[メソッド仕様]

```
void putFuncID(int funcID);
```

[説明]

funcID で指定された関数番号をバッファに格納する. データのエンコード前に本メソッドを使用して関数番号を設定することで, 先行してバッファの送信を行う事ができる.

[返却値]

本メソッドに返却値はない.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
int funcID = 100;
:
outBuf.putFuncID(funcID);
// エンコード処理
:
```

putObj

[概要]

バッファにデータを格納する.

[メソッド仕様]

```
void putObj(float value) throws IOException;
void putObj(double value) throws IOException;
void putObj(short value) throws IOException;
void putObj(int value) throws IOException;
void putObj(long value) throws IOException;
void putObj(char value) throws IOException;
void putObj(byte value) throws IOException;
void putObj(Float value) throws IOException;
void putObj(Double value) throws IOException;
void putObj(Short value) throws IOException;
void putObj(Integer value) throws IOException;
void putObj(Long value) throws IOException;
void putObj(Character value) throws IOException;
void putObj(Byte value) throws IOException;
void putObj(String value) throws IOException;
void putObj(float value[], int index, int count) throws IOException;
void putObj(double value[], int index, int count) throws IOException;
void putObj(short value[], int index, int count) throws IOException;
void putObj(int value[], int index, int count) throws IOException;
void putObj(long value[], int index, int count) throws IOException;
void putObj(char value[], int index, int count) throws IOException;
void putObj(byte value[], int index, int count) throws IOException;
void putObj(Float value[], int index, int count) throws IOException;
void putObj(Double value[], int index, int count) throws IOException;
void putObj(Short value[], int index, int count) throws IOException;
void putObj(Integer value[], int index, int count)
            throws IOException;
void putObj(Long value[], int index, int count) throws IOException;
void putObj(Character value[], int index, int count)
            throws IOException;
void putObj(Byte value[], int index, int count) throws IOException;
```

[説明]

送信用バッファに, value で指定されたデータを格納する. value が配列の場合は, index で指定された配列の要素番号から count で指定された数だけバッファに格納する.

value が String 型の場合, バッファに格納するデータ前4バイトに, String の長さを格納するため, アダプタ側のスケルトン関数でデータを取り出す際には SCEGetString を使用する必要がある.

[返却値]

本メソッドに返却値はない. ただし, バッファの送信に失敗した場合は, IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);
int i;
char ch[] = new char[10];
:
try {
outBuf.putObj(i);
outBuf.putObj(ch, 0, 10);
}
catch(IOException e){
        // エラー処理
}

try {
        sceSendRecv(funcID, outBuf, inBuf);
}
catch(SceUserException e) {
        // エラー処理
}
catch(IOException e){
        // エラー処理
}
```

clear

[概要]
バッファをクリアする.

[メソッド仕様]

```
quote clear();
```

[説明]
バッファをクリアする. バッファに設定されていた情報は失われる.

[返却値]
本メソッドに返却値はない.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
:
outBuf.clear();
:
```

4.2.6 SceInBuffer クラス

SceInBuffer クラスは, アダプタ側のスケルトン関数からの応答を受信するためのバッファを提供するクラスである.

get_*TYPE*

[概要]

バッファからデータを取り出す.

[メソッド仕様]

```
float get_float() throws
                  OutOfBufferException, IOException;
double get_double() throws
                  OutOfBufferException, IOException;
short get_short() throws
                  OutOfBufferException, IOException;
int get_int() throws
                  OutOfBufferException, IOException;
long get_long() throws
                  OutOfBufferException, IOException;
char get_char() throws
                  OutOfBufferException, IOException;
byte get_byte() throws
                  OutOfBufferException, IOException;
Float get_Float() throws
                  OutOfBufferException, IOException;
Double get_Double() throws
                  OutOfBufferException, IOException;
Short get_Short() throws
                  OutOfBufferException, IOException;
Integer get_Integer() throws
                  OutOfBufferException, IOException;
Long get_Long() throws
                  OutOfBufferException, IOException;
Characterget_Character() throws
                  OutOfBufferException, IOException;
Byte get_Byte() throws
                  OutOfBufferException, IOException;
String get_String() throws
                  OutOfBufferException, IOException;
```

[説明]

受信用バッファから, TYPE 型の値としてデータを取り出す.

get_String メソッドは, 実際に取り出すデータの前 4 バイトに String の長さが格納されていることを前提とするため, アダプタ側のスケルトン関数でデータを格納する際には, SCEPutString 関数を使用する必要がある.

[返却値]

バッファから取り出した値を返す.

バッファの終端に達してデータを取り出せない場合は, OutOfBufferException を発行する. また, データの受信に失敗した場合は, IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);
int i;
float f;

// エンコード処理

try {
        sceSendRecv(funcID, outBuf, inBuf);
}
catch(SceUserException e) {
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}

try {
        i = inBuf.get_int();
        f = inBuf.get_float();
}
catch(OutOfBufferException e) {
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}
```

getObj

[概要]

バッファからデータを配列として取り出す.

[メソッド仕様]

```
void getObj(float value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(double value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(short value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(int value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(long value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(char value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(byte value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(Float value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(Double value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(Short value[],
           int index,
            int count) throws OutOfBufferException, IOException;
void getObj(Integer value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(Long value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(Character value[],
            int index,
            int count) throws OutOfBufferException, IOException;
void getObj(Byte value[],
            int index,
            int count) throws OutOfBufferException, IOException;
```

[説明]

受信用バッファから, value で指定された配列の index で指定した要素番号から count 分, データを取り出す.

[返却値]

本メソッドに返却値はない. ただし, バッファの終端に達してデータを取り出せない場合は, OutOfBufferException を発行する. また, データの受信に失敗した場合は, IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);
int i[] = new int[10];
float f[] = new float[30];

// エンコード処理

try {
        sceSendRecv(funcID, outBuf, inBuf);
}
catch(SceUserException e) {
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}

try {
        inBuf.getObj(i, 0, 10);
        inBuf.getObj(f, 5, 25);
}
catch(OutOfBufferException e) {
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}
```

skip

[概要]

バッファのデータを読みとばす.

[メソッド仕様]

```
void skip(int count);
```

[説明]

引数 count で指定された数分, バッファのデータを読みとばす.

[返却値]

本メソッドに返却値はない. ただし, バッファの終端に達してデータを取り出せない場合は, OutOfBufferException を発行する. また, データの受信に失敗した場合は, IOException を発行する.

[使用例]

```
SceOutBuffer outBuf = new SceOutBuffer(this);
SceInBuffer inBuf = new SceInBuffer(this);
int i;
float f;

// エンコード処理

try {
        sceSendRecv(funcID, outBuf, inBuf);
}
catch(SceUserException e) {
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}

try {
        i = inBuf.get_int();
        inBuf.skip(i);
        f = inBuf.get_float();
}
catch(OutOfBufferException e) {
```

```
        // エラー処理
}
catch(IOException e) {
        // エラー処理
}
```

4.2.7 SceToolManager クラス

getServers

[概要]

利用可能なサーバ一覧を取得する.

[メソッド仕様]

```
String[] getServers();
```

[説明]

利用可能なサーバ一覧を取得して返却値として返す.

引数なし.

[返却値]

返却値として, サーバ一覧を返す. データが登録されていない場合, null を返す.

[使用例]

```
String   servers[];
 :
servers = toolManager.getServers();
if (servers != null) {
  // 取得成功
} else {
  // 取得失敗
}
```

getAvailableServers

[概要]

使用可能なサーバ一覧を取得する.

[メソッド仕様]

```
String[] getServers();
```

[説明]

サーバのツール管理サブシステムが使用可能となったサーバの一覧を返す。

MasterConnection に登録されているサーバ情報オブジェクトからチケット番号を取得する. ツール管理サブシステムが使用可能となり正常なチケット番号がプロキシから返却されているサーバ名を配列として返却している.

引数なし.

[返却値]

返却値として, 使用可能なサーバ一覧を返す. 使用可能なサーバがない場合, null を返す.

[使用例]

```
String   servers[];
 :
servers = toolManager.getAvailableServers();
if (servers != null) {
  // 取得成功
} else {
  // 取得失敗
}
```

getTools

[概要]

利用可能なツール一覧を取得.

[メソッド仕様]

```
String[] getTools(String server, String category);
```

[説明]

指定したサーバのカテゴリの中で利用可能なツール一覧を取得し, 返却値として返す. serverには, 一覧を取得するサーバ名を指定する. categoryには一覧を取得するカテゴリ名を指定する.

[返却値]

返却値として, 利用可能なツール一覧を返す. データが登録されていない場合, nullを返す.

[使用例]

```
String   tools[];
 :
tools = toolManager.getTools("host01","Editor");
if (tools != null) {
  // 取得成功
} else {
  // 取得失敗
}
```

requestModule

[概要]

STA アプリケーション GUI プログラムの起動要求または連携要求を行う.

[メソッド仕様]

```
SceHandle requestModule(String hostname, String category,
                        String toolname, SceWindow parent,
                        boolean new_request);
SceHandle requestModule(String hostname, String category,
                        String toolname, SceWindow parent,
                        boolean new_request, Vector params);
SceHandle requestModule(String hostname, String category,
                        String toolname, SceWindow parent,
                        boolean new_request, Vector params,
                        SceToolExecuteType newType);
SceHandle requestModule(String hostname, String category,
                        String toolname, SceWindow parent,
                        boolean new_request, Vector params,
                        SceToolExecuteType newType, int ID);
```

[説明]

STA アプリケーション GUI プログラムから, 他の STA アプリケーション GUI プログラムの起動もしくは連携を, ツール制御モジュールに依頼する.

指定されたツール名、カテゴリ名を元にクラスをロードし新規インスタンスを生成する.

new_request が true の場合, 指定されたツール実行モジュールのコンストラクタを呼んで新しいインスタンスを生成し, その後生成したインスタンスの onRequestModule メソッドを呼ぶ.

new_request が false の場合, 指定されたツールが既に起動していれば新しいインスタンスは生成せず, 起動しているインスタンスの onRequestModule メソッドを呼ぶ.

本メソッドでは, STA アプリケーション GUI プログラムの画面の表示は行わないため,
java.awt.Frame.setVisible() メソッドを使用する必要がある.

hostnem には, 依頼先のホスト名を指定する. category には, 依頼先のカテゴリ名を指定する. toolname には, 依頼先のツール名を指定する. parent には, 依頼元の SceWindow を指定する. new_request には, 新規起動要求フラグを指定する. params には, ユーザ定義のパラメタを指定する. newType には, ツール起動情報オブジェクトを指定する. ID には, ツール連携 ID を指定する（指定されない場合-1 が入る）.

[返却値]

返却値として, SCEHandle を返す. ツール起動・連携に失敗した場合, null を返す.

[使用例]

```
SceHandle handleEditor;
SceWindow windowEditor;
Vector vectorParams = new Vector();
 :
// ベクトル vectorParams の中身を埋める処理
 :
handleEditor = toolMnanager.requestModule("host01","category01",
                                "Editor",this,true,vectorParams);
if (handleEditor != null) {
  // 起動に成功
  windowEditor = handleEditor.getWindow(); // 連携先の SceWindow が
入る.
   :
} else {
  // 起動に失敗
}
 :
// ツール起動時の処理
 :
windowEditor.getWindow().setVisible(true); // ツール画面表示
 :
```

requestCategory

[概要]

STA アプリケーション GUI プログラムの起動要求または連携要求を行う.

[メソッド仕様]

```
SceHandle requestCategory(String hostname, String category,
                          SceWindow parent, boolean new_request);
SceHandle requestCategory(String hostname, String category,
                          SceWindow parent, boolean new_request,
                          Vector params);
SceHandle requestCategory(String hostname, String category,
                          SceWindow parent, boolean new_request,
                          Vector params, SceToolExecuteType newType);
SceHandle requestCategory(String hostname, String category,
                          SceWindow parent, boolean new_request,
                          Vector params, SceToolExecuteType newType,
                          int ID);
```

[説明]

カテゴリ名だけを指定して, STA アプリケーション GUI プログラムから, 他のツール実行モジュールの起動もしくは連携を, ツール制御モジュールに依頼する.

指定されたカテゴリ内にユーザの選択したツールがある場合そのツールは優先的に起動される. カテゴリ中に複数のツールが存在しユーザが 1 つも選択していなかった場合、SCE が起動するツールを決定する.

new_request が true の場合, 指定されたツール実行モジュールのコンストラクタを呼んで新しいインスタンスを生成し, その後生成したインスタンスの onRequestModule メソッドを呼ぶ.

new_request が false の場合, 指定されたツールが既に起動していれば新しいインスタンスは生成せず, 起動しているインスタンスの onRequestModule メソッドを呼ぶ.

本メソッドでは STA アプリケーション GUI プログラムの画面の表示は行わないため,
java.awt.Frame.setVisible() メソッドを使用する必要がある.

hostnem には, 依頼先のホスト名を指定する. category には, 依頼先のカテゴリ名を指定する. parent には, 依頼元の SceWindow を指定する. new_request には, 新規起動要求フラグを指定する. params には, ユーザ定義のパラメタを指定する. newType には, ツール起動情報オブジェクトを指定する. ID には, ツール連携 ID を指定する (指定されない場合－ 1 が入る) .

[返却値]

返却値として, SCEHandle を返す. ツール起動・連携に失敗した場合, null を返す.

[使用例]

```
SceHandle handleEditor;
SceWindow windowEditor;
Vector vectorParams = new Vector();
 :
// ベクトル vectorParams の中身を埋める処理
 :
handleEditor = toolMnanager.requestCategory("host01",
                                            "category01",
                                            this, true,
                                            vectorParams);
if (handleEditor != null) {
  // 起動に成功
  windowEditor = handleEditor.getWindow(); // 連携先の SceWindow
                                           // が入る
   :
} else {
  // 起動に失敗
}
 :
// ツール起動時の処理
 :
windowEditor.getWindow().setVisible(true); // ツール画面表示
 :
```

requestModuleVisible

[概要]

STA アプリケーション GUI プログラムの起動要求または連携要求を行い, ツール実行モジュール画面を表示する.

[メソッド仕様]

```
SceHandle requestModuleVisible(String hostname, String category,
                               String toolname, SceWindow parent,
                               boolean new_request);
SceHandle requestModuleVisible(String hostname, String category,
                               String toolname, SceWindow parent,
                               boolean new_request, Vector params);
SceHandle requestModuleVisible(String hostname, String category,
                               String toolname, SceWindow parent,
                               boolean new_request, Vector params,
                               SceToolExecuteType newType);
SceHandle requestModuleVisible(String hostname, String category,
                               String toolname, SceWindow parent,
                               boolean new_request, Vector params,
                               SceToolExecuteType newType, int ID);
```

[説明]

STA アプリケーション GUI プログラムから, 他の STA アプリケーション GUI プログラムの起動もしくは連携を, ツール制御モジュールに依頼し, STA アプリケーション GUI プログラムの画面を表示する.

本メソッドでは, requestModule の処理を行った後ツール実行モジュールの画面を表示する, またはアクティブにする処理を行う.

hostnem には, 依頼先のホスト名を指定する. category には, 依頼先のカテゴリ名を指定する. toolname には, 依頼先のツール名を指定する. parent には, 依頼元の SceWindow を指定する. new_request には, 新規起動要求フラグを指定する. params には, ユーザ定義のパラメタを指定する. newType には, ツール起動情報オブジェクトを指定する. ID には, ツール連携 ID を指定する (指定されない場合－1 が入る).

[返却値]

返却値として, SCEHandle を返す. ツール起動・連携に失敗した場合, null を返す.

[使用例]

```
SceHandle handleEditor;
SceWindow windowEditor;
Vector vectorParams = new Vector();
 :
// ベクトル vectorParams の中身を埋める処理
```

```
 :
handleEditor = toolMnanager.requestModuleVisible("host01",
                                                  "category01",
                                                  "Editor",
                                                  this, true,
                                                 vectorParams);
if (handleEditor != null) {
  // 起動に成功
  windowEditor = handleEditor.getWindow(); // 連携先のSceWindowが
入る.
   :
} else {
  // 起動に失敗
}
```

requestCategoryVisible

[概要]

STA アプリケーション GUI プログラムの起動要求または連携要求を行い, ツール実行モジュール画面を表示する.

[メソッド仕様]

```
SceHandle requestCategoryVisible(String hostname, String category,
                                 SceWindow parent,
                                 boolean new_request);
SceHandle requestCategoryVisible(String hostname, String category,
                                 SceWindow parent,
                                 boolean new_request,
                                 Vector params);
SceHandle requestCategoryVisible(String hostname, String category,
                                 SceWindow parent,
                                 boolean new_request,
                                 Vector params,
                                 SceToolExecuteType newType);
SceHandle requestCategoryVisible(String hostname, String category,
                                 SceWindow parent,
                                 boolean new_request,
                                 Vector params,
                                 SceToolExecuteType newType, int ID);
```

[説明]

カテゴリ名だけを指定して, STA アプリケーション GUI プログラムから, 他のツール実行モジュールの起動もしくは連携を, ツール制御モジュールに依頼し, ツール実行モジュールの画面を表示する.

指定されたカテゴリ内にユーザの選択したツールがある場合そのツールは優先的に起動される. カテゴリ中に複数のツールが存在しユーザが 1 つも選択していなかった場合、SCE が起動するツールを決定する.

本メソッドでは, requestModule の処理を行った後ツール実行モジュールの画面を表示する, またはアクティブにする処理を行う.

hostnem には, 依頼先のホスト名を指定する. category には, 依頼先のカテゴリ名を指定する. parent には, 依頼元の SceWindow を指定する. new_request には, 新規起動要求フラグを指定する. params には, ユーザ定義のパラメタを指定する. newType には, ツール起動情報オブジェクトを指定する. ID には, ツール連携 ID を指定する (指定されない場合 − 1 が入る) .

[返却値]

返却値として, SCEHandle を返す. ツール起動・連携に失敗した場合, null を返す.

[使用例]

```
SceHandle handleEditor;
SceWindow windowEditor;
Vector vectorParams = new Vector();
 :
// ベクトル vectorParams の中身を埋める処理
 :
handleEditor = toolMnanager.requestCategoryVisible("host01",
                                                    "category01",
                                                    this, true,
                                                    vectorParams);
if (handleEditor != null) {
  // 起動に成功
  windowEditor = handleEditor.getWindow(); // 連携先の SceWindow が
入る.
   :
} else {
  // 起動に失敗
}
```

getRunningTool

[概要]

実行中の全ツールの一覧情報を取得する.

[メソッド仕様]

```
ToolList getRunningTool();
```

[説明]

利用者が動作させているツールの一覧情報を取得する. 返却型 ToolList については 4.2.13 ToolList クラスを参照のこと.

[返却値]

返却値として, ToolList を返す. ツールの一覧情報の取得に失敗した場合, null を返す.

[使用例]

```
ToolList  tlist;
int       execToolNum;
 :
tlist = toolManager.getRunnningTool();
if (tlist != null) {
  // ツール一覧情報の取得に成功
  execToolNum = tlist.size(); // 起動中のツール数を取得
   :
} else {
  // ツール一覧情報の取得に失敗
}
```

getCategories

[概要]

利用可能なカテゴリ一覧を取得する.

[メソッド仕様]

```
String [] getCategories(String hostname);
```

[説明]

hostnameで指定したサーバで利用可能なカテゴリの一覧を取得する. カテゴリの中で利用可能なツールを問い合わせる場合は, 本メソッドでカテゴリの一覧を取得した後にgetToolsメソッドを使用して, 利用可能なツールの一覧を取得する.

[返却値]

返却値として, カテゴリ名を格納したString配列を返す. 利用可能なカテゴリが存在しない場合, あるいはカテゴリ一覧の取得に失敗した場合, nullを返す.

[使用例]

```
String    categories[];
int       categoryNum;
 :
categories = toolManager.getCategories("chobi.koma.jaeri.go.jp");
if (categories != null) {
  // カテゴリ一覧情報の取得に成功
  categoryNum = categories.length; // 利用可能なカテゴリ数を取得
   :
} else {
  // カテゴリ一覧情報の取得に失敗
}
```

setToolConfigData

[概要]

ツール設定情報を保存する.

[メソッド仕様]

```
public boolean setToolConfigData(String tool, String key,
                                 String data);
```

[説明]

ツールごとの利用者のカスタマイズ情報を設定ファイルに保存する. tool には, ツール名称を指定する. key には設定する情報のキーを指定する. data には, 設定するデータを設定する.

カスタマイズ情報設定ファイルは, STA サーバサブシステムの WWW サーバドキュメントディレクトリ内にある GUI サブシステム格納ディレクトリ (以下 *GUIDIR* と記述) 内に, 以下のファイルとして保存される.

GUIDIR/userinf/ユーザ *ID*/toolinf/ツール名.txt

[返却値]

true : 設定情報の保存に成功
false : 設定情報の保存に失敗

[使用例]

```
 :
String data;
data = toolManager.getToolConfigData("editor", "width");
if(data == null) {
  toolManager.setToolConfigData("editor", "width", "80");
}
 :
```

getToolConfigData

[概要]

ツール設定情報を取得する.

[メソッド仕様]

```
public String getToolConfigData(String tool, String key);
```

[説明]

ツールごとの利用者のカスタマイズ情報を取得する. toolには, ツール名称を指定する. keyには, 設定ファイル中のキーを指定する.

カスタマイズ情報の設定はsetToolConfigDataメソッドにより行う.

[返却値]

keyに対応する設定情報を返却する. keyに対応する設定情報が存在しない場合, nullを返却する.

[使用例]

```
 :
String data;
data = toolManager.getToolConfigData("editor", "width");
if(data == null) {
  toolManager.setToolConfigData("editor", "width", "80");
}
 :
```

4.2.8 SceToolExecuteManage クラス

getToolObject

[概要]

呼びもとのツールにツール起動情報オブジェクトを渡す

[メソッド仕様]

```
SceToolExecuteType getToolObject(SceWindow wnd);
```

[説明]

呼びもとのツールが管理しているツール起動情報オブジェクトを渡す．返却された起動情報オブジェクトの利用方法は次項を参照する．SceWindowには当オブジェクトの為の toolTypeCounter が変数として設定されている．wnd にはツール自身の SceWindow を指定する．

[返却値]

SceToolExecuteType 起動情報オブジェクトが返却される．取得に失敗した場合（起動情報オブジェクトの登録に失敗している場合）null を返却する．

[使用例]

```
SceToolExecuteManager toolExMan;
SceToolExecuteType    toolExeType;
 :
toolExMan = toolManager.getToolExecuter();
toolExeType = toolExMan.getToolObject(this);
if (toolExeType != null) {
  // 取得成功
} else {
  // 取得失敗
}
```

setToolObject

[概要]

ツール連携時に既存のツール起動情報オブジェクトを上書きする

[メソッド仕様]

```
SceToolExecuteType setToolObject(SceWindow wnd, boolean shift,
                                 boolean cntl);
SceToolExecuteType setToolObject(SceWindow wnd,
                                 SceToolExecuteType obj);
```

[説明]

呼びもとのツールが管理しているツール起動情報オブジェクトを渡す．返却された起動情報オブジェクトの利用方法は次項を参照する．SceWindowには当オブジェクトの為の toolTypeCounter が変数として設定されている．wnd にはツール自身の SceWindow を指定する．shift には設定したい場合 true を指定する．cntl には設定したい場合 true を指定する．obj には設定したい SceToolExecuteType を作成し指定する．

[返却値]

なし

[使用例]

```
SceToolExecuteType   toolExeType;
toolExeType = new SceToolExecuteType(true, false);
 :
toolExMan.setToolObject(wnd, toolExeType);
```

4.2.9 SceToolExecuteType クラス

getShiftKey

[概要]
ツール起動時シフトキー情報取得.

[メソッド仕様]

```
boolean getShiftKey();
```

[説明]
ツール起動情報としてシフトキーが押されているかを問い合わせる

引数なし.

[返却値]
シフトキーが押されていた場合 true を、押されていなかった場合 false を返却する.

[使用例]

```
boolean   shift;
 :
shift = SceToolExecuteType.getShiftKey();
if (shift != true) {
  // シフトキーが押されていた
} else {
  // シフトキーは押されていなかった
}
```

getCntlKey

[概要]

ツール起動時コントロールキー情報取得.

[メソッド仕様]

```
boolean getCntlKey();
```

[説明]

ツール起動情報としてコントロールキーが押されているかを問い合わせる

引数なし.

[返却値]

コントロールキーが押されていた場合 true を、押されていなかった場合 false を返却する.

[使用例]

```
boolean   cntl;
 :
cntl = SceToolExecuteType.getCntlKey();
if (cntl != true) {
  // コントロールキーが押されていた
} else {
  // コントロールキーは押されていなかった
}
```

setShiftKey

[概要]
ツール起動時シフトキー情報設定.

[メソッド仕様]

```
void setShiftKey(boolean shift);
```

[説明]
ツール起動情報としてのシフトキー情報を,shift に設定する.

[返却値]
なし

[使用例]

```
boolean   shift = true;
 :
SceToolExecuteType.setShiftKey(shift);
```

setCntlKey

[概要]
ツール起動時コントロールキー情報設定.

[メソッド仕様]

```
void setCntlKey(boolean cntl);
```

[説明]
ツール起動情報としてのコントロールキー情報を,cntl に設定する.

[返却値]
なし

[使用例]

```
boolean   cntl = true;
 :
SceToolExecuteType.setCntlKey(cntl);
```

4.2.10 SceFileManager クラス

getFileName

[概要]

選択ファイル名取得.

[メソッド仕様]

```
String[] getFileName();
```

[説明]

ファイル管理モジュールで選択されたファイル名を, ファイル管理モジュールから取得する.

引数なし.

[返却値]

返却値として, 取得したファイル名を返す. ファイル名が取得できなかった場合, null を返す.

[使用例]

```
String   file[];
 :
file = fileManager.getFileName();
if (file != null) {
  // 取得に成功
} else {
  // 取得に失敗
}
```

getServerName

[概要]

選択サーバ名取得.

[メソッド仕様]

```
String getServerName();
```

[説明]

ファイル管理モジュールで選択されたファイルの存在するサーバ名を, ファイル管理モジュールから取得する.

引数なし.

[返却値]

返却値として, 取得したサーバ名を返す. サーバ名が取得できなかった場合, null を返す.

[使用例]

```
String   serv;
 :
serv = fileManager.getServerName();
if (serv != null) {
  // 取得に成功
} else {
  // 取得に失敗
}
```

changeDirectory

[概要]

カレントディレクトリ変更.

[メソッド仕様]

```
int changeDirectory(String host, String dir);
```

[説明]

ファイルマネージャのカレントディレクトリを変更する. hostにはカレントディレクトリ変更後のホスト名を, dirには変更後のディレクトリを指定する.

[返却値]

0 : 正常終了
-1 : 引数の指定が正しくない
-2 : 指定されたサーバと接続していない
-3 : 指定されたサーバにおいてディレクトリの変更に失敗した

[使用例]

```
String   serv;
 :
serv = fileManager.changeDirectory("desr01","/tmp")
if (serv != null) {
  // 取得に成功
} else {
  // 取得に失敗
}
```

getStub

[概要]

ファイルマネージャスタブ取得.

[メソッド仕様]

```
SceFileManagerStub getStub(String hostName);
```

[説明]

hostName で指定されたスタブを取得する.

[返却値]

返却値として, 取得したスタブを返す. スタブが取得できなかった場合, null を返す.

[使用例]

```
SceFileManagerStub   stub;
 :
stub = fileManager.getStub("host01");
if (stub != null) {
  // 取得に成功
} else {
  // 取得に失敗
}
```

4.2.11 SceFileManagerStub クラス

getHome

[概要]

ホームディレクトリ取得

[メソッド仕様]

```
String getHome();
```

[説明]

当クラスの通信先であるアダプタのあるサーバにおけるホームディレクトリを取得する. アダプタでは getpwuid を発行し返却する.

[返却値]

返却値として, ホームディレクトリを返却する. 取得に失敗した場合 null を返却する.

[使用例]

```
SceFileManagerStub stub;
String strHome;
 :
stub = fileManager.getStub("host01");
strHome = stub.getHome();
if (strHome != null) {
  // 取得に成功
   System.out.println("host01's home == "+strHome);
   :
} else {
  // 取得に失敗
}
```

changeDirectory

[概要]

カレントディレクトリ移動

[メソッド仕様]

```
boolean changeDirectory(String directory);
```

[説明]

directoryで指定されたスタブを取得する.

[返却値]

返却値として, 成功時はtrue失敗時にはfalseを返却する.

[使用例]

```
SceFileManagerStub   stub;
boolean bsuccess;
 :
stub = fileManager.getStub("host01");
bsuccess = stub.changeDirectory("/usr/local/SCE");
if (bsuccess == true) {
  // 取得に成功
} else {
  // 取得に失敗
  if(stub.errMsg != null){
    System.out.println("changeDirectory failed " + stub.errMsg);
  }
}
```

subDirs

[概要]

ディレクトリ一覧取得.

[メソッド仕様]

```
String[] subDirs();
```

[説明]

現在のカレントディレクトリ上にあるディレクトリ一覧を取得する. アダプタでは readdir を実行しファイルステータスによってディレクトリのみを取得する.

[返却値]

返却値として, ディレクトリ一覧を返す. 取得に失敗した場合, null を返す.

[使用例]

```
SceFileManagerStub   stub;
String[] strDirs;
 :
stub = fileManager.getStub("host01");
strDirs = stub.subDirs();
if (strDirs != null) {
  // 取得に成功
} else {
  // 取得に失敗
  if(stub.errMsg != null){
    System.out.println("getDirectrys failed " + stub.errMsg);
  }
}
```

files

[概要]

ファイル一覧取得.

[メソッド仕様]

```
String[] files();
```

[説明]

現在のカレントディレクトリ上にあるファイル一覧を取得する. アダプタでは readdir を実行しファイルステータスによってファイルのみを取得する.

[返却値]

返却値として, ファイル一覧を返す. 取得に失敗した場合, null を返す.

[使用例]

```
SceFileManagerStub   stub;
String[] strFiles;
 :
stub = fileManager.getStub("host01");
strFiles = stub.subDirs();
if (strFiles != null) {
  // 取得に成功
} else {
  // 取得に失敗
  if(stub.errMsg != null){
    System.out.println("getFiles failed " + stub.errMsg);
  }
}
```

search

[概要]

正規表現で指定した名称に該当するファイル、ディレクトリ名検索.

[メソッド仕様]

```
String[] search(String regular, int mode);
```

[説明]

カレントディレクトリ上からregularで指定した正規表現に該当するファイル、ディレクトリ名を返却する. 利用できる正規表現は*、?、[]、である. modeには，以下の値を指定することにより，返却値を制御できる.

1 : ファイルのみ
2 : ディレクトリのみ
3 : ファイル、ディレクトリ両方

アダプタでは正規表現をコンパイルしregcomp、を実行しregexec ファイルステータスによって返却値を決定する.

本メソッドにおけるは検索対象ディレクトリはカレントディレクトリのみであるため，本メソッドを呼び出す前にchangeDirectoryメソッドを呼び出し，検索したいディレクトリへ移動する必要がある.

[返却値]

正規表現で指定した名称に該当するファイル、ディレクトリ名を返す. 指定した名称に該当するファイル、ディレクトリがなかった場合、要素数0個のString配列を返す. 取得に失敗した場合、nullを返す.

[使用例]

```
SceFileManagerStub   stub;
String[]   strFind;
 :
stub = fileManager.getStub("host01");
strFind = stub.serch("A*", 1);
if (strFind != null) {
  if(strFind.length() == 0)[
  //実行は成功したが該当するファイルは見つからなかった
  }
  else{
  // 取得に成功
  }
```

```
} else {
  // 取得に失敗
  if(stub.errMsg != null){
    System.out.println("search failed " + stub.errMsg);
  }
}
```

makeDir

[概要]

ディレクトリ作成.

[メソッド仕様]

```
boolean makeDir(String dir);
boolean makeDir(String[] dir);
```

[説明]

dirで指定されたディレクトリを作成する. ディレクトリ名称は"/"(slash)を含まないディレクトリ名称のみを指定する. そのため, 呼び出しの前にchangeDirectoryメソッドを呼び出し, 作成したいディレクトリの親ディレクトリへ移動する必要がある.

[返却値]

返却値として, 成功時はtrue失敗時にはfalseを返却する.

[使用例]

```
SceFileManagerStub   stub;
boolean bsuccess;
 :
stub = fileManager.getStub("host01");
bsuccess = stub.makeDir("/usr/local/SCE");
if (bsuccess == true) {
  // 作成に成功
} else {
  // 作成に失敗
  if(stub.errMsg != null){
    System.out.println("makeDirectory failed " + stub.errMsg);
  }
}
```

remove

[概要]

ファイルやディレクトリの削除.

[メソッド仕様]

```
boolean remove(String name);
boolean remove(String[] name);
```

[説明]

name で指定されたディレクトリ、ファイルを削除する. ディレクトリが指定された場合、指定ディレクトリ下のファイル及びディレクトリは可能な限り削除される.

アダプタではファイル削除スレッド関数が実行され、ディレクトリは open され書き込み権限のあるファイルをすべて削除しようとする. ファイルがすべて削除されると指定ディレクトリも削除される.

[返却値]

返却値として, 成功時は true 失敗時には false を返却する.

[使用例]

```
SceFileManagerStub   stub;
boolean bsuccess;
 :
stub = fileManager.getStub("host01");
bsuccess = stub.remove("/usr/local/SCE");
if (bsuccess == true) {
  // 作成に成功
} else {
  // 作成に失敗
  if(stub.errMsg != null){
    System.out.println("remove failed " + stub.errMsg);
  }
}
```

copy

[概要]

同一サーバ内コピー.

[メソッド仕様]

```
boolean copy(String path, String file);
boolean copy(String[] path, String file);
```

[説明]

pathで指定されたコピー元をfileで指定されたコピー先へコピーする. pathは絶対パスで指定する. コピー先は"/"(slash)を含まないディレクトリ名のみを指定する. そのため、本メソッドを呼び出す前にchangeDirectoryでコピー先ディレクトリの1つ上のディレクトリへ移動する必要がある.

アダプタではcpコマンドを実行している.

[返却値]

返却値として, 成功時はtrue失敗時にはfalseを返却する.

[使用例]

```
SceFileManagerStub   stub;
boolean bsuccess;
 :
stub = fileManager.getStub("host01");
bsuccess = stub.copy("/usr/local/etc/htdocs/SCE","public_html");
if (bsuccess == true) {
  // 作成に成功
} else {
  // 作成に失敗
  if(stub.errMsg != null){
    System.out.println("copy failed " + stub.errMsg);
  }
}
```

move

[概要]

同一サーバ内のファイルの移動

[メソッド仕様]

```
boolean move(String path, String file);
boolean move(String[] path, String file);
```

[説明]

pathで指定されたファイルをfileで指定されたファイルに移動する. pathは絶対パスで指定する. 移動先は、"/"(slash)を含まないディレクトリ名のみを指定する. そのため、呼び出しの前にchangeDirectoryで移動先ディレクトリの1つ上のディレクトリへ移動する必要がある.

アダプタではmvコマンドを実行している. 本メソッド実行後, 移動元ファイルは無くなる.

[返却値]

返却値として, 成功時はtrue失敗時にはfalseを返却する.

[使用例]

```
SceFileManagerStub   stub;
boolean bsuccess;
 :
stub = fileManager.getStub("host01");
bsuccess = stub.move("/usr/local/etc/htdocs/SCE","public_html");
if (bsuccess == true) {
  // 移動に成功
} else {
  // 移動に失敗
  if(stub.errMsg != null){
    System.out.println("move failed " + stub.errMsg);
  }
}
```

remoteCopy

[概要]

サーバ間コピー.

[メソッド仕様]

```
int remoteCopy(String fromServer,String[] fromFiles,
               String toServer,String toDirectory);
```

[説明]

fromServerで指定されたホストのfromFilesを、toServerで指定されたホストのtoDirectoryへコピーする. アダプタではコピー元のホストに対してアダプタ間連携を実行し、コピー元ファイルをオープンしバッファに読み込み、そのバッファを取得する. コピー先ディレクトリでファイル名のファイルを作成し、オープンし、バッファの中身を書き込む.

引数にコピー先のホストを指定するが、当メソッドはコピー先のホストのスタブを使用する. 当メソッドはディレクトリの移動をサポートしていない為、呼び出しの前にchangeDirectoryでコピーしたいディレクトリの1つ上のディレクトリへ移動する必要がある.

[返却値]

返却値として, 成功時は0、失敗時には負値を返却する.

[使用例]

```
SceFileManagerStub   stub;
int iret;
 :
stub = fileManager.getStub("host01");
stub.changeDirectory("/home/user01");
bsuccess = stub.remoteCopy("host02', "/usr/local/SCE",
                           "host01", "main");
if (iret == 0) {
  // コピーに成功
} else {
  // コピーに失敗
  if(stub.errMsg != null){
    System.out.println("copy failed " + stub.errMsg);
  }
}
```

remoteMove

[概要]

サーバ間のファイル移動

[メソッド仕様]

```
int remoteMove(String fromServer,String[] fromFiles,
               String toServer,String toDirectory);
```

[説明]

fromServerで指定されたホストのfromFilesを、toServerで指定されたホストのtoDirectoryへ移動する．アダプタではコピー元のホストに対してアダプタ間連携を実行し、ムーブ元ファイルをオープンしバッファに読み込み、そのバッファを取得する．ムーブ先ディレクトリでファイル名のファイルを作成し、オープンし、バッファの中身を書き込む．

引数に移動先のホストを指定するが、当メソッドはムーブ先のホストのスタブを使用する．本メソッドはディレクトリの移動をサポートしていない為、呼び出しの前にchangeDirectoryで移動したいディレクトリの１つ上のディレクトリへ移動する必要がある．

本メソッド実行後，移動元ファイルは無くなる．

[返却値]

返却値として，成功時は０、失敗時には負値を返却する．

[使用例]

```
SceFileManagerStub   stub;
int iret;
 :
stub = fileManager.getStub("host01");
stub.changeDirectory("/home/user01");
bsuccess = stub.remoteMove("host02', "/usr/local/SCE",
                           "host01", "main");
if (iret == 0) {
  // ムーブに成功
} else {
  // ムーブに失敗
  if(stub.errMsg != null){
    System.out.println("move failed " + stub.errMsg);
  }
}
```

4.2.12 SceHandle クラス

getWindow

[概要]

起動・連携先の STA アプリケーション GUI プログラムの取得.

[メソッド仕様]

```
SceWindow getWindow();
```

[説明]

ツール起動・連携先の STA アプリケーション GUI プログラムを返却値として返す. ユーザは, この返却値を使用してツール連携を実現する.

[返却値]

返却値として, 起動・連携先の STA アプリケーション GUI プログラムを返す.

[使用例]

```
SceHandle handleEditor;
SceWindow windowEditor;
 :
handleEditor = toolMnanager.requestModule("host01","category01",
                                          "Editor",this,
                                           true,vectorParams);
if (handleEditor != null) {
  // 起動に成功
  windowEditor = handleEditor.getWindow(); // 連携先のSceWindowが
入る.
   :
} else {
  // 起動に失敗
}
```

getHandleId

[概要]

ツール連携 ID の取得.

[メソッド仕様]

```
int getHandleId();
```

[説明]

連携先の STA アプリケーション GUI プログラムの onRequestModule メソッド内で設定されたツール連携 ID を取得する.

[返却値]

返却値として, ツール連携 ID を返す.

[使用例]

```
SceHandle handleEditor;
int idEditor;
 :
handleEditor = toolMnanager.requestModule("host01","category01","Editor",this
                                                    true,vectorParams);
if (handleEditor != null) {
  // 起動に成功
  idEditor = handleEditor.getHandleId(); // ツール連携 ID が入る.
   :
} else {
  // 起動に失敗
}
```

getRequestMode

[概要]

起動モードの取得.

[メソッド仕様]

```
boolean getRequestMode();
```

[説明]

起動されたSTAアプリケーションGUIプログラムの実際の起動モードを取得する.

[返却値]

true : 新規にSTAアプリケーションGUIプログラムを起動
false : 既存のSTAアプリケーションGUIプログラムとの連携

[使用例]

```
SceHandle handleEditor;
boolean modeEditor;
 :
handleEditor = toolMnanager.requestModule("host01","category01",
                                          "Editor", this,
                                          true, vectorParams);
if (handleEditor != null) {
  // 起動に成功
  modeEditor = handleEditor.getRequestMode(); // 起動モードが入る.
   :
} else {
  // 起動に失敗
}
```

setWindow

[概要]

起動・連携先のSTAアプリケーションGUIプログラムを設定する.

[メソッド仕様]

```
public boolean setWindow(SceWindow window);
```

[説明]

起動・連携先のSTAアプリケーションGUIプログラムを設定する.

[返却値]

true : STAアプリケーションGUIプログラムの設定に成功した
false : STAアプリケーションGUIプログラムの設定に失敗した

[使用例]

```
public SceHandle onRequestModule(SceWindow caller,
                                 boolean new_request,
                                 Vector params)
{
  SceHandle handle;
  if(caller == null) {
    // ツール制御モジュールのボタンが押された場合
    return null;
  }
  // ツール連携の場合
  handle = new SceHandle();
  // SceHandle の設定
  if(!new_request) {
    handleID++;
    handle.setHandleId(handleID);
  } else {
    handle.setHandleId(0);
  }
  handle.setWindow(this);
   :
  // ツールの初期化処理を行う
   :
    return handle;
  }
```

setHandleId

[概要]

ツール連携 ID の設定.

[メソッド仕様]

```
public boolean setHandleId(int id);
```

[説明]

ツール連携 ID を設定する.

[返却値]

true : ツール連携 ID の設定に成功した
false : ツール連携 ID の設定に失敗した

[使用例]

```
public SceHandle onRequestModule(SceWindow caller,
                                 boolean new_request,
                                 Vector params)
{
  SceHandle handle;
  if(caller == null) {
    // ツール制御モジュールのボタンが押された場合
    return null;
  }
  // ツール連携の場合
  handle = new SceHandle();
  // SceHandle の設定
  if(!new_request) {
    handleID++;
    handle.setHandleId(handleID);
  } else {
    handle.setHandleId(0);
  }
  handle.setWindow(this);
   :
  // ツールの初期化処理を行う
   :
  return handle;
}
```

4.2.13 ToolList クラス

size

[概要]
動作中のツールの数を返却する.

[メソッド仕様]

```
int size();
```

[説明]
動作中のツールの個数を返却する. ツールは動作中のツールの個数を使用してツール連携の場合の new,renew のハンドリングを行なう.

[返却値]
toolname で指定したツールが動作中の ningTool メソッド呼び出し時に動作していたツールの数を返す.

[使用例]

```
ToolList  tlist;
int       toolNum;
 :
tlist = toolManager.getRunningTool();
if (tlist != null) {
  // 動作中のツール一覧情報の取得に成功
  toolNum = tlist.size(); // 動作中のツールの個数を取得.
   :
} else {
  // 動作中のツール一覧情報の取得に失敗
}
```

containsTool

[概要]

指定したツールが動作中かを返却する.

[メソッド仕様]

```
boolean containsTool(String toolname);
```

[説明]

toolname で指定したツールが動作中であるかを返却する. 動作中であるかどうかの情報を使用して, ツール連携の場合の new,renew のハンドリングを実現する.

[返却値]

true : toolname で指定したツールが動作している
false : toolname で指定したツールが動作していない

[使用例]

```
ToolList  tlist;
boolean   viExecf;
boolean   execNewf;
 :
tlist = toolManager.getRunningTool();
if (tlist != null) {
  // 動作中のツール一覧情報の取得に成功
  viExecf = tlist.containsTool("vi"); // vi が動作しているかの情報
を取得
  if (viExecf) {
    //vi が動作中：vi を renew で起動
    execNewf = false;
  } else {
    //vi が動作していない：vi を new で起動
    execNewf = true;
  }
   :
} else {
  // 動作中のツール一覧情報の取得に失敗
}
```

containsCategory

[概要]

指定したカテゴリに属するツールが動作中かを返却する.

[メソッド仕様]

```
boolean containsCategory(String category);
```

[説明]

categoryで指定したカテゴリに属するツールが動作中であるかを返却する. 動作中であるかどうかの情報を使用して, ツール連携の場合の new,renew のハンドリングを行なう.

[返却値]

true : categoryで指定したカテゴリに属するツールが動作している
false : categoryで指定したカテゴリに属するツールが動作していない

[使用例]

```
ToolList  tlist;
boolean   editorExecf;
boolean   execNewf;
 :
tlist = toolManager.getRunningTool();
if (tlist != null) {
  // 動作中のツール一覧情報の取得に成功
  editorExecf = tlist.containsCategory("Editor"); // Editor が動作
中か
  if (editorExecf) {
    //editor が動作中：editor を renew で起動
    execNewf = false;
  } else {
    //editor が動作していない：editor を new で起動
    execNewf = true;
  }
   :
} else {
  // 動作中のツール一覧情報の取得に失敗
}
```

toolAt

[概要]

指定したインデックスに対応するツール名称を返却する.

[メソッド仕様]

```
String toolAt(int index);
```

[説明]

indexで指定したインデックスに格納されている動作中のツールの情報からツール名を返却する.

[返却値]

indexで指定したインデックスに格納されている動作中のツールの情報からツールの名称を返す. 指定したインデックスが0未満あるいは動作中ツールの個数以上の場合はnullを返す.

[使用例]

```
ToolList  tlist;
int       toolNum;
String    execToolList[];
 :
tlist = toolManager.getRunningTool();
if (tlist != null) {
  // 動作中のツール一覧情報の取得に成功
  toolNum = tlist.size(); // 動作中のツールの個数を返却
  if (toolNum > 0) {
    //ツールが動作している
    execToolList = new String [toolNum];
    for(int li=0; li<toolNum; li++) {
      execToolList[li] = tlist.toolAt(li);
    }
  } else {
    //ツールが動作していない
  }
   :
} else {
  // 動作中のツール一覧情報の取得に失敗
}
```

categoryAt

[概要]

指定したインデックスに対応するカテゴリ名称を返却する.

[メソッド仕様]

```
String categoryAt(int index);
```

[説明]

indexで指定したインデックスに格納されている動作中のツールの情報からカテゴリ名を返却する.

[返却値]

indexで指定したインデックスに格納されている動作中のツールの情報からカテゴリの名称を返す. 指定したインデックスが0未満あるいは動作中ツールの個数以上の場合はnullを返す.

[使用例]

```
ToolList  tlist;
int       toolNum;
String    execCategoryList[];
 :
tlist = toolManager.getRunningTool();
if (tlist != null) {
  // 動作中のツール一覧情報の取得に成功
  toolNum = tlist.size(); // 動作中のツールの個数を返却
  if (toolNum > 0) {
    //ツールが動作している
    execCategoryList = new String [toolNum];
    for(int li=0; li<toolNum; li++) {
      execCategoryList[li] = tlist.toolAt(li);
    }
  } else {
    //ツールが動作していない
  }
   :
} else {
  // 動作中のツール一覧情報の取得に失敗
}
```

4.2.14 SceFileSelectDialog クラス

setTitle

[概要]
SCE ファイルセレクトダイアログボックスのタイトルを設定する.

[メソッド仕様]

```
void setTitle(String title);
```

[説明]
title で指定したタイトル文字列を SCE ファイルセレクトダイアログボックスのタイトルに設定する.

[返却値]
本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
 :
// ファイルセレクトダイアログを表示
selectDlg = new SceFileSelectDialog(this);
selectDlg.setTitle("file open");
 :
selectDlg.setVisible(true);
}
```

setHostName

[概要]

SCEファイルセレクトダイアログボックスで選択可能なホスト名を設定する.

[メソッド仕様]

```
void setHostName(String [] hostname);
```

[説明]

hostnameで指定したホスト名をSCEファイルセレクトダイアログボックスのホスト名選択リストに設定する.

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
String              hostname[];
 :
// ファイルセレクトダイアログを表示
hostName = new String [1];
hostname[0] = "chobi.koma.jaeri.go.jp";
selectDlg = new SceFileSelectDialog(this);
selectDlg.setHostName(hostname);
 :
selectDlg.setVisible(true);
}
```

setDefaultHostName

[概要]

ホスト名に初期表示するホストを設定する.

[メソッド仕様]

```
void setDefaultHostName(String  defaultHost);
```

[説明]

defaultHost で指定したホスト名を SCE ファイルセレクトダイアログボックスのホスト名選択リストに初期設定する.

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
String              hostname[];
 :
// ファイルセレクトダイアログを表示
hostName = new String [1];
hostname[0] = "chobi.koma.jaeri.go.jp";
selectDlg = new SceFileSelectDialog(this);
selectDlg.setDefaultHostName(hostname);
 :
selectDlg.setVisible(true);
}
```

setDirectoryName

[概要]

SCEファイルセレクトダイアログボックスで初期表示するディレクトリを設定する.

[メソッド仕様]

```
void setDirectoryName(String directoryName);
```

[説明]

directoryNameで指定したディレクトリ名をSCEファイルセレクトダイアログボックス表示の際の初期表示ディレクトリとして設定する.

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
String              directory;
 :
// ファイルセレクトダイアログを表示
directory = "/work002/SCE.main/IR2.2/adapter";
selectDlg = new SceFileSelectDialog(this);
selectDlg.setDirectoryName(directory);
 :
selectDlg.setVisible(true);
}
```

setDirectoryName

[概要]

SCEファイルセレクトダイアログボックスで表示するディレクトリを複数設定する.

[メソッド仕様]

```
void setDirectoryName(String[] directoryName);
```

[説明]

directoryNameで指定したディレクトリ名をSCEファイルセレクトダイアログボックス表示の際の表示ディレクトリとして設定する. その際、setHostNameで設定されたホストと対応する順で登録する事が必要である.

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
String[]            directory;
 :
// ファイルセレクトダイアログを表示
directory[1] = "/work002/SCE.main/IR2.2/adapter";
directory[2] = "/work002/SCE.develop/IR2.2/adapter";
directory[3] = "/work002/SCE.develop_h/IR2.2/adapter";
selectDlg = new SceFileSelectDialog(this);
selectDlg.setDirectoryName(directory);
 :
selectDlg.setVisible(true);
}
```

setFileTypeList

[概要]

SCEファイルセレクトダイアログボックスでファイル一覧に表示対象とするフィルタを設定する.

[メソッド仕様]

```
void setFileTypeList(String [] fileType);
```

[説明]

fileTypeで指定したフィルタを通過したファイルをSCEファイルセレクトダイアログボックスのファイル一覧の表示対象とすることを設定する. フィルタに指定可能なワイルドカードはアスタリスクのみである.

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
String              fileType[];
 :
// ファイルセレクトダイアログを表示
fileType = new String [1];
fileType[0] = "*.f";
selectDlg = new SceFileSelectDialog(this);
selectDlg.setFileTypeList(fileType);
 :
selectDlg.setVisible(true);
}
```

setMultiSelect

[概要]

SCE ファイルセレクトダイアログボックスでファイル一覧から選択可能なファイルを単一選択とするか複数選択を可能とするかを指定する.

[メソッド仕様]

```
void setMultiSelect(boolean multiSel);
```

[説明]

multiSel で true を指定した場合, ファイル一覧からの複数選択を許可する. multiSel で false を指定した場合, ファイル一覧からは単一ファイルのみの選択を許可する.

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
 :
// ファイルセレクトダイアログを表示
selectDlg = new SceFileSelectDialog(this);
selectDlg.setMultiSel(false);              //単一選択設定
 :
selectDlg.setVisible(true);
}
```

setFileDirectory

[概要]

SCEファイルセレクトダイアログボックスでファイル一覧に表示を行なう対象を指定する.

[メソッド仕様]

```
void setFileDirectory(int fileDirectory);
```

[説明]

fileDirectoryに指定するために，本クラスには以下のクラス変数が定義されている.

FILE : ファイルのみ表示 DIRECTORY : ディレクトリのみ表示 BOTH : ファイル及びディレクトリを表示

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
int                 fileDirectory;
 :
// ファイルセレクトダイアログを表示
selectDlg = new SceFileSelectDialog(this);
fileDirectory = selectDlg.BOTH;
selectDlg.setFileDirectory(fileDirectory); //ファイル及びディレクトリ表示
 :
selectDlg.setVisible(true);
}
```

setFileUse

[概要]

SCEファイルセレクトダイアログボックスで取得したファイルの使用方法を指定する．使用方法としては，“読み込み” 又は “書き込み” のいずれかを指定する．

[メソッド仕様]

```
void setFileUse(int fileUse);
```

[説明]

fileUseに指定するために，本クラスには以下のクラス変数が定義されている．

READTYPE : 読み込み
WRITETYPE : 書き込み

READTYPEを指定した場合，SCEファイルセレクトダイアログでは，すでに存在するファイルのみ選択できるようになる．また，この場合，ファイル名入力領域は入力不可となる．

WRITETYPEを指定した場合，存在しないファイルを指定する必要があるため，ファイル名入力領域は入力可能となる．

[返却値]

本メソッドの返却値はない

[使用例]

```
SceFileSelectDialog selectDlg;
int                 fileUse;
 :
// ファイルセレクトダイアログを表示
selectDlg = new SceFileSelectDialog(this);
fileUse = selectDlg.READTYPE;
selectDlg.setFileUse(fileUse); //使用方法はRead
 :
selectDlg.setVisible(true);
}
```

getButtonName

[概要]

SCEファイルセレクトダイアログボックスを終了させたボタンの種別を取得する.

[メソッド仕様]

```
int getButtonName();
```

[説明]

SCEファイルセレクションダイアログはOKボタンを押下するかあるいはCancelボタンを押下することにより,表示を終了する.本メソッドはSCEファイルセレクトダイアログがどちらのボタン押下によって表示を終了したのかを返す.

[返却値]

PUSH_OK : OKボタンの押下
PUSH_CANCEL : CANCELボタンの押下

[使用例]

```
SceFileSelectDialog selectDlg;
int                 button;
 :
// ファイルセレクトダイアログを表示
selectDlg = new SceFileSelectDialog(this);
fileDirectory = selectDlg.READTYPE;
selectDlg.setTitle("file open");
 :
selectDlg.setVisible(true);
button = selectDlg.getButtonName();
if (button == selectDlg.PUSH_OK) {
  // OKボタン押下
 :
} else {
  // CANCELボタン押下
 :
}
```

getFileNames

[概要]

SCEファイルセレクトダイアログボックスで選択したファイル名一覧を取得する.

[メソッド仕様]

```
String [] getFileNames();
```

[説明]

SCEファイルセレクションダイアログはOKボタンを押下するかあるいはCancelボタンを押下することにより,表示を終了する.本メソッドはSCEファイルセレクトダイアログの終了時に選択されていたファイル名の一覧を返す.,

[返却値]

SCEファイルセレクトダイアログで選択したファイル名一覧を返す.ファイル名の形式は「ホスト名:絶対パスのファイル名」で返却する.

[使用例]

```
SceFileSelectDialog selectDlg;
int                 button;
String              selectFiles[]
 :
// ファイルセレクトダイアログを表示
selectDlg = new SceFileSelectDialog(this);
fileDirectory = selectDlg.READTYPE;
selectDlg.setTitle("file open");
 :
selectDlg.setVisible(true);
button = selectDlg.getButtonName();
if (button == selectDlg.PUSH_OK) {
  // OKボタン押下:選択したファイル名を取得
  selectFiles = selectDlg.getFileNames();
 :
} else {
  // CANCELボタン押下
 :
}
```

4.2.15 SceMessageDialog クラス

setMessage

[概要]

SCE メッセージダイアログボックスに表示するタイトル, メッセージ, 埋字, ボタンの各情報を設定する.

[メソッド仕様]

```
void setMessage(String title, String msg, String inMsg[], int btn);
```

[説明]

SCE メッセージダイアログのタイトル, メッセージ, 埋字, ボタンを設定する. title には SCE メッセージダイアログボックスのタイトルを指定する. msg には SCE メッセージダイアログに表示するメッセージを指定する. inMsg にはメッセージ中に挿入する埋字を指定する. btn には SCE メッセージダイアログの下部に表示するボタン種別を指定する. bnt に指定できる値として, 以下のクラス変数が定義されている.

OK : [OK] ボタンを表示
OKCANCEL : [OK], [Cancel] ボタンを表示
YESNO : [Yes], [No] ボタンを表示
YESNOCANCEL : [Yes], [No], [Cancel] ボタンを表示

[返却値]

本メソッドの返却値はない.

[使用例]

```
SceMessageDialog msgDlg;
 :
if (rtn != 0) {
  // メッセージダイアログを表示
  msgDlg = new SceMessageDialog(this);
  msgDlg.setMessage("Error", "File open error",
                    null, msgDlg.OKCANCEL);
 :
  msgDlg.setVisible(true);
```

setTitle

[概要]

SCEメッセージダイアログボックスに表示するタイトルを設定する.

[メソッド仕様]

```
void setTitle(String title);
```

[説明]

titleにはSCEメッセージダイアログボックスのタイトルを指定する. タイトルはクラス変数として定義している, 使用可能なクラス変数は次の通り

QUESTIONTITLE : 問合せダイアログ
INFORMATIONTITLE : インフォメーションダイアログ
WARNINGTITLE : 警告ダイアログ
ERRORTITLE : エラーメッセージダイアログ

[返却値]

本メソッドの返却値はない.

[使用例]

```
SceMessageDialog msgDlg;
 :
if (rtn != 0) {
  // メッセージダイアログを表示
  msgDlg = new SceMessageDialog(this);
  msgDlg.setTitle(msgDlg.ERRORTITLE);
 :
  msgDlg.setVisible(true);
```

setMsg

[概要]

SCEメッセージダイアログボックスに表示するメッセージと埋字を設定する.

[メソッド仕様]

```
void setMsg(String msg, String inMsg[]);
```

[説明]

msgにはSCEメッセージダイアログボックスに表示するメッセージを指定する. inMsgにはメッセージ中に挿入する埋字を指定する. 埋字はメッセージの中の%Sの部分と置き換えが行われる。

[返却値]

本メソッドの返却値はない.

[使用例]

```
SceMessageDialog msgDlg;
 :
if (rtn != 0) {
  // メッセージダイアログを表示
  msgDlg = new SceMessageDialog(this);
  msgDlg.setMsg("File open error.",null);
  :
  msgDlg.setVisible(true);
```

setBtn

[概要]

SCEメッセージダイアログボックスに表示するボタンの情報を設定する.

[メソッド仕様]

```
void setBtn(int btn);
```

[説明]

btnにはSCEメッセージダイアログボックスに表示するボタンの種別を指定する.
btnには本クラスで定義されている以下の値を指定することができる.

OK : OKボタンのみの表示を指定する
OKCANCEL : OKボタンとCANCELボタンの表示を指定する
YESNO : YESボタンとNOボタンの表示を指定する
YESNOCANCEL : YESボタンとNOボタンとCANCELボタンの表示を指定する

[返却値]

本メソッドの返却値はない.

[使用例]

```
SceMessageDialog msgDlg;
 :
if (rtn != 0) {
  // メッセージダイアログを表示
  msgDlg = new SceMessageDialog(this);
  msgDlg.setBtn(msgDlg.OKCANCEL);
 :
  msgDlg.setVisible(true);
```

getButtonName

[概要]

SCEメッセージダイアログボックスで押下されたボタンの情報を取得する.

[メソッド仕様]

```
int getButtonName();
```

[説明]

SCEメッセージダイアログボックスで押下されたボタン種別を取得する. SCEメッセージダイアログボックスはいずれかのボタンを押下された場合に表示を終了するため, 本メソッドは, SCEメッセージダイアログボックスの表示終了後に呼び出す.

[返却値]

ボタン種別は本クラスのクラス変数に以下のように定義されている.

PUSH_OK : OKボタンが押下された
PUSH_CANCEL : CANCELボタンが押下された
PUSH_YES : YESボタンが押下された
PUSH_NO : NOボタンが押下された

[使用例]

```
SceMessageDialog msgDlg;
int              btn;
 :
if (rtn != 0) {
 // メッセージダイアログを表示
 msgDlg = new SceMessageDialog(this);
 msgDlg.setBtn(msgDlg.OKCANCEL);
 :
 msgDlg.setVisible(true);
 btn = msgDlg.getButtonName();
 if (btn == nmsgDlg.PUSH\_OK) {
   // OKボタン押下時の処理
 } else {
   // CANCELボタン押下時の処理
 }
```

4.2.16 SceSocket クラス

本クラスは，アダプタとの間でSceソケット通信を行うためのソケットを提供するクラスである．

SceSocketクラスは，JDKのSocketクラスの派生クラスである．本クラスを利用する際は，Socketクラスのメソッドを使用する．

4.2.17 SceSocketBuffer クラス

本クラスは，Sceソケット通信に使用する送信用バッファを提供するクラスである．

SceSocketBufferクラスは，SceOutBuffer クラスの派生クラスである．本クラスを利用する際は，SceOutBuffer クラスのメソッドを使用する．ただし，本クラスでは，SceOutBuffer クラスで実現されている関数番号指定によるバッファの先行送信は行われないため，注意が必要である．

4.2.18 SceException クラス

setCode

[概要]

例外コードを設定する．

[メソッド仕様]

```
void setCode(int code);
```

[説明]

codeで指定されたコードをその例外中の例外コードとして設定する．

[返却値]

本メソッドに返却値はない．

[使用例]

```
SceException e = new SceException();
e.setCode(-100);
```

getCode

[概要]
例外コードを取得する.

[メソッド仕様]

```
int getCode();
```

[説明]
例外中に設定された例外コードを取得する.

[返却値]
例外コードを返す. コードが設定されていない場合は0を返す.

[使用例]

```
try {
  :
}
cacth (SceException e) {
  int code = e.getCode();
}
```

4.2.19 ScePackable インタフェース

ScePackableインタフェースを実装するクラスでは, アダプタへの送信用バッファのエンコードを行う.

putObj

[概要]

送信用バッファにデータを格納する.

[メソッド仕様]

```
void putObj(SceOutBuffer outBuf) throws IOException;
```

[説明]

outBuf で指定されたバッファに対して, データのエンコードを行うコードを記述する.

[返却値]

本メソッドに返却値はない. ただし, データの受信に失敗した場合は, IOException を発行する.

[使用例]

```
public void putObj(SceOutBuffer outBuf) throws IOException {
    outBuf.putObj(i);
    outBuf.putObj(f);
    outBuf.putObj(str);
}
```

4.2.20 SceUnpackable インタフェース

SceUnpackable インタフェースを実装するクラスでは, アダプタからの受信用バッファのデコードを行う.

getObj

[概要]
受信用バッファからデータを取り出す.

[メソッド仕様]

```
void getObj(SceInBuffer inBuf)throws IOException;
```

[説明]
inBuf で指定されたバッファから, データのデコードを行うコードを記述する.

[返却値]
本メソッドに返却値はない. ただし, データの受信に失敗した場合は, IOException を発行する.

[使用例]

```
public void getObj(SceInBuffer inBuf) throws IOException {
    try {
        i = inBuf.get_int();
        f = inBuf.get_float();
        str = inBuf.get_String();
    }
    catch(OutOfBufferException e) {
            // エラー処理
    }
}
```

5 STA通信基盤へのSTAアプリケーションの登録

STAアプリケーションをSTA通信基盤システムに組込むためには

1. アダプタの登録
2. ツール設定ファイルの編集
3. STAアプリケーションGUIプログラムの登録

を行う必要がある。

本章では，これらの方法について説明する．

なお，各種格納ディレクトリは図7に示す構成になっているものと仮定する．

表7: ディレクトリ構成例

No.	通称	ディレクトリ
1	アダプタライブラリ格納ディレクトリ	/usr/local/SCE/main/lib/
2	NEXUSライブラリ格納ディレクトリ	/usr/local/SCE/nexus-4.1.1/
3	アダプタ格納ディレクトリ	/usr/local/SCE/main/adapter/
4	アダプタ設定ファイル格納ディレクトリ	/usr/local/SCE/main/adplib/
5	ツール固有ワークファイル格納ディレクトリ	/usr/local/SCE/main/work/
6	STAアプリケーションGUIプログラム格納ディレクトリ	/usr/local/etc/httpd/htdocs/SCE/main/tool/

5.1 アダプタの登録

ツール提供者は、アダプタをツール管理サブシステムに登録する必要がある。アダプタ(実行ファイル)の登録と、アダプタ設定ファイルの登録を行うことで、ツール管理サブシステムからアダプタを実行できるようになる。

5.1.1 実行ファイルの登録

表7で示すアダプタ格納ディレクトリに“ツール名称”のファイル名でアダプタ(実行ファイル)を格納する。

例えば、“diff”という名称のツールの場合では，

```
/usr/local/SCE/main/adapter/diff
```

としてアダプタを格納する。

5.1.2 アダプタ設定ファイルの登録

表7に示すアダプタ設定ファイル格納ディレクトリに“ツール名称”のファイル名でアダプタ設定ファイルを格納する。

アダプタ設定ファイルは、設定項目の集合体であり、設定項目の書式は

項目名: 設定値

である．アダプタ共通の設定項目として、Comment, Data, Registration, Versionが予約されている。各STAアプリケーション開発者は，これら予約された設定項目の他にアダプタ固有の設定項目を定義することができる。各設定値は，C言語ライブラリを使用して，アダプタから容易に取得することができる．以下に本章の例におけるアダプタ設定ファイルの例を示す。

```
Comment: show differences between two text files.
Data: Mon 30 10:00 JST 1997
Registration: aaa@sr01.koma.jaeri.go.jp
Version: 1.00
DiffOption: -c
```

5.1.3 ツール固有ワークファイル格納ディレクトリの作成

表(7)に示すツール固有ワークファイル格納ディレクトリは各STAアプリケーション固有のファイルを格納するディレクトリである。各STAアプリケーションはこのディレクトリを情報の格納や一時ファイルの作成等の目的で，自由に使用することができる．

ただし，ワークファイル格納ディレクトリは各STAアプリケーション間で衝突することが予想させるため、ツール固有ワークファイル格納ディレクトリ内のアダプタ名と同一のディレクトリをツール固有ワークファイル格納ディレクトリとすることを奨励する．

5.1の場合，以下のコマンドによって，ツール固有ワークファイル格納ディレクトリを作成する．

```
# cd /usr/local/SCE/main/work
# mkdir diff
```

5.2 ツール設定ファイル

ツール設定ファイルは、すべてのSTAアプリケーションの情報を記述するファイルである。

ツール設定ファイルは、STAアプリケーションGUIプログラム格納ディレクトリに“toolinf.txt”のファイル名で存在する。

本章の例におけるツール設定ファイルは

/usr/local/etc/httpd/htdocs/SCE/main/tool/toolinf.txt

である．

表 8: ツール設定ファイルの設定項目

番号	項目名	説明	必須/任意
1	Tool	ツールを記述する。	必須
2	Category	ツールが属するカテゴリ名称を記述する。	必須
3	Gui	ツールのGUI名称を記述する。	必須
4	Adapter	ツールが使用するアダプタ名称を記述する。	必須
5	LinkTool	他ツールへリンクしている場合は、リンク先のツール名称を記述する。	任意
6	LinkCategory	他ツールへリンクしている場合は、リンク先のツールが属するカテゴリ名を記述する。	任意
7	Server	ツールが使用できるサーバ名を記述する。複数のサーバで使用できる場合は、空白で区切って記述する。サーバに依存しないツールの場合は、“ any ”と記述する。	必須
8	#	コメントを記述する。	任意

ツールを登録する際は、このファイルにツールの情報を追加する必要がある。

ここでは5.1の場合を例として登録方法を説明する．以下，登録するSTAアプリケーションGUIプログラムのカテゴリ名はfile, STAアプリケーションGUIプログラムのメインのクラス (SceWindowクラスの派生クラス) の名称がdiffであるとする．

5.2.1 ツール設定ファイルの設定項目

ツール設定ファイルには，各STAアプリケーションの情報を記述する．

ツール設定ファイルは設定項目の集合体であり，設定項目の書式は

項目名: 設定値

である．ツール設定ファイルの設定項目を表8に示す．

尚，Toolの設定は，各STAアプリケーションの設定項目の最初の行に記述する必要がある．Tool行から次のTool行の直前の行までが1つのSTAアプリケーションに対する設定となる．

ツール設定ファイルの例を以下に示す。

```
# Compiler
Tool: compiler
Category: compiler
Gui: compiler
Adapter: compiler
Server: any
# diff tool
Tool: diff
Category: file
Gui: diff
Adapter: diff
Server: desr01.koma.jaeri.go.jp,hi00011.koma.jaeri.go.jp
```

5.3 STA アプリケーション GUI プログラムの登録

STA アプリケーション GUI プログラムは、GUI 管理サブシステムの WWW サーバに登録する。ツール設定ファイルへのツールの登録、STA アプリケーション GUI プログラム格納ディレクトリのサブディレクトリの作成、STA アプリケーション GUI プログラムの登録、STA アプリケーション GUI プログラム設定ファイルの登録を行うことで、GUI サブシステムから STA アプリケーション GUI プログラムが利用できるようになる。

5.3.1 STA アプリケーション GUI プログラム格納ディレクトリのサブディレクトリ作成

STA アプリケーション GUI プログラム格納ディレクトリ内に個々の STA アプリケーション GUI プログラムを格納するサブディレクトリとして,

STA アプリケーション GUI プログラム格納ディレクトリ/tool/カテゴリ名称
/*STA アプリケーション GUI プログラムクラス名*

を作成する。ここで, *STA アプリケーション GUI プログラムクラス名* は, STA アプリケーション GUI プログラムのメインのクラス (SceWindow クラスの派生クラス) の名称である.

本章の例 (ツール名が "diff", カテゴリ名が "file") では,

```
# cd /usr/local/etc/httpd/htdocs/SCE/main/tool # mkdir file # cd file
# mkdir diff
```

を実行し, STA アプリケーション GUI プログラム格納ディレクトリを作成する.

5.3.2 クラスファイルの登録

次に, 5.3.1 で作成したディレクトリにツール実行モジュールのクラスファイルを格納する.

本章の例では,

```
/usr/local/etc/httpd/htdocsSCE/main/tool/file/diff/diff.class
```

としてSTAアプリケーションGUIプログラムを格納する。この他にSTAアプリケーションGUIプログラムが使用するクラスがあれば，それらもこのディレクトリに格納する.

5.3.3 STAアプリケーションGUIプログラム設定ファイルの登録

5.3.1で作成したディレクトリにSTAアプリケーションGUIプログラム設定ファイルを作成する。設定ファイルには、上記で、格納したツール実行モジュールの設定を格納する。設定ファイルのファイル名称は、STAアプリケーションGUIプログラムのファイル名称の拡張子を .txt に変更した名称とする。

本章の例の場合，

```
/usr/local/etc/httpd/htdocs/SCE/main/tool/file/diff.txt
```

としてSTAアプリケーションGUIプログラム設定ファイルを格納する。

STAアプリケーションGUIプログラム設定ファイルの書式はアダプタ設定ファイルと同様とする.

以下に本章の例における，STAアプリケーションGUIプログラム設定ファイルの例を示す。

```
Comment: show difference between two text files.
Data: Mon 30 10:00 JST 1997
Registration: aaa@sr01.koma.jaeri.go.jp
Version: 1.01
```

5.4 STA アプリケーション利用権限の管理

STA アプリケーションにおいても，通常のアプリケーションと同様，個人あるいはグループでの利用を想定し，それ以外の利用者からは利用できないようなものが存在する．本章では，STA アプリケーションに対し，このような制限を課すための設定方法を記述する．

本機能を使用することにより，各 STA アプリケーションに対し，そのアプリケーションを利用できる利用者を限定することができる．また,，複数のユーザから構成されるユーザグループを定義し，定義したユーザグループを利用者として指定することもできる．

5.4.1 ユーザグループ管理ファイルの設定

ユーザグループ管理ファイルは，複数のユーザから構成されるユーザグループの定義を行うファイルである．

ユーザグループ管理ファイルは，

GUIDIR/tool/group.txt

として作成する．

ユーザグループ定義ファイルは，ユーザグループ定義の集合である．ユーザグループ定義は，

ユーザグループ名 ： ユーザ *ID*, ユーザ *ID*,..., ユーザ *ID*

である．

以下にユーザグループ管理ファイルの例を示す。

```
guest:usr1,usr2,usr3
user:usr2,usr4,usr5,usr6
admin:usr8,usr9
```

5.4.2 アプリケーション利用権限管理ファイルの設定

STA アプリケーション利用権限管理ファイルは，STA アプリケーション毎の利用可能/不可能なユーザ ID を設定するファイルである．

STA アプリケーション利用権限管理ファイルは，

GUIDIR/tool/permit.txt

として作成する．

アプリケーション利用権限管理ファイルは，利用可否定義の集合である．利用可否定義は，

STA アプリケーション名：ユーザリスト

または，

カテゴリ名：ユーザリスト

である.

“ユーザリスト”は，各要素間を“,”(カンマ)で区切った，ユーザID及びユーザグループのリストである．また、“!”をユーザIDやユーザグループ名の先頭に付加することにより、そのユーザIDやユーザグループに対する禁止を指示することが出来る。但し、一つの利用可否定義内に、許可と禁止の指示を混在することは不可とする。

空のユーザリストが指定された場合は、全ユーザに対する利用可能を意味する.

また、STAアプリケーション利用権限ファイルに記述の無いSTAアプリケーションは全ユーザが使用可能であるものとする。

以下にSTAアプリケーション利用権限管理ファイルの例を示す。

```
PPEditor:Editor:usr1,usr3:usr,admin
CompileManager:Compiler::!guest
ExecuteManager:Executer::
```

6 STAアプリケーションの利用方法

6.1 環境設定

STA通信基盤は，通信路の確立の際に内部で rsh コマンド(リモート計算機上のコマンドを実行するコマンド)を使用するため、STAアプリケーションを利用するためには，STAサーバ機から各計算サーバに対して rsh ができるように.rhosts ファイルを設定する必要がある。.rhosts ファイルは各計算機のホームディレクトリに作成する．

.rhosts ファイルの設定が有効かどうかは、STAサーバ機で以下のコマンドを実行することにより確認できる．

```
% rsh 計算サーバのホスト名 -l 計算サーバでのユーザID ls
```

環境によっては rsh の代わりに remsh を使用することもある。またホスト名でなくIPアドレスを用いる機種もある．コマンドを実行して "Permission denied." などのエラーが発生する場合は、.rhosts ファイルを見直して修正する必要がある．

.rhosts ファイルの設定例を以下に示す．

- ホスト名を使った設定例

```
stasrv1.koma.jaeri.go.jp      STAサーバ機でのユーザID
```

- IPアドレスを使った設定例

```
133.53.187.77      STAサーバ機でのユーザID
```

6.2 STAアプリケーションの起動

PCまたはWSのNetscape Communicatorを起動する．その後、Netscape CommunicatorにSTA基本システムメインページのURLを指定する．ここで，STA基本システムの正常動作が保証されているブラウザは、Netscape Communicator 4.5以上 (Windows版、WS版)である．

6.3 ログイン

利用者はSTA基本システムログイン画面 (図10)
において STAサーバ機に登録されたユーザIDとパスワードを入力し、STA基本システムへログインする．

User IDフィールドにはSTAサーバ機のユーザIDを入力する。また、Passwordフィールドにパスワードを入力する．入力後，OKボタンをクリックする．

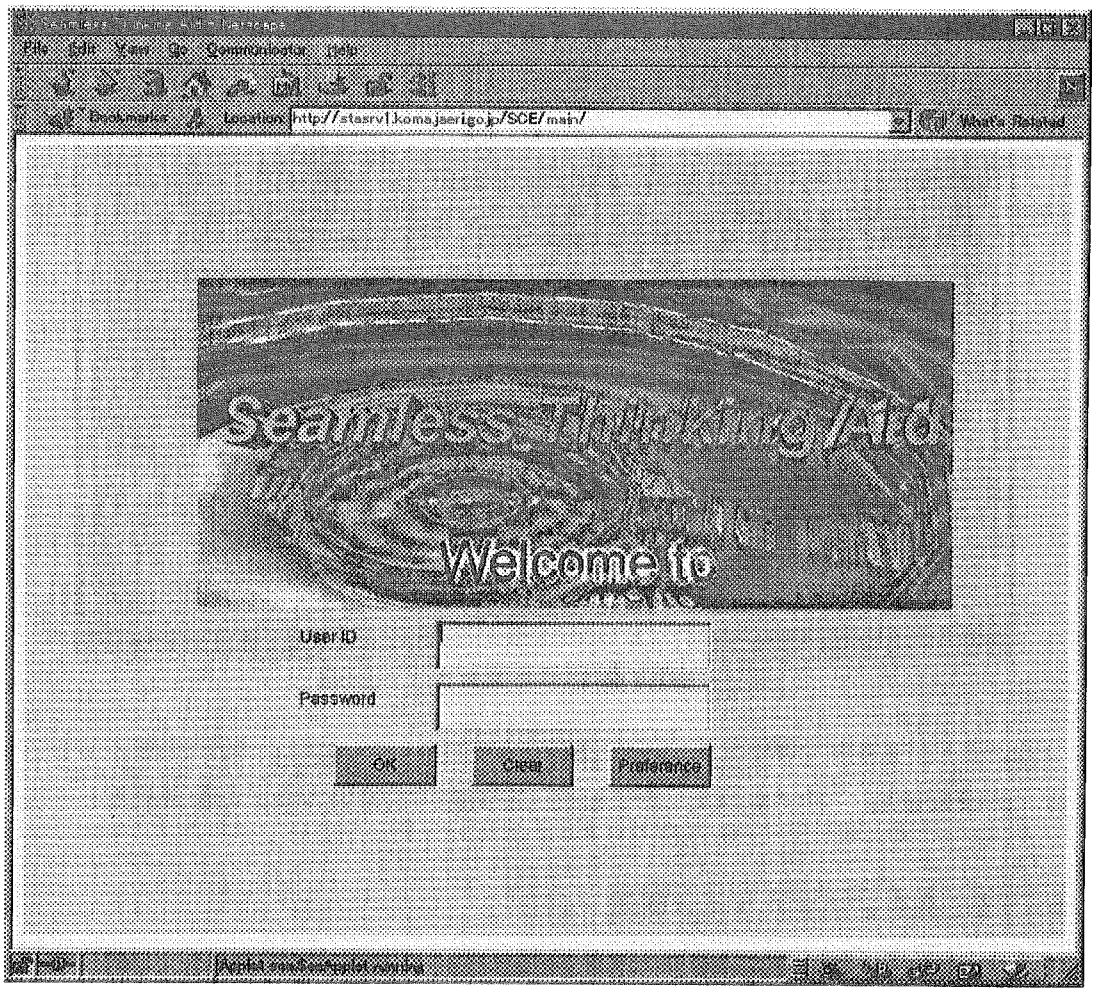

図 10: STA 基本システムログイン画面

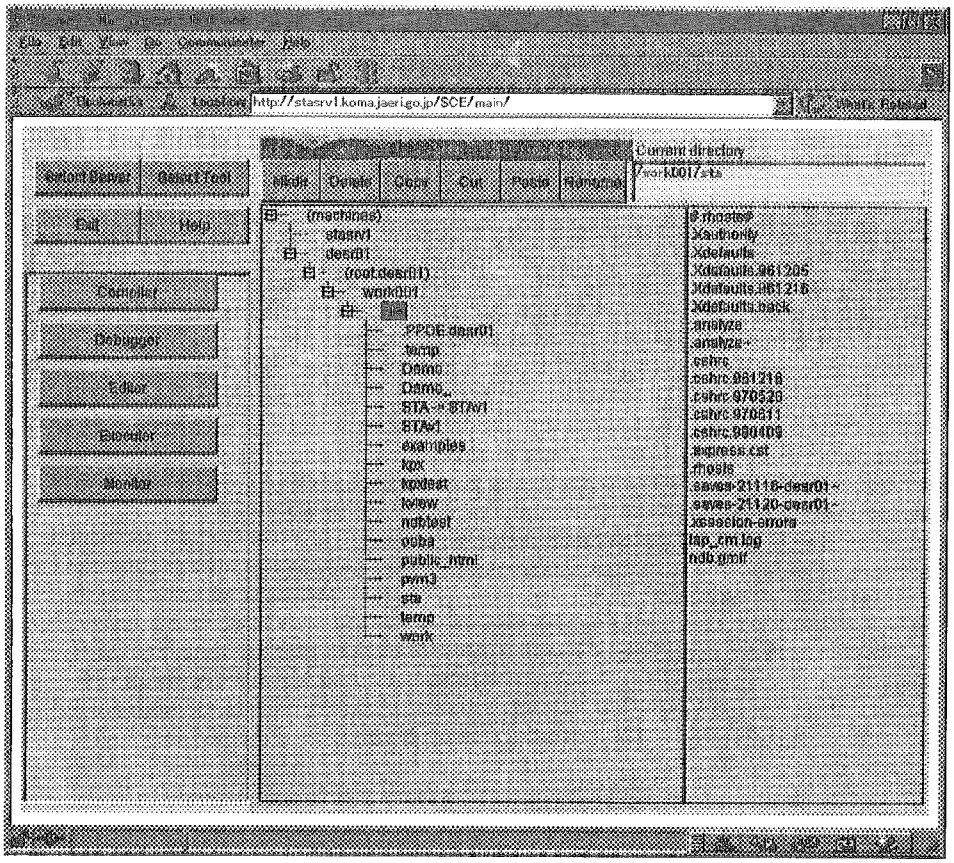

図 11: STA 基本システムメイン画面

Clear ボタンをクリックすると User ID フィールドと Password フィールドがクリアされる．Preference ボタンをクリックすると、画面表示領域の大きさを変更できる．

6.4 STA 基本システムメイン画面

ログインした後、STA 基本システムメイン画面 (図 11) が表示される．ここでは、ファイル操作，利用計算機の登録，利用する STA アプリケーションの登録を行うことができる．

以下でそれらの方法について説明する．

6.5 ファイルマネージャの利用

STA 基本システムメイン画面 (図 11) の右側の領域がファイルマネージャ画面である。この画面は、ディレクトリ階層をツリー表示するディレクトリ表示領域と、選択されているディレクトリ下のファイル名を一覧表示するファイル表示領域の 2 つの領域から構成されている．

ここでは、ファイルマネージャの操作法について説明する．

6.5.1 計算サーバの選択と解除

STA 基本システムメイン画面表示後、ファイルマネージャ画面のディレクトリ表示領域には利用可能な計算機の一覧が表示される。この状態では、どの計算機も選択されていない状態にある.

一覧で表示されている利用計算機名をクリックすると、その計算機が選択状態となり、選択した計算機のホームディレクトリ下のファイルとサブディレクトリを操作することができる.

ファイルマネージャ画面において計算機名表示部分をマウスで再度クリックすると、サーバの選択を解除することができる.

6.5.2 ファイルおよびディレクトリの表示と選択

ファイルマネージャ画面で計算機を選択し、ホームディレクトリ下のディレクトリとファイルが一覧表示されている状態で、他のディレクトリをクリックすると、そのディレクトリに移動することができる。ディレクトリに移動すると、移動先ディレクトリ下のディレクトリとファイルの一覧が表示される。このように、ディレクトリのクリックを繰り返すことによって、目的とするディレクトリに移動することができる.

ディレクトリをクリックしたとき、クリックされたディレクトリは反転表示され、選択状態にあることを表す. ファイルマネージャ画面のディレクトリ表示領域で反転表示をされているディレクトリをカレントディレクトリと呼ぶ. カレントディレクトリは、ファイル操作及び STA アプリケーション実行において基点のディレクトリとなる。

ファイルマネージャ画面のファイル表示領域でファイル名をクリックすると、ファイル名が反転表示され、選択状態となる. Shift キーや Ctrl キーを押しながらクリックすることにより、複数のファイルを選択することができる. 選択されているファイルが、ファイル操作や STA アプリケーション実行の処理対象となる.

6.5.3 ファイル操作

ファイルマネージャ画面では以下のファイル操作を行うことができる.

- ディレクトリの作成 (Mkdir)
- ファイルおよびディレクトリの削除 (Delete)
- ファイルまたはディレクトリの複写 (Copy)
- ファイルまたはディレクトリの移動 (Cut)
- ファイルまたはディレクトリの貼り付け (Paste)
- ファイルまたはディレクトリの名称変更 (Rename)

ファイルやディレクトリに対する操作は、まず操作対象ファイルやディレクトリを選択し、次に、画面上の操作ボタンを押すことにより実行される. 例えば、ホームディレクトリ下に "dir1" という名称のディレクトリを作成する場合は、ホームディレクトリを選択

し、Mkdir ボタンを押し、表示されたダイアログボックスに “dir1” と入力し OK ボタンを押す、という操作を行う.

複写や移動は、Copy または Cut ボタンと Paste ボタンの操作を組み合わせて行う. 複写を行う場合は、複写対象のファイルやディレクトリを選択して Copy ボタンを押し、その後複写先ディレクトリを選択して Paste ボタンを押す. 移動の場合は Copy ボタンの代わりに Cut ボタンを押す.

6.5.4 ポップアップメニューによるファイル操作

ファイルマネージャ画面上で右ボタンをクリックすることで、マウスカーソル位置にポップアップメニューを表示することができる.

ポップアップメニューの “Mkdir”、“Delete”、“Copy”、“Cut”、“Paste”、“Rename” は、ファイルマネージャ画面の同名のボタンの代わりに使用することができる (ボタンクリックの代わりにポップアップメニューから機能を選択できる)。

ポップアップメニューには、Mkdir ボタン等と同等の機能の他、以下の機能が含まれる. ポップアップメニュー固有機能を以下に示す.

(1) ホーム復帰機能

機能

ホームディレクトリへ復帰する

操作

ファイルマネージャ画面でマウス右ボタンクリックによりポップアップメニューを開き、“Return Home” を選択する.

(2) オプション設定機能

機能

Option ダイアログを開き、ファイルマネージャに対するオプションを設定する.

操作

ファイルマネージャ画面でマウス右ボタンクリックによりポップアップメニューを開き、“Option” を選択すると Option ダイアログが開く。ここで、先頭にピリオド (“.”) の付いた特殊ファイルやディレクトリの表示モードを変更する.

6.6 利用計算機の登録

STA 基本システムにおいて STA アプリケーションを実行する対象となる計算機は、Select Serverダイアログ(図 12)によって登録する必要がある.

STA基本システムメイン画面 (図11) でSelect Server ボタンをクリックすると、Select Server ダイアログが開く.

ダイアログには、SCE で利用可能な計算機群が表示される (Host Name フィールド)。使用する計算機の選択は、Host Name フィールドの計算機に対応する User ID フィールドに当該利用計算機での利用者のユーザ ID を入力することによって行う.

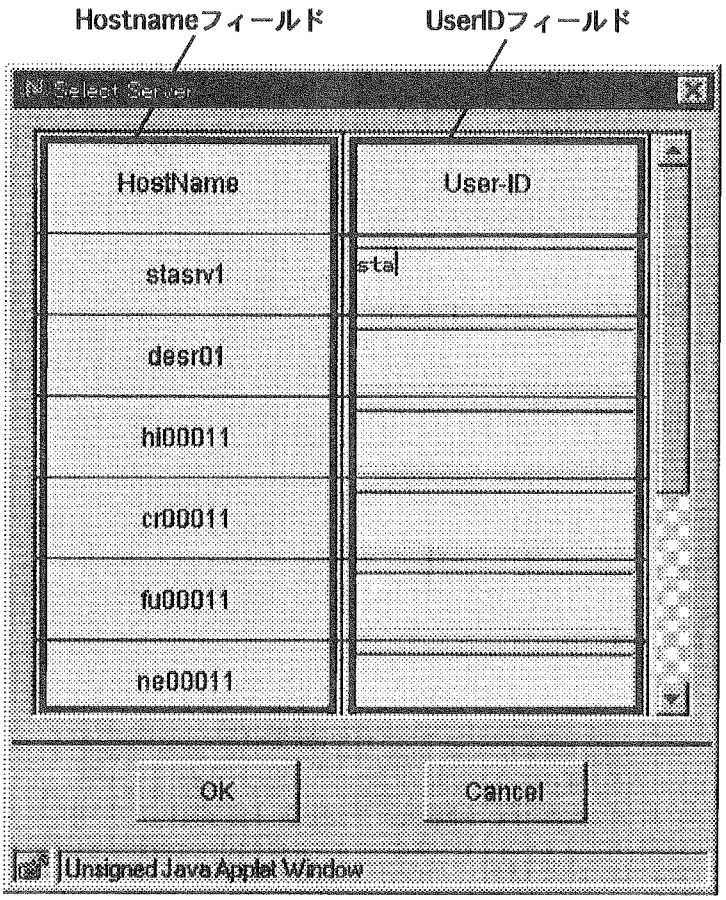

図 12: Select Server ダイアログ

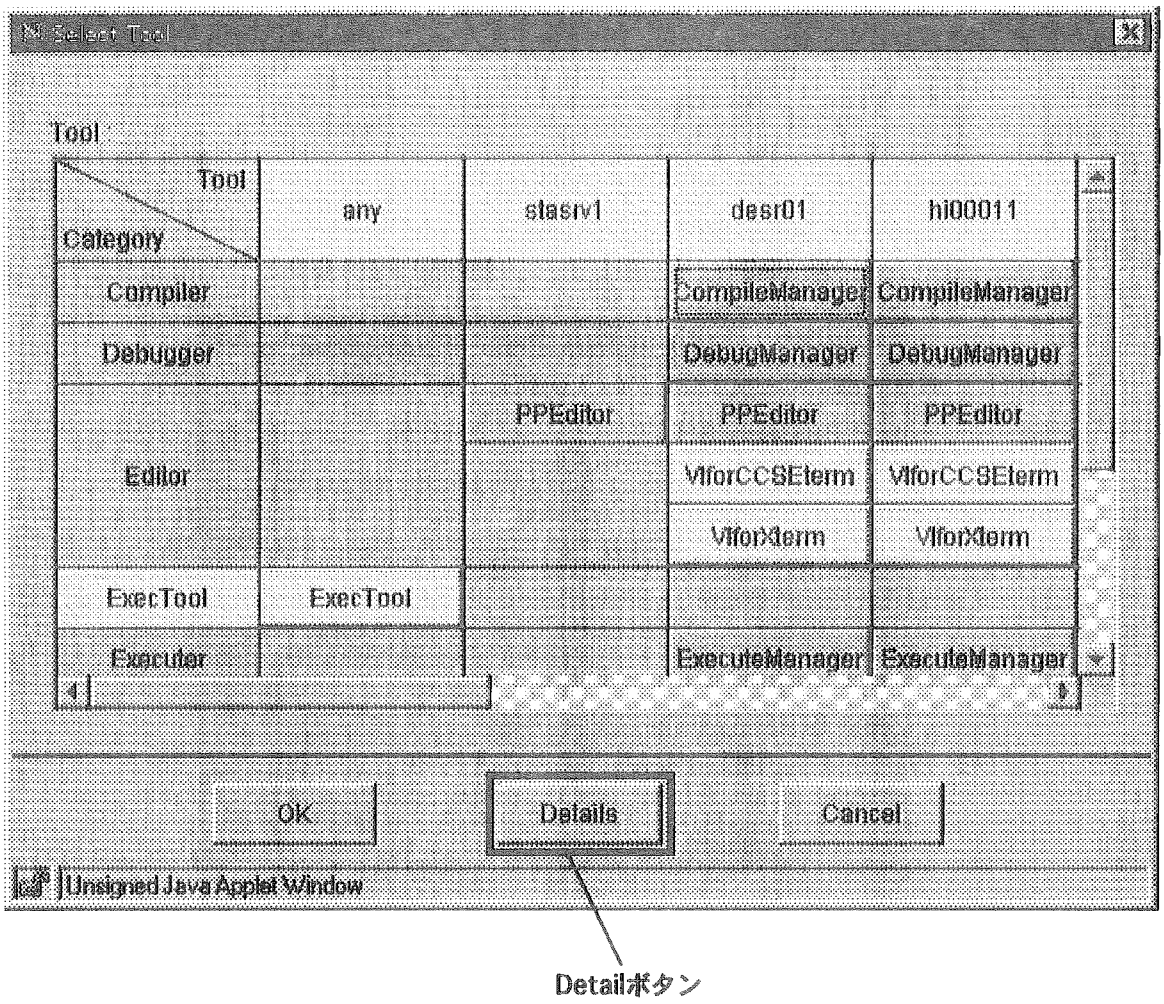

図 13: Select Tool ダイアログ

6.7 利用する STA アプリケーションの登録

STA アプリケーションを利用するには、STA アプリケーション選択ダイアログボックス (図 13 によって利用者自身が利用する STA アプリケーションを登録する必要がある.

6.7.1 STA アプリケーション選択ダイアログの表示

STA 基本システムメイン画面 (図:11) の Select Tool ボタンをクリックすると、Select Tool ダイアログが開く (図 13)。ダイアログには、利用可能な計算機名 (Select Server ダイアログを用いて選択した計算機) と、計算機毎に利用可能な STA アプリケーション群が表示される。計算機 any に登録されている STA アプリケーションは、特定の計算機を必要としない STA アプリケーションである。

STA アプリケーションの選択 使用する STA アプリケーションの選択は、Select Tool ダイアログに表示されている STA アプリケーション名をマウスでクリックすることによっ

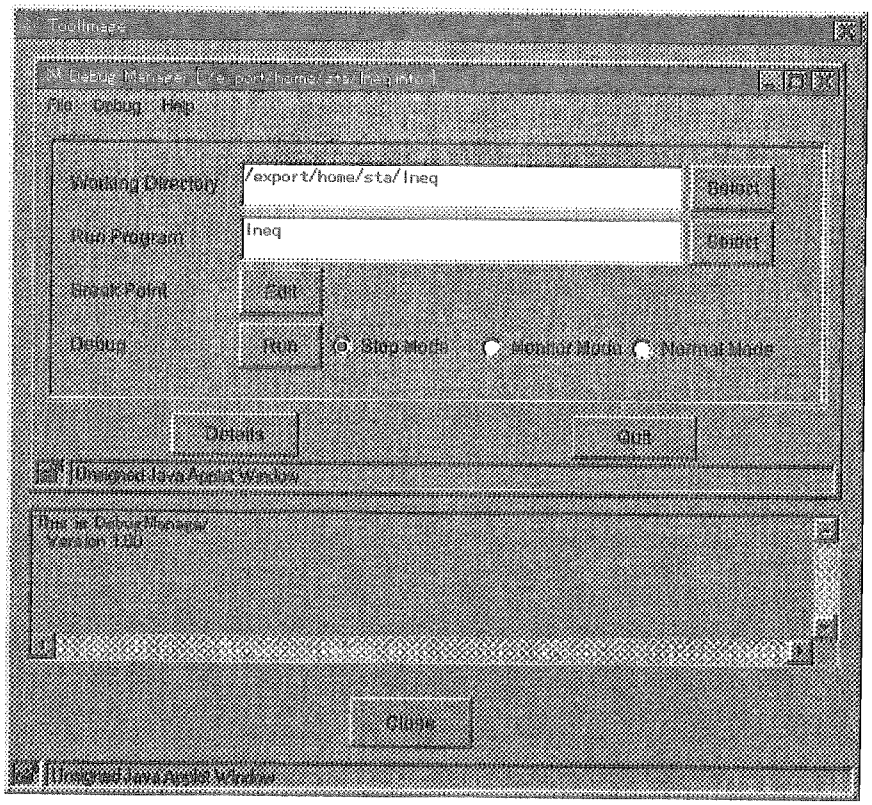

図 14: STA アプリケーション詳細情報表示画面

て行う。選択状態の STA アプリケーションは非選択状態の STA アプリケーションと異なる背景色で表示される。既に選択状態の STA アプリケーションを再びマウスで選択した場合、STA アプリケーションは非選択状態となる。

STA アプリケーションは，種類毎に分類されている．分類基準の名称をカテゴリと呼ぶ．

カテゴリには複数の STA アプリケーションが存在することがあり、複数の STA アプリケーションを共に選択状態とすることができる。この場合、メイン画面上でツール起動ボタンを押下すると，ポップアップメニューが表示され、希望する STA アプリケーションを起動することができる．

Detail ボタン Detail ボタンをクリックすることで、STA アプリケーションの詳細情報を参照できる．Detail ボタンをクリックするとマウスカーソルが指形になる．この状態で STA アプリケーションをクリックすると、その STA アプリケーションの詳細情報が表示される．(図 14)。

6.7.2 STA アプリケーションの起動

STA アプリケーションはメイン画面に表示されているカテゴリボタンをクリックすることによって起動される．

あるカテゴリにおいてSTAアプリケーションが1つだけ利用登録されている場合は、選択したSTAアプリケーションが直ちに起動される．複数のSTAアプリケーションが利用登録されている場合は、カテゴリボタンを押下すると起動できるSTAアプリケーションのリストを表示するポップアップメニューが表示されるので、希望するSTAアプリケーションを選択する．

6.8 Help画面

STA基本システムメイン画面(図11)のHelpボタンをクリックすることによって、STA基本システムのヘルプを参照することができる．

6.9 STA基本システムの終了

STA基本システムを終了する場合は、STA基本システムメイン画面(図11)のExitボタンをクリックする．

STA基本システムを終了すると、STA基本システムログイン画面(図10)に戻る．

7 おわりに

異機種並列計算機間の通信を支援するライブラリ Starpc について説明するとともに，Starpc を用いたアプリケーションの構築手法，及び利用方法について述べた．Starpc は現時点で 11 機種の並列計算機上で動作が確認されている．また，エディタ，コンパイラ，チューナ等のプログラム開発支援ツール群，あるいはスケジューラ，プログラム実行マネジャ等のプログラム実行支援ツール群等，多数のツールが Starpc を利用して開発されている．

ネットワーク通信速度の向上，多様なアーキテクチャを持った計算機の出現により，ネットワークを介して複数の計算機上のツールを連携させて効率的な処理を行うことは，今後一層一般的になっていくと考えられる．Starpc がそのような処理形態の実現に資することができれば幸いである．

謝辞

本報告書をまとめるにあたり，日本原子力研究所 計算科学技術推進センター 並列処理基本システム開発グループの平山グループリーダには細部にわたり数多くの貴重なご助言をいただいた．ここに深謝いたします．

参考文献

[1] 例えば，W. スティーブンス：UNIX ネットワークプログラミング，トッパン (1992)

[2] I. Foster, J.Geisler, C.Kesselman, S.Tuecke : Managing Multiple Communication Methods in Highperformance Networked Computing Systems, Journal of Parallel and Distributed Computing, Vol.40, pp.35-48 (1997)

付録 A 利用可能プラットフォーム

平成12年1月現在において，Starpcライブラリの動作が確認された計算機を以下に示す.

A.1 C言語用Starpcライブラリ

(1) 日本原子力研究所 小型並列計算機
(2) 株式会社日立製作所 SR2201
(3) 富士通株式会社 VPP300
(4) 日本電気株式会社 SX-4
(5) IBM Corp. SP2
(6) Cray Inc. T90
(7) 富士通株式会社 VPP500
(8) 富士通株式会社 AP3000
(9) Intel Paragon
(10) Sun Micro Systems SparcStation及びSparcServer
(11) Dec Alpha Workstation

A.2 Java言語用Starpcライブラリ

(1) NetscapeCommunicator V4.05以降 (ただし，JDK1.1.5以降が動作すること) の動作する任意のWorkstation.
(2) NetscapeCommunicator V4.05以降 (ただし，JDK1.1.5以降が動作すること) の動作する任意のWindowsPC.

A.3 GUI管理サブシステム

Starpcによる通信を中継するGUI管理サブシステムは，以下のプラットフォームで動作している.

(1) OSがSolaris 2.5.1以降であるSun Micro Systems SparcStation及びSparcServer.
(2) 日本原子力研究所 小型並列計算機
(3) 株式会社日立製作所 SR2201

国際単位系（SI）と換算表

表1　SI基本単位および補助単位

量	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd
平面角	ラジアン	rad
立体角	ステラジアン	sr

表2　SIと併用される単位

名称	記号
分，時，日	min，h，d
度，分，秒	°，′，″
リットル	l，L
トン	t
電子ボルト	eV
原子質量単位	u

1 eV=1.60218×10^{-19}J

1 u=1.66054×10^{-27}kg

表5　SI接頭語

倍数	接頭語	記号
10^{18}	エクサ	E
10^{15}	ペタ	P
10^{12}	テラ	T
10^{9}	ギガ	G
10^{6}	メガ	M
10^{3}	キロ	k
10^{2}	ヘクト	h
10^{1}	デカ	da
10^{-1}	デシ	d
10^{-2}	センチ	c
10^{-3}	ミリ	m
10^{-6}	マイクロ	μ
10^{-9}	ナノ	n
10^{-12}	ピコ	p
10^{-15}	フェムト	f
10^{-18}	アト	a

表3　固有の名称をもつSI組立単位

量	名称	記号	他のSI単位による表現
周波数	ヘルツ	Hz	s^{-1}
力	ニュートン	N	$m\cdot kg/s^2$
圧力，応力	パスカル	Pa	N/m^2
エネルギー，仕事，熱量	ジュール	J	N·m
工率，放射束	ワット	W	J/s
電気量，電荷	クーロン	C	A·s
電位，電圧，起電力	ボルト	V	W/A
静電容量	ファラド	F	C/V
電気抵抗	オーム	Ω	V/A
コンダクタンス	ジーメンス	S	A/V
磁束	ウェーバ	Wb	V·s
磁束密度	テスラ	T	Wb/m^2
インダクタンス	ヘンリー	H	Wb/A
セルシウス温度	セルシウス度	℃	
光束	ルーメン	lm	cd·sr
照度	ルクス	lx	lm/m^2
放射能	ベクレル	Bq	s^{-1}
吸収線量	グレイ	Gy	J/kg
線量等量	シーベルト	Sv	J/kg

表4　SIと共に暫定的に維持される単位

名称	記号
オングストローム	Å
バーン	b
バール	bar
ガル	Gal
キュリー	Ci
レントゲン	R
ラド	rad
レム	rem

1 Å=0.1nm=10^{-10}m

1 b=100fm^2=10^{-28}m^2

1 bar=0.1MPa=10^{5}Pa

1 Gal=1cm/s^2=10^{-2}m/s^2

1 Ci=3.7×10^{10}Bq

1 R=2.58×10^{-4}C/kg

1 rad=1cGy=10^{-2}Gy

1 rem=1cSv=10^{-2}Sv

(注)

1．表1～5は「国際単位系」第5版，国際度量衡局1985年刊行による。ただし，1 eVおよび1 uの値はCODATAの1986年推奨値によった。

2．表4には海里，ノット，アール，ヘクタールも含まれているが日常の単位なのでここでは省略した。

3．barは，JISでは流体の圧力を表わす場合に限り表2のカテゴリーに分類されている。

4．EC関係理事会指令ではbar，barnおよび「血圧の単位」mmHgを表2のカテゴリーに入れている。

換算表

力	N(=10^{5}dyn)	kgf	lbf
	1	0.101972	0.224809
	9.80665	1	2.20462
	4.44822	0.453592	1

粘度　1 Pa·s(N·s/m^2)=10 P(ポアズ)(g/(cm·s))

動粘度　1m^2/s=10^{4}St(ストークス)(cm^2/s)

圧力	MPa(=10bar)	kgf/cm^2	atm	mmHg(Torr)	lbf/in^2(psi)
	1	10.1972	9.86923	7.50062×10^3	145.038
	0.0980665	1	0.967841	735.559	14.2233
	0.101325	1.03323	1	760	14.6959
	1.33322×10^{-4}	1.35951×10^{-3}	1.31579×10^{-3}	1	1.93368×10^{-2}
	6.89476×10^{-3}	7.03070×10^{-2}	6.80460×10^{-2}	51.7149	1

エネルギー・仕事・熱量	J(=10^7 erg)	kgf·m	kW·h	cal(計量法)	Btu	ft·lbf	eV
	1	0.101972	2.77778×10^{-7}	0.238889	9.47813×10^{-4}	0.737562	6.24150×10^{18}
	9.80665	1	2.72407×10^{-6}	2.34270	9.29487×10^{-3}	7.23301	6.12082×10^{19}
	3.6×10^6	3.67098×10^5	1	8.59999×10^5	3412.13	2.65522×10^6	2.24694×10^{25}
	4.18605	0.426858	1.16279×10^{-6}	1	3.96759×10^{-3}	3.08747	2.61272×10^{19}
	1055.06	107.586	2.93072×10^{-4}	252.042	1	778.172	6.58515×10^{21}
	1.35582	0.138255	3.76616×10^{-7}	0.323890	1.28506×10^{-3}	1	8.46233×10^{18}
	1.60218×10^{-19}	1.63377×10^{-20}	4.45050×10^{-26}	3.82743×10^{-20}	1.51857×10^{-22}	1.18171×10^{-19}	1

1 cal= 4.18605J（計量法）

= 4.184J（熱化学）

= 4.1855J（15℃）

= 4.1868J（国際蒸気表）

仕事率　1 PS(仏馬力)

= 75 kgf·m/s

= 735.499W

放射能	Bq	Ci
	1	2.70270×10^{-11}
	3.7×10^{10}	1

吸収線量	Gy	rad
	1	100
	0.01	1

照射線量	C/kg	R
	1	3876
	2.58×10^{-4}	1

線量当量	Sv	rem
	1	100
	0.01	1

(86年12月26日現在)

並列計算機クラスタ上のツール間通信を支援するライブラリ：Starpc
—Starpc利用及び開発手引書—