次世代ロボット知能化技術開発プロジェクト

対話フロー制御コンポーネント
外部仕様書

２０１１年　２月　(Ver. E)

株式会社国際電気通信基礎技術研究所
知能ロボティクス研究所

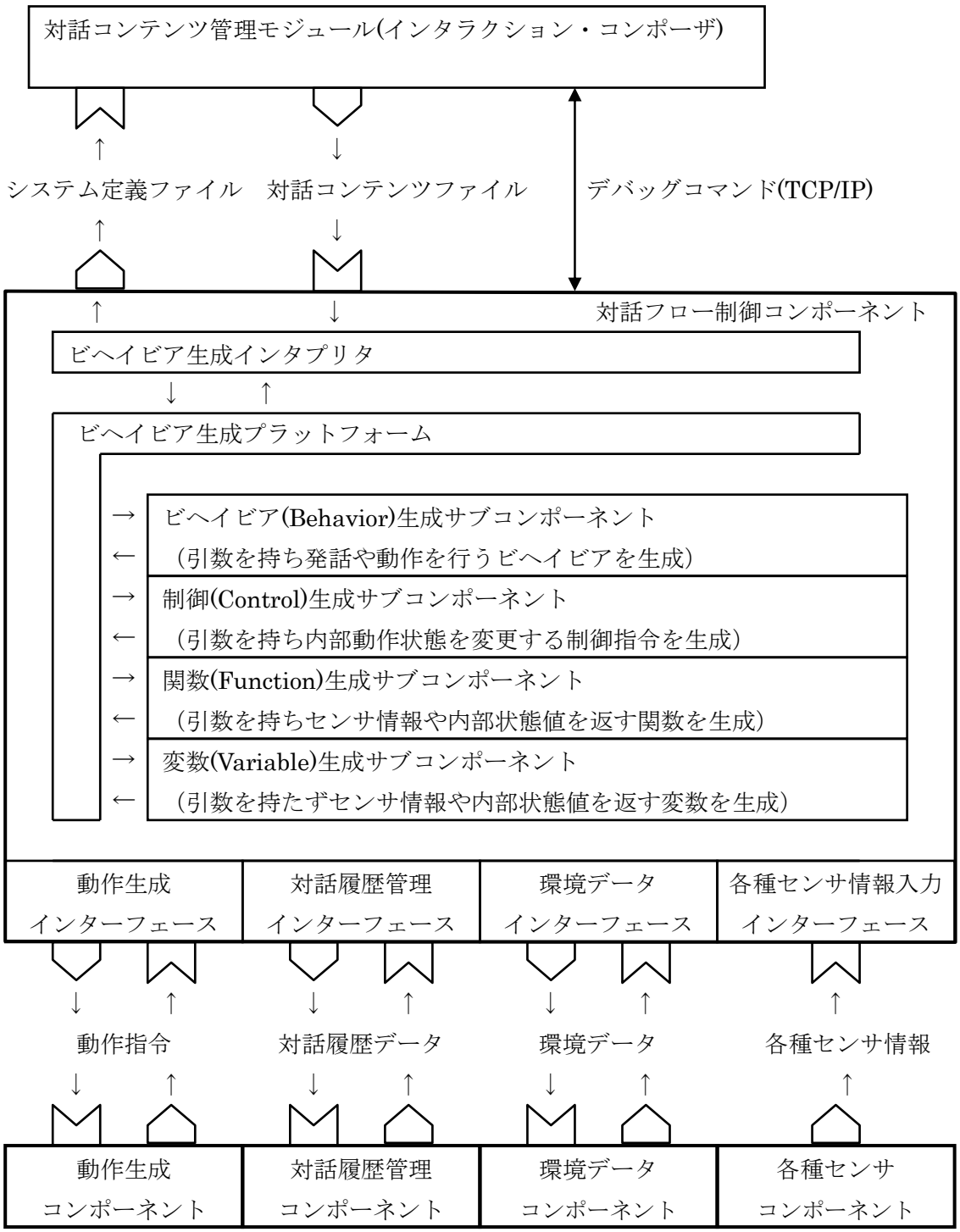
・改訂履歴

Ver.	改訂日付	改訂内容	作成
NC	2008-07-31	初版	岩崎
A	2009-03-31	機能追加	岩崎
B	2009-06-30	機能追加	岩崎
C	2010-01-08	機能追加	岩崎
D	2011-01-18	機能追加	岩崎
E	2011-02-21	誤記訂正	岩崎

目次

1. 対話フロー制御コンポーネントの目的

対話フロー制御コンポーネントは、対話コンテンツ管理モジュール(インタラクション・コンポーザ)からの対話コンテンツ中間ファイルを入力とし、ファイルに格納されているコンテンツ・コマンドを解析実行してビヘイビアを生成し、下位層の動作生成コンポーネントに対して一連の動作指令スクリプトを送る働きをする。



(1) ビヘイビア生成インタプリタ

対話コンテンツ管理モジュール(インタラクション・コンポーザ)より入力される対話コンテンツファイルには、一連の動作スクリプト（プログラム）が格納されている。このスクリプトを解釈実行し、条件判断やスケジュール管理、ビヘイビア生成関数を呼び出す働きを行う。

(2)ビヘイビア生成サブコンポーネント

引数を持ち発話や動作を行うビヘイビアを生成する実行関数群であり、タスクに応じて追加作成が可能。4.1 項参照。

(3) 制御(Control)生成サブコンポーネント

引数を持ち内部動作状態を変更する制御指令を生成する実行関数群であり、タスクに応じて追加作成が可能。4.2 項参照。

(4) 関数(Function)生成サブコンポーネント

引数を持ちセンサ情報や内部状態により動的に値が変化する関数を生成する実行関数群であり、タスクに応じて追加作成が可能。4.3 項参照。

(5) 変数(Variable)生成サブコンポーネント

引数を持たずセンサ情報や内部状態により動的に値が変化する変数を生成する実行関数群であり、タスクに応じて追加作成が可能。4.4 項参照。

(6)動作生成インターフェース

ビヘイビア生成関数や制御生成関数より呼び出され、動作生成コンポーネントに対するコマンド（動作指令スクリプト）を生成し、応答を受け取る処理を行う。5.1 項参照。

(7)対話履歴管理インターフェース

ビヘイビア生成関数や制御生成関数や関数生成関数より呼び出され、対話履歴管理コンポーネントに対するコマンドを生成し、応答を受け取る処理を行う。

(8)環境データインターフェース

人位置やロボット位置、顔認識結果や音声認識結果などの環境データを受け取る。

(9)各種センサ情報入力インターフェース（オプション機能）

各種センサ情報を入力する。

2. プログラム仕様

2.1 動作環境

・使用 OS： WindowsXP

・Visual C++ 2008 Express Edition
・RT ミドルウェア(OpenRTM-aist-1.0.0)

2.2 プログラムファイル構成

```
BehaviorGenModule                    ： 実行形式の RTM コンポーネント
 ├ BehaviorGenModule_vc9.sln      ：VC++2008 用設定ファイル
 ├ BehaviorGenModule_vc9.vcproj   ：VC++2008 用設定ファイル
 ├ BehaviorGenModuleComp_vc9.vcproj  ：VC++2008 用設定ファイル
 ├ rtm_config.vsprops             ：VC++2008 用設定ファイル
 ├ user_config.vsprops            ：VC++2008 用設定ファイル
 │
 ├ BehaviorGenBase.lib            ：BehaviorGenBase で作成されたライブラリファイル
 ├ BehaviorGenBaseComp.lib        ：BehaviorGenBase で作成されたライブラリファイル
 │
 ├ behaviorGeneratorApi.h         ：API 部定義
 │
 ├ bhvr/                          ： ビヘイビア生成プログラムフォルダ
 │   ├ bhvr Submodule.cpp         ：ビヘイビア生成サブコンポーネント共通部
 │   ├ bhvr Method1.cpp           ：ビヘイビア生成メソッド 1
 │   ├ bhvr Method2.cpp           ：ビヘイビア生成メソッド 2
 │   ├         :
 │   └ bhvr Submodule.h           ：ビヘイビアのクラス定義
 │
 ├ cont/                          ： 制御生成プログラムフォルダ
 │   ├ contSubmodule.cpp          ： 制御生成サブコンポーネント共通部
 │   ├ contMethod1.cpp            ： 制御生成メソッド 1
 │   ├ contMethod2.cpp            ： 制御生成メソッド 2
 │   ├         :
 │   └ contSubmodule.h            ： 制御生成のクラス定義
 │
 ├ func/                          ： 関数生成プログラムフォルダ
 │   ├ funcSubmodule.cpp          ： 関数生成サブコンポーネント共通部
 │   ├ funcMethod1.cpp            ： 関数生成メソッド 1
 │   ├ funcMethod2.cpp            ： 関数生成メソッド 2
 │   ├         :
 │   └ funcSubmodule.h            ： 関数生成のクラス定義
 │
 └var/                            ： 静的変数生成プログラムフォルダ
     ├ varSubmodule.cpp           ： 静的変数生成サブコンポーネント共通部
```

```
├ varMethod1.cpp          :: 静的変数生成メソッド1
├ varMethod2.cpp          :: 静的変数生成メソッド2
├          :
└ varSubmodule.h          :: 静的変数生成のクラス定義
```

2.3 データファイル

(1) 入力ファイル

rtc.conf - RTM 設定ファイル

project.icp - インタラクション・コンポーザから出力された中間ファイル。
漢字コード：UTF-8N

project.lab - インタラクション・コンポーザから出力されたラベルリスト。
漢字コード：UTF-8N

project.lst - インタラクション・コンポーザから出力されたビヘイビアリスト。
漢字コード：UTF-8N

(2) 出力ファイル

system-defs.xml - ビヘイビア生成部のシステム定義ファイル。
BehaviorGenModule 起動時の引数に "/s" を指定すると作成される。
漢字コード：UTF-8N

2.4 プログラムの使用方法

(1) BehaviorGenModule のあるディレクトリで、Windows のコマンドプロンプトを開く。

※コマンドプロンプトの推奨プロパティ設定
現在のコードページ：日本語 Shift-JIS
フォント：MS ゴシック

(2) UTF-8N 漢字コードでモニタする場合、コマンドプロンプトにて、"chcp 65001"と入力する。

(3) " BehaviorGenModuleComp"で起動する。
システム定義ファイルを出力させる場合は、"BehaviorGenModuleComp /s"で起動する。
インタラクション・リモート・コントローラ(IRC)に接続して起動させる場合は、
"BehaviorGenModuleComp /m IRC のホスト名（IP アドレス）"
で起動する。

(4)RTSystemEditor で接続する。

(5)RTSystemEditor から Activate で、対話コンテンツファイルを読み込んで実行開始。
インタラクション・コンポーザに接続する場合は、インタラクション・コンポーザの
デバッグ開始で、対話コンテンツファイルを読み込んで実行開始。

(6)RTSystemEditor から Deactivate で実行終了。

(7)対話コンテンツファイルを再び読み込んで実行するには、(5)へ。

(8) RTSystemEditor で Dectivate して Exit した後、Ctrl-C で終了する。

3. 外部インターフェース

(1)動作指令スクリプト出力ポート

ポート名：BG_BOSout
型：OutPort TimedString×1
接続先：動作生成コンポーネント
漢字コード：UTF-8N

(2)動作指令スクリプト応答入力ポート

ポート名：BG_BOSRin
型：InPort TimedString×1
接続先：動作生成コンポーネント

(3)対話履歴管理コマンド出力ポート

ポート名：BG_HMCout
型：OutPort TimedString×1
接続先：対話履歴管理コンポーネント
漢字コード：UTF-8N

(4)対話履歴管理コマンド応答入力ポート

ポート名：BG_HMCRin
型：InPort TimedString×1
接続先：対話履歴管理コンポーネント
漢字コード：UTF-8N

(5)環境データコマンド出力ポート

ポート名：BG_EDCout
型：OutPort TimedString×1
接続先：環境情報コンポーネント
漢字コード：UTF-8N

(6)環境データコマンド応答入力ポート

ポート名：BG_EDCRin
型：InPort TimedString×1
接続先：環境情報コンポーネント
漢字コード：UTF-8N

(7)デバッガ専用 TCP/IP ソケット

接続先：インタラクション・コンポーザのデバッガ接続用ポート
接続待ちポート番号：5555 (サーバ接続)

漢字コード：UTF-8N

(8)インタラクション・リモート・コントローラ用 TCP/IP ソケット
接続先：インタラクション・リモート・コントローラ接続用ポート
接続先ポート番号：5571 (クライアント接続)
漢字コード：UTF-8N

4. メソッド関数の形式

ビヘイビア生成などを実行するメソッド関数には、以下の 4 種類がある。

(1)ビヘイビア(Behavior)生成メソッド
引数を持ち、主に発話や動作を行うビヘイビアを生成するメソッド。

(2)制御(Control)生成メソッド
引数を持ち、主に内部動作状態を変更する制御指令を生成するメソッド。

(3)関数(Function)生成メソッド
引数を持ち、主にセンサ情報や内部状態値を返す関数を生成するメソッド。

(4)静的変数(Variable)生成メソッド
引数を持たず、主にセンサ情報や内部状態値を返す静的変数を生成するメソッド。
注：ユーザ定義の静的変数以外の変数として、ビヘイビア生成インタプリタ動作中に分岐(Branch)コマンドなどで自動的に生成される動的変数がある。

4.1 ビヘイビア生成メソッド

4.1.1 ビヘイビア生成メソッド関数の一般形式

behaviorID に関連付けられたビヘイビア生成関数が呼び出される。
・関数定義：int bhvr メソッド名(int inmode, string args[], int argc);
・関数引数入力：inmode - 関数の呼び出しモード。
BG_INIT(0) - プログラム初期化
BG_START(1) - ビヘイビア開始
BG_RECALL(2) - リコール
BG_INTRPT(3) - 割り込み発生
BG_EXIT(5) - プログラム終了
args - 文字列引数配列。
argc - 文字列引数の数。
・関数引数出力：なし。
・関数戻り値：inmode 入力が BG_INIT, BG_INTRPT, BG_EXIT のとき

BG_OK(0)　　- 正常終了。
BG_ERR(-1)　　- 異常終了。

inmode 入力が BG_START, BG_RECALL のとき
BG_OK(0)　　- 正常終了。分岐出力ポート 1 に分岐。
BG_OK+1(1)　　- 正常終了。分岐出力ポート 2 に分岐。
　:　　:
BG_OK+n-1(n-1) - 正常終了。分岐出力ポート n に分岐。
BG_ERR(-1)　　- 異常終了。
BG_REQREC(-2)　- リコール要求終了。処理の途中なので、再呼び出しが必要。

・補足：1)メソッド関数は、自分自身の中で待ち状態になってはならない。
応答待ちなどが必要な場合は、一旦 BG_REQREC の戻り値で関数を終了し、
一定時間(標準 10ms)後、inmode=BG_RECALL で、再び呼び出されるようにすること。

2)inmode=BG_INIT で呼び出された際には、メソッドパラメータをシステム登録する関数呼び出しを行うこと。(4.1.2 参照)

3)inmode=BG_INTRPT で呼び出された時は、リコール要求終了はできない。
また、動作生成モジュールへのスクリプト出力は、動作が保証されないので、するべきでない。
必要最小限の内部処理（内部フラグのセット等）で復帰する必要がある。

4)メソッド関数を作成したら、システムに登録するため、bhvrSubmodule.cpp と bhvrSubmodule.h に登録情報を追加して再ビルドすること。(4.1.3 参照)

4.1.2 ビヘイビア生成メソッドのパラメータ

メソッドパラメータの書式は、以下の通りである。

```
--- ここから ----------
const char *mparm[] =
  {
    "ビヘイビア名",  // behaviorID
    "ビヘイビアのグループ&#9;Behavior Group",  // group
    "ビヘイビアのタイトル&#9;Behavior Title",  // title
    "ビヘイビアのコメント&#9;Behavior description",  // behavior description
    "parameter type=¥"入力引数型 1¥" description=¥"入力引数 1 の説明&#9;Parameter 1 explanation¥"",  // parameter 1
    "parameter type=¥"入力引数型 2¥" description=¥"入力引数 2 の説明&#9;Parameter 2 explanation¥" history=¥"入力履歴名¥"",  // parameter 2
        :
    "parameter type=¥"入力引数型 n¥" description=¥"入力引数 n の説明&#9;Parameter n explanation¥"",  // parameter n
```

```
    "branch name=¥"出力分岐名 1&#9;Branch Name 1¥"",   // branch 1
    "branch name=¥"出力分岐名 2&#9;Branch Name 2¥"",   // branch 2
          :
    "branch name=¥"出力分岐名 n&#9;Branch Name n¥"",   // branch n
    ""   // end of data
  };
--- ここまで ----------
```

メソッドパラメータは、inmode 入力が BG_INIT のとき、

registMethodParm(mparm, KANJI_CODE_SJIS);　// mparm の使用漢字コードが SJIS

registMethodParm(mparm, KANJI_CODE_EUC);　// mparm の使用漢字コードが EUC

registMethodParm(mparm, KANJI_CODE_UTF8); // mparm の使用漢字コードが UTF-8N

のいずれかの関数呼び出しによりシステムに設定する。

メソッドパラメータに漢字を使用していない時は、KANJI_CODE_UTF8 を指定しておく。

ビヘイビア名は英数字および半角記号が使用できる。

ビヘイビアのグループ、タイトルおよびコメントには、漢字も使用できる。

ビヘイビアのグループを指定しない時は、"-"と指定する。

出力分岐名は、英数字および半角記号を推奨するが、漢字も使用できる。

inmode 入力が BG_START, BG_RECALL のときのメソッド関数の戻り値により

関数戻り値	インタプリタの動作
BG_OK(=0)	- メソッド正常終了。分岐出力ポート 1 に分岐。
BG_OK+1(=1)	- メソッド正常終了。分岐出力ポート 2 に分岐。
:	:
BG_OK+n-1(=n-1)	- メソッド正常終了。分岐出力ポート n に分岐。

となる。

入力引数型は、以下のものが指定できる。

"Int"	- 整数型(定数と変数を選択可)
"ConstInt"	- 整数型定数
"VarInt"	- 整数型変数
"Float"	- 浮動小数点数型(定数と変数を選択可)
"ConstFloat"	- 浮動小数点数型定数
"VarFloat"	- 浮動小数点数型変数
"String"	- 文字列型(定数と変数を選択可)
"ConstString"	- 文字列型定数
"VarString"	- 文字列型変数

"String"　　- 文字列型(定数と変数を選択可)
"ConstString"　- 文字列型定数
"VarString"　　- 文字列型変数
"Script"　　- 動作生成スクリプト(文字列定数)
"Bool"　　- 論理型(true/false)(定数と変数を選択可)
"ConstBool"　　- 論理型定数
"VarBool"　　- 論理型変数
"Enum(A,B,C)"　- 列挙型定数

Script 型は、実体は文字列型定数である。
Bool 型は、実体は整数型であり、true=1, false=0 に対応する。
Enum 型は、カッコ内に指定した列挙定数リストが、整数型定数の 0,1,2～に対応する。
history を指定すると、インタラクション・コンポーザでの引数入力時に、その入力が指定した入力履歴名毎に保存され、次回の引数入力時に過去の入力履歴を参照できる。

4.1.3 ビヘイビア生成メソッドのシステム登録

(1)bhvrSubmodule.cpp の指定場所に、以下の行を追加する。

```
registBhvrMethod(BhvrSubmodule::メソッド関数名);
```

(2)bhvrSubmodule.h の指定場所に、以下の行を追加する。

```
static int メソッド関数名(int inmode, std::string args[], int argc);
```

(3)コンパイラのプロジェクトに、追加したビヘイビア生成メソッドのファイルを追加してビルドする。
(Visual C++の場合、ビルドは、Release モードで行うこと。)

4.2 制御生成メソッド

4.2.1 制御生成メソッド関数の一般形式

controlID に関連付けられた制御生成関数が呼び出される。

・関数定義：int cont メソッド名(int inmode, string args[], int argc);
・関数引数入力：inmode - 関数の呼び出しモード。

BG_INIT(0)　- プログラム初期化
BG_START(1)　- 制御開始
BG_RECALL(2) - リコール
BG_INTRPT(3) - 割り込み発生
BG_EXIT(5)　- プログラム終了

args - 文字列引数配列。
argc - 文字列引数の数。

・関数引数出力：なし。

・関数戻り値：inmode 入力が BG_INIT, BG_INTRPT, BG_EXIT のとき

BG_OK(0)　　- 正常終了。

BG_ERR(-1)　　- 異常終了。

inmode 入力が BG_START, BG_RECALL のとき

BG_OK(0)　　- 正常終了。分岐出力ポート 1 に分岐。

BG_OK+1(1)　　- 正常終了。分岐出力ポート 2 に分岐。

：　　：

BG_OK+n-1(n-1) - 正常終了。分岐出力ポート n に分岐。

BG_ERR(-1)　　- 異常終了。

BG_REQREC(-2)　- リコール要求終了。処理の途中なので、再呼び出しが必要。

・補足：1)メソッド関数は、自分自身の中で待ち状態になってはならない。

応答待ちなどが必要な場合は、一旦 BG_REQREC の戻り値で関数を終了し、

一定時間(標準 10ms)後、inmode=BG_RECALL で、再び呼び出されるようにすること。

2)inmode=BG_INIT で呼び出された際には、メソッドパラメータをシステム登録する関数呼び出しを行うこと。(4.2.2 参照)

3)inmode=BG_INTRPT で呼び出された時は、リコール要求終了はできない。

また、動作生成モジュールへのスクリプト出力は、動作が保証されないので、

するべきでない。

必要最小限の内部処理（内部フラグのセット等）で復帰する必要がある。

4)メソッド関数を作成したら、システムに登録するため、contSubmodule.cpp と contSubmodule.h に登録情報を追加して再ビルドすること。(4.2.3 参照)

4.2.2 制御生成メソッドのパラメータ

メソッドパラメータの書式は、以下の通りである。

--- ここから ----------

```
const char *mparm[] =
  {
    "制御名",  // controlID
    "制御のグループ&#9;Control Group",  // group
    "制御のタイトル&#9;Control Title",  // title
    "制御のコメント&#9;Control description",  // description
    "parameter type=¥"入力引数型 1¥" description=¥"入力引数 1 の説明&#9;Parameter 1 explanation¥"",  // parameter 1
    "parameter type=¥"入力引数型 2¥" description=¥"入力引数 2 の説明&#9;Parameter 2 explanation¥" history=¥"入力履歴名¥"",  // parameter 2
          :
```

```
        "parameter type=¥"入力引数型 n¥" description=¥"入力引数 n の説明&#9;Parameter n explanation¥"",  // parameter n
        "branch name=¥"出力分岐名 1&#9;Branch Name 1¥"",  // branch 1
        "branch name=¥"出力分岐名 2&#9;Branch Name 2¥"",  // branch 2
              :
        "branch name=¥"出力分岐名 n&#9;Branch Name n¥"",  // branch n
        ""  // end of data
    };
--- ここまで ----------
```

メソッドパラメータは、inmode 入力が BG_INIT のとき、

```
registMethodParm(mparm, KANJI_CODE_SJIS);  // mparm の使用漢字コードが SJIS
registMethodParm(mparm, KANJI_CODE_EUC);  // mparm の使用漢字コードが EUC
registMethodParm(mparm, KANJI_CODE_UTF8); // mparm の使用漢字コードが UTF-8N
```

のいずれかの関数呼び出しによりシステムに設定する。
メソッドパラメータに漢字を使用していない時は、KANJI_CODE_UTF8 を指定しておく。

制御名は英数字および半角記号が使用できる。
制御のグループ、タイトルおよびコメントには、漢字も使用できる。
制御のグループを指定しない時は、"-"と指定する。

出力分岐名は、英数字および半角記号を推奨するが、漢字も使用できる。
inmode 入力が BG_START, BG_RECALL のときのメソッド関数の戻り値により

関数戻り値	インタプリタの動作
BG_OK(=0)	- メソッド正常終了。分岐出力ポート 1 に分岐。
BG_OK+1(=1)	- メソッド正常終了。分岐出力ポート 2 に分岐。
:	:
BG_OK+n-1(=n-1)	- メソッド正常終了。分岐出力ポート n に分岐。

となる。

入力引数型は、以下のものが指定できる。

"Int"	- 整数型(定数と変数を選択可)
"ConstInt"	- 整数型定数
"VarInt"	- 整数型変数
"Float"	- 浮動小数点数型(定数と変数を選択可)
"ConstFloat"	- 浮動小数点数型定数
"VarFloat"	- 浮動小数点数型変数
"String"	- 文字列型(定数と変数を選択可)

"ConstString" - 文字列型定数
"VarString" - 文字列型変数
"Script" - 動作生成スクリプト(文字列定数)
"Bool" - 論理型(true/false)(定数と変数を選択可)
"ConstBool" - 論理型定数
"VarBool" - 論理型変数
"Enum(A,B,C)" - 列挙型定数

Script 型は、実体は文字列型定数である。
Bool 型は、実体は整数型であり、true=1, false=0 に対応する。
Enum 型は、カッコ内に指定した列挙定数リストが、整数型定数の 0,1,2～に対応する。
history を指定すると、インタラクション・コンポーザでの引数入力時に、その入力が指定した入力履歴名毎に保存され、次回の引数入力時に過去の入力履歴を参照できる。

4.2.3 制御生成メソッドのシステム登録

(1)contSubmodule.cpp の指定場所に、以下の行を追加する。

```
registContMethod(ContSubmodule::メソッド関数名);
```

(2)contSubmodule.h の指定場所に、以下の行を追加する。

```
static int メソッド関数名(int inmode, std::string args[], int argc);
```

(3)コンパイラのプロジェクトに、追加した制御生成メソッドのファイルを追加してビルドする。
(Visual C++の場合、ビルドは、Release モードで行うこと。)

4.3 関数生成メソッド

4.3.1 関数生成メソッド関数の一般形式

functionID 名に関連付けられた関数生成関数が呼び出される。

・関数定義：int func メソッド名(int inmode, string args[], int argc, string& ret);
・関数引数入力：inmode - 関数の呼び出しモード。
BG_INIT(0) - プログラム初期化
BG_FNCGET(1) - 関数値取得
BG_FNCMON(3) - 関数モニタ値取得
BG_EXIT(5) - プログラム終了
args - 文字列引数配列。
argc - 文字列引数の数。
・関数引数出力：ret - 関数の戻り値。
・関数戻り値：リターンコード

BG_OK(0)　　- 正常終了。
BG_ERR(-1)　- 異常終了。

・補足：1)メソッド関数は、自分自身の中で待ち状態になってはならない。
2)inmode=BG_INIT で呼び出された際には、メソッドパラメータをシステム登録する関数呼び出しを行うこと。(4.3.2 参照)
3)メソッド関数を作成したら、システムに登録するため、funcSubmodule.cpp と funcSubmodule.h に登録情報を追加して再ビルドすること。(4.3.3 参照)

4.3.2 関数生成メソッドのパラメータ

メソッドパラメータの書式は、以下の通りである。

```
--- ここから ----------------------------
const char *mparm[] =
  {
    "関数名",   // functionID
    "関数の戻り値型",   // type
    "関数のタイトル&#9;Function Title",   // title
    "関数のコメント&#9;Function description",   // description
    "parameter type=¥"入力引数型 1¥" description=¥"入力引数 1 の説明&#9;Parameter 1 explanation¥"",   // parameter 1
    "parameter type=¥"入力引数型 2¥" description=¥"入力引数 2 の説明&#9;Parameter 2 explanation¥" history=¥"入力履歴名¥"",   // parameter 2
          :
    "parameter type=¥"入力引数型 n¥" description=¥"入力引数 n の説明&#9;Parameter n explanation¥"",   // parameter n
    ""   // end of data
  };
--- ここまで ----------------------------
```

メソッドパラメータは、inmode 入力が BG_INIT のとき、
registMethodParm(mparm, KANJI_CODE_SJIS);　// mparm の使用漢字コードが SJIS
registMethodParm(mparm, KANJI_CODE_EUC);　// mparm の使用漢字コードが EUC
registMethodParm(mparm, KANJI_CODE_UTF8); // mparm の使用漢字コードが UTF-8N
のいずれかの関数呼び出しによりシステムに設定する。
メソッドパラメータに漢字を使用していない時は、KANJI_CODE_UTF8 を指定しておく。

関数名は英数字および半角記号が使用できる。
関数の戻り値型は、Int, Float, String, Bool が指定できる。
関数のタイトルおよびコメントには、漢字が使用できる。

入力引数型は、以下のものが指定できる。

"Int" - 整数型(定数と変数を選択可)
"ConstInt" - 整数型定数
"VarInt" - 整数型変数
"Float" - 浮動小数点数型(定数と変数を選択可)
"ConstFloat" - 浮動小数点数型定数
"VarFloat" - 浮動小数点数型変数
"String" - 文字列型(定数と変数を選択可)
"ConstString" - 文字列型定数
"VarString" - 文字列型変数
"Script" - 動作生成スクリプト(文字列定数)
"Bool" - 論理型(true/false)(定数と変数を選択可)
"ConstBool" - 論理型定数
"VarBool" - 論理型変数
"Enum(A,B,C)" - 列挙型定数

Script 型は、実体は文字列型定数である。
Bool 型は、実体は整数型であり、true=1, false=0 に対応する。
Enum 型は、カッコ内に指定した列挙定数リストが、整数型定数の 0,1,2～に対応する。
history を指定すると、インタラクション・コンポーザでの引数入力時に、その入力が指定した入力履歴名毎に保存され、次回の引数入力時に過去の入力履歴を参照できる。

4.3.3 関数生成メソッドのシステム登録

(1)funcSubmodule.cpp の指定場所に、以下の行を追加する。

```
registFuncMethod(FuncSubmodule::メソッド関数名);
```

(2)funcSubmodule.h の指定場所に、以下の行を追加する。

```
static int メソッド関数名(int inmode, std::string args[], int argc, string& ret);
```

(3)コンパイラのプロジェクトに、追加した関数生成メソッドのファイルを追加してビルドする。
(Visual C++の場合、ビルドは、Release モードで行うこと。)

4.4 静的変数生成メソッド

4.4.1 静的変数生成メソッド関数の一般形式

変数名に関連付けられた静的変数生成関数が呼び出される。

・関数定義：int var メソッド名(int inmode, string& arg);
・関数引数入力：inmode - 関数の呼び出しモード。

BG_INIT(0)　- プログラム初期化
BG_VARGET(1) - 変数値取得
BG_VARSET(2) - 変数値設定(option)
BG_VARMON(3) - 変数モニタ値取得
BG_EXIT(5)　- プログラム終了

arg - 設定する変数値。(inmode 入力が BG_VARSET の時のみ)

・関数引数出力：arg - 取得した変数値。

・関数戻り値：リターンコード

BG_OK(0)　- 正常終了。
BG_ERR(-1)　- 異常終了。

・補足：1)メソッド関数は、自分自身の中で待ち状態になってはならない。
2)inmode=BG_INIT で呼び出された際には、メソッドパラメータをシステム登録する関数呼び出しを行うこと。(4.4.2 参照)
3)メソッド関数を作成したら、システムに登録するため、varSubmodule.cpp と varSubmodule.h に登録情報を追加して再ビルドすること。(4.4.3 参照)

4.4.2 静的変数生成メソッドのパラメータ

メソッドパラメータの書式は、以下の通りである。

```
--- ここから ----------
const char *mparm[] =
  {
    "静的変数名",  // variableID
    "静的変数の型",  // type
    "静的変数のタイトル&#9;Variable Title",  // title
    "静的変数のコメント&#9;Variable Comment",  // description
    ""  // end of data
  };
--- ここまで ----------
```

メソッドパラメータは、inmode 入力が BG_INIT のとき、
registMethodParm(mparm, KANJI_CODE_SJIS);　// mparm の使用漢字コードが SJIS
registMethodParm(mparm, KANJI_CODE_EUC);　// mparm の使用漢字コードが EUC
registMethodParm(mparm, KANJI_CODE_UTF8); // mparm の使用漢字コードが UTF-8N
のいずれかの関数呼び出しによりシステムに設定する。
メソッドパラメータに漢字を使用していない時は、KANJI_CODE_UTF8 を指定しておく。

静的変数名は英数字および半角記号が使用できる。
静的変数の型は、Int, Float, String, Bool が指定できる。
静的変数のタイトルおよびコメントには、漢字も使用できる。

4.4.3 静的変数生成メソッドのシステム登録

(1)varSubmodule.cpp の指定場所に、以下の行を追加する。

registVarMethod(VarSubmodule::メソッド関数名);

(2)varSubmodule.h の指定場所に、以下の行を追加する。

static int メソッド関数名(int inmode, std::string& arg);

(3)コンパイラのプロジェクトに、追加した変数生成メソッドのファイルを追加してビルドする。
(Visual C++の場合、ビルドは、Release モードで行うこと。)

5. メソッド関数のユーザ公開サンプルソースコードの概要

ユーザ自身でメソッド関数を作成する際の参考とするため、サンプルのソースコードを添付している。

Windows と Linux のソースコードを共存させるときは、

```
#ifdef LINUX
#else
#endif
```

を使用すること。

LINUX の make によるコンパイル時には、自動的に LINUX=1 が設定されるよう make に記述しておくこと。

5.1 ユーザ公開 API の概要

ユーザがビヘイビア／制御／関数／変数生成メソッド関数内で使用できる関数が、
API として準備されている。
API 関数の詳細については、behaviorGeneratorApi.h を参照のこと。

6. システム定義ファイルの概要

システム定義ファイル<system-defs.xml>は XML で記述されており、以下の構造を持つ。

```
--- ここから ----------------------------
<?xml version="1.0" encoding="utf-8"?>
<definitions>
  <types>
    データ型(type)定義要素 1
    データ型定義要素 2
      :
    データ型定義要素 n
```

```
    </types>
    <setup>
      プロジェクト初期設定パラメータ定義要素 1
      プロジェクト初期設定パラメータ定義要素 2
        :
      プロジェクト初期設定パラメータ定義要素 n
    </setup>      <behaviors>
      ビヘイビア(behavior)定義要素 1
      ビヘイビア定義要素 2
        :
      ビヘイビア定義要素 n
    </behaviors>
    <controls>
      制御(control)定義要素 1
      制御定義要素 2
        :
      制御定義要素 n
    </controls>
    <functions>
      関数(function)定義要素 1
      関数定義要素 2
        :
      関数定義要素 n
    </functions>
    <variables>
      静的変数(variable)定義要素 1
      静的変数定義要素 2
        :
      静的変数定義要素 n
    </variables>
  </definitions>
  --- ここまで ----------------------------
```

6.1 データ型定義要素

データ型(type)定義要素は、以下の構造を持つ。

```
  <type name="tname" description="tcomment">
    <operator OP="operator 1" />
    <operator OP="operator 2" />
```

```
                :
  <operator OP="operator n" />
</type>
```

属性：tname　　- データ型名（Int/Float/String/Script/Bool）
　　tcomment　- データ型のコメント
　　operator n - データ型の持つ演算子 n
補足：description 属性は、オプションである。
　　Enum 型は、Int 型と同じであるので、ここでは定義しない。

ここで定義されるデータ型とそれに対応する演算子は、ビヘイビアインタプリタ内部の実装に依存するため、ユーザによる追加・更新は出来ない。

6.2 プロジェクト初期設定パラメータ定義要素

プロジェクト初期設定パラメータ(parameter)定義要素は、以下の構造を持つ。

```
 <parameter name="pname" title="ptitle">
  <type name="tname 1" />
  <type name="tname 2" />
                :
  <type name="tname n" />
 </parameter>
```

属性：
　pname　　- プロジェクト初期設定パラメータ名
　ptitle　- プロジェクト初期設定パラメータのタイトル（バイリンガル対応）
　tname n　- パラメータのタイプ名
補足：バイリンガルに対応するには、"日本語文字列	EnglishText"と記述する。

6.3 ビヘイビア定義要素

ビヘイビア(behavior)定義要素は、以下の構造を持つ(4.1.2 項参照)。

```
<behavior behaviorID="bname" group="gname" title="btitle" description="bcomment">
  <parameter type="ptype 1" description="pcomment 1" />
  <parameter type="ptype 2" description="pcomment 2" />
                :
  <parameter type="ptype n" description="pcomment n" />
  <branch name="branch 1" />
  <branch name="branch 2" />
```

```
                :
  <branch name="branch n" />
</behavior>
```

属性：bname　- ビヘイビアのメソッド名
　　gname　- ビヘイビアのグループ（バイリンガル対応）
　　btitle　- ビヘイビアのタイトル（バイリンガル対応）
　　bcomment - ビヘイビアのコメント（バイリンガル対応）
　　ptype n　- 入力引数型 n (Int/ConstInt/VarInt/
　　　　Float/ConstFloat/VarFloat/
　　　　String/ConstString/VarString/
　　　　Script/Bool/ConstBool/VarBool/
　　　　Enum(A,B,…,N))
　　pcomment n　- 入力引数 n のコメント（バイリンガル対応）
　　branch n - 出力分岐名 n（バイリンガル対応）
補足：1)group 属性と description 属性は、オプションである。
　　2)バイリンガルに対応するには、"日本語文字列	EnglishText"と記述する。

6.4 制御定義要素

制御(control)定義要素は、以下の構造を持つ(4.2.2 項参照)。

```
<control controlID="cname" group="gname" title="ctitle" description="ccomment">
  <parameter type="ptype 1" description="pcomment 1" />
  <parameter type="ptype 2" description="pcomment 2" />
                :
  <parameter type="ptype n" description="pcomment n" />
  <branch name="branch 1" />
  <branch name="branch 2" />
                :
  <branch name="branch n" />
</control>
```

属性：cname　- 制御のメソッド名
　　gname　- 制御のグループ（バイリンガル対応）
　　ctitle　- 制御のタイトル（バイリンガル対応）
　　ccomment - 制御のコメント（バイリンガル対応）
　　ptype n　- 入力引数型 n (Int/ConstInt/VarInt/
　　　　Float/ConstFloat/VarFloat/
　　　　String/ConstString/VarString/

Script/Bool/ConstBool/VarBool/
Enum(A,B,…,N))

pcomment n　- 入力引数 n のコメント（バイリンガル対応）

branch n - 出力分岐名 n（バイリンガル対応）

補足：1)group 属性と description 属性は、オプションである。

2)バイリンガルに対応するには、"日本語文字列	EnglishText"と記述する。

6.5 関数定義要素

関数(function)定義要素は、以下の構造を持つ(4.4.2 項参照)。

```
<function functionID="fname" type="ftype" title="ftitle" description="fcomment">
  <parameter type="ptype 1" description="pcomment 1" />
  <parameter type="ptype 2" description="pcomment 2" />
                :
  <parameter type="ptype n" description="pcomment n" />
</function>
```

属性：fname　　- 関数のメソッド名

ftype　　- 戻り値型（Int/Float/String/Bool）

btitle　- 関数のタイトル（バイリンガル対応）

fcomment - 関数のコメント（バイリンガル対応）

ptype n　- 入力引数型 n (Int/ConstInt/VarInt/
Float/ConstFloat/VarFloat/
String/ConstString/VarString/
Script/Bool/ConstBool/VarBool/
Enum(A,B,…,N))

pcomment n　- 入力引数 n のコメント（バイリンガル対応）

補足：1)description 属性は、オプションである。

2)バイリンガルに対応するには、"日本語文字列	EnglishText"と記述する。

6.6 静的変数定義要素

静的変数(variable)定義要素は、以下の構造を持つ(4.4.2 項参照)。

```
<variable name="vname" type="vtype" title="vtitle" description="vcomment" />
```

属性：vname　　- 静的変数のメソッド名

vtype　　- 静的変数の型（Int/Float/String/Bool）

vtitle　- 静的変数のタイトル（バイリンガル対応）

vcomment - 静的変数のコメント（バイリンガル対応）

補足：1)description 属性は、オプションである。
2)バイリンガルに対応するには、"日本語文字列	EnglishText"と記述する。

7. 対話コンテンツ中間ファイルの概要

インタラクション・コンポーザからコンパイル出力され、シーケンス実行時に
ビヘイビア生成インタプリタにより実行されるスクリプトファイルである。

(1)1 行が 1 アドレスに対応し、最初の行のアドレスは 0 である。
(2)行中の項目は TAB コード<TAB>で区切られる。
(3)行の最後には必ず改行コード<LF>が必要。
(4)アドレスの変更がなければ、一行下(アドレス+1)に実行遷移。
(5)割り込みトリガ条件設定(IT コマンド)は START コマンドの直後に置かれる。
（コンパイル時に配置される）
(6)ファイルの最後は"EndOfFile"<LF>。

行の基本形式
コマンド<TAB>パラメータ 1<TAB>パラメータ 2<TAB>・・・<TAB>パラメータ n<LF>

7.1 START コマンド

最初のシーケンス(メインシーケンス)であれば､プロジェクト初期設定パラメータ設定を実行し、
次アドレスに行く。
サブシーケンスであれば、何もせずに次アドレスに行く。

命令フォーマット:
(0) START [sequenceID] [プロジェクト初期設定パラメータ 1]･･･[プロジェクト初期設定パラメータ N]

注 1：sequenceID は、シーケンスファイル名(拡張子あり)
注 2：プロジェクト初期設定パラメータは、メインシーケンスのみ存在する

7.2 IT(InterruptTrigger)コマンド

割り込みベクタに割り込み条件を追加する。
割り込み条件は複数記述でき、全ての条件が真の時に割り込みがかかる。
START コマンドの直後にのみ記述でき、複数の IT コマンドを続けて記述できる。

命令フォーマット:
(0) IT [instanceID] [条件数 N]
(1) [値①#1 種類(Func/Var)] [値①#1] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(2) [演算子#1] [値②#1 種類(Func/Var/Const)] [値②#1] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(3) [値①#2 種類(Func/Var)] [値①#2] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(4) [演算子#2] [値②#2 種類(Func/Var/Const)] [値②#2] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

：

(N*2-1) [値①#N 種類(Func/Var)] [値①#N] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(N*2)　[演算子#N] [値②#N 種類(Func/Var/Const)] [値②#N] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(N*2+1) [割り込み発生時に実行されるコマンド(GOTO or IRET)]

注：[[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]] は、値①②が関数(Func)の時のみ存在する。

7.3 END コマンド

処理を終了してプログラムの実行を停止する。

命令フォーマット:

(0) END [sequenceID]

注：sequenceID は、シーケンスファイル名(拡張子あり)

7.4 GOTO コマンド

指定アドレスへジャンプする。

命令フォーマット:

(0) GOTO [飛び先アドレス]

7.5 BHV(Behavior)コマンド

ビヘイビアを実行する。

BHV の実行結果により分岐アドレス 1～N の GOTO に分岐する。

命令フォーマット:

(0) BHV [instanceID] [behaviorID] [引数の数 M] [引数 1 種類(Func/Var/Const)][引数 1]･･･[引数 M]

(1) GOTO [分岐アドレス 1]

：

(N) GOTO [分岐アドレス N]

注：引数 n 種類が関数(Func)の場合、[引数 n]の後に[[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]
が挿入される。

7.6 BM(BehaviorMultiple)コマンド

多重化されたビヘイビアをInt型の選択変数(現状はSpeedSelectのみ)により選択して実行する。
選択条件 1＜選択条件 2＜･･･＜選択条件 N とする。
instanceID は BM のみで指定し、BHV の instanceID__x はコンパイラで自動生成される。
各 BHV に指定する behaviorID は、全て同一とする。
BHV の実行結果により分岐アドレス 1～L の GOTO に分岐する。(全ての BI で共通に使用)

命令フォーマット:
(0) BM [instanceID] [選択数 N] [選択変数] [選択条件 1]･･･[選択条件 N]
(1) BHV [instanceID__1] [behaviorID] [引数の数 M] [引数 1 種類(Func/Var/Const)][引数 1]･･･[引数 M]
:
(N) BHV [instanceID__N] [behaviorID] [引数の数 M] [引数 1 種類(Func/Var/Const)][引数 1]･･･[引数 M]
(N+1) GOTO [分岐アドレス 1]
:
(N+L) GOTO [分岐アドレス L]

注：引数 n 種類が関数(Function)の場合、[引数 n]の後に[[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]
が挿入される。

選択条件:
behaviorID__1 が選択される条件
選択変数値 ＜ (選択条件 1 + 選択条件 2) / 2
behaviorID__2 が選択される条件
(選択条件 1 + 選択条件 2) / 2 ≦ 選択変数値 ＜ (選択条件 2 + 選択条件 3) / 2
behaviorID__N が選択される条件
(選択条件 N-1 + 選択条件 N) / 2 ≦ 選択変数値

7.7 CTL(Control)コマンド

CTL の実行結果により分岐アドレス 1～N の GOTO に分岐する。

命令フォーマット:

(0) CTL [instanceID] [controlID] [引数の数 M] [引数 1 種類(Func/Var/Const)][引数 1]･･･[引数 M]

(1) GOTO [分岐アドレス 1]

：

(N) GOTO [分岐アドレス N]

注：引数 n 種類が関数(Func)の場合、[引数 n]の後に[[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

が挿入される。

7.8 IF(SimpleCondition)コマンド

単一条件分岐処理。

条件が真なら(2)へ、偽なら(3)へと分岐する。

命令フォーマット:

(0) IF [instanceID] [値①種類(Func/Var)] [値①] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(1) [演算子] [値②種類(Func/Var/Const)] [値②] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(2) GOTO [分岐アドレス YES]

(3) GOTO [分岐アドレス NO]

注：[[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]] は、値①②種類が関数(Func)の時のみ存在する。

7.9 MC(MultipleCondition)コマンド

複数条件分岐処理。

(n*2+1) の条件が真なら(n*2+2)へ、偽なら(n*2+2)を飛ばして次のアドレス（n++）の条件を判断。

命令フォーマット:

(0) MC [instanceID] [条件数 N] [値①種類(Func/Var)] [値①] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(1) [演算子#1] [値②#1 種類(Func/Var/Const)] [値②#1] [[関数引数の数 M] [関数引数 1 種類(Var/Const)][関数引数 1]･･･[関数引数 M]]

(2) GOTO [分岐アドレス 1]

：

(N*2-1) [演算子#N] [値②#N 種類(Func/Var/Const)] [値②#N] [[関数引数の数 M] [関数引数 1

種類(Var/Const)][関数引数 1]･･･[関数引数 M]]
(N*2) GOTO [分岐アドレス N]
(N*2+1) GOTO [分岐アドレス ELSE]

注：[[関数引数の数 M] [関数引数 1 種類(Variable/Const)][関数引数 1]･･･[関数引数 M]] は、値①②が関数(Func)の時のみ存在する。

7.10 SBR(SequentialBranch)コマンド

コマンドが実行されるたびに、分岐先が 1,…,N-1,N,N,… と順に変化する。

命令フォーマット:
(0) SBR [instanceID] [リセットなし(U)/リセットあり(R)] [分岐数 N]
(1) GOTO [分岐アドレス 1]
：
(N) GOTO [分岐アドレス N]

7.11 CBR コマンド(CyclicBranch)

コマンドが実行されるたびに、分岐先が 1,2,…,N,1,… と循環して変化する。

命令フォーマット:
(0) CBR [instanceID] [リセットなし(U)/リセットあり(R)] [分岐数 N]
(1) GOTO [分岐アドレス 1]
：
(N) GOTO [分岐アドレス N]

7.12 RBR(RandomBranch)コマンド

コマンドが実行されるたびに、分岐先が 1～N からランダムに変化する。
なお、直前に分岐したものは、直後の分岐からは回避される。

命令フォーマット:
(0) RBR [instanceID] [分岐数 N]
(1) GOTO [分岐アドレス 1]
：
(N) GOTO [分岐アドレス N]

7.13 CALL コマンド

サブシーケンスのコール。
スタックに戻りアドレス(CALL の次のアドレス)と割り込みベクタを push して
指定アドレスにジャンプする。

サブシーケンスからのリターン時は、RET の戻り先引数により 1～N の GOTO に分岐する。

命令フォーマット:
(0) CALL [アドレス]
(1) GOTO [戻り先アドレス 1]
：
(N) GOTO [戻り先アドレス N]

7.14 RET(Return)コマンド

サブシーケンスからのリターン。
割り込みベクタをクリアし、スタックから pop した割り込みベクタを設定し、
スタックから pop したアドレス＋戻り先引数 - 1 へジャンプする。

命令フォーマット:
(0) RET [戻り先引数]

7.15 IRET(InterruptReturn)コマンド

①割り込みからのリターン。
スタックから pop したアドレスへジャンプする。

命令フォーマット:
(0) IRET

②割り込みからの直接ジャンプ
スタックを pop しそのアドレスを破棄した後、[アドレス]にジャンプ。

命令フォーマット:
(0) IRET [アドレス]

7.16 EXEC(Execute)コマンド

別プロジェクトのプログラムを実行。
実行後は、元のプログラムには戻らない。

命令フォーマット:
(0) EXEC [instanceID] [パスを含むプロジェクト名]

注：パスを含むプロジェクト名は、カレントワークディレクトリからの相対パスとする。

8. デバッグファイルの概要

インタラクション・コンポーザからコンパイル出力され、オンラインデバッグ時に
使用するファイルである。

(1)1 行が 1 アドレスに対応し、最初の行のアドレスは 0 である。
(2)対話コンテンツ中間ファイル(.icp)の同じ行番号のものが対応。。
(3)行中の項目は TAB コード<TAB>で区切られる。
(4)行の最後には必ず改行コード<LF>が必要。
(5)ファイルの最後は"EndOfFile"<LF>。

ファイルフォーマット:
・[T or F] [ファイル名] [エディタ内部での固有 ID]
　T：ブレークする
　F：ブレークしない

デバッガ実行時に、次に実行する行のファイル名と固有 ID を、
インタプリタからエディタに送信する。

9. 割り込み時動作の概要

9.1 割り込みタイミング

(1)START, IRET
　割り込みは無視される。
(2)END
　実行前。
　プログラム終了後は、割り込みは無視される。
(3)GOTO, IF, MC, SBR, CBR, RBR, IT, CALL, RET, EXEC
　実行前。
(4)BHV, CTL
　実行前と実行中。
　実行中に割り込みがあった場合、割り込み処理終了後の IRET で、
　割り込まれた BHV,CTL は再実行される。

10. デバッグ時通信コマンド概要

インタラクション・コンポーザのデバッガ部とビヘイビア生成モジュールのインタプリタ部
との通信フォーマットである。
行中の項目は、指定のないものはスペースで区切られる。
行の最後には必ず改行コード<LF>が必要。

10.1 インタプリタからデバッガへのコマンド

Send ProgFile　：対話コンテンツ中間ファイル(.icp)送信要求

Send DebugFile：デバッガファイル(.dbg)送信要求

Send LabelFile：ラベルファイル(.lbl)送信要求

Send BehaviorFile：ビヘイビアファイル(.bhv)送信要求

Break [FileName] [NodeNumber]：ブレークポイント到達で応答待ち(引数は、次実行ノード)

Run [FileName] [NodeNumber]：通常実行中で応答待ち(引数は、次実行ノード)

Error [エラーメッセージ<TAB>ErrorMessage]：エラーメッセージを表示して応答待ち

Abort [実行中断メッセージ<TAB>AbortMessage]：実行中断メッセージを表示して終了

End　：プログラム実行終了

Value [TAB 区切りの変数値リスト]：Monitor コマンドの応答

書式の詳細：Value<TAB>val1<TAB>val2<TAB>……<TAB>val

未定義変数は、"UNDEFINED"という文字列を返す。

10.2 デバッガからインタプリタへのコマンド

Next：次実行

End　：デバッグ終了

Continue：エラーメッセージ確認し実行再開

Monitor [カンマ区切りの変数リスト]：変数のモニタ

書式の詳細：Monitor val1,val2,……,valN

10.3 動作シーケンス

(1)接続時シーケンス

　　← インタプリタ側の接続待ち(listen)

デバッガ側から connect →

　　← "Send ProgFile"

対話コンテンツ中間ファイル(.icp)一括送信(最後の"EndOfFile"行を含む) →

　　← "Send DebugFile"

デバッガファイル(.dbg)一括送信(最後の"EndOfFile"行を含む)　→

　　← "Send LabelFile"

ラベルファイル(.lbl)一括送信(最後の"EndOfFile"行を含む)　→

　　← "Send BehaviorFile"

ビヘイビアファイル(.bhv)一括送信(最後の"EndOfFile"行を含む)　→

(2)実行時シーケンス

　　← "Run/Break …"

"Monitor …"(Break、Step 動作時のみ) →

　　← "Value …"

"Next" →

← "Run/Break …"
"Monitor …"(Break、Step 動作時のみ) →
← "Value …"
"Next" →
:
:
"Next" →
← "End"
デバッガ側のクローズ →
← インタプリタ側のクローズ

(3)ユーザによる強制終了時
← "Run/Break …"
"End" →
← "End" (デバッガ側の受信タイムアウト 5 秒)
デバッガ側のクローズ →
← インタプリタ側のクローズ

(4)Error 発生時(実行継続)
← "Run/Break …"
"Next" →
← "Error …"
"Continue" →
← "Run/Break …"
"Monitor …"(Break、Step 動作時のみ) →
← "Value …"
"Next" →
:

(5)Error 発生時(実行停止)
← "Run/Break …"
"Monitor …"(Break、STEP 動作時のみ) →
← "Value …"
"Next" →
← "Error …"
"End" →
← "End" (デバッガ側の受信タイムアウト 5 秒)
デバッガ側のクローズ →
← インタプリタ側のクローズ

(6)Abort 発生時

← "Run/Break …"

"Next" →

← "Abort …"

デバッガ側のクローズ →

← インタプリタ側のクローズ

11. その他コマンド形式

以下のコマンドについては、それぞれの仕様書を参照のこと。

・動作指令スクリプトコマンド(BG->MG) … 動作生成コンポーネント仕様書

・対話履歴管理コマンド(BG->HM) … 対話履歴管理コンポーネント仕様書

・環境データコマンド(BG->ED) … 環境データコンポーネント仕様書

12. インタラクション・リモート・コントローラ入出力データの概要

12.1 入力コマンド（リモコン→対話フロー制御）

"<BG:ITC x>¥n" - 割り込みテスト用入力文字

"<BG:ITS text>¥n" - 割り込みテスト用入力文字列

12.2 出力ステータス（対話フロー制御→リモコン）

"<BG:ACT>¥n" - ACTIVE 状態遷移

"<BG:DAC>¥n" - INACTIVE 状態遷移

"<BG:ERR>¥n" - ERROR 状態遷移

"<BG:MCL c1¥tc2¥t・・・¥tcn>¥n" - 音声認識候補(直後の MC 条件のリスト)

"<BG:SRB n>¥n" - 音声認識開始(BGN)

"<BG:SRS n,text>¥n" - 音声認識開始(SEL)

"<BG:SRR text>¥n" - 音声認識結果

"<BG:SRC>¥n" - 音声認識停止