HITACHI
Inspire the Next

COBOL2002　使用の手引　操作編

操作書

4010-1J-803

COBOL2002

前書き

■ 対象製品

P-9W36-1251 COBOL2002 Net Server Suite(64) 05-00（適用 OS：Linux Server 9 (64-bit x86_64)）
P-9W36-2251 COBOL2002 Net Server Runtime(64) 05-00（適用 OS：Linux Server 9 (64-bit x86_64)）

これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

COBOL2002 は，COBOL2002 規格（ISO/IEC 1989:2002）の主な機能に対応しています。
COBOL85 互換機能（コンパイラオプション）指定時，COBOL2002 は，COBOL85 規格（ISO85，ANSI85，JIS88，JIS92）の上位水準に準拠します。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

■ 発行

2025 年 4 月　4010-1J-803

■ 著作権

はじめに

このマニュアルは，次に示すプログラムプロダクトの機能と操作方法について説明したものです。

- P-9W36-1251 COBOL2002 Net Server Suite(64)
- P-9W36-2251 COBOL2002 Net Server Runtime(64)

AIX(64) COBOL2002，Linux(x64) COBOL2002 をお使いになる場合は，必要に応じて「9.　64bit アプリケーションの開発」をお読みください。

Linux(x86) COBOL2002，Linux(x64) COBOL2002 をお使いになる場合は，初めに「10. COBOL2002 での UTF-8 ロケールの対応」をお読みください。

■ 対象読者

このマニュアルは，COBOL2002 のテストデバッガ，カバレージ，TD コマンド生成機能の使用方法を知りたい方を対象としています。また，COBOL の基本的な言語仕様と UNIX の操作方法について理解していることを前提としています。

■ このマニュアルで使用する記号

このマニュアルで使用する記号を次に示します。

記号	意味
[　] キー	文字キーや F（ファンクション）キーを意味する。
[　] ＋ [　] キー	＋の前のキーを押したまま，あとのキーを押すことを意味する。
{　}	この記号で囲まれている複数の項目のうちから一つを選択することを意味する。項目が縦に複数行にわたって記述されている場合は，そのうちの 1 行分を選択する。
[　]	この記号で囲まれている項目は省略してもよいことを意味する。 複数の項目が縦または横に並べて記述されている場合には，すべてを省略するか，記号 {　} と同じく，どれか一つを選択する。
…	記述が省略されていることを意味する。 この記号の直前に示された項目を繰り返して複数個指定できる。
下線	括弧で囲まれた複数の項目のうち 1 項目に対して使用され，括弧内のすべてを省略したときにシステムがとる標準値を意味する。
|	横に並べられた複数の項目に対して項目間の区切りを示し，「または」を意味する。
□	オペランドの名称の短縮形を示す。この記号で囲まれている文字だけを指定すれば短縮形となる。

記号	意味
[ ]	ウィンドウのメニューバーから選択するメニュー，コマンド，またはボタンを意味する。

■ プログラム例について

このマニュアルのプログラム例は，断り書きがない場合は AIX(32) COBOL2002 および Linux(x86) COBOL2002 用です。プログラム例を AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用するには，プログラムの記述に変更が必要な場合がありますのでご注意ください。

目次

1 テストデバッガ，カバレージ，TD コマンド生成機能，および Unicode 機能の概要

テストデバッガ，カバレージ，TD コマンド生成機能，および Unicode 機能の概要について説明します。

1.1　テストデバッガとは

テストデバッガは，TD コマンドなどでプログラムの実行を制御しながらプログラムをテストしたり，デバッグしたりする機能です。テストデバッガで，プログラムの実行を実行文単位に中断し，データ項目に対して値を表示，代入，比較してデバッグできます。また，連動するプログラムやファイルなどの実行環境が整わない段階でも，それらの処理をシミュレーションできる機能もあるため，単体でプログラムをテストすることもできます。テストデバッガを使用するときは，-TDInf コンパイラオプションを指定します。

テストデバッガの主な機能について説明します。

1.1.1　デバッグ機能

(1)　実行プログラムの中断

プログラムの実行を実行文単位に中断し，データの代入などができます。機能の概要については「2.2.1 プログラムの中断」を参照してください。

- 中断点の設定による中断
- データ監視条件の設定による中断
- 実行時エラーが発生した場合の中断

(2)　実行の制御

設定した中断点などに基づき実行できます。機能の概要については「2.2.2　プログラムの実行」を参照してください。

- 連続実行
- ステップイン
- ステップオーバー
- ジャンプ
- ジャンプ実行
- 実行の強制終了
- 実行の開始と開始時にできる指定

(3)　データの操作

COBOL プログラムのデータに対して，値の表示，値の代入，比較ができます。

データの操作については「2.2.5　データの操作」を参照してください。

Unicode 機能を使用するときのデータの操作については，「1.4　Unicode 機能」を参照してください。
Unicode 機能を使用しないときのデータの操作については「2.2.5　データの操作」を参照してください。

- データ値の表示
- データ値の代入
- データ値の比較

1.1.2　テスト機能

(1)　単体テスト

連動するプログラムやファイルなどの実行環境が整わない段階でも，それらの処理をシミュレーションして，プログラムを単体でテストできます。呼び出し元のプログラム，呼び出し先のプログラム，ファイル，または DC（データコミュニケーション）を，TD コマンドによってシミュレーションできます。

機能の概要については「2.2.7　プログラムの単体テスト」を参照してください。

- 主プログラムシミュレーション
- 副プログラムシミュレーション
- ファイル入出力文のシミュレーション
- DC シミュレーション

1.1.3　ラインモードとバッチモード

テストデバッガには，ラインモードとバッチモードの二つの方法があります。

- ラインモード

  TD コマンドの入力によって，対話的にテストデバッグができます。TD コマンドによる入出力だけで実行できるので，画面での操作がなく，簡単に早くテストとデバッグができます。

  ラインモードでのテストデバッグの手順については，「3.2.1　テストの手順」を参照してください。
- バッチモード

  あらかじめ，実行したい TD コマンドをファイルにまとめて記述しておき，そのファイルを起動することによって，一括してテストデバッグを実行する方法です。端末に，TD コマンドを記述したファイルを指定した起動コマンドを入力すると，ファイルに記述された TD コマンドを一括して実行します。一度実行を開始すれば，利用者の操作を必要としないので，効率良く大量のプログラムがテストできます。

  バッチモードでのテストデバッグの手順については，「4.2.1　テストの手順」を参照してください。

1.1.4　テストデバッガの機能一覧

テストデバッガの機能一覧とラインモード，バッチモードでの機能の有無を表 1-1 に示します。

表 1-1　テストデバッガの機能一覧とラインモード，バッチモードでの機能の有無

機能の概要	機能の詳細		ラインモード	バッチモード
プログラムの中断	中断点の設定による中断	中断点の設定／解除	○	○
		中断点の一覧表示	○	○
	データ監視条件の設定による中断		○	○
	実行時エラーが発生した場合の中断		○	×
プログラムの実行	連続実行		○	○
	ステップイン		○	○
	ステップオーバー		○	○
	ジャンプ		○	○
	ジャンプ実行		○	○
	実行の強制終了		○	○
プログラムの実行の追跡	トレース表示		○	○
	フロー情報の蓄積と表示		○	○
	現在位置の表示		○	○
データの操作	データ値の表示		○	○
	オブジェクトデータ値の表示	オブジェクト参照データ値の表示	○	○
	データ値の代入		○	○
	データ値の比較		○	○
領域の操作	領域の確保／解放		○	○
	任意のデータ項目のアドレスの取得		○	○
プログラムの単体テスト	主プログラムシミュレーション		○	○
	副プログラムシミュレーション		○	○
	ファイルシミュレーション		○	○
	ファイル入出力条件発生のシミュレーション	ファイル終了条件	○	○
		ページ終了条件	○	○
		無効キー	○	○
		入出力誤り	○	○

機能の概要	機能の詳細		ラインモード	バッチモード
	DC シミュレーション		○	○
バッチによるテスト	テストケースの指定		×	○
テストデバッグのための準備	翻訳単位またはソース要素の変更		○	○
	TD コマンド格納ファイルの TD コマンドの実行		○	○
	等価規則の変更		○	○
	結果のファイル出力	結果出力先の指定	○	○
		テストデバッガの表示内容のファイル出力	○	×
その他	装置名へのファイルの割り当て		○	○
	UNIX コマンドの実行		○	×
	テスト結果への注釈の表示		○	○

(凡例)

○：機能を使用できる

×：機能を使用できない

ラインモード，バッチモードそれぞれで使用できる TD コマンドについては，「5.3　TD コマンドの一覧」を参照してください。

1.2　カバレージとは

カバレージを使用して，テスト終了後，C0 メジャー，C1 メジャー，未実行文情報などのカバレージ情報を集計して表示できます。カバレージ情報を確認することでテストの進捗状況や性能を数値で把握できます。カバレージには，テストの進捗状況を定量的に表すカバレージ情報，文の実行回数をカウントするカウント情報の表示機能があります。

端末から入力するコマンドの指示によって，カバレージの機能を操作できます。プログラムからカバレージを連動実行させ，一括してカバレージの蓄積およびカウント情報を表示することもできます。一度実行を開始すれば，利用者の操作を必要としないので，効率良く大量のプログラムのカバレージができます。

カバレージを使用するときは，プログラムのコンパイル時に-CVInf コンパイラオプションを指定します。テストデバッガ上でカバレージを使用するときは，-TDInf コンパイラオプションも指定します。

カバレージの手順については，「7.　カバレージ」を参照してください。

1.2.1　カバレージ情報

カバレージ情報とは，テストによって実行された文の割合を表示したもので，テストの進捗状況を確認するのに使用します。

カバレージ情報を表示できるリストには，カバレージ情報一覧，まとめ表示，全ソース表示などがあります。また，カバレージ情報を 0%化したり，カバレージ情報ファイルをマージしたりする操作もできます。

1.2.2　カウント情報

カウント情報とは，プログラムごとに文の実行回数を表示したもので，プログラムの実行時の性能を評価するのに使用します。

1.2.3　カバレージの機能一覧

カバレージの機能一覧を表 1-2 に示します。

表 1-2　カバレージの機能一覧

機能の概要	機能の詳細		
カバレージ情報の表示	カバレージ情報の蓄積		
	カバレージ情報の表示	テキスト形式のファイルの場合	C0，C1，S1 メジャーの表示
			未実行の文の表示
			差分カバレージ情報の表示

機能の概要	機能の詳細		
		CSV 形式のファイルの場合	C0，C1，S1 メジャーの表示
			未実行の文の表示
			差分カバレージ情報の表示
	カバレージ情報の操作		カバレージ情報の 0%化
			差分カバレージ情報の 0%化
			差分カバレージ情報のクリア
			カバレージ情報のマージ
カウント情報の表示	実行回数の表示		

1.3　TD コマンド生成機能とは

TD コマンド生成機能とは，テスト対象の COBOL ソースファイルから，テストデバッガで使用する TD コマンド群を生成する機能です。TD コマンド群とは，テストデバッガの実行を制御するための TD コマンドの集まりです。生成されたファイルを利用し，必要に応じて修正することで，TD コマンド群を効率良く作成できます。

生成される TD コマンド群には，中断点情報，プログラムのシミュレーション情報，およびファイルのシミュレーション情報の 3 種類があります。TD コマンド生成機能については，「8.　TD コマンド生成機能」を参照してください。

1.3.1　TD コマンド生成機能を使用するときに指定するコンパイラオプション

TD コマンド群は，-TestCmd コンパイラオプションを指定してコンパイルしたときに生成されます。

1.3.2　TD コマンド生成機能の対象となるプログラム

TD コマンド生成機能の対象となるプログラムは，コンパイルしたときすべてのプログラムで，S レベル，U レベルのコンパイルエラーのなかった原始プログラムファイルだけです。

1.4 Unicode 機能

1.4.1 機能の概要

Unicode をデータ値として扱う COBOL プログラムをデバッグするための機能です。

AIX の場合

テストデバッガは，データを表示するときに，Unicode からシフト JIS への変換を行い，代入や比較をするときは，シフト JIS から Unicode への変換を行います。そのため，ユーザはデータの操作をシフト JIS でできます。

この機能は，カバレージ情報の蓄積およびカウント情報の表示を対象にしません。ただし，カバレージ情報の蓄積またはカウント情報の表示を実行した場合は，この機能を使用しないで実行した場合と同じ動作をします。

Linux の場合

テストデバッガは，用途が NATIONAL の項目に対して，データを表示するときに，UTF-16 から UTF-8 への変換を行い，代入や比較をするときは，UTF-8 から UTF-16 への変換を行います。そのため，ユーザはデータの操作を UTF-8 でできます。

なお，COBOL ソースがシフト JIS であることから，デバッガでは，文字コードをシフト JIS で扱います。このため，UTF-8，または UTF-16 からシフト JIS に変換できない文字がある場合は，16 進数や別の文字に置き換えて実行，または，エラーとして，実行を中止することがあります。詳細については，以降の Linux の場合の説明での注意事項などを参照してください。

Unicode 機能のその他の機能については，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」を参照してください。

(1)　使用するコード変換ライブラリ

Unicode 機能を利用するには，コード変換ライブラリをインストールしておく必要があります。次のプログラムプロダクト（以降これらのプログラムプロダクトをコード変換ライブラリと表記します）をインストールしてください。

- AIX(32)，Linux(x86)の場合
  - 日立コード変換 - Runtime
  - Hitachi Code Converter - Runtime for C/COBOL
- AIX(64)の場合
  - 日立コード変換 - Runtime(64)
  - Hitachi Code Converter - Runtime for C/COBOL(64)
- Linux(x64)の場合
  - Hitachi Code Converter - Runtime for C/COBOL(64)

また，COBOL プログラムから直接コード変換ライブラリを呼び出す場合は，上記のプログラムプロダクトに加えて次のプログラムプロダクトが必要です。

- AIX(32)，Linux(x86)の場合
  - 日立コード変換 - Development Kit
  - Hitachi Code Converter - Development Kit for C/COBOL
  - Hitachi Code Converter - Development Kit for C/COBOL(64)
- AIX(64)の場合
  - 日立コード変換 - Development Kit(64)
  - Hitachi Code Converter - Development Kit for C/COBOL(64)
- Linux(x64)の場合
  - Hitachi Code Converter - Development Kit for C/COBOL(64)

コード変換ライブラリが次のどれかの場合で，テストデバッガを起動したときはエラーメッセージを出力してテストデバッガは終了します。

- コード変換ライブラリをインストールしていない。
- 正しくインストールされていない。
- 前提バージョンより前のバージョンがインストールされている。

出力するメッセージを次に示します。
「KCCC0202T-U　コード変換ライブラリが正しくインストールされていません。」

外字を表示するための外字マッピングファイルは，コード変換ライブラリのインストールディレクトリにあるファイルを使用します。外字マッピングファイル名は，インストール時のファイル名にします。

(2)　コンパイラオプションと環境変数について

Unicode 機能を利用してデバッグをするときに必要なコンパイラオプションと環境変数を次に示します。

なお，コンパイラオプションおよび環境変数については，マニュアル「COBOL2002　使用の手引　手引編」の「コンパイラオプション」および「環境変数」を参照してください。

Unicode 機能に必要なコンパイラオプションを表 1-3 に，環境変数を表 1-4 に示します。

表 1-3　Unicode 機能に必要なコンパイラオプション

コンパイラオプション	説明
-UniObjGen	デバッグ対象プログラムは，必ずこのコンパイラオプションを指定してコンパイルしなければならない。
-UniEndian	UTF-16 のバイトオーダを指定したいときだけ指定する。

表 1-4　Unicode 機能に必要な環境変数

環境変数	説明
CBLLANG	必ず設定しなければならない。 設定は，テストデバッガを起動する前に設定しなければならない。
CBLUNIENDIAN	UTF-16 のバイトオーダを指定したいときだけ設定する。 設定は，テストデバッガを起動する前に設定しなければならない。
CBLSRCENCODING※	必ず設定しなければならない。 設定は，テストデバッガを起動する前に設定しなければならない。

注※　Linux の場合に指定する。

Unicode 機能に必要なコンパイラオプションの指定と環境変数を設定した場合，テストデバッガは，次の規則に従い動作します。

1. すべてのデバッグ対象プログラムに-UniObjGen コンパイラオプションを指定してコンパイルする必要があります。また，環境変数 CBLLANG に UNICODE を指定した環境でデバッグをしてください。それ以外の環境での動作は保証しません。
2. -UniEndian コンパイラオプションで指定したバイトオーダと環境変数 CBLUNIENDIAN で指定したバイトオーダを一致させてください。一致しない場合の動作は保証しません。
3. 環境変数の値は，テストデバッガの起動前に設定してください。テストデバッガを起動中に環境変数の値を設定または変更しても，有効になりません。

(3)　デバッグ手順

Unicode 機能を利用したデバッグの手順を次に示します。

1. デバッグ対象の COBOL プログラムをコンパイルするときに，-UniObjGen コンパイラオプションを指定します。Linux の場合は，環境変数 CBLSRCENCODING に SJIS を同時に設定します。
2. 環境変数 CBLLANG に値として UNICODE を設定します。Linux の場合は，環境変数 CBLSRCENCODING に SJIS を同時に設定します。
3. テストデバッガを起動してデバッグをします。

用途が NATIONAL の項目（UTF-16）に対してバイトオーダを指定する場合は，上記手順に合わせて，次の手順になります。

1. デバッグ対象の COBOL プログラムをコンパイルするときに，-UniEndian コンパイラオプションで UTF-16 のバイトオーダを指定します。
2. 環境変数 CBLUNIENDIAN に値として BIG または LITTLE を設定して，UTF-16 のバイトオーダを指定します。
3. テストデバッガを起動して，デバッグをします。

バイトオーダの指定については，「1.4.3　その他の注意事項」の「(1)　バイトオーダ」を参照してください。

(4)　その他の注意事項

AIX の場合

1. この機能を使用したテストデバッグを実行するには，シフト JIS 環境下でなくてはなりません。この機能を EUC 環境下で実行した場合はエラーになります。次のエラーメッセージを出力してテストデバッガは終了します。

   「KCCC0204T-U　日本語 EUC 環境下では，環境変数 CBLLANG に UNICODE を指定してテストデバッガを実行できません。」

Linux の場合

1. この機能を使用したテストデバッグを実行するには，UTF-8 環境下でなくてはなりません。また，あわせて，環境変数 CBLLANG，および CBLSRCENCODING を設定しなければなりません。環境変数で設定する条件が満たされない場合は，エラーになります。

   ロケール，環境変数の設定値によるテストデバッガの動作を表 1-5 に示します。

表 1-5　ロケール，環境変数の設定値によるテストデバッガの動作

ロケール	環境変数 CBLLANG の値	環境変数 CBLSRCENCODING の値	テストデバッガの動作
UTF-8	UNICODE	SJIS	正常
		SJIS 以外 または設定なし	エラー※
	UNICODE 以外 または設定なし	SJIS	エラー※
		SJIS 以外 または設定なし	

注※

KCCC0205T-S のエラーメッセージを出力し，テストデバッガは終了します。

2. この機能を使用したカバレージ（カバレージ情報の蓄積，カウント情報の表示，カバレージ情報の表示（テキスト形式および CSV 形式），カバレージ情報の操作）を実行するには，UTF-8 環境下でなくてはなりません。また，あわせて，環境変数 CBLSRCENCODING を設定しなければなりません。環境変数で設定する条件が満たされない場合は，エラーになります。

   ロケール，環境変数の設定値によるカバレージの動作を表 1-6 に示します。

表 1-6　ロケール，環境変数の設定値によるカバレージの動作

ロケール	環境変数 CBLSRCENCODING の値	カバレージの動作
UTF-8	SJIS	正常
	SJIS 以外 または設定なし	エラー※

注※

KCCC0206T-S のエラーメッセージを出力し，カバレージは終了します。

1.4.2　機能の詳細

(1)　データ値の表示

AIX の場合

データ値をデータ属性表示するときは，テストデバッガが文字コードを Unicode（UTF-8 または UTF-16）からシフト JIS に変換して，変換後の値を表示します。

Linux の場合

用途が NATIONAL の項目のデータ値をデータ属性表示するときは，テストデバッガが文字コードを UTF-16 から UTF-8 に変換して，変換後の値を表示します。

また，データ値を 16 進数で表示するときは，文字コードの変換は行わないで，Unicode（UTF-8 または UTF-16）の 16 進数で表示します。UTF-16 の 16 進数での表示については，「1.4.3　その他の注意事項」の「(1)　バイトオーダ」を参照してください。

データ値の表示は，DISPLAY DATA コマンド，SET TRACE コマンド，DISPLAY OBJECT コマンド，DISPLAY FACTORY コマンドで，データの値をデータ属性または 16 進数で表示できます。

DISPLAY DATA コマンド，SET TRACE コマンド，DISPLAY OBJECT コマンド，DISPLAY FACTORY コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.9　SET TRACE／RESET TRACE（トレース表示の開始と中止）」，「5.4.13　DISPLAY DATA（データの値表示）」，「5.4.14　DISPLAY OBJECT／DISPLAY FACTORY（オブジェクトのデータ値の表示）」を参照してください。

データ値を表示するときの規則を次に示します。

1. AIX の場合，データ値をデータ属性表示するときに，用途が DISPLAY の項目および用途が NATIONAL の項目を変換して表示します。
   - 用途が DISPLAY の項目の場合は，UTF-8 からシフト JIS に変換して表示します。
   - 用途が NATIONAL の項目の場合は，UTF-16 からシフト JIS に変換して表示します。

   各データ項目について変換するかどうかを表 1-7 に示します。

表 1-7　シフト JIS へ変換するデータ項目

表示するデータ項目の種類	Unicode からシフト JIS への変換
集団項目	○
英字項目	−
英数字項目	○
英数字編集項目	○

表示するデータ項目の種類	Unicode からシフト JIS への変換
数字編集項目	―※1
整数項目※2	―
非整数項目※2	―
COMP－X 項目	―
外部浮動小数点項目	―
内部浮動小数点項目	―
指標名	―
指標データ項目	―
外部ブール項目	―
内部ブール項目	―
日本語集団項目	●
日本語項目	●
日本語編集項目	●
アドレス名	―
アドレスデータ項目	―
ポインタ項目	―
オブジェクト参照データ項目	―

（凡例）

○：UTF-8 からシフト JIS へ変換する

●：UTF-16 からシフト JIS へ変換する

－：変換の必要がないため変換しない

注※1

文字がある場合は，UTF-8 からシフト JIS に変換します。

注※2

整数，非整数項目はそれぞれ外部 10 進項目，内部 10 進項目，および 2 進項目を含みます。

2. Linux の場合，データ値をデータ属性表示するときに，用途が NATIONAL の項目を，UTF-16 から UTF-8 に変換して表示します。

3. 表示機能を使用してデータ値を 16 進数で表示する場合は，変換しないで，Unicode（UTF-8 または UTF-16）の 16 進数で表示します。データ属性表示で表示エラーが発生した場合も変換しないで，Unicode （UTF-8 または UTF-16）の 16 進数で表示します。

4. AIX の場合，Unicode（UTF-8 または UTF-16）からシフト JIS へ変換できないコードがある場合は，そのコードを置き換えて表示します。

   - 用途が DISPLAY の項目の場合は，半角ピリオドに置き換えて表示します。

- 用途が NATIONAL の項目の場合は，全角' = '（げた記号）（シフト JIS:X'81AC'）に置き換えて表示します。

5. Linux の場合，Unicode（UTF-8 または UTF-16)からシフト JIS に変換できないコードがある場合は，そのコードを置き換えて表示します。
   - 用途が DISPLAY の項目の場合，半角ピリオドに置き換えて表示します。
   - 用途が NATIONAL の項目の場合，全角' = '（げた記号）（UTF-8:X'E38093'）に置き換えて表示します。
6. Linux の場合，データ値をデータ属性表示するときに，用途が NATIONAL の項目で UTF-16 から UTF-8 に変換すると 1 バイトコードになる文字は，表示できる文字であれば，そのまま 1 バイトコードで表示します。 UTF-8 では表示できない文字の場合は，UTF-8 でのバイト数分を半角ピリオドで表示します。
7. Unicode（UTF-8 または UTF-16）で多バイトとして扱われる文字の途中で終了している場合は，そのバイトを半角ピリオドに置き換えて表示します。
8. コード変換中にエラーが発生したときは，データ属性表示で表示エラーになった場合と同様に変換しないで，Unicode（UTF-8 または UTF-16）の 16 進数で表示します。

(2)　データ値の代入

AIX の場合

データ名に文字定数（英数字定数，日本語文字定数，ALL 定数）を代入する場合は，テストデバッガが，文字定数の文字コードをシフト JIS から Unicode（UTF-8 または UTF-16）に変換して，データに代入するので，ユーザは文字定数をシフト JIS で指定できます。

Linux の場合

用途が NATIONAL のデータ名に文字定数（日本語文字定数，ALL 定数）を代入する場合は，テストデバッガが，文字定数の文字コードを UTF-8 から UTF-16 に変換して，データ名に代入するので，ユーザは文字定数を UTF-8 で指定できます。

ただし，TD コマンド格納ファイルを使用して，データ名に文字定数（英数字定数，日本語文字定数，ALL 定数）を代入する場合は，テストデバッガが，文字定数の文字コードをシフト JIS から Unicode（UTF-8 または UTF-16）に変換して，データ名に代入します。

また，16 進英数字定数または 16 進日本語文字定数を使用すれば，文字コードを直接指定した代入ができます。16 進日本語文字定数での代入については，「1.4.3　その他の注意事項」の「(1)　バイトオーダ」を参照してください。

データ値の代入は，ASSIGN DATA コマンドで，データに値を代入して，値を変更できます。

ASSIGN DATA コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.15　ASSIGN DATA (データの値代入)」を参照してください。

データ値を代入するときの規則を次に示します。

1. AIX の場合，代入する値に文字定数を指定した場合は次のように変換して代入します。
   - 英数字定数の場合は，シフト JIS から UTF-8 に変換して代入します。
   - 日本語文字定数の場合は，シフト JIS から UTF-16 に変換して代入します。

   代入は変換後のサイズで行われるため，変換によってデータのサイズが受け取り側のデータ項目のサイズを超えた場合，超えた部分は代入されません。
2. Linux の場合，代入する値に日本語文字定数を指定した場合は，UTF-8 から UTF-16 に変換して代入します。ただし，TD コマンド格納ファイルを使用して，代入する値に文字定数を指定した場合は，次のように変換して代入します。
   - 英数字定数の場合は，シフト JIS から UTF-8 に変換して代入します。
   - 日本語文字定数の場合は，シフト JIS から UTF-16 に変換して代入します。

   代入は変換後のサイズで行われるため，変換によってデータのサイズが受け取り側のデータ項目のサイズを超えた場合，超えた部分は代入されません。
3. 16 進英数字定数または 16 進日本語文字定数を指定した場合は，文字コードの変換はしないでデータに代入します。
4. 16 進英数字定数または 16 進日本語文字定数の場合，次のように指定します。
   - 代入先の用途が DISPLAY の場合は，UTF-8 のコード値を指定します。
   - 代入先の用途が NATIONAL の場合は，UTF-16 のコード値を指定します。
5. 集団項目を除く，用途が DISPLAY の項目と，用途が NATIONAL の項目との間では代入はできません。
6. 用途が NATIONAL の項目を含む集団項目と，用途が DISPLAY の項目との間の代入はしないでください。また，用途が DISPLAY の項目を含む集団項目と，用途が NATIONAL の項目との間の代入はしないでください。代入をした場合は，代入後の値を保証しません。
7. 作用対象の用途が DISPLAY と，用途が NATIONAL の代入規則を表 1-8 に示します。なお，表 1-8 に示されていないデータ項目の組み合わせについては，「2.3.1　データの比較・代入規則」の「表 2-7　代入規則」を参照してください。
8. コード変換中に未定義コードを検出するなどのエラーが発生したときは，次のエラーメッセージを出力して代入処理を中止します。

   AIX の場合

   「KCCC4302T-E　値が不正なため処理できません。値（XXXXX）」

   Linux の場合

   「KCCC4328T-E 値が不正なため処理できません。値（X' XXXXXX'）」

表 1-8　代入規則（Unicode 機能）

送り出し側	受け取り側													
	集団項目	英字項目	英数字項目	英数字編集項目	数字編集項目	外部10進項目	整数※6	非整数※6	外部浮動小数点※7	内部浮動小数点※8	指標名	日本語集団項目	日本語項目	日本語編集項目
集団項目	※1	※4	※4	※4	※4	※4	○	○	※4	※4		※5	※5	※5
英字項目	※2	○	○	○										
英数字項目	※2	○	○	○	○	○	○	○	○	○				
英数字編集項目	※2	○	○	○										
数字編集項目	※2		○	○	○	○	○	○	○	○				
HIGH-VALUE	○	○	○	○								○	○	○
LOW-VALUE	○	○	○	○								○	○	○
SPACE	○	○	○	○								○	○	○
ZERO	○		○	○	○	○	○	○	○	○	○	○	○	○
英数字定数	※2	○	○	○										
英数字定数（数字だけ）	※2	○	○	○	○	○	○	○						
16進英数字定数	※2	○	○	○	○	○	○	○						
整数定数※6	○		○	○	○	○	○	○	○	○	※9			
非整数定数※6	○		○	○	○	○	○	○	○	○				
外部10進項目	※2		○	○	○	○	○	○	○	○	○			
整数※6	○		○	○	○	○	○	○	○	○	○			
非整数※6	○		○	○	○	○	○	○	○	○				
16進数字定数	※2		○	○	○	○	○	○	○	○	○			
外部浮動小数点※7	※2		○		○	○	○	○	○	○				
内部浮動小数点※8	※2		○		○	○	○	○	○	○				
浮動小数点数字定数	※2		○		○	○	○	○	○	○				
指標名						○	○				○			
日本語集団項目	※3											○	○	○
日本語項目	※3											○	○	○
日本語編集項目	※3											○	○	○
日本語文字定数	※3											○	○	○
16進日本語文字定数	※3											○	○	○
拡張16進定数	○	○	○	○	○	○	○	○	○	○		○	○	○

（凡例）

○：代入できる

空白：代入できない

注※1

送り出し側の集団項目に用途が NATIONAL の項目を含んでいて，受け取り側の集団項目に用途が DISPLAY の項目を含んでいる場合は，代入後の値を保証しません。また，送り出し側の集団項目に用途が DISPLAY の項目を含んでいて，受け取り側の集団項目に用途が NATIONAL の項目を含んでいた場合も，代入後の値は保証しません。

注※2

受け取り側の集団項目に用途が NATIONAL の項目を含んでいた場合は，代入後の値は保証しません。

注※3

受け取り側の集団項目に用途が DISPLAY の項目を含んでいた場合は，代入後の値は保証しません。

注※4

送り出し側の集団項目に用途が NATIONAL の項目を含んでいた場合は，代入後の値は保証しません。

注※5

送り出し側の集団項目に用途が DISPLAY の項目を含んでいた場合は，代入後の値は保証しません。

注※6

整数は，次のデータ項目で小数点けたを持たない指定です。非整数は，次のデータ項目で小数点けたを持つ指定です。

- 内部 10 進項目
- 2 進項目（COMP-X も含む）

注※7

外部浮動小数点形式の数字項目です。

注※8

内部浮動小数点形式の数字項目です。

注※9

正の整数（0 を含むが，-0 は含まない）だけ代入できます。

(3) 比較条件式の設定

AIX の場合

比較条件式に文字定数（英数字定数，日本語文字定数，ALL 定数）を指定した場合は，テストデバッガが，文字定数の文字コードをシフト JIS から Unicode（UTF-8 または UTF-16）に変換して比較するため，ユーザは文字定数をシフト JIS で指定できます。

Linux の場合

用途が NATIONAL のデータ名に対する比較条件式に文字定数（日本語文字定数，ALL 定数）を指定した場合は，テストデバッガが，文字定数の文字コードを UTF-8 から UTF-16 に変換して比較するため，ユーザは文字定数を UTF-8 で指定できます。

また，16 進英数字定数または 16 進日本語文字定数を使用できるため，文字コードを直接比較条件式に設定できます。16 進日本語文字定数での比較については，「1.4.3　その他の注意事項」の「(1)　バイトオーダ」を参照してください。

比較は，バイト単位で比較します。大小順もバイト単位の大小順になります。

比較条件式は，IF コマンド，または SET WATCH コマンドを使用して設定できます。

IF コマンド，SET WATCH コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.16　IF（データの値比較）」，「5.4.3　SET WATCH／RESET WATCH（データ監視条件の設定と解除）」を参照してください。

比較するときの規則を次に示します。

1. AIX の場合，比較条件式に文字定数を指定した場合は，文字定数の文字コードを次のように変換します。比較は変換後の値で比較します。
   - 英数字定数の場合は，UTF-8 に変換して比較します。
   - 日本語文字定数の場合は，UTF-16 に変換して比較します。

2. Linux の場合，用途が NATIONAL のデータ名に対する比較条件式に文字定数（日本語文字定数，ALL 定数)を指定した場合は，文字定数の文字コードを UTF-8 から UTF-16 に変換します。比較は変換後の値で比較します。

   ただし，TD コマンド格納ファイルを使用して，比較条件式に文字定数を指定した場合は，文字定数の文字コードをシフト JIS から次のように変換します。比較は変換後の値で比較します。

   - 英数字定数の場合は，UTF-8 に変換して比較します。
   - 日本語文字定数の場合，UTF-16 に変換して比較します。

3. 16 進英数字定数または 16 進日本語文字定数を指定した場合は，文字コードを変換しないで比較します。
4. 16 進英数字定数または 16 進日本語文字定数を指定する場合，次のように指定してください。
   - 用途が DISPLAY の項目と比較する場合は，UTF-8 を指定します。
   - 用途が NATIONAL の項目と比較する場合は，UTF-16 を指定します。
5. コード変換中にエラーが発生したときは，エラーメッセージを出力して比較処理を中止します。

   AIX の場合

   「KCCC4302T-E　値が不正なため処理できません。値（XXXXX)」

   Linux の場合

   「KCCC4328T-E 値が不正なため処理できません。値（X' XXXXXX')」

1.4.3　その他の注意事項

(1)　バイトオーダ

テストデバッガは，文字コードの変換だけではなく，UTF-16 のバイトオーダの変換をします。

UTF-16 のバイトオーダの変換の規則を次に示します。

1. AIX の場合，シフト JIS から UTF-16，または UTF-16 からシフト JIS に変換するときの UTF-16 のバイトオーダは，環境変数 CBLUNIENDIAN の値に従います。環境変数 CBLUNIENDIAN の値と仮定するバイトオーダの関係を表 1-9 に示します。
2. Linux の場合，UTF-8 から UTF-16，または UTF-16 から UTF-8 に変換するときの UTF-16 のバイトオーダは，環境変数 CBLUNIENDIAN の値に従います。環境変数 CBLUNIENDIAN の値と仮定するバイトオーダの関係を表 1-9 に示します。

表 1-9　環境変数 CBLUNIENDIAN の値と仮定するバイトオーダ

データ項目	環境変数 CBLUNIENDIAN の値		
	LITTLE	BIG	LITTLE，BIG 以外 （指定なしも含む）
用途が NATIONAL	UTF-16LE※1 を仮定する	UTF-16BE※2 を仮定する	AIX の場合 UTF-16BE※2 を仮定する Linux の場合 UTF-16LE※1 を仮定する

注※1

UTF-16LE: UTF-16 をリトルエンディアンで表現します。

注※2

UTF-16BE: UTF-16 をビッグエンディアンで表現します。

3. データ名への代入，および比較条件式を設定するときの 16 進日本語文字定数は UTF-16BE を指定してください。16 進日本語文字定数に指定された UTF-16BE のコード値は，環境変数 CBLUNIENDIAN の指定に従い，テストデバッガが自動的にバイトオーダの変換をします。
4. 用途が NATIONAL の項目を 16 進数で表示する場合，およびデータ属性表示で表示エラーが発生した場合に表示する 16 進数の値は，UTF-16BE の 16 進数で表示します。
5. データ名に値を代入する場合，拡張 16 進定数を使用してコード値を直接指定したときは，バイトオーダの変換はしません。
6. 比較条件式を設定する場合，拡張 16 進定数を使用してコード値を直接指定したときは，バイトオーダの変換はしません。

(2)　テストデバッガとカバレージが入出力するファイル

TD コマンド格納ファイルはシフト JIS で作成してください。ただし，外字を指定する場合など文字のコード値を直接指定したい場合は，16 進英数字定数，16 進日本語文字定数を使用して，指定したいコード値を Unicode で指定します。

AIX の場合

テストデバッガまたはカバレージが出力するエラーメッセージおよび次に示すファイルはシフト JIS で出力します。

- ログ出力ファイル
- 結果出力ファイル
- 結果蓄積ファイル
- 実行結果出力ファイル
- カバレージ情報リストファイル
- カウント情報リストファイル

- カバレージ統計情報 CSV ファイル
- ソースカバレージ情報 CSV ファイル

Linux の場合

テストデバッガまたはカバレージが出力するエラーメッセージおよび次に示すファイルは UTF-8 で出力します。

- ログ出力ファイル
- 結果出力ファイル
- 結果蓄積ファイル
- 実行結果出力ファイル
- カバレージ情報リストファイル
- カウント情報リストファイル
- カバレージ統計情報 CSV ファイル
- ソースカバレージ情報 CSV ファイル

(3) 表意定数の扱い

空白文字，表意定数 SPACE の文字コードは次のようになります。

- 用途が DISPLAY の場合は，UTF-8 の半角空白文字（X'20'）になります。
- 用途が NATIONAL の場合は，環境変数 CBLUNIENDIAN の設定に従い，UTF-16LE の全角空白文字（X'0030'）または UTF-16BE の全角空白文字（X'3000'）になります。仮定するバイトオーダについては「表 1-9　環境変数 CBLUNIENDIAN の値と仮定するバイトオーダ」を参照してください。

(4) UTF-8 環境下での注意事項（Linux の場合）

テストデバッガやカバレージの UTF-8 環境下での注意事項を次に示します。

1. 次に示す個所に UTF-8 の多バイト文字を使用した場合，テストデバッガやカバレージの動作は保証しません。エラーメッセージが正しく表示されないなどの現象が起きます。
   - 起動コマンドのパラメタ※
   - 環境変数に指定する値※
   - カレントディレクトリ名
   - COBOL プログラムの実行時にローディングされるすべての共用ライブラリの絶対パス名

   注※

   ファイル名または相対パス名が指定されたとき，それらの絶対パス名も含みます。
2. 次に示すコンパイラオプション，環境変数を同時に指定し，コンパイルして生成したプログラム情報ファイルだけ使用できます。

- -UniObjGen コンパイラオプション
- 環境変数 CBLSRCENCODING に SJIS を指定

これらのコンパイラオプション，環境変数については，「1.4.1　機能の概要」の「(2)　コンパイラオプションと環境変数について」を参照してください。

3. ラインモードから TD コマンドを入力するとき，UTF-8 からシフト JIS に変換できない文字コードを含んではいけません。
4. TD コマンド格納ファイルから入力するとき，次に示す TD コマンドのオペランドのファイル名に，シフト JIS から UTF-8 に変換できない文字コードは使用できません。
   - #INCLUDE（TD コマンド格納ファイルの取り込み）コマンド
   - SET PRINT（テスト結果蓄積先の設定）コマンド
   - SET LOG（ログ出力ファイルの設定）コマンド
   - ASSIGN DEVICE（装置名へのファイルの割り当て）コマンド

   #INCLUDE（TD コマンド格納ファイルの取り込み）コマンド，SET PRINT コマンド，SET LOG コマンド，および ASSIGN DEVICE コマンドについては，「5.4　TD コマンドの詳細」の「5.4.33　#INCLUDE（TD コマンド格納ファイルの取り込み)」，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除)」の「(1)　SET PRINT（テスト結果蓄積先の設定)」，「5.4.36　SET LOG／RESET LOG（ログ出力ファイルの設定／解除)」の「(1)　SET LOG（ログ出力ファイルの設定)」および「5.4.37　ASSIGN DEVICE（装置名へのファイルの割り当て)」のそれぞれを参照してください。
5. TD コマンド格納ファイルから入力するとき，！（UNIX コマンドの実行）コマンドで指定する UNIX コマンドまたはオペランドにシフト JIS から UTF-8 に変換できない文字コードは使用できません。

   ！（UNIX コマンドの実行）コマンドについては，「5.4　TD コマンドの詳細」の「5.4.38　！（UNIX コマンドの実行)」を参照してください。
6. TD コマンド格納ファイルから入力するとき，GO（実行の開始／再開）コマンド，STEP IN（ステップイン実行の開始／再開）コマンド，および STEP OVER（ステップオーバー実行の開始／再開）コマンドの PARAMETER オペランドで指定する引数の文字列に，シフト JIS から UTF-8 に変換できない文字コードは使用できません。

   GO（実行の開始／再開）コマンド，STEP IN コマンド，および STEP OVER コマンドについては，「5.4　TD コマンドの詳細」の「5.4.4　GO（実行の開始／再開)」，「5.4.5　STEP IN（ステップイン実行の開始／再開)」，および「5.4.6　STEP OVER（ステップオーバー実行の開始／再開)」のそれぞれを参照してください。
7. TD コマンドの実行結果およびメッセージに含まれるシフト JIS から UTF-8 に変換できない文字コードは，ログ出力ファイル，結果出力ファイル，結果蓄積ファイル，実行結果出力ファイル，およびラインモードのコンソールに出力するとき，ピリオド（.）に置き換えます。
8. TD コマンド格納ファイルの TD コマンドに含まれるシフト JIS から UTF-8 に変換できない文字コードは，結果出力ファイルに TD コマンドを出力するとき，ピリオド（.）に置き換えます。

9. COBOL のソースプログラムに含まれるシフト JIS から UTF-8 に変換できない文字コードは，カバレージ情報リスト・カウント情報リストでは，半角ピリオド（.）に置き換えます。

2 テストデバッガの概要と規則

テストデバッガの前提条件，概要，設定値，使用例などについて説明します。

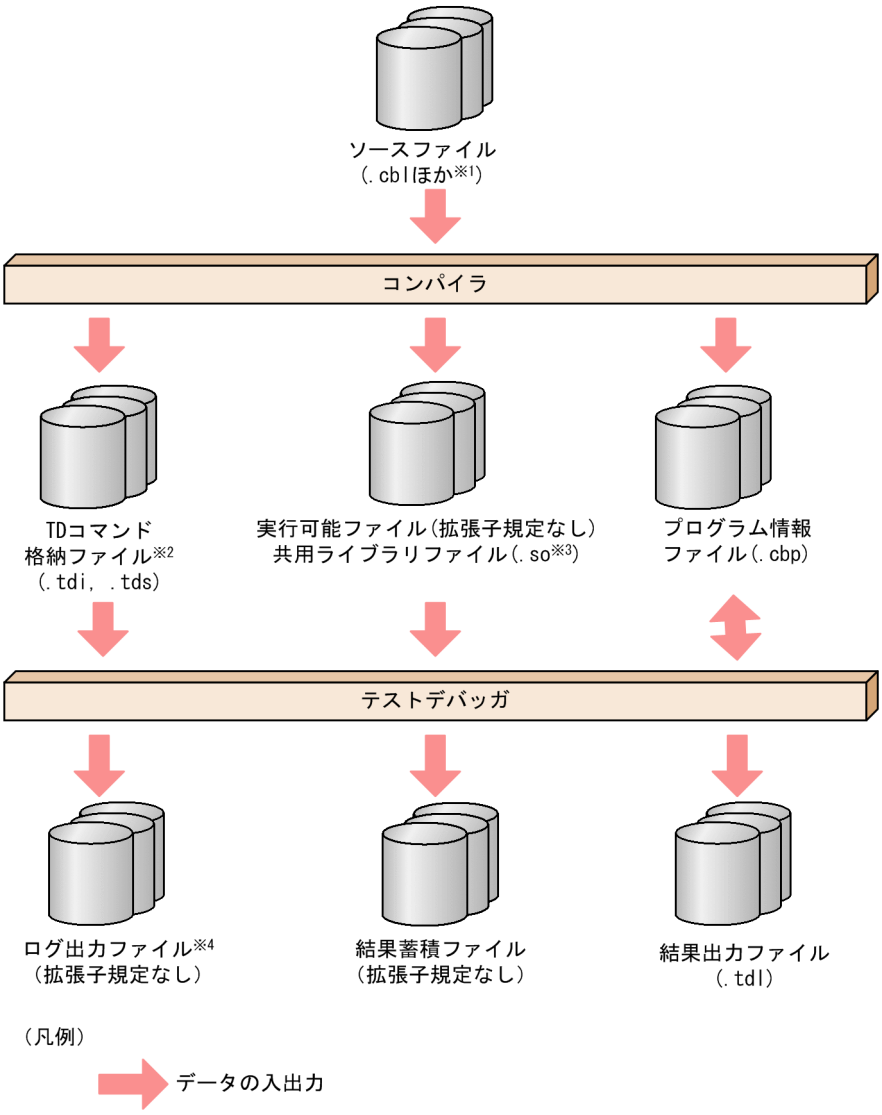
2.1　テストデバッガの前提条件

2.1.1　テストデバッガの入出力構成と使用するファイル

(1)　入出力構成

テストデバッガを使ったプログラムのテスト時の入出力構成を図 2-1 に示します。

図 2-1　テスト時の入出力構成



注※1

目的に応じて次の拡張子を使用します。

・固定形式正書法で書かれた原始プログラムをコンパイルする場合

.cbl，.CBL，.cob，.ocb，または環境変数 CBLFIX で指定した拡張子

・自由形式正書法で書かれた原始プログラムをコンパイルする場合

.cbf，.ocf，または環境変数 CBLFREE で指定した拡張子

注※2

コンパイラによる生成と，利用者による作成が可能です。

#INCLUDE コマンドで指定するときは，拡張子は任意に指定できます。

注※3

システムによって違いがあります。詳細は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注※4

ラインモードの場合だけ使用できます。

(2)　使用するファイル

テストデバッガを使ったテストで使用するファイルについて説明します。

- ソースファイル

  テキストエディタ（vi や FSED など）を使用して開かれるファイルです。

- 実行可能ファイル

  テストデバッグの対象となる実行可能ファイルです。

- 共用ライブラリファイル

  テストデバッグの対象となる共用ライブラリファイルです。共用ライブラリについては，「2.4.5　共用ライブラリ」を参照してください。

- プログラム情報ファイル

  テストデバッガで使用するプログラムの情報を格納するファイルです。テストデバッグのために，-TDInf コンパイラオプションを指定してコンパイルした場合，コンパイラはソースファイル名の拡張子を「.cbp」に変えたプログラム情報ファイルを出力します。

  プログラム情報ファイルは，使用するテストデバッガと同じバージョンのコンパイラで作成されたものでなければなりません。バージョンが異なる場合は，次に示すエラーメッセージが表示されます。

  KCCC4436T-E 異なるバージョンでコンパイルされたプログラム情報ファイルのため，対象としません。ファイル（xxxxx）

  シミュレーションのために，-SimMain コンパイラオプション，-SimSub コンパイラオプション，または-SimIdent コンパイラオプションと-DynamicLink コンパイラオプションを指定してコンパイルした場合，コンパイラは拡張子に「.cbs」を付けたプログラム情報ファイルを出力します。

  プログラム情報ファイルの詳細については，マニュアル「COBOL2002　使用の手引　手引編」の「主／副プログラムシミュレーションで生成する擬似プログラム用プログラム情報ファイル」の説明を参照してください。

- TD コマンド格納ファイル

  TD コマンドを記述するテキストファイルです。

  ラインモードでは，#INCLUDE コマンドで取り込むことができます。

バッチモードでは，入力ファイルとして用意する必要があります。バッチモードの入力として指定する場合の拡張子は，「.tdi」または「.tds」である必要があります。#INCLUDE コマンドで指定する場合は，ファイル名と拡張子は任意に指定できます。

- ログ出力ファイル

  端末に表示される内容を出力するファイルです。SET LOG コマンドで指定できます。バッチモードでは使用できません。ファイル名と拡張子は任意に指定できます。

- 結果蓄積ファイル

  TD コマンドの実行結果を出力するファイルです。SET PRINT コマンドでファイルを指定します。PRINT オペランドを持つ TD コマンドで PRINT を指定すると，そのコマンドの実行結果がファイルに出力されます。

- 結果出力ファイル

  バッチモードの入力となった TD コマンド格納ファイル中の TD コマンド操作によるメッセージおよび結果を出力するファイル（.tdl）です。

注意事項

- プログラム情報ファイルは，実行可能ファイルまたは共用ライブラリファイルのあるディレクトリを参照します。任意のディレクトリのプログラム情報ファイルを参照する場合は，環境変数 CBLPIDIR にディレクトリ名を指定します。
- 環境変数 CBLPIDIR を指定した場合のプログラム情報ファイルの検索順序は，「3.2.2　環境変数の指定」の注意事項を参照してください。

2.1.2　プログラムのコンパイル

テストデバッガを使用するためには，プログラムのコンパイル時に-TDInf コンパイラオプションを必ず指定します。また，必要に応じて次のコンパイラオプションもあわせて指定します。

-CVInf　-SimMain　-SimSub　-SimIdent　-DynamicLink

テストデバッガの機能を使用するために指定する COBOL2002 のコンパイラオプションを次に示します。コンパイラオプションの詳細については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

表 2-1　テストデバッガの機能とコンパイラオプション

テストデバッガの機能	コンパイラオプション
テストデバッガを使用する	-TDInf
カバレージを使用する	-CVInf
主プログラムシミュレーションを使用する	-SimMain（擬似主プログラムを生成する）
副プログラムシミュレーションを使用する	-SimSub，-SimIdent と-DynamicLink,Call

テストデバッガの機能	コンパイラオプション
	または-DynamicLink,IdentCall（擬似副プログラムを生成する）

ccbl2002 コマンド以外でリンクする場合に必要な，リンク時の指定を次に示します。

表 2-2　テストデバッガのためのリンク時の指定

対象	機能	リンク時の指定
実行可能ファイル	実行可能ファイルをテストデバッガの対象とする場合	指定しない

注※

-lcbl2ktd は，-lcbl2k よりも前に指定してください。

ccbl2002 を使用しないで，cc コマンドを使用する場合の使用例を次に示します。

(使用例)

実行可能ファイルを cc コマンドで作成する

AIX(32)の場合

```
cc TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -ldl -lm
```

AIX(64)の場合

```
cc -q64 TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -ldl -lm
```

Linux(x86)の場合

```
cc TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -ldl -lm
```

Linux(x64)の場合

```
cc TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k64/lib -lcbl2k -lcbl2kml -lm
```

共用ライブラリを ld コマンドで作成するときは，「2.4.5　共用ライブラリ」を参照してください。

注意事項

-Optimize,2 コンパイラオプションによる最適化の指定をして作成したプログラムでは，文が最適化されることがあり，テストデバッグ時にプログラムの実行順序を変更する操作によって，プログラムの動作が保証されない場合があります。また，中断点で参照できるデータ名が，最適化によって参照できない場合があります。

2.1.3　テストしたいプログラムの実行

テストデバッガの対象となるプログラムは，一つの実行可能ファイルと複数の共用ライブラリファイルで構成されます。

注意事項

- ld コマンドの-s オプションや strip コマンドなどで実行可能ファイルや共用ライブラリのシンボル情報を削除した場合，その実行可能ファイルや共用ライブラリは，テストデバッグの対象になりません。

  なお，Linux では，cbltd2k コマンドや cbltl2k コマンドの-Execute オプションに指定する実行可能ファイルは，テストデバッグの対象にしない場合でもシンボル情報が必要です。

- 共用ライブラリのロードは，COBOL プログラムの実行時の動作に従います。指定した共用ライブラリが，プログラムの実行時にロードされた場合に，テストデバッガの対象になります。ロードされなかった場合は，テストデバッガの対象になりません。

- テストデバッガでは，共用ライブラリがロードされた場合に次のチェックをして，指定された共用ライブラリとロードされた共用ライブラリが同じであるかを判断しています。次のチェックで，指定された共用ライブラリとは異なると判断された場合は，テストデバッガの対象になりません。

  チェック内容

  ・ロードされた共用ライブラリのファイルが共用ライブラリを検索するパスにある。

  共用ライブラリについては，「2.4.5　共用ライブラリ」を参照してください。

- STOP RUN 文を記述しない場合は，「COBOL プログラムの実行を終了しました」のメッセージが表示されません。

- テストデバッガを実行するシステム以外で作成された実行可能ファイルを対象にすると，KCCC4216T-S または KCCC4412T-E のメッセージが出力され，テストデバッガまたはカバレージが終了します。

- テストデバッガを実行するシステム以外で作成された共用ライブラリを対象にすると，KCCC4216T-S または KCCC4412T-E，および KCCC0331T-I のメッセージが出力され，テストデバッガの対象になりません。テストデバッガは続行します。

2.1.4　テストプログラムの構成

テストデバッグの対象となるプログラムの構成，プログラム定義，クラス定義，および関数定義の詳細については，マニュアル「COBOL2002　使用の手引　手引編」の翻訳グループについての説明を参照してください。

2.1.5　文字コード

環境変数 LANG を指定すると，テストデバッガまたはカバレージの入出力は，指定された文字コードに対応します。環境変数 LANG には，次の文字コード名が指定できます。

OS	文字コード名	出力メッセージの文字コード
AIX	Ja_JP	シフト JIS

OS	文字コード名	出力メッセージの文字コード
	ja_JP	EUC
Linux	ja_JP.UTF-8	UTF-8
	ja_JP.utf8	

注意事項

- 環境変数 LANG による文字コード名の指定は，コンパイル時とテストデバッガまたはカバレージの実行時で同じである必要があります。異なった環境変数 LANG を指定し，コンパイルして出力したプログラム情報ファイルは，テストデバッガではプログラム開始時または TD コマンド入力時にエラーになり，カバレージではプログラムの実行開始時にエラーになります。
- 環境変数 LANG に，日本語 EUC の文字コードを指定した場合，起動コマンドおよび TD コマンドの入力で，EUC の 3 バイトの外字は使用できません。
- Linux の場合，環境変数 CBLSRCENCODING に SJIS を指定する必要があります。

2.1.6　その他の注意事項

(1)　ファイルの書き込み時の制約

- 複数の ccbl2002 コマンドによって，同時にプログラムがコンパイルされた場合，コンパイラが生成する次のファイルは，あとから生成されたファイルが有効になります。
  - プログラム情報ファイル
  - TD コマンド格納ファイル
- 同時にカバレージ情報の蓄積をした場合，プログラムの終了後にプログラム情報ファイルに書き込みを同時にすると，エラーになる場合があります。

(2)　作業用ファイルについて

COBOL2002 では，テストデバッグ時に作業用ファイルを生成して使用しています。作業用ファイルは，次の表にあるディレクトリに，先頭に「CBLTD」，その後ろに任意の文字列が付いたファイル名で生成されます。テストデバッグ時に強制終了した場合など，作業用ファイルが正しく削除されないことがあります。そのときは，OS のコマンドなどを使用して削除してください。

OS	出力ディレクトリ
AIX	/tmp ディレクトリに生成されます。ただし，環境変数 TMPDIR を指定したときは，環境変数 TMPDIR に指定したディレクトリに生成されます。
Linux	/tmp ディレクトリに生成されます。

2.2.1　プログラムの中断

テスト中のプログラムの実行を任意に中断させることができます。中断させた状態で，次の操作をし，プログラムを調査できます。

- データの値を表示，変更する。
- プログラムの実行経路を表示する。

プログラムを中断させるには，次の方法があります。

- 中断点を設定する。
- データ監視条件を設定する。

また，ラインモードでは，次の場合にもプログラムを中断させます。

- プログラムで実行時エラーが発生した。
- ファイル管理記述項で指定した装置名にファイルの割り当てをしていないファイルをオープンした。

(1)　中断点の設定による中断

プログラムの実行を，任意の個所で中断させるには，中断点を設定します。中断点を設定すると，実行中のプログラムが中断点に達したときに，中断状態になります。中断中は，中断状態で実行できる TD コマンドだけを指定できます。処理が繰り返し実行されるような個所に中断点を設定する場合は，スキップ回数を指定できます。この指定によって，不要な中断回数を減らすことができ，効率良くプログラムをテストできます。

TD コマンドの場合，SET BREAK コマンドや RESET BREAK コマンドで，中断点を設定したり，解除したりできます。SET BREAK コマンドおよび RESET BREAK コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.1　SET BREAK／RESET BREAK（中断点の設定と解除）」を参照してください。

(2)　データ監視条件の設定による中断

データの値を監視し，その状態によってプログラムを中断できます。監視したいデータの値に条件式を指定することもできます。

TD コマンドの場合，SET WATCH コマンドや RESET WATCH コマンドで，データ監視条件を設定したり，解除したりできます。SET WATCH コマンドおよび RESET WATCH コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.3　SET WATCH／RESET WATCH（データ監視条件の設定と解除）」を参照してください。

データ監視条件の設定による中断の詳細は次のとおりです。

- データ値の変化の監視

  データ値が次の変化をしたときに，プログラムの実行が中断されます。

  - データ値が変化した。
  - アドレス名によって参照されるデータの場合，アドレス不正（アドレスが設定されていない，不正なアドレスが設定されている）によって参照できない状態から，参照できる状態に変化した。または，参照できる状態から，アドレス不正（アドレスが設定されていない，不正なアドレスが設定されている）によって参照できない状態に変化した。

- 比較条件式の監視

  比較条件式の判定結果が次のとき，プログラムの実行が中断されます。比較条件式の評価については，比較の規則「2.3.1　データの比較・代入規則」の「(2)　データの比較規則」を参照してください。

  - 比較条件式が，不成立の状態から，成立の状態に変わった（ただし，成立の状態からデータが変更されたとき，さらに成立の状態が継続しても，中断の状態とはならない）。
  - アドレス名によって参照されるデータの場合，アドレス不正（アドレスが設定されていない，不正なアドレスが設定されている）によって参照できない状態から参照できる状態に変化したときに，比較条件式が成立した。
  - 左辺・右辺の両方のデータが参照できるソース要素に制御が渡ったときに，比較条件式が成立した。

  （例）

  次のプログラムに，（A ＞ B）の比較条件式のデータ監視をすると一連番号 3000 の行で中断します。一連番号 7000 の行で比較条件式は成立しますが，すでに一連番号 3000 の行で成立していて，不成立な状態から成立な状態に変化したわけではないので中断されません。再び比較条件式が不成立な状態から成立の状態になり，中断されるのは一連番号 11000 の行となります。

プログラム

```
001000     MOVE  10 TO A.
002000     MOVE   5 TO B.
003000     ～
005000     ～
006000     MOVE  11 TO A.
007000     ～.
008000     MOVE   1 TO A.
009000     ～
010000     MOVE  10 TO A.
011000     ～
```

  データの監視範囲については，「2.3.2　データ項目・ファイル名が参照できる範囲」を参照してください。

注意事項

- COBOL プログラムの連絡節や局所場所節に定義されたデータ項目および特殊レジスタは，次に示す実行単位ごとに，それぞれ異なるデータとして監視されます。

  ・オブジェクト指向のインスタンス単位

  ・シングルスレッドやマルチスレッドなどのスレッド単位

・再帰できるソース要素※の再帰単位

注※

再帰できるソース要素とは次の COBOL プログラムを示します。

- RECURSIVE 指定を伴うプログラム
- 利用者定義関数
- メソッド

- プログラムの中断時に，データ監視条件を設定した場合，監視対象データが参照できればその位置から，参照できなければ参照できる実行時要素に制御が渡ったときにデータの監視が開始されます。

(3) 実行時エラーが発生した場合の中断

ラインモードの場合，実行時エラーが発生したときプログラムが中断します。実行時エラーが発生した文で，プログラムの状態を調べることができます。この中断のための設定は特に必要ありません。

次の条件がすべて当てはまるときに，プログラムが中断します。

- 実行時エラーのメッセージのレベルが S または U のとき，または，CBLABN サービスルーチンによってユーザプログラムが終了したとき（KCCC0900R-I のメッセージが表示されます）。
- プログラムが，実行時エラーによって中断したあとに，データ参照などデバッガの操作の対象にできる状態であるとき。
- -TDInf コンパイラオプションでコンパイルした翻訳単位で実行時エラーが発生したとき。

AIX の場合，実行時エラーによってユーザプログラムが中断したときは，次の手順でユーザプログラムを続行できます。

1. データの値を変えるなどのデバッガの操作によって実行時エラーの原因を解決する。
2. 実行時エラーの原因を解決したあとに，ジャンプまたはジャンプ実行によって，実行時エラーが発生した文を再び実行させる。

ただし，実行時エラーの種類によっては，ジャンプまたはジャンプ実行ができない場合があります。ジャンプまたはジャンプ実行をしないで，ユーザプログラムを続行したときは，ユーザプログラムが終了します。

2.2.2 プログラムの実行

プログラムの実行を制御する機能には次の種類があります。

(1) 連続実行

GO コマンドで，プログラムを連続して実行できます。

GO コマンドについては「5.4　TD コマンドの詳細」の「5.4.4　GO（実行の開始／再開）」を参照してください。

(2)　ステップイン

STEP IN コマンドで，プログラムを 1 文ずつ実行して中断できます。CALL 文，INVOKE 文および関数呼び出しをする文は，呼び出したプログラムに制御が渡って中断されます。

STEP IN コマンドについては「5.4　TD コマンドの詳細」の「5.4.5　STEP IN（ステップイン実行の開始／再開）」を参照してください。

(3)　ステップオーバー

STEP OVER コマンドで，プログラムを 1 文ずつ実行して中断できます。ただし，ステップインと異なり，CALL 文，INVOKE 文および関数呼び出しをする文は，呼び出したプログラムに制御が渡って中断されることはありません。

STEP OVER コマンドについては「5.4　TD コマンドの詳細」の「5.4.6　STEP OVER（ステップオーバー実行の開始／再開）」を参照してください。

注意事項

- 呼び出し先プログラムで，中断点の設定があるとき，データ監視の条件が成立したとき，割り込みが実行されたとき，および実行時エラーが発生したときは，これらの条件が起きた文でプログラムが中断されます。
- USE 手続きにプログラムの制御が渡る文でステップオーバーした場合は，USE 手続きの文では中断されないで，USE 手続きを終了した次の文で中断されます。
- プログラムの入口を示すものとして，PROCEDURE または ENTRY を使用します。プログラムの出口を示すものとして，END PROGRAM または END METHOD を使用します。省略された場合は空白行が挿入され，使用されます。

(4)　ジャンプ

STEP TO コマンドで，中断点と異なる文に中断位置をジャンプできます。次にプログラムの実行を再開したときは，移動した中断位置からプログラムが実行されます。

STEP TO コマンドについては「5.4　TD コマンドの詳細」の「5.4.7　STEP TO（ステップツー実行の再開）」を参照してください。

注意事項

- 中断点が属するソース要素の文へジャンプできます。手続き部に宣言部分があるときにできるジャンプは次のとおりです。

  ・宣言部分以外で中断しているとき

  宣言部分以外にジャンプできる。宣言部分にはジャンプできない。

・宣言部分で中断しているとき
中断点が属する節の中の文にジャンプできる。

- ソース要素の入口・出口および ENTRY 文へのジャンプはできません。
- -Optimize,2 コンパイラオプションによる最適化を指定してコンパイルしたプログラムでは文が最適化されることがあります。このため，文を選択してもジャンプ，ジャンプ実行ができないことがあります。また，プログラムの実行順序を変更するジャンプ操作をすると，その後のプログラムの動作が保証されない場合があります。

(5) ジャンプ実行

GO コマンドで，中断点と異なる文から，プログラムの実行を再開できます。

GO コマンドについては「5.4 TD コマンドの詳細」の「5.4.4 GO（実行の開始／再開）」を参照してください。

(6) 実行の強制終了

STOP コマンドで，プログラムの実行を強制的に終了できます。

STOP コマンドについては，「5.4 TD コマンドの詳細」の「5.4.8 STOP（プログラムの強制終了）」を参照してください。

(7) 実行の開始と開始時にできる指定

プログラムの実行は，次の操作で開始できます。

- 連続実行
- ステップイン
- ステップオーバー

プログラムが終了したあとも，上記の操作で，再びプログラムを開始させることができます。

プログラムの開始時には，次の指定ができます。

- プログラムへ渡す引数
- カバレージ情報の蓄積

プログラムへ渡す引数は，文字列を指定します。指定された文字列は，空白の区切りによって，複数の引数に分けられます。空白を含む文字列を一つの引数として指定するときは，引用符（"）で囲みます。

実行の開始と開始時にできる指定の詳細については，「5.4 TD コマンドの詳細」の「5.4.4 GO（実行の開始／再開）」を，ステップインについては，「5.4 TD コマンドの詳細」の「5.4.5 STEP IN（ステップイン実行の開始／再開）」を，ステップオーバーについては，「5.4 TD コマンドの詳細」の「5.4.6 STEP OVER（ステップオーバー実行の開始／再開）」を参照してください。

カバレージ情報の蓄積を指定すると，プログラムの実行中にカバレージ情報が蓄積され，プロセスが終了したときに，蓄積されたカバレージ情報がプログラム情報ファイルに蓄積されます。蓄積されたカバレージ情報は，カバレージの表示機能で表示できます。

カバレージ情報の表示については，「7.3　カバレージ情報の表示（テキスト形式）」および「7.4　カバレージ情報の表示（CSV 形式）」を参照してください。

注意事項

プログラムへ渡す引数は，C インタフェース形式（-Main,System コンパイラオプションまたは C 言語のプログラムから呼ばれるプログラム），および VOS3 形式（-Main,V3 コンパイラオプション）の二つがあります。

プログラムへ渡す引数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

使用例 1

C インタフェース形式の引数に，GO コマンドで値を設定します。

COBOL プログラム

```
01 ARGC PIC 9(8) USAGE COMP.
01 ARGV.
  02 ARGV1 ADDRESS.
  02 ARGV2 ADDRESS.
  02 ARGV3 ADDRESS.
PROCEDURE DIVISION USING BY VALUE ARGC BY REFERENCE ARGV.
             .
             .
             .
```

TD コマンド

- ARGC に 3，ARGV1 にプログラム名のアドレス，ARGV2 に文字列 APPLE のアドレス，ARGV3 に文字列 ORANGE のアドレスを設定します。

```
   GO PARAMETER('APPLE ORANGE')
```

- ARGC に 2，ARGV2 に文字列 APPLE ORANGE のアドレスを設定します。

```
   GO PARAMETER('"APPLE ORANGE"')
```

使用例 2

VOS3 形式の引数に，GO コマンドで値を設定します。

COBOL プログラム

```
01 PARM.
  02 PLEN  PIC S9(4) USAGE COMP.
  02 PCHAR PIC X(30).
PROCEDURE DIVISION USING PARM.
             .
             .
             .
```

TD コマンド

- PLEN に 5，PCHAR に APPLE を設定します。

```
GO PARAMETER('APPLE')
```

- PLEN に 12，PCHAR に APPLE ORANGE を設定します。

```
GO PARAMETER('"APPLE ORANGE"')
```

2.2.3　プログラムの状態の表示

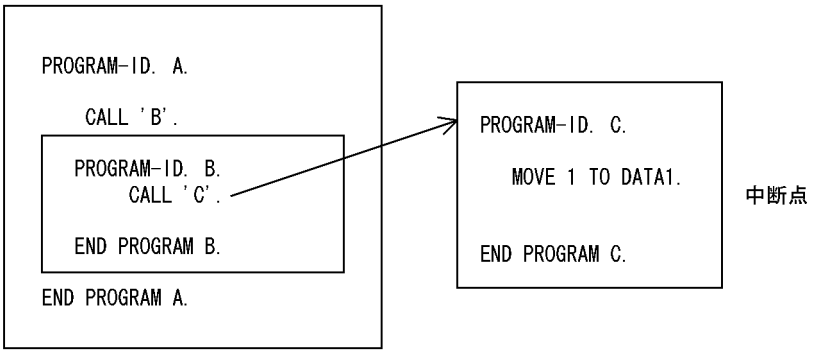
プログラムの状態を表示します。

(1)　呼び出し順序番号

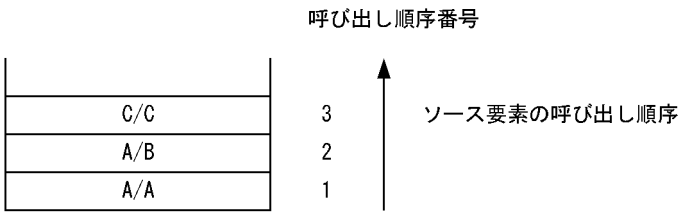
呼び出し順の番号を，呼び出し順序番号とします。

呼び出し順序番号は，最初に制御が渡る実行時要素を 1 として実行時要素が生成されるたびに一つずつ増加します。呼び出し元に戻るときは一つ減少します。

（例）



上記の順でプログラムが呼び出された場合，呼び出し順序番号は次のとおりになります。



(2)　現在位置の表示

ラインモードではプログラムの中断位置の情報を表示します。表示する形式については，「2.5　ソース要素・中断点・通過点の表示形式」を参照してください。

2.2.4　プログラムの実行の追跡

プログラムの実行経過を追跡して，その経路を表示します。

(1)　トレース表示

SET TRACE コマンド，または RESET TRACE コマンドで，プログラムの実行を追跡して，通過点を表示します。同時に，ソース要素の引数のデータ項目の値を表示することもできます。

SET TRACE コマンドおよび RESET TRACE コマンドについては「5.4　TD コマンドの詳細」の「5.4.9　SET TRACE／RESET TRACE（トレース表示の開始と中止）」を参照してください。

(2)　フロー情報の表示

SET FLOW コマンド，または RESET FLOW コマンドでプログラムの実行経路を示す通過点の情報を追跡して，プログラムの中断時にまとめて表示します。プログラムの実行時エラーが発生したときにも，実行時エラー発生までのプログラムの実行経路を表示できます。

SET FLOW コマンドおよび RESET FLOW コマンドについては「5.4　TD コマンドの詳細」の「5.4.10　SET FLOW／RESET FLOW（フロー情報の蓄積開始と終了）」を参照してください。

2.2.5　データの操作

COBOL プログラムのデータに対して，データの表示，データの代入，データの比較ができます。データの表示によって，プログラムの実行状態を確認できます。また，値を変更することによって，状態を変更してテストができます。

データの代入規則については「2.3.1　データの比較・代入規則」の「(1)　データの代入規則」を，データの比較規則については「(2)　データの比較規則」を参照してください。

(1)　データの表示

DISPLAY DATA コマンドについては，「5.4　TD コマンドの詳細」の「5.4.13　DISPLAY DATA（データの値表示）」を参照してください。

(a)　データ属性表示

データの属性に従い値を表示します。データの値が属性で表示できないときは，16 進数で表示されます。

(表示例)

- AIX(32)，Linux(x86)の場合

```
名　称　　データ1
値　　123
```

```
名　称　　データ2
エラー
*001E87C0  0000
```

- AIX(64)，Linux(x64)の場合

```
名　称　　データ1
値　　123

名　称　　データ2
エラー
*00000000004172F8  0000
```

(b)　16 進数表示

メモリに位置するアドレスとともに，データの値を 16 進数で表示します。アドレスは，先頭に”*”を付けて表示され，表示できないときは,”*-------- ” が表示されます。データの値は，4 バイト（8 けた）単位に区切って表示されます。

(表示例)

- AIX(32)，Linux(x86)の場合

```
名　称　　データ3
*001E87C0  31323334 353637
```

- AIX(64)，Linux(x64)の場合

```
名　称　　データ3
*00000000004172F8  31323334 353637
```

注意事項

表示されるデータのサイズは，32,767 バイトまでです。これを超えるときは，データの先頭から 32,767 バイトを表示します。32,768 バイト以降のデータは，DISPLAY DATA コマンドにデータ名の部分参照を指定することによって表示できます。

(2)　データの代入

ASSIGN DATA コマンドで，データに値を代入して，値を変更できます。

ASSIGN DATA コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.15　ASSIGN DATA (データの値代入)」を参照してください。

(3)　データの比較

IF コマンドを使用して，比較条件式によって，データとデータの比較，データと定数の比較ができます。比較条件式によって，TD コマンドの実行の条件を指定できます。また，比較条件式は，データ監視条件の設定にも使用できます。

IF コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.16　IF（データの値比較）」を参照してください。

2.2.6　領域の操作

ALLOCATE AREA コマンドを使用して，領域を確保し，そのアドレスをアドレス名，アドレスデータ項目またはポインタ項目に設定できます。また，任意のデータ項目のアドレスをアドレスデータ項目またはポインタ項目に設定できます。

この機能は，連動する別のプログラムで確保されるデータ領域を参照するテストを実行する場合，その別プログラムを動作させることなくデータ領域だけ確保したいときに使用します。確保した領域は，FREE AREA コマンドで解放します。

(1)　領域の確保・解放

アドレス名によって参照されるデータ項目と同じサイズの領域を確保して，アドレス名に設定できます。また，指定したサイズの領域を確保して，アドレスデータ項目，ポインタ項目に設定できます。

アドレス名に設定されているアドレスの領域が不要となったときは，これを解放できます。領域を解放したときは，アドレス名に ZERO が設定されます。

主プログラムシミュレーションのパラメタや，副プログラムシミュレーションの RETURNING にアドレス名を使用するとき，その領域を確保して値を設定できます。

なお，アドレスデータ項目，ポインタ項目も同様です。

ALLOCATE AREA コマンドおよび FREE AREA コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.17　ALLOCATE AREA／FREE AREA（領域の確保と解放）」を参照してください。

注意事項

FREE AREA コマンドが解放する領域は，ALLOCATE AREA コマンドで確保した領域です。このため，FREE AREA コマンドには，ALLOCATE AREA コマンドで確保した領域のアドレスが設定されているアドレス名を指定する必要があります。なお，アドレスデータ項目，ポインタ項目も同様です。

(2)　任意のデータ項目のアドレスの取得

任意のデータのアドレスをアドレス名，アドレスデータ項目，またはポインタ項目に設定できます。

アドレス名で参照されたデータ名のアドレスを取得した場合は，そのアドレス名が指すアドレスが設定されます。

ASSIGN ADDRESS コマンドの詳細については，「5.4.18　ASSIGN ADDRESS（アドレスの取得）」を参照してください。

2.2.7　プログラムの単体テスト

テストデバッガには，主，副プログラムシミュレーション機能，ファイルシミュレーション機能，DC シミュレーション機能などの単体テスト機能があります。単体テスト機能を使用すると，プログラムやファイルなどの実行環境が整わない段階でも，作成したプログラムを単体でテストできます。

テストデバッガでは，プログラムの単体テストをするために，次のシミュレーションをする機能があります。

- 主プログラムシミュレーション
- 副プログラムシミュレーション
- ファイルシミュレーション
- DC シミュレーション

(1)　主プログラムシミュレーション

SIMULATE MAIN コマンドで，ソース要素を呼び出す主プログラムをシミュレーションします。ソース要素に渡す引数に，TD コマンドで値を設定するなどの操作ができます。呼び出されるソース要素は，プログラム定義です。

SIMULATE MAIN コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.19　SIMULATE MAIN（主プログラムシミュレーションの設定）」を参照してください。

主プログラムシミュレーションをする手順を次に示します。

1. -SimMain コンパイラオプションを指定して，プログラムをコンパイルする。
2. テストデバッガを起動する。
3. テストデバッグ対象のソース要素を開始するときに必要な設定を SIMULATE MAIN コマンドで設定する。テストデバッグ対象のソース要素に渡す引数に値の設定ができる。
4. プログラムを実行して，テストデバッグの操作を開始する。

注意事項

SIMULATE MAIN コマンドで入口名を指定したとき，テストデバッグ対象プログラム中に SIMULATE MAIN コマンドで指定したプログラム名と同じ名前の別のプログラムがある場合，別のプログラムにも同じ入口名が指定された ENTRY 文が必要です。同じ入口名を持つ ENTRY 文がない場合は，エラーメッセージを出力して，主プログラムシミュレーションは設定されません。

主プログラムシミュレーションを設定する場合は，同じ名前を持ち，主プログラムシミュレーションの対象としないプログラムを次のどちらかの方法でデバッグ対象から外してください。

1. 主プログラムシミュレーションの対象としないプログラムは-TDInf コンパイラオプションを付けないでコンパイルしてください。
2. 主プログラムシミュレーションの対象としないプログラムが属する共用ライブラリを，-Library オプションで指定しないでください。

主プログラムシミュレーションの使用例を次に示します。

使用例 1

プログラム SUB01 の開始時に，データ名 PARAM01・PARAM02 に値を設定します。

```
SIMULATE MAIN( #PROG( SUB01 ) )
  ASSIGN DATA( PARAM01 ) VALUE( 1 )
  ASSIGN DATA( PARAM02 ) VALUE(100)
ENDSIMULATE
GO    *>プログラムの実行を開始する。
```

使用例 2

ENTRY 文 'ENTRYSUB'からプログラムの実行を開始します。

```
SIMULATE  MAIN( #PROG( SUB02 ) 'ENTRYSUB' )
  ASSIGN DATA( PARAM01 ) VALUE( 2 )
ENDSIMULATE
GO    *>プログラムの実行を開始する。
```

(2)　副プログラムシミュレーション

SIMULATE SUB コマンドで，テストデバッグ対象のソース要素が呼び出す副プログラムのインタフェースを TD コマンドでシミュレーションします。副プログラムに渡される引数の値を表示して確認したり，副プログラムが値を返す引数に値を設定したりできます。シミュレーションする副プログラムのソース要素は，プログラム定義です。

SIMULATE SUB コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.20　SIMULATE SUB (副プログラムシミュレーションの設定)」を参照してください。

副プログラムシミュレーションをする手順を次に示します。

1. 呼び出す副プログラムがない場合は，次のコンパイラオプションを指定してコンパイルする。

   ・-SimSub コンパイラオプションで呼び出すプログラム名を指定する。

   ・一意名指定の CALL 文によって副プログラムを呼び出す場合は，-SimIdent コンパイラオプションおよび-DynamicLink コンパイラオプションを指定する。

2. テストデバッガを起動する。

3. 副プログラムがする処理を SIMULATE SUB コマンドで設定する。副プログラムが，引数や RETURNING で返すデータ項目に値の設定ができる。

4. プログラムを実行して，テストデバッグの操作を開始する。

呼び出す副プログラムのソース要素がある場合は，-SimSub コンパイラオプションを指定してコンパイルする必要はありません。存在する副プログラムの処理を，SIMULATE SUB コマンドの設定によって，TD コマンドの処理に置き換えて実行させることができます。

CALL 文の引数と RETURNING のデータ項目は，記号名を割り当ててシミュレーションの手続きで参照できます。複数の CALL 文で異なるデータ名の引数や RETURNING が指定されている場合でも，記号名を割り当てることで，同一の名称によって，シミュレーションの手続きで参照できます。

副プログラムシミュレーションの実行回数を，カウンタ変数によってカウントし，参照できます。

注意事項

- 再帰属性の付いたプログラムが自プログラムを呼び出している場合，そのプログラム中に中断している状態で，自プログラムに対しての副プログラムシミュレーションは設定できません。
- SIMULATE SUB コマンドの TD コマンド群中に書かれた ASSIGN DATA コマンドによって，特殊レジスタ RETURN-CODE に設定した値は，シミュレーション対象の副プログラムに RETURNING 指定がない場合だけ，副プログラムを呼び出すプログラムの RETURN-CODE に戻されます。
- ソース要素がある副プログラムにシミュレーションを設定しても，呼び出すプログラムと副プログラムの間の引数および返却項目に矛盾があると，COBOL プログラムが異常終了します。異常終了を避けるためには，環境変数 CBLPRMCHKW を設定してください。

副プログラムシミュレーションの使用例を次に示します。

使用例 1

プログラム PROG1 をシミュレーションします。

引数 X，Y に記号名 A，B を対応させます。

COBOL プログラム

- データ定義

```
01 X  PIC X(10).
01 Y.
    02 Y1  PIC X(10).
    02 Y2  PIC X(10).
01 RTN.
    02 R-NAME.
       03 N-DATA  PIC X(10).
```

- 呼び出し文

```
CALL  'PROG1'  USING  X,Y  RETURNING  RTN.
```

TD コマンド

```
 SIMULATE SUB( #PROGRAM(PROG1) )  USING(A,B)  RETURNING(R)
   DEFINE
     01 A
     01 B
       02 B1
       02 B2
     01 R
       02 R1
         03 R11
```

```
    ENDDEFINE
    IF CONDITION(A=1)
      ASSIGN DATA( B1 )  VALUE('APPLE')
      ASSIGN DATA( B2 )  VALUE ('PANDA')
      ASSIGN DATA( R11 )  VALUE ('NORMAL')
    ELSE
      DISPLAY  DATA ( A )
      ASSIGN DATA( R11 )  VALUE ('ABNORMAL')
    ENDIF
  ENDSIMULATE
```

(3)　ファイルシミュレーション

SIMULATE FILE コマンドで，プログラムから入出力するファイルがないときに，入出力文によるインタフェースをシミュレーションできます。

SIMULATE FILE コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.21　SIMULATE FILE (ファイルシミュレーションの設定)」を参照してください。

(a)　ファイル入出力文のシミュレーション

入力文のシミュレーションでは，ファイルのレコード領域への値の設定ができます。出力文のシミュレーションでは，レコード領域の内容を表示できます。また，入出力文で，擬似的に入出力条件を発生させることができます。

ファイルシミュレーションは，ファイル結合子とオープンモード（INPUT，OUTPUT，I-O，EXTEND）の単位に，シミュレーションでする手続きを指定します。さらに，入出力文ごとに手続きを指定できます。

ファイルシミュレーションの実行回数を，カウンタ変数によってカウントし，参照できます。

注意事項

- ファイルシミュレーションは，実体ファイルが割り当てられているときでも指定できます。シミュレーションの手続きが実行されると，実体ファイルへのアクセスは行われません。
- INTO の指定がある READ 文は，シミュレーションの手続きの実行後に，READ 文によるデータが転記されます。レコード領域に代入した値が書き換えられるため，注意が必要です。
- SORT 文，MERGE 文の USING 指定，GIVING 指定で指定するファイルは，READ 文，WRITE 文としてシミュレーション対象となります。ただし，USING 指定に指定されたファイルは GO END コマンドを指定しないかぎりシミュレーションが終わりません。
- ファイルシミュレーションは，シミュレーションを設定したあとの OPEN 文で開始します。ファイルを OPEN したあとにファイルシミュレーションを指定した場合は，ファイルが CLOSE されたあと，再度，OPEN 文が実行されたときにシミュレーションは開始します。
- ファイルシミュレーションの設定と，ファイル名とオープンモードが合致する OPEN 文で，ファイルシミュレーションは開始します。EXTERNAL 句指定のあるファイルでは，同じシミュレーショ

ン手続きが実行されます。EXTERNAL 句指定がないファイルでは，ファイルごとに別のシミュレーション手続きが実行されます。

- シミュレーションが設定されているファイルに対して，再度シミュレーションを設定した場合は，設定されていた TD コマンド群を置き換えます。このとき，カウンタ変数および REPEAT 回数は初期値に戻ります。
- EXTERNAL 句指定のファイルのシミュレーションでは，OPEN 文と入出力文が記述された翻訳単位の両方を-TDInf コンパイラオプションでコンパイルします。OPEN 文または入出力文のどちらか記述された翻訳単位が-TDInf コンパイラオプションでコンパイルされていないときは，シミュレーションをしません。

使用例 1

カウンタ変数を使用して，カウンタ変数の値ごとにレコードの値を設定できます。

```
SET QUALIFICATION( #PROG( 社員/社員 ) )
 SIMULATE FILE( 給与ファイル ) OPENMODE( IO ) COUNTER(CNT)
   IF CONDITION(CNT = 1)
      ASSIGN DATA(基本給) VALUE(100000)
   ENDIF
   IF CONDITION(CNT=2)
      ASSIGN DATA(基本給) VALUE(101000)
   ENDIF
   IF CONDITION(CNT=3)
      ASSIGN DATA(基本給) VALUE(104000)
   ENDIF
 ENDSIMULATE
```

同じレコードの値を連続して設定する場合は，反復回数指定を使用できます。

使用例 2

繰り返し指定を使用して，同じレコードの値を連続して設定できます。

```
 SET QUALIFICATION( #PROG( 社員/社員 ) )
 SIMULATE FILE( 給与ファイル ) OPENMODE(IO)
   REPEAT  TIMES(2)
      ASSIGN DATA(基本給) VALUE(100000)
   ENDREPEAT
   REPEAT
      ASSIGN DATA(基本給) VALUE(101000)
   ENDREPEAT
   REPEAT  TIMES(3)
      ASSIGN DATA(基本給) VALUE(104000)
   ENDREPEAT
 ENDSIMULATE
```

繰り返し指定（REPEAT）は，指定された反復回数（TIMES）の回数分を入出力文に対して実行します。指定した反復回数分の実行が終了すると，次の繰り返し指定に移ります。

- 反復回数を省略した場合は，TIMES(1)が仮定されます。
- 繰り返し指定が終了しても，まだ，入出力文が現れた場合は，最後の繰り返し指定が実行されます。

OPEN 文のオープンモードが I-O の場合の入出力文は，複数あります。この場合は入出力文単位にレコード値を設定できる入出力文選択指定を使用すると便利です。

使用例 3

入出力文を指定します。

```
SET QUALIFICATION( #PROG( 社員/社員 ) )
SIMULATE FILE( 給与ファイル )
    OPENMODE(IO)
  SELECT ACTION(DELETE)
     REPEAT TIMES(2)
       DISPLAY DATA(名前)
     ENDREPEAT
  ENDSELECT
  SELECT ACTION(READ)
     REPEAT
       ASSIGN DATA(基本給) VALUE(104000)
     ENDREPEAT
     REPEAT TIMES(3)
       ASSIGN DATA(基本給) VALUE(108000)
   ENDREPEAT
ENDSELECT
ENDSIMULATE
```

文種別単位に処理を繰り返すことができます。文種別指定は SELECT を使用して指定します。

- SELECT 指定のない入出力文で制御が渡った場合，この入出力文に対してはコマンドが実行されません。例えば上の例のようにシミュレーション手続きを実行した場合，START 文の文種別指定がないのでシミュレーションが実行されないで COBOL プログラムが続行します。
- 各文種別単位に反復回数がなくなったときは，使用例 2 のように最後の REPEAT を繰り返します。
- 同じ SELECT 指定が現れた場合は，繰り返し（REPEAT）が設定順に追加されます。

(b) 入出力条件のシミュレーション

ファイルシミュレーションの手続き中で，次の入出力条件を擬似的に発生させることができます。

表 2-3 シミュレーション対象入出力条件

コマンド	シミュレーション対象入出力条件	入出力文
GO END	ファイル終了条件 (END OF FILE)	READ
GO EOP	ページ終了条件 (END OF PAGE)	WRITE
GO INVALID	無効キー条件 (INVALID KEY)	READ，WRITE， REWRITE， START，DELETE
GO ERROR	入出力誤り	READ，WRITE， REWRITE， START，DELETE

(c)　論理誤りチェックと入出力状態の値

ファイルシミュレーションの手続きで，入出力状態の値（FILE STATUS 句で指定されたデータ名の値）に任意の値を設定できます。次の場合は，入出力状態の値にテストデバッガが値を設定します。この場合も，ASSIGN DATA コマンドによって値を変更できます。

ファイルの入出力処理の論理誤りをチェックして，エラーがあった場合，シミュレーションは実行しません。シミュレーション対象ファイルに入出力状態の値（FILE STATUS 句で指定されたデータ名の値）の指定があるときは，シミュレーションの終了状態によって「表 2-4　論理誤りチェック条件」の値を入出力状態の値に設定します。

また，シミュレーション手続きによって「表 2-3　シミュレーション対象入出力条件」で示したコマンドが実行されたときは，「表 2-5　論理誤りチェック以外の設定条件」で示した値が設定されます。

シミュレーション手続きで指定がない場合は，「表 2-6　成功完了の設定条件」の成功完了時の入出力状態の値が設定されます。

表 2-4　論理誤りチェック条件

入出力状態の値	内容
41	開かれているファイル結合子に対して OPEN 文を実行しようとした。
46	GO END コマンド実行後に再度 READ 文を実行しようとした。
47	入力モード（INPUT）や入出力モード（I-O）で開かれていないファイル結合子に対して READ 文や START 文を実行しようとした。
48	正しいオープンモードでは開かれていないファイル結合子に対して WRITE 文を実行しようとした。順アクセス法の場合，ファイル結合子が出力モード（OUTPUT）や拡張モード（EXTEND）で開かれていない。動的アクセス法や乱アクセス法の場合，ファイル結合子が出力モード（OUTPUT）や入出力モード（I-O）で開かれていない。
49	入出力モード（I-O）で開かれていないファイル結合子に対して REWRITE 文や DELETE 文を実行しようとした。

表 2-5　論理誤りチェック以外の設定条件

入出力状態の値	内容
10	GO END コマンド実行による終了。
23	GO INVALID コマンド実行による終了（相対ファイル，索引ファイル）。
34	GO INVALID コマンド実行による終了（順ファイル）。
90	GO ERROR コマンド実行による終了。または，論理誤りが発生した。

表 2-6　成功完了の設定条件

入出力状態の値	内容
00	正常終了。一般のコマンド（入出力状態の値を設定するコマンドは除く）で終了。

入出力状態の値	内容
07	REEL／UNIT の CLOSE 文を実行した（CLOSE 処理を行わない。OPEN 状態を残す）（順ファイル）

注意事項

入出力状態の値は，ファイルシミュレーションが開始されたとき，00 に初期化されます。

(4) 記号名

記号名は，次のシミュレーションでデータ項目の名前を置き換えるときに使用します。

- 副プログラムシミュレーション

  CALL 文の呼び出し文の引数に対して，USING と RETURNING で指定するデータ項目を記号名の名前で置き換えて参照できます。

- ファイルシミュレーション

  レコード記述項のデータ項目を記号名の名前で置き換えて参照できます。ファイル記述項に EXTERNAL 句を指定し，複数の翻訳単位から参照されるファイルのシミュレーションで，翻訳単位ごとにレコード記述項の名前に違いがあるときでも，記号名を対応させることによって同一の名称でレコード領域を参照できます。

記号名で置き換えるデータ項目が集団項目の場合は，DEFINE オペランドによって，基本項目をレベル番号と構造定義で指定できます。基本項目を参照しないなど，必要がないときは，DEFINE オペランドの指定は不要です。

レベル番号の指定については，「5.4　TD コマンドの詳細」の「5.4.26　レベル番号（記号名構造指定）」を参照してください。

注意事項

- 記号名の構造定義は参照が必要な部分までを記述すればよく，原始プログラムのデータ項目の残りのレベル番号は，指定しなくてもかまいません。ただし，途中の引数，レコード記述項を省略して定義することはできません。
- 記号名の指定を省略した場合，データ値の表示や，エラーが発生したときのメッセージ中に記号名を表示できません。このため記号名の代わりに「*****」と表示します。記号名の表示が必要なときは，記号名を記述してください。

（例）

A1 までが記号名に対応し，A11 以降が無視されます。

COBOL プログラム

- データ定義

```
01 A.
   02 A1.
      03 A11 PIC X(10).
```

```
          03 A12 PIC X(10).
       02 A2.
```

- 呼び出し文

```
   CALL 'SUB1' USING A.
```

TD コマンド

```
   SIMULATE SUB(#PROGRAM(SUB1)) USING(P)
     DEFINE
       01 P          *>  Aに対応する。
         02 P1       *>  A1に対応する。
     ENDDEFINE
       :
   ENDSIMULATE
```

- 記号名の定義は，01 から始まらなければなりません。構造が原始プログラムと合っていれば，レベル番号を一致させる必要はありません。

（例）

P1 は A1 に対応づけられます。

COBOL プログラム

- データ定義

```
   01 A.
      02 A1.
         03 A11 PIC X(10).
         03 A12 PIC X(10).
      02 A2 PIC X(10).
```

- 呼び出し文

```
   CALL 'SUB2' USING A.
```

TD コマンド

```
   SIMULATE SUB(#PROGRAM(SUB2)) USING(P)
     DEFINE
       01 P          *>  Aに対応する。
         04 P1       *>  A1に対応する。
     ENDDEFINE
       :
   ENDSIMULATE
```

- データ項目の数より記号名の数が多いとき，対応するデータ項目のない記号名を使用した TD コマンドは実行できません。

（例）

P2 に対応するデータ項目の定義がありません。

COBOL プログラム

- データ定義

```
01 A.
   02 A1 PIC X(10).
```

- 呼び出し文

```
CALL 'SUB3' USING A.
```

TD コマンド

```
SIMULATE SUB(#PROGRAM(SUB3)) USING(P)
  DEFINE
   01 P             *>  Aに対応する。
     02 P1          *>  A1に対応する。
     02 P2          *>  対応するデータ項目がない。
  ENDDEFINE
    :
 DISPLAY DATA(P)     *> 実行される。
 DISPLAY DATA(P1)    *> 実行される。
 DISPLAY DATA(P2)    *> 実行できない。
    :
ENDSIMULATE
```

- 記号名を対応づけるデータ項目に TYPE 句または SAME AS 句が指定されているときは，記号名の構造定義には，TYPE 句または SAME AS 句によって展開された構造による指定をします。

（例）

P3 は A1 に，P4 は A2 に対応づけられます。

COBOL プログラム

- データ定義

```
01 TYPE1 IS TYPEDEF.
  02 A1  PIC X(1).
  02 A2  PIC X(1).

01 U1.
  02 PRM1  PIC X(1).
  02 PRM2  TYPE TO TYPE1.
```

- 呼び出し文

```
CALL 'SUB4' USING U1.
```

TD コマンド

```
SIMULATE SUB(#PROGRAM(SUB4)) USING(P)
  DEFINE
   01 P             *>  U1に対応する。
     02 P1          *>  PRM1に対応する。
     02 P2          *>  PRM2(TYPE1)に対応する。
       03 P3        *>  A1に対応する。
       03 P4        *>  A2に対応する。
  ENDDEFINE
    :
ENDSIMULATE
```

使用例 1

副プログラムシミュレーションの引数に記号名を設定します。

COBOL プログラム

- データ定義

```
01 X.
  02 X1 OCCURS 5.
    03 X11 PIC X(2).
  02 X2 PIC X(2).
01 Y  PIC 9.
01 RTN.
  02 R OCCURS 5 PIC X(10).
```

- 呼び出し文

```
CALL  'PROG1'  USING  X,X11(2), Y   RETURNING  R(1).
```

TD コマンド

```
SIMULATE SUB(#PROGRAM(PROG1)) USING(A,B,C)  RETURNING(R)
  DEFINE
     01 A               *>  Xに対応する。
      02 A1             *>  X1に対応する。
       03 A11           *>  X11に対応する。
      02 A2             *>  X2に対応する。
  ENDDEFINE
  IF CONDITION( C=1 )
    ASSIGN DATA(A11(1)) VALUE('OK')
    ASSIGN DATA(R) VALUE('normal')
  ELSE
    ASSIGN DATA(B) VALUE('NG')
    ASSIGN DATA(R) VALUE('abnormal')
  ENDIF
ENDSIMULATE
```

使用例 2

ファイルシミュレーションのレコードに記号名を設定します。

COBOL プログラム

```
  FD 給与ファイル DATA RECORD IS 形式1.
  01 形式1.
   02 社員コード PIC X(9).
   02 氏名.
    03 名字  PIC N(10).
    03 名前  PIC N(10).
  01 形式2.
   02 契約コード.
    03 種別 PIC X(9).
    03 コード PIC X(40).
  01 形式3 PIC X(49).
```

TD コマンド

```
SIMULATE FILE(給与ファイル)  OPENMODE(IO)  RECORD(A,B,C)
  DEFINE
    01 A
     02 A1
     02 A2
      03 A21
      03 A22
    01 B
     02 B1
      03 B11
      03 B12
  ENDDEFINE
      :
ENDSIMULATE
```

使用例 3

ファイルシミュレーションのレコードに記号名を設定します。

COBOL プログラム

```
FD 給与ファイル.
01 形式1.
 02 社員コード PIC 9(8).
 02 氏名.
   03 名字 PIC X(30).
   03 名前 PIC X(30).
01 形式2.
 02 契約コード.
   03 種別 PIC 99.
   03 コード PIC 9(4).
01 形式3 PIC X(80).
01 形式4 PIC X.
```

TD コマンド

```
SIMULATE FILE(給与ファイル) OPENMODE(IO) RECORD(A,B,C)
   DISPLAY DATA(B)
   :
ENDSIMULATE
```

RECORD 指定で対応づけをします。使用例 2 および使用例 3 の場合，形式 1 が A，形式 2 が B，形式 3 が C に対応します。対応づけた記号化名称をコマンド中でデータの代わりに使用できます。

- B 指定だけを省略するなど，途中の省略はできません。後ろは省略できます。
- レコードが構造定義のとき，構造内を参照するために DEFINE～ENDDEFINE を使用して記号名を定義できます。

(5) DC シミュレーション

SIMULATE DC コマンドによって，次の DC 文をシミュレーションします。DC 文を使用した OpenTP1 のインタフェースをシミュレーションすることもできます。SIMULATE DC コマンドの詳細については，

「5.4　TD コマンドの詳細」の「5.4.24　SIMULATE DC（DC シミュレーションの設定）」を参照してください。

- COMMIT 文
- ENABLE 文
- ROLLBACK 文
- DISABLE 文
- RECEIVE 文
- SEND 文

対象とする DC 文の種別と通信記述名を指定して，シミュレーションで実行する手続きを TD コマンドで記述します。手続きでは，メッセージファイル領域の参照，値の設定ができます。また，状態コード（STATUS KEY 句で指定したデータ項目）に対して，値を設定できます。

DC シミュレーションの実行回数を，カウンタ変数によってカウントし，参照することもできます。

注意事項

DC シミュレーションは，プログラムの実行開始前でも実行中でも指定できます。DC シミュレーション手続き設定後の DC 文からシミュレーションが実行されます。

使用例 1

DISABLE 文のシミュレーションをします。

```
SIMULATE DC( DISABLE )  CDNAME(CD1)
  ASSIGN DATA( STATUS-KEY ) VALUE( 1 )
ENDSIMULATE
```

使用例 2

COMMIT 文のシミュレーションをします。

```
SIMULATE  DC( COMMIT )
  ASSIGN DATA( STATUS-KEY ) VALUE( 2 )
ENDSIMULATE
```

2.2.8　バッチによるテスト

(1)　テストケース

バッチモードでは，TD コマンドによる処理を，テストケースに分けて記述できます。記述した複数のテストケースは，順番に実行されます。次の方法によって，テストケースを選択して実行することもできます。

- ケース識別子の指定
- ケースコードによるケース選択条件の判定

TEST コマンドは，テストケースの集まりを定義します。CASE コマンドは，ケース識別子による名前と，一連の処理を実行するための TD コマンドの集まりによるテストケースを指定します。

TEST コマンドについては「5.4　TD コマンドの詳細」の「5.4.27　TEST（テストケースの選択）」を，CASE コマンドについては「5.4　TD コマンドの詳細」の「5.4.28　CASE（テストケースの指定）」を参照してください。

一つのテストケースの実行が終了したとき，次に示すテストデバッガの機能が設定されていた場合，テストケースの設定が解除されます。解除されたテストケースの設定は，次のテストケースには引き継がれません。プログラムが実行中のときは，強制的に終了されます。テストケースを複数実行している場合でも，データなどは次のテストケースに引き継がれません。

- 中断点
- データ監視条件
- トレース表示
- フロー情報の蓄積
- ファイルシミュレーション
- DC シミュレーション
- 主プログラムシミュレーション
- 副プログラムシミュレーション
- SET QUALIFICATION

(2)　ケース識別子の指定

ケース識別子の指定によって，テストケースを選択して実行できます。ケース識別子の末尾に*を付けて指定すると，*より前の文字列が一致するケース識別子のテストケースが選択されて，実行されます。

ケース識別子が指定されている場合にケース識別子の指定がないテストケースは実行されません。

使用例 1

指定されたテストケースから，TEST1 と TEST3 を選んで実行します。テストケースは CASE コマンドが指定された順に実行されます。TEST コマンドの CASEID オペランドに指定したケース識別子の順序は，テストケースの実行順序にはかかわりません。

```
TEST CASEID( TEST1, TEST3 )
  CASE ID(TEST1)              *> ケース識別子TEST1のテストケース
    SET BREAK ST(100)         *> 実行される
    GO
    DISPLAY DATA( RETURN-CODE )
    IF CONDITION(RETURN-CODE NOT= 0)
      ASSIGN CASECODE( 8 )
    ENDIF
    GO
  ENDCASE

  CASE ID(TEST2)              *> ケース識別子TEST2のテストケース
    GO                        *> TESTコマンドで指定されていないため,
 ENDCASE                      *> 実行されない

  CASE ID(TEST3)              *> ケース識別子TEST3のテストケース
    SET TRACE                 *> 実行される
    GO
  ENDCASE


  CASE                        *> ケース識別子の指定がないテストケース
    GO                        *> 実行されない
  ENDCASE

ENDTEST
```

使用例 2

CASEID オペランドに TEST*を指定することによって，TEST で始まるケース識別子のテストケースを実行します。

```
TEST CASEID( TEST* )
  CASE ID(TEST1)              *> ケース識別子TEST1のテストケース
    ASSIGN CASECODE( 0 )      *> 実行される
  ENDCASE

  CASE ID(TEST2)              *> ケース識別子TEST2のテストケース
    :                         *> 実行される
  ENDCASE

  CASE ID(SAMP1)              *> ケース識別子SAMP1のテストケース
    :                         *> 実行されない
  ENDCASE
ENDTEST
```

(3)　ケースコードによる条件判定

ASSIGN CASECODE コマンドによって，テストケースの実行結果をケースコードに設定できます。ケース選択条件は，このケースコードの値を条件判定して，あとに続くテストケースの実行を決めます。

ASSIGN CASECODE コマンドについては「5.4　TD コマンドの詳細」の「5.4.29　ASSIGN CASECODE（ケースコードの設定）」を参照してください。

ケース選択条件は，次のどちらかを，<，>，<=，>=，=，NOT= の比較演算子によって比較します。

- 最後に設定されたケースコード（#LASTCODE）と定数
- ケース識別子のケースコードと定数

ケース選択条件に複数の条件式を指定すると，すべての条件を満たしたときに，テストケースを実行します。

使用例 1

最後に設定されたケースコードによって，テストケースを実行します。

```
 TEST
   CASE ID(TEST1)
    :
   IF CONDITION(RETURN-CODE NOT= 0)
     ASSIGN CASECODE( 8 )                  *> ケースコードに8を設定する
     ENDIF
   ENDCASE

   CASE ID(TEST2) CONDITION( #LASTCODE=0 )  *> TEST1のケースコードが
                                            *>   8のときは，実行されない
    :
   ENDCASE

   CASE ID(TEST3) CONDITION( #LASTCODE=8 )  *> TEST1のケースコードが
                                            *> 8のとき，実行される
    :
   ENDCASE
 ENDTEST
```

使用例 2

直前に実行されたテストケースがケースコードに値を設定しないとき，ケースコードによるケース選択条件が指定されたテストケースは実行されません。

```
 TEST
   CASE ID(TEST1)             *> テストケースを実行する
     ASSIGN CASECODE( 8 )     *> ケースコードに8を設定する
   ENDCASE

   CASE ID(TEST2)              *> 実行する
      :                        *> ケースコードに値を設定しない
   ENDCASE

   CASE ID(TEST3) CONDITION ( #LASTCODE = 8 )       *> TEST2が
                                                    *> ケースコードの
                                                    *> 設定をしないため
      :                                             *> 実行されない
   ENDCASE
ENDTEST
```

使用例 3

最初のテストケースにケースコードによるケース選択条件を指定すると，そのテストケースは実行されません。

```
 TEST
   CASE ID(TEST1) CONDITION( #LASTCODE = 8 )      *> 実行されない
     ASSIGN CASECODE( 8 )
   ENDCASE
 ENDTEST
```

使用例４

同一のケース識別子を持つテストケースが複数あるときは，最後に実行されたテストケースのケースコードを判定します。ケース選択条件に指定されたケース識別子のテストケースがまだ実行されていないときは，判定が偽となり，実行されません。

```
TEST
  CASE ID(TEST1)                          *> 実行される
    ASSIGN CASECODE( 4 )                  *> ケースコードに4を設定する
  ENDCASE

  CASE ID(TEST1)                          *> 実行される
    ASSIGN CASECODE( 8 )                  *> ケースコードに8を設定する    ★1
  ENDCASE

  CASE ID(TEST2) CONDITION(TEST1 = 8)     *> ★1で設定されたケースコードを判定
        :                                 *> 実行される
    ASSIGN CASECODE( 0 )                  *> ケースコードに0を設定する    ★2
  ENDCASE

  CASE ID(TEST2) CONDITION(TEST3 = 8)     *> TEST3はまだ実行されていない
    ASSIGN CASECODE( 4 )                  *> 実行されない
  ENDCASE

  CASE ID(TEST3) CONDITION(TEST2 = 0)     *> ★2で設定されたケースコードを判定する
        :                                 *> 実行される
    ASSIGN CASECODE( 3 )                  *> ケースコードに3を設定する    ★3
  ENDCASE

  CASE CONDITION(TEST3 = 3)               *> ★3で設定されたケースコードを判定する
        :                                 *> 実行される
    ASSIGN CASECODE( 5 )                  *> ケースコードに5を設定する    ★4
  ENDCASE

  CASE CONDITION(#LASTCODE = 5)           *> ★4で設定されたケースコードを判定する
        :                                 *> 実行される
  ENDCASE
ENDTEST
```

使用例５

TEST コマンドで選択され，かつ，ケース選択条件の判定が真のときに，テストケースは実行されます。

```
TEST  CASEID(TEST1,TEST3)    *> TEST1, TEST3の実行を指定する ★1
  CASE ID(TEST1)             *> 実行される
    :
  ENDCASE

  CASE ID(TEST2)             *> 実行されない
    :
  ENDCASE

  CASE ID(TEST1)             *> 実行される
    ASSIGN CASECODE( 0 )     *> ケースコードに0を設定する  ★2
  ENDCASE

  CASE ID(TEST2) CONDITION( #LASTCODE = 0 ) *> ★2で設定されたケースコードを判定する
    :                                       *> ケース選択条件は成立するが,
    :                                       *> ★1のTESTコマンド選択されていないため,
  ENDCASE                                   *> 実行されない

  CASE ID(TEST3) CONDITION( #LASTCODE = 4 ) *> ★2で設定されたケースコードを判定する
    :                                       *> ケース選択条件は成立しない
    :                                       *> ★1のTESTコマンド選択されているが
  ENDCASE                                   *> 実行されない
ENDTEST
```

注意事項

TEST コマンドを複数記述したとき，前の TEST コマンドのテストケースで設定されたケースコードは，次の TEST コマンドのケース選択条件では参照されません。

2.2.9　テストデバッグのための設定

テストデバッグをするための設定について説明します。

(1)　入力する文字の扱い（等価規則）

#OPTION コマンドで，入力する文字の扱い（等価規則）を変更できます。詳細については，「5.4　TD コマンドの詳細」の「5.4.34　#OPTION（オプションの変更）」を参照してください。

TD コマンドに使用される文字コードでは，英数字文字は 1 バイトの標準コードに，拡張文字は 2 バイトの拡張コードになります。標準コードの文字を英数字文字（例えば大文字 A と小文字 a）と呼び，拡張コードの文字を拡張文字（例えば大文字Ａと小文字ａ）と呼びます。

英大文字と英小文字，英数字文字と拡張文字を，同じに扱うか区別するかの規則が等価規則です。等価規則の指定がない場合は，同じ文字を表す英大文字と英小文字，英数字文字と拡張文字は同じものとして扱われます。例えば，次に示す，すべての形でコマンドが実行できます。このとき，データ名はすべて英数字文字の大文字 ABC として扱われます。

使用例 1

#OPTION NOCASE を指定した場合は，次の指定はすべて同じ結果となります。

```
DISPLAY DATA(ABC)
display data(abc)
ＤＩＳＰＬＡＹ　ＤＡＴＡ（ＡＢＣ）
```

使用例 2

TD コマンドのオペランド#PROGRAM の#は必ず英数字文字で記述します。

データ名　＃ＤＡＴＡ＠０１は，拡張文字の＃が使用できます。

```
ＤＩＳＰＬＡＹ　ＤＡＴＡ（#ＰＲＯＧＲＡＭ(Ｐ／Ｐ）＃ＤＡＴＡ＠０１）
```

等価規則は，次の入力に適用されます。また，アポストロフィ（'）やダブルコーテーション（"）で囲まれた文字列は，等価規則の対象となりません。

- TD コマンドとオペランドで指定するプログラム名，データ名

(2)　TD コマンド格納ファイルの読み込み

#INCLUDE コマンドで，ファイルに格納した TD コマンドを入力し，実行させることができます。シミュレーション手続きなど，テストデバッグのたびに使用する手続きを TD コマンドで記述しファイルに保存することによって，繰り返し実行できます。

詳細については，「5.4　TD コマンドの詳細」の「5.4.33　#INCLUDE（TD コマンド格納ファイルの取り込み）」を参照してください。

TD コマンド格納ファイルは，利用者がテキストエディタ（vi や FSED など）によって作成できます。また，-TestCmd コンパイラオプションを指定してコンパイルすることによって自動生成し，必要に応じて TD コマンドの修正や追加をすることで作成できます。

コンパイラが自動生成する TD コマンド格納ファイルについては，「8.　TD コマンド生成機能」を参照してください。

(3)　テスト結果の出力

(a)　結果出力先の指定

SET PRINT コマンド，RESET PRINT コマンド，SET LOG コマンド，または RESET LOG コマンドで，結果出力の出力先を変更できます。バッチモードでは，結果出力ファイル以外のファイルへ出力先を変更できます。詳細については，「5.4　TD コマンドの詳細」の「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）」，「5.4　TD コマンドの詳細」の「5.4.36　SET LOG／RESET LOG（ログ出力ファイルの設定／解除）」を参照してください。

(b)　注釈の表示

DISPLAY COMMENT コマンドで，任意の文字列をテスト結果に表示できます。テスト結果に説明を追加することによって，わかりやすくできます。詳細については，「5.4　TD コマンドの詳細」の「5.4.31 DISPLAY COMMENT（注釈の表示）」を参照してください。

2.3　データ操作の規則

2.3.1　データの比較・代入規則

(1)　データの代入規則

テストデバッガの代入規則は，原則として COBOL 言語の転記規則に従います。COBOL 言語の転記規則との差異を次に示します。

COBOL 言語の転記規則については，マニュアル「COBOL2002　言語　標準仕様編」の MOVE 文についての説明を参照してください。

(a)　けたあふれ・不正なデータ

次の代入は，COBOL プログラムの実行時にされる同様の転記と代入される値が異なる場合があります。

- 整数・非整数のデータへの代入でけたあふれが生じる。
- データ項目の属性に適さない値を保持するデータ名から，整数・非整数にデータを代入する。
- 外部浮動小数点・内部浮動小数点の代入の精度が異なる。

(b)　コンパイラオプション

次のコンパイラオプションは適用されません。

-StdVersion,1，-StdVersion,2，-H8Switch，-MinusZero，-International，-V3Rec,Fixed，-V3Rec,Variable，-EquivRule,StdCode，-SpaceAsZero

(c)　拡張 16 進定数

テストデバッガ独自の定数であり，COBOL 言語の 16 進英数字定数・16 進数字定数・16 進日本語定数が代入できないデータ項目へ，16 進数で指定した値を代入できます。

- 受け取り側作用対象が次の項目のとき，送り出し側作用対象の 16 進のけた数が 2 の倍数でなければなりません。
  - 英数字集団項目
  - 英字項目
  - 英数字項目
  - 英数字編集項目
  - 数字編集項目
- 受け取り側作用対象が次の項目のとき，送り出し側作用対象の 16 進のけた数が 4 の倍数でなければなりません。

- 日本語集団項目
- 日本語項目
- 日本語編集項目

- 受け取り側作用対象が次の項目のとき，送り出し側作用対象と受け取り側作用対象のサイズが等しくなければなりません。
  - 整数の数字項目
  - 非整数の数字項目
  - 外部浮動小数点の数字項目
  - 内部浮動小数点の数字項目

(d) 代入規則

代入規則を表 2-7 に示します。

表 2-7　代入規則

送り出し側	受け取り側																			
	英数字集団項目	英字項目	英数字項目	英数字編集項目	数字編集項目	整数※4	非整数※4	外部浮動小数点※5	内部浮動小数点※6	指標名	指標データ項目	外部ブール項目	内部ブール項目	日本語項目	日本語集団項目	日本語編集項目	アドレス名	アドレスデータ項目	ポインタ項目	オブジェクト参照データ項目
英数字集団項目	○	○	○	○	○	○	○	○	○					○	○	○				
英字項目	○	○	○	○																
英数字項目	○	○	○	○	○	○	○	○	○			○	○							
英数字編集項目	○	○	○	○																
数字編集項目	○		○	○	○	○	○	○	○											
HIGH-VALUE	○	○	○	○										○	○	○				
LOW-VALUE	○	○	○	○										○	○	○				
SPACE	○	○	○	○										○	○	○				
QUOTE	○	○	○	○																
ZERO	○		○	○	○	○	○	○	○	○		○	○	○	○	○	○	○	○	
英数字定数	○	○	○	○																
英数字定数（数字だけ）	○	○	○	○	○	○	○													
16進英数字定数	○	○	○	○	○	○	○													
整数定数 ※4	○		○	○	○	○	○	○	○	※1										
非整数定数 ※4	○		○	○	○	○	○	○	○											
整数 ※4	○		○	○	○	○	○	○	○	○										
非整数 ※4	○		○	○	○	○	○	○	○											
16進数字定数	○		○	○	○	○	○	○	○	○										
外部浮動小数点 ※5	○		○		○	○	○	○	○											
内部浮動小数点 ※6	○		○		○	○	○	○	○											
浮動小数点数字定数	○		○		○	○	○	○	○											
指標名						○				○	○									
指標データ項目										○	○									
外部ブール項目			○									○	○							
内部ブール項目			○									○	○							
ブール定数			○									○	○							
日本語集団項目	○													○	○	○				
日本語項目	○													○	○	○				
日本語編集項目	○													○	○	○				
日本語文字定数	○		※2	※2										○	○	○				
16進日本語文字定数	○		※2	※2										○	○	○				
アドレス名																	○	○	○	
アドレスデータ項目																	○	○	○	
ポインタ項目																	○	○	○	
NULL																	○	○	○	○
オブジェクト参照データ項目																				○
既定義オブジェクト※3																				○
拡張16進定数	○	○	○	○	○	○	○	○	○					○	○	○				

(凡例)

○：代入できる　空白：代入できない

注※1

正の整数（0 を含むが，-0 は含まない）だけ代入できます。

注※2

-JPN コンパイラオプションを指定してコンパイルされたプログラム内データ項目の場合，日本語項目／日本語編集項目は，英数字項目／英数字編集項目の代入規則に従います。受け取り側作用対象がこれに該当する場合，代入できます。

日本語集団項目も上記と同様に，英数字集団項目の代入規則に従います。ただし，-JPN,V3JPNSpace オプションを指定した場合，-JPN コンパイラオプションは無効となります。

注※3

既定義オブジェクトは，SELF，EXCEPTION-OBJECT とします。

注※4

整数は，次のデータ項目で小数点けたを持たない指定です。非整数は，次のデータ項目で小数点けたを持つ指定です。

・外部 10 進項目

・内部 10 進項目

・2 進項目（COMP-X も含む）

注※5

外部浮動小数点形式の数字項目です。

注※6

内部浮動小数点形式の数字項目です。

注意事項

- 次のデータ項目に NULL または不正な値を代入すると，その後，領域が解放されないで残る場合があります。

  NULL または不正な値が代入される場合とは，これらのデータ項目が所属する英数字集団項目に基本項目の属性を考慮しないで値を設定したときなどです。

  ・オブジェクト参照データ項目

- テストデバッガでは，TYPEDEF 句の STRONG 指定があるデータ項目への代入は，STRONG 指定がないデータ項目と同様に扱われ代入できます。
- 一般に浮動小数点数は，数値を正確に表現したものではなく，概略の値を表現するもので誤差を含みます。このため，代入元の値と代入後の値が一致しないことがあります。
- 連絡節で定義した LIMIT 指定がない動的長基本項目には代入できません。代入すると，KCCC3433T-E のメッセージが出力されて，処理が中止されます。ただし，主プログラムシミュレーション中で，SIMULATE MAIN コマンドの TD コマンド群に指定した副プログラム内でだけ代入が可能です。また，この動的長基本項目は主プログラム内で LIMIT 1024 を定義されているものと仮定されます。値を代入すると，その長さが最大長となって，それ以降はその長さ以下しか代入できなくなります。

(2)　データの比較規則

テストデバッガの比較条件の組み合わせは，次の場合を除いて COBOL 言語の比較規則に従います。

COBOL 言語の比較規則については，マニュアル「COBOL2002　言語　標準仕様編」の「いろいろな式」の「条件式（Conditional expressions)」を参照してください。

(a)　けたあふれ・不正なデータ

次の比較は，COBOL プログラムの実行時に行う同様の比較と結果が異なる場合があります。

- データ項目の属性に適さない値を保持するデータ名の整数・非整数を比較する。
- 外部浮動小数点・内部浮動小数点の比較の精度が異なる。

(b) コンパイラオプション

次のコンパイラオプションは適用されません。

-StdVersion,1，-StdVersion,2，-H8Switch，-MinusZero，-International，-V3Rec,Fixed，-V3Rec,Variable，-EquivRule,StdCode，-SpaceAsZero

(c) 拡張 16 進定数

テストデバッガ独自の定数であり，COBOL 言語の 16 進英数字定数・16 進数字定数・16 進日本語定数とは比較できないデータ項目と，16 進数で指定した値で比較ができます。

- 左辺または右辺が次の項目のとき，けた数が 2 の倍数でなければなりません。
  - 英数字集団項目
  - 英字項目
  - 英数字項目
  - 英数字編集項目
  - 数字編集項目
- 左辺または右辺が次の項目のとき，けた数が 4 の倍数でなければなりません。
  - 日本語項目
  - 日本語編集項目
  - 日本語集団項目
- 左辺または右辺が次の項目のとき，両辺のサイズは等しくなければなりません。
  - 整数の数字項目
  - 非整数の数字項目
  - 外部浮動小数点の数字項目
  - 内部浮動小数点の数字項目
  - アドレス名
  - アドレスデータ項目
  - ポインタ項目

(d) 比較規則

比較規則を表 2-8 に示します。

表 2-8　比較規則

右辺＼左辺	英数字集団項目	英字項目	英数字項目	英数字編集項目	数字編集項目	HIGH-VALUE／LOW-VALUE	QUOTE	SPACE	ZERO	英数字定数	16進英数字定数	整数定数※4	非整数定数※4	16進数字定数	外部10進項目	内部10進項目	2進項目・COMP-X	外部浮動小数点※5	浮動小数点数字定数	内部浮動小数点※6	指標名	指標データ項目	外部ブール項目	内部ブール項目	ブール定数	日本語集団項目	日本語項目	日本語編集項目	日本語文字定数	16進日本語文字定数	アドレス名	アドレスデータ項目	ポインタ項目	NULL	オブジェクト参照データ項目	既定義オブジェクト	拡張16進定数
英数字集団項目	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○						○	○	○	○	○							○
英字項目	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○			○	○																		○
英数字項目	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○			○	○										※2	※2							○
英数字編集項目	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○			○	○										※2	※2							○
数字編集項目	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○			○	○																		○
HIGH-VALUE／LOW-VALUE	○	○	○	○	○										○			○								○	○	○									
QUOTE	○	○	○	○	○										○			○																			
SPACE	○	○	○	○	○										○			○								○	○	○									
ZERO	○	○	○	○	○										○	○	○	○		○	○		○	○		○	○	○			○	○	○				
英数字定数	○	○	○	○	○										○			○																			
16進英数字定数（X'xxxx'）	○	○	○	○	○										○			○																			
整数定数※4	○	○	○	○	○										○	○	○	○		○	○																
非整数定数※4	○	○	○	○	○										○	○	○	○		○																	
16進数字定数（H'xxxx'）	○	○	○	○	○										○	○	○	○		○	○																
外部10進項目	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	※1																○
内部10進項目	○								○			○	○	○	○	○	○	○	○	○	※1																○
2進項目・COMP-X	○								○			○	○	○	○	○	○	○	○	○	※1																○
外部浮動小数点※5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○																	○
浮動小数点数字定数	○	○	○	○	○										○	○	○	○		○																	
内部浮動小数点※6	○								○			○	○	○	○	○	○	○	○	○																	○
指標名									○			○		○	※1	※1	※1				○	○															
指標データ項目																					○	○															
外部ブール項目※7									○														○	○	○												
内部ブール項目※7									○														○	○	○												
ブール定数																							○	○													
日本語集団項目	○					○		○	○																	○	○	○	○	○							○
日本語項目	○					○		○	○																	○	○	○	○	○							○
日本語編集項目	○					○		○	○																	○	○	○	○	○							○
日本語文字定数	○		※2	※2																						○	○	○									
16進日本語文字定数	○		※2	※2																						○	○	○									
アドレス名									○																						○	○	○	○			○
アドレスデータ項目									○																						○	○	○	○			○
ポインタ項目									○																						○	○	○	○			○
NULL																															○	○	○		○	○	
オブジェクト参照データ項目※7																																		○	○	○	
既定義オブジェクト※3																																		○	○	○	
拡張16進定数	○	○	○	○	○										○	○	○	○		○						○	○	○			○	○	○				

(凡例)

○：比較できる　空白：比較できない

注※1

整数の場合は比較できます。非整数の場合は比較できません。

注※2

-JPN コンパイラオプションを指定してコンパイルされたプログラム内データの場合，日本語項目／日本語編集項目は，英数字項目／英数字編集項目の比較規則に従います。左辺がこれに該当する場合，比較できます。ただし，左辺が定数の場合は，右辺データがこれに該当する場合，比較できます。

日本語集団項目も上記と同様に，英数字集団項目の代入規則に従います。ただし，-JPN,V3JPNSpace オプションを指定した場合，-JPN コンパイラオプションは無効になります。

注※3

既定義オブジェクトは，SELF，EXCEPTION-OBJECT とします。

注※4

整数は，次のデータ項目で小数点けたを持たない指定です。非整数は，次のデータ項目で小数点けたを持つ指定です。

- 外部 10 進項目
- 内部 10 進項目
- 2 進項目（COMP-X も含む）

注※5

外部浮動小数点形式の数字項目です。

注※6

内部浮動小数点形式の数字項目です。

注※7

比較条件式は，次のデータ項目を除いて，>，<，>=，<=，=，NOT=の比較演算子を用います。次のデータ項目では=，NOT=の 2 種類の比較演算子を用います。

- ブール項目
- オブジェクト参照データ項目
- 既定義オブジェクト

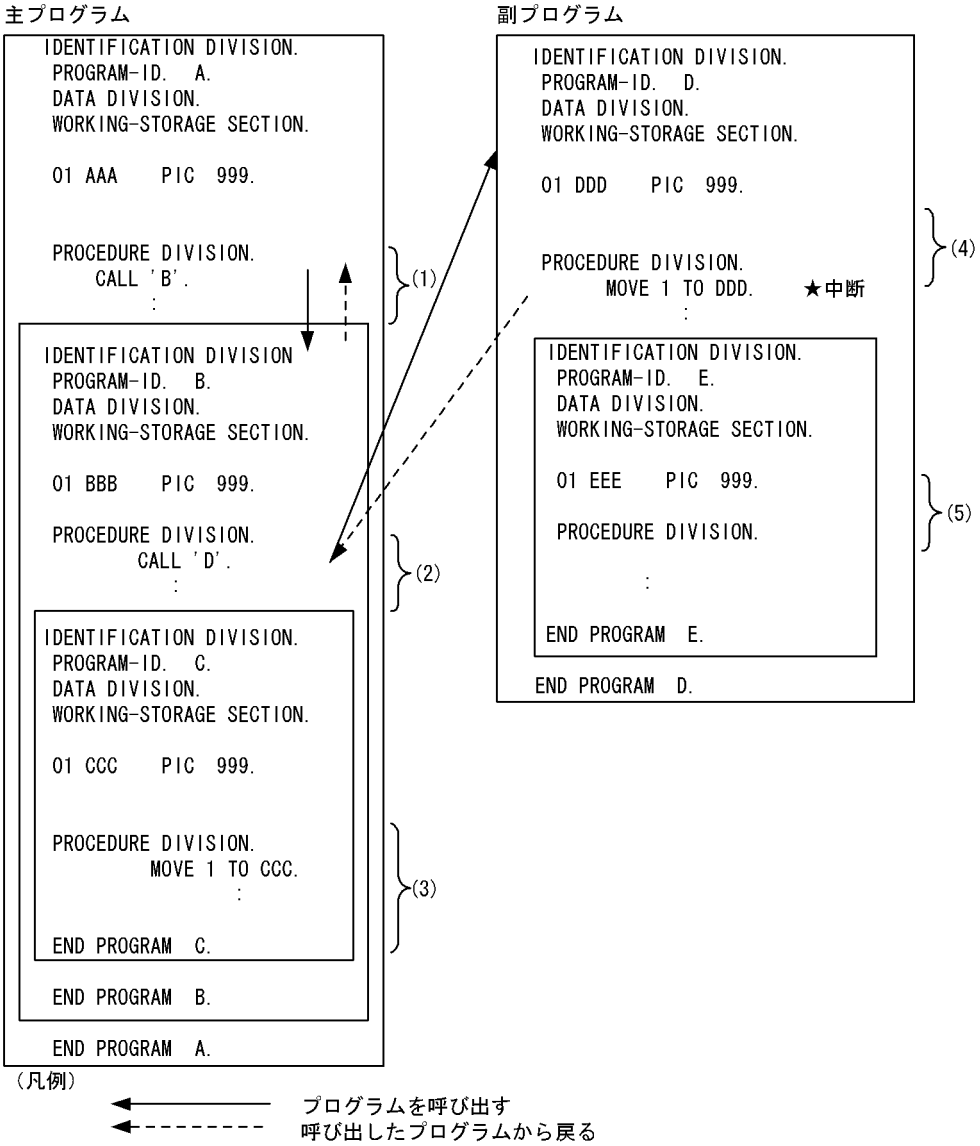
注意事項

一般に浮動小数点数は，数値を正確に表現したものではなく，概略の値を表現するもので誤差を含むことが多いです。このため，DISPLAY DATA TD コマンドなどで表示されたデータ値と比較しても等しくならないなど，比較条件の真理値が一致しない場合があります。

2.3.2 データ項目・ファイル名が参照できる範囲

テストデバッガで，データに対する操作ができる範囲を次に示します。

(1)　プログラム定義

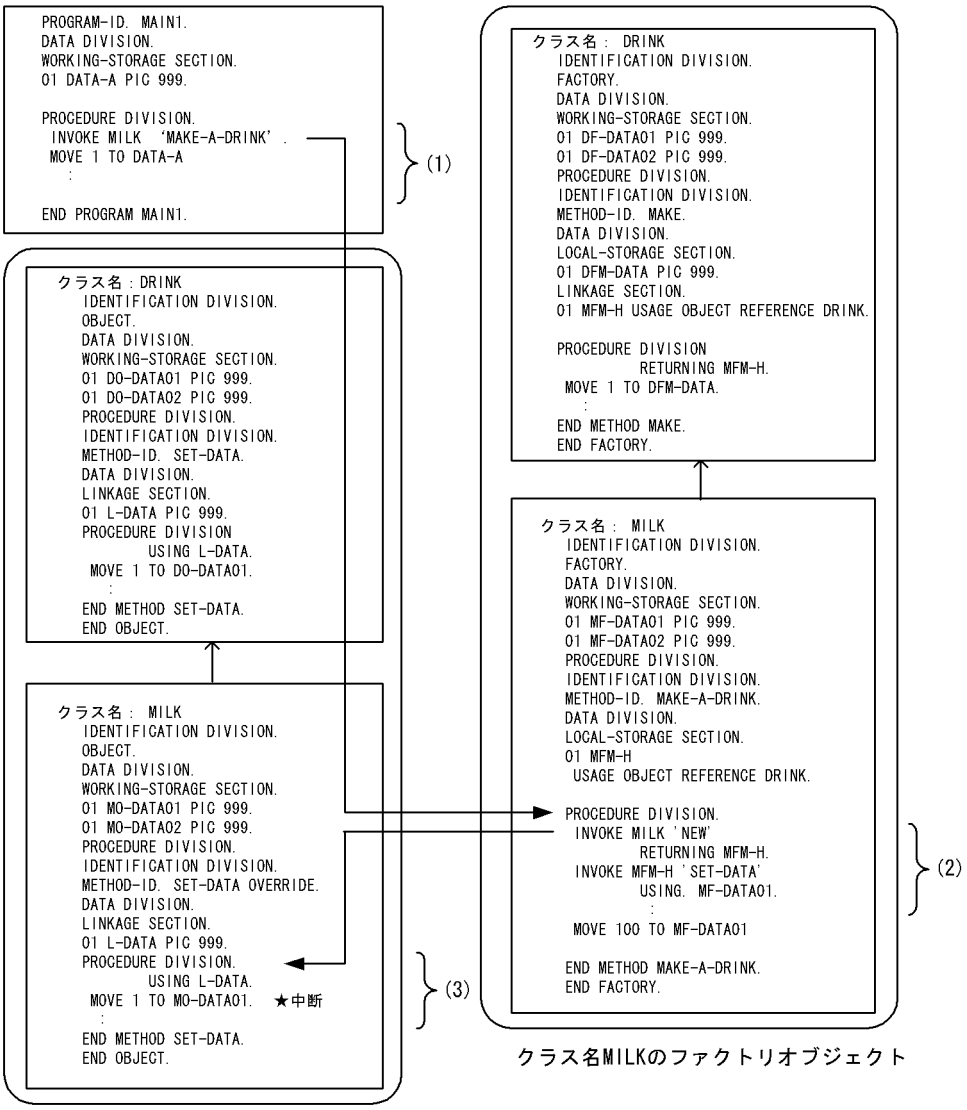


(1)→(2)→(4)の順でプログラムが呼び出され，★で中断しているとき，表示・代入・比較ができるデータ項目は，(1) (2) (4)の各プログラムで参照できる，次のデータです。

(1) データ名 AAA　(2) データ名 BBB　(4) データ名 DDD

データ監視条件で指定した監視対象データは，データの定義のあるプログラムに制御が渡ってから呼び出し元に戻るまでの間，監視されます。プログラムが(1)→(2)→(4)の順で呼ばれ，(4)→(2)→(1)の順で戻り，監視対象のデータがBBBのとき，データは(2) (4)のプログラム実行中に監視されます。
また，左辺・右辺がデータ項目の比較条件式の監視では，左辺・右辺両方のデータが参照できる間に監視されます。

(2)　クラス定義



(1)→(2)→(3)の順にメソッドが呼び出され，★で中断しているとき，操作できるデータ項目は(1)，(2)，(3)の各メソッドで参照できる次のデータ項目です。

- (1)のDATA-A
- (2)のMF-DATA01 MF-DATA02 MFM-H
- (3)のMO-DATA01 MO-DATA02 L-DATA

データ監視条件で指定したデータは，データの参照できるメソッドに制御が渡ってから呼び出し元に戻る間に監視します。プログラムからメソッドが(1)→(2)→(3)の順で呼ばれ，(3)→(2)→(1)の順で戻る場合に，監視対象データをMF-DATA01としたときは，(2)，(3)のメソッドの実行中に監視されます。

（凡例）　：インスタンス　：継承　：制御の流れ

2.3.3　プログラムの実行環境の文字コード

データに対する操作は，プログラムの実行環境の文字コードに従います。

(1) シフト JIS コード（AIX で有効）

データを表示するときに，次の変換をします。

- 印刷不能文字は，半角ピリオドに置き換える。
- 外字は使用する環境に登録されている文字を表示する。

(2) EUC コード（AIX で有効）

データを表示するときに，次の変換をします。

- 印刷不能文字は，半角ピリオドに置き換える。
- 外字は使用する環境に登録されている文字を表示する。

(3) UTF-8 コード（Linux で有効）

データを表示するときに，次の変換をします。

- 印刷不能文字は，半角ピリオドに置き換える。
- 外字は使用する環境に登録されている文字を表示する。

2.3.4 ファクトリオブジェクトとインスタンスオブジェクト

クラス定義のファクトリ定義（FACTORY）とインスタンス定義（OBJECT）のデータの操作について説明します。

(1) オブジェクト

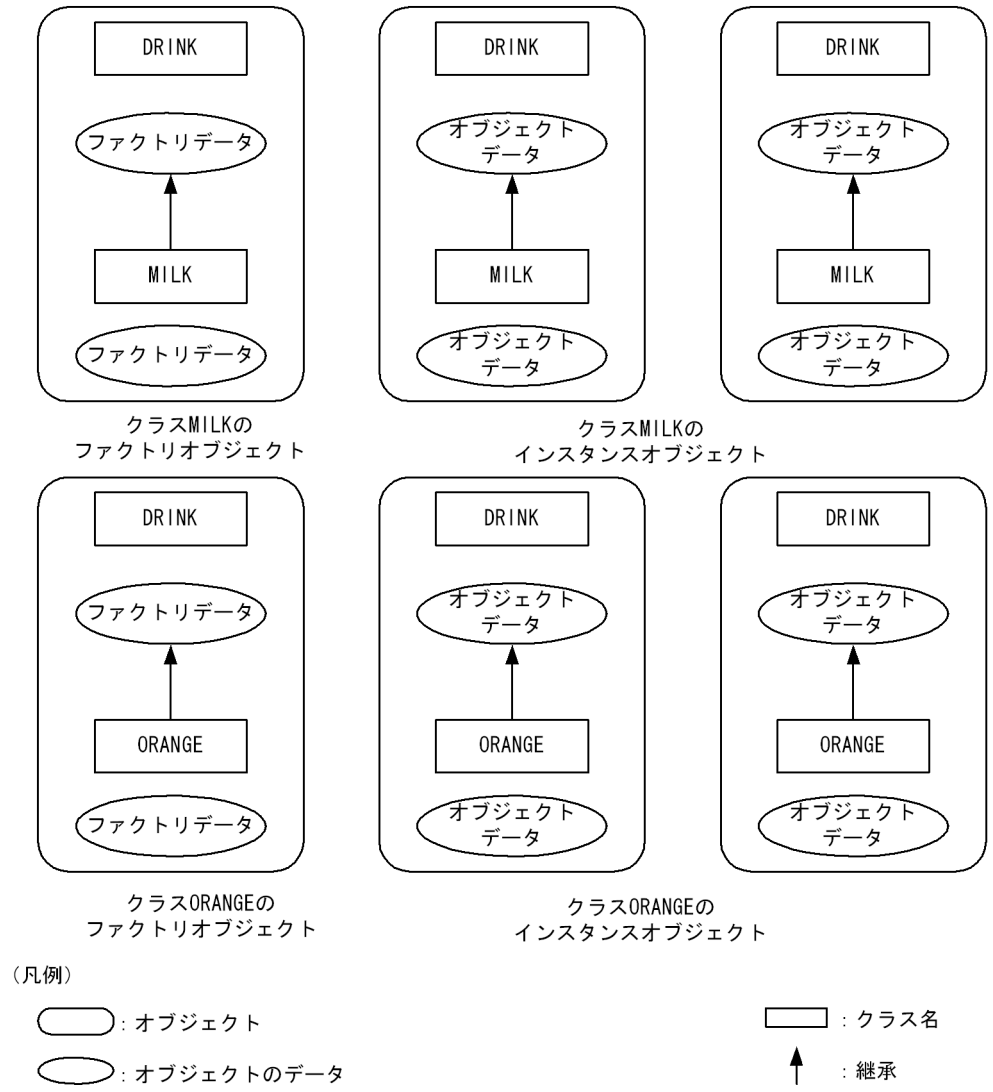
ファクトリオブジェクトは，ファクトリ定義のデータの集まりと，そのデータに対する操作の集まりです。インスタンスオブジェクトは，クラスのインスタンス定義のデータと，そのデータに対する操作の集まりです。クラス定義からは，一つのファクトリオブジェクトと，複数のインスタンスオブジェクトが生成されます。

ファクトリオブジェクトとインスタンスオブジェクトを合わせて，オブジェクトと呼びます。オブジェクトは，オブジェクト参照データ項目または既定義オブジェクトの値によって，一意となります。ファクトリオブジェクトまたはインスタンスオブジェクトに定義されるデータを，オブジェクトデータとします。

ファクトリオブジェクトとインスタンスオブジェクトの例を次に示します。

図 2-2　ファクトリオブジェクトとインスタンスオブジェクト

クラスDRINKを，クラスMILKとクラスORANGEが継承します。
クラスMILKとクラスORANGEのインスタンスオブジェクトがそれぞれ2個ずつ生成された図を示します。



(2)　オブジェクトデータの表示

DISPLAY OBJECT／DIPLAY FACTORY コマンドで，次のオブジェクトデータが表示できます。

- データ名を指定

  ほかのソース要素のデータと同様に，中断点から参照できるデータを表示できます。

  詳細については，「2.2.5　データの操作」の「(1)　データの表示」を参照してください。

- オブジェクト参照データ項目・既定義オブジェクトを指定

  DIPLAY OBJECT コマンドで，オブジェクト参照データ項目または既定義オブジェクト SELF，EXCEPTION-OBJECT を指定して，参照するオブジェクトのデータの値を表示できます。

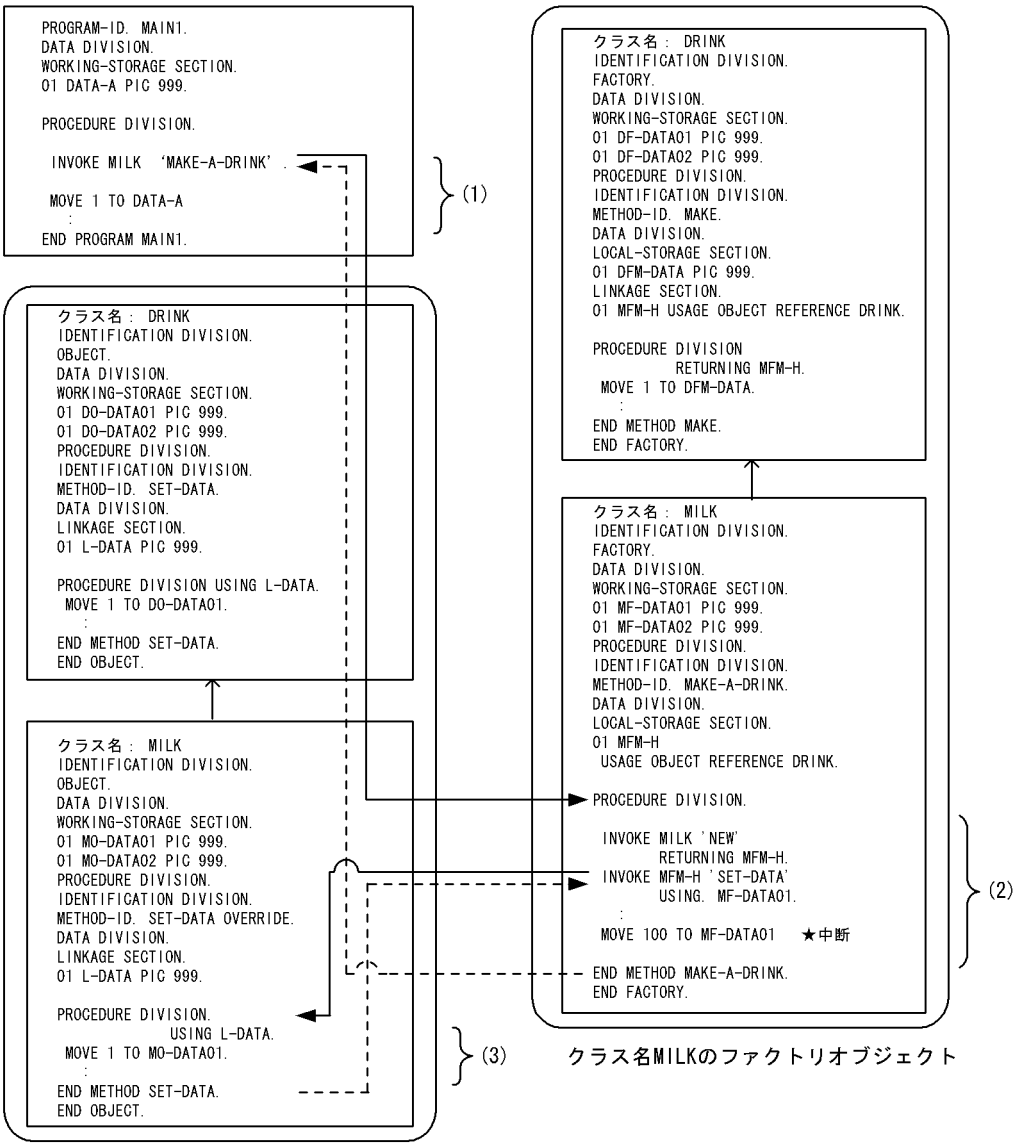
  DISPLAY FACTORY コマンドで，クラス名を指定して，ファクトリオブジェクトのデータの値を表示できます。

詳細については，「5.4　TD コマンドの詳細」の「5.4.14　DISPLAY OBJECT／DISPLAY FACTORY（オブジェクトのデータ値の表示）」を参照してください。

注意事項

- オブジェクト生成時のクラスの，スーパクラスのデータを表示できます。表示をするときは，スーパクラスも-TDInf コンパイラオプションを指定してコンパイルします。
- 集団項目に属し，OCCURS 句に DEPENDING の指定のあるデータ項目よりあとにあるデータ項目は，オブジェクトが生成されてもそのメソッドが呼ばれるまで値の表示ができません。

中断点から参照できるデータ項目の例を次に示します。



(1)→(2)→(3)の順にメソッドが呼び出され，(3)のメソッドから戻った後に★で中断しているとき，オブジェクト参照データ項目MFM-Hを使用して参照できるデータ項目は，次のインスタンスオブジェクトのデータ項目です。

・DRINKクラスのDO-DATA01　DO-DATA02
・MILKクラスのMO-DATA01　MO-DATA02

また，同じ★で中断しているとき，既定義オブジェクトSELFを用いて参照できるデータ項目は，ファクトリオブジェクトの次のデータ項目です。

・DRINKクラスのDF-DATA01　DF-DATA02
・MILKクラスのMF-DATA01　MF-DATA02

（凡例）　：インスタンス　：継承　：制御の流れ

(3)　オブジェクトデータの監視

ファクトリオブジェクト・インスタンスオブジェクトのデータは，オブジェクトごとのデータの変化を監視します。また，一つのオブジェクトのデータだけの監視を指定できます。一つのオブジェクトのデータの監視を指定したとき，監視の対象とするオブジェクトデータは次のとおりです。

- 指定されたクラスのオブジェクトデータが中断位置で参照できるときは，参照できるオブジェクトデータ

- プログラムの起動前または指定されたオブジェクトデータが中断位置で参照できないときは，データ監視の設定後に，指定されたオブジェクトデータが参照できるメソッドに制御が渡ったときに参照できたオブジェクトデータ

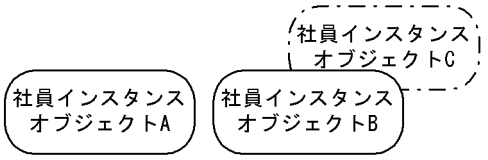
(4) ファイルシミュレーション

一つのファイル定義で複数のファイル結合子がある場合は，ファイル結合子単位にシミュレーション手続きが実行されます。

(例)

```
IDENTIFICATION DIVISION.
CLASS-ID. 社員 INHERITS BASE.
  ⋮
IDENTIFICATION DIVISION.
OBJECT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT  給与ファイル   ASSIGN TO SYS001.
DATA DIVISION.
FILE SECTION.
FD 給与ファイル.
 01 給与レコード.
  02  コード PIC 999.
  02  基本給 PIC 9(8).
  ⋮
METHOD-ID. 計算.
  ⋮
  OPEN INPUT給与ファイル.
  READ 給与ファイル.
  ⋮
END METHOD 計算.
          ⋮
```

```
SET QUALIFICATION( #CLASS (社員/#OBJECT) )
SIMULATE FILE(給与ファイル) OPENMODE( INPUT ) COUNTER( CNT )
  IF CONDITION(CNT = 1)
     ASSIGN DATA(コード) VALUE(1)
     ASSIGN DATA(基本給) VALUE(100000)        (1)
  ENDIF
  IF CONDITION(CNT=2)
     ASSIGN DATA(コード) VALUE(2)
     ASSIGN DATA(基本給) VALUE(101000)
  ENDIF
  IF CONDITION(CNT=3)
     ASSIGN DATA(コード) VALUE(3)
     ASSIGN DATA(基本給) VALUE(104000)
  ENDIF
ENDSIMULATE
```



社員インスタンスオブジェクト A，B…は，それぞれファイル結合子を持つため，一つのファイルシミュレーション定義が個々に有効となります。

社員インスタンスオブジェクト A の計算の READ に対して(1)が実行され，CLOSE 前に社員インスタンスオブジェクト B の計算の READ が実行されても個々にファイルシミュレーション定義が有効なので(1)が実行されます。

カウンタ変数または繰り返し指定の反復回数はファイル結合子単位に実行されるシミュレーション単位にカウントされます。

2.4　その他の機能

2.4.1　マルチスレッド対応 COBOL プログラムのテストデバッグ

マルチスレッド対応 COBOL プログラムのデバッグのための機能と注意事項を説明します。

マルチスレッド対応 COBOL プログラムは，プロセス内で動作するすべての COBOL プログラムを，-MultiThread コンパイラオプションを使ってコンパイルしないと，実行時の動作は保証しません。

マルチスレッド対応 COBOL プログラムの詳細については，マニュアル「COBOL2002　使用の手引　手引編」の，マルチスレッド環境での実行についての説明を参照してください。

(1)　プログラムの実行

マルチスレッド対応 COBOL プログラムを実行するときの仕様は次のとおりです。

- 中断点の設定による中断

  スレッドに依存しないで，中断点に達すれば中断します。
- データ監視条件による中断

  マルチスレッド対応 COBOL プログラムでは，スレッドごとにデータ値の変化を監視します。どれかのスレッドでデータの値が変化するか，または，比較条件式が成立したときに中断します。
- 特定スレッドに対応するデータの値だけを監視する設定もできます。この場合，設定時のプログラムによって，監視するデータは次のとおりになります。
  - プログラムの中断時

    中断しているスレッドの指定ソース要素のデータ
  - プログラムの起動前

    プログラムの起動後に，最初に生成された指定ソース要素を実行するスレッドのデータ
- ステップイン

  ステップインが実行されると，スレッドが切り替わるタイミングに関係なく，次に実行される文で中断します。つまり，ステップインを実行した直後にスレッドの切り替えが発生すると，ステップイン実行前の中断位置とステップイン実行後の中断位置は別のスレッドで実行されるプログラムとなります。
- ステップオーバー

  ステップオーバーを実行したスレッドと同じスレッドで，CALL 文，INVOKE 文および関数呼び出しを行う文を 1 文として実行して中断します。中断点の設定，データ監視，実行時エラー，割り込みによる中断がなければ，ステップオーバーを実行したスレッド以外で中断しません。
- ジャンプ，ジャンプ実行

  ジャンプ，ジャンプ実行したスレッドで，指定された文にジャンプします。ただし，ジャンプ，ジャンプ実行の直後に，別スレッドに切り替わり，そのプログラムで中断点の設定・データ監視・実行時エラー・割り込みなどによって中断した場合は，ジャンプ，ジャンプ実行は行われません。

- 呼び出し順序番号

  マルチスレッドで実行するプログラムの場合は，スレッドごとに呼び出し順序番号の増加と減少が行われます。

- ファイルシミュレーション

  ファイルシミュレーションは，COBOL 言語のファイル結合子ごとに行います。スレッドごとにファイル結合子が生成されるファイルでは，スレッドごとにファイルシミュレーションされます。

- 使用できない機能

  AIX の場合，次の機能は，マルチスレッドプログラムでは使用できません。

  - DC シミュレーション
  - 領域の確保と解放

注意事項

- AIX の場合，マルチスレッドプログラムをデバッグ対象とする場合は，各ユーザスレッドが一つのカーネルスレッドにマップされる 1：1 モデルで動作する必要があります。1：1 モデル以外で動作した場合，テストデバッグの動作は保証しません。1：1 モデルの設定は，環境変数 AIXTHREAD_SCOPE に S を設定することで行えます。

  設定例

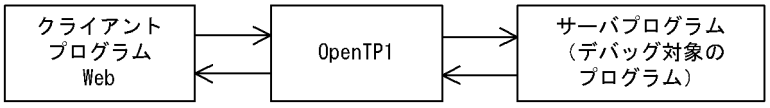
  ```
  AIXTHREAD_SCOPE=S
  export AIXTHREAD_SCOPE
  ```

- Linux の場合，マルチスレッド対応 COBOL プログラムのテストデバッグはできません。

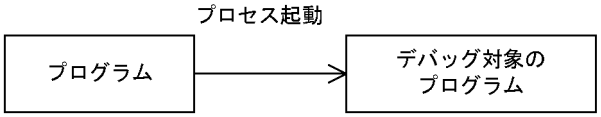
2.4.2　プログラムからの連動実行

ほかのプロセスから起動されることを前提にするプログラムのテストデバッグをするときは，プログラムからの連動実行ができます。次の環境で動作するプログラムのテストデバッグをするときに有効です。詳細については，「3.3.1　プログラムからの連動実行」を参照してください。

- OpenTP1 から起動されるサーバプログラム



- ほかのプログラムからプロセス起動されるプログラム



2.4.3　再帰によるソース要素の実行

プログラムのソース要素が再帰によって実行され，一つのソース要素から複数の実行時要素が生成されたときに，実行時要素を指定してデータの表示ができます。

データ監視では，連絡節，局所場所節に定義されているデータ，特殊レジスタは，実行時要素ごとに監視します。カウンタ変数も実行時要素ごとにカウントされます。

2.4.4　共通例外を発生させるプログラムのテストデバッグ

プログラムで共通例外が発生したあとのプログラムの動きは，COBOL の言語仕様に従います。

共通例外の詳細については，マニュアル「COBOL2002　使用の手引　手引編」の共通例外処理の説明を参照してください。

共通例外が発生し伝播が起きることによって，共通例外が発生した実行時要素を呼び出した実行時要素の手続き部の宣言部分が実行される場合があります。このとき，共通例外が発生した実行時要素の出口を通過してから，手続き部の宣言部分が実行されます。次の機能が設定されているときは，実行時要素の出口では対応する機能がそれぞれ動作します。

- 出口指定の中断点による中断
- データ監視条件による中断
- トレース
- フロー
- ステップイン・ステップオーバーによる中断

注意事項

テストデバッグの対象となるプログラムで実行時エラーが発生して KCCC0015R-S のメッセージを表示した場合，KCCC1207T-I のメッセージが示す行番号は，KCCC0015R-S のメッセージが示す共通例外を引き起こした文の行番号となります。また，そのときに DISPLAY POINT コマンドで表示される文番号は，共通例外を引き起こした文番号となります。

2.4.5　共用ライブラリ

(1)　コンパイルと実行

テストデバッグで共用ライブラリを使用するときは，-PIC,Std，および-TDInf コンパイラオプションを指定してプログラムをコンパイルします。コンパイラオプションについては，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

AIX の場合，ar コマンドでアーカイブファイルとした共用ライブラリ中のサブプログラムを，テストデバッグ対象に指定できます。ar コマンドについては，システムのリファレンスマニュアルを参照してください。

共用ライブラリファイルは，cbltd2k コマンドや，cbltl2k コマンドの-Library オプションで，パスプレフィクスが付いていないファイル名を指定した場合，環境変数の指定有無によって検索順序が決まります。

-Library オプションで指定された共用ライブラリが，環境変数 CBLLSLIB，CBLLPATH，または LD_LIBRARY_PATH（Linux の場合）に設定したパスで見つかり，そのパスがロードされた共用ライブラリのパスと一致する場合は，ローディングされた共用ライブラリをテストデバッグの対象とし，KCCC1203T-I メッセージを出力します。

「KCCC1203T-I ユーザプログラムのライブラリファイルがロードされました。ファイル（*** 1 ***)」

検索した結果，共用ライブラリが見つかってもパスが一致しない場合は，テストデバッグの対象としないで，KCCC1203T-I メッセージも出力しません。

検索した結果，共用ライブラリが見つからない場合は，KCCC4207T-E のエラーメッセージを出力し，テストデバッグの対象となりません。

「KCCC4207T-E　ユーザプログラムの実行可能ファイル又はライブラリファイルが存在しません。ファイル（*** 1 ***)」

OS ごとに，環境変数の指定有無による検索順序を説明します。

AIX の場合

1. 環境変数 CBLLSLIB の指定がある場合

   cbltd2k コマンドや cbltl2k コマンドの-Library オプションで指定したファイル名と同一名称のファイル名が，パスプレフィクス付きで環境変数 CBLLSLIB に指定されていれば，そのパスだけが検索されます。同一名称のファイル名が指定されていない，またはパスプレフィクスなしで指定されているときは，次の項目のとおりに検索します。

2. 環境変数 CBLLPATH の指定がある場合

   指定されているパス，カレントディレクトリの順序で検索します。

3. 環境変数が何も指定されていない場合

   カレントディレクトリを検索します。

Linux の場合

1. 環境変数 CBLLSLIB の指定がある場合

   - 環境変数 CBLLSLIB にパスプレフィクス付きのファイル名の指定があり，cbltd2k コマンドや cbltl2k コマンドの-Library オプションのファイル名と一致するとき

     環境変数 CBLLSLIB に指定されているパスだけ検索します。

   - 環境変数 CBLLSLIB にパスプレフィクスなしのファイル名の指定だけがあるとき

環境変数 CBLLSLIB に指定されているファイルを，環境変数 CBLLPATH または環境変数 LD_LIBRARY_PATH に指定されているパスから検索します。

2. 環境変数 CBLLPATH の指定がある場合

   環境変数 CBLLSLIB に指定されているファイルを，環境変数 CBLLPATH に指定されているパスから検索します。カレントディレクトリを検索対象とするときは，パスにカレントディレクトリを追加する必要があります。

3. 環境変数 LD_LIBRARY_PATH の指定がある場合

   環境変数 CBLLSLIB に指定されているファイルを，環境変数 LD_LIBRARY_PATH に指定されているパスから検索します。カレントディレクトリを検索対象とするときは，パスにカレントディレクトリを追加する必要があります。

   なお，環境変数 CBLLPATH と同時に指定されているときは，環境変数 CBLLPATH に指定されているパスを検索したあと，環境変数 LD_LIBRARY_PATH に指定されているパスを検索します。

検索についての注意事項（AIX の場合）

環境変数 LIBPATH に指定されているパスは検索されません。この環境変数で共用ライブラリの検索パスを指定している場合は，-Library オプションにパスプレフィクスを付けて指定する必要があります。

注意事項

1. AIX の場合の注意事項を次に示します。

   - ar コマンドでアーカイブファイルとして作成した共用ライブラリ中に，同じ名称のメンバ（共用オブジェクト）が複数あるときは，KCCC4251T-W メッセージを出力し，先に配置されているメンバ（共用オブジェクト）をテストデバッグ対象とします。
   - -L および-l オプションを指定して共用ライブラリをリンケージしたプログラムを実行し，-DynamicLink,Call または-DynamicLink,IdentCall コンパイラオプションを指定して再びリンケージした場合，この共用ライブラリをテストデバッグの対象とできない場合があります。その場合は，次のどちらかの指定をします。

     ・共用ライブラリの検索ディレクトリ指定の，-L オプションと同じディレクトリを，環境変数 CBLLPATH に指定。

     ・-L オプションと同じディレクトリを指定した共用ライブラリファイルを，環境変数 CBLLSLIB に指定。
   - アーカイブファイルの中に含まれるアーカイブファイルは，テストデバッグ対象になりません。
   - メッセージでアーカイブファイル名を表示する場合は，ファイル名を次の形式で出力します。

     アーカイブファイル名〔メンバ名〕

     （例）アーカイブファイル名が/usr/test/usersub.a，メンバ名が sub1.o の場合

     KCCC1203T-I　ユーザプログラムのライブラリファイルがロードされました。

     　　ファイル（/usr/test/usersub.a〔sub1.o〕）
   - 環境変数 CBLLTAG による検索動作の指定は，デバッガでは有効になりません。

2. 遅延ロードされる共用ライブラリはテストデバッグの対象になりません。遅延ロードについては，システムのマニュアルを参照してください。

3. ld コマンドの-s オプションや strip コマンドなどで共用ライブラリのシンボル情報を削除した場合，その共用ライブラリは，テストデバッグの対象になりません。

   なお，Linux では，cbltd2k コマンドや cbltl2k コマンドの-Execute オプションに指定する実行可能ファイルは，テストデバッグの対象にしない場合でもシンボル情報が必要です。

使用例 1

ccbl2002 コマンドに-L および-l オプションを使用して実行可能ファイルを作成します。

1. sub1.cbl をコンパイルする。

   AIXの場合

```
ccbl2002 -PIC,Std -TDInf sub1.cbl
```

   Linuxの場合

```
ccbl2002 -PIC,Std -TDInf -UniObjGen sub1.cbl
```

2. 共用ライブラリを作成する。

   AIX(32)の場合

```
ld -o libsub1.a sub1.o -bpT:0x10000000 -bpD:0x20000000
 -bnoentry -bM:SRE -bexpall -L/opt/HILNGcbl2k/lib -lcbl2k
 -lcbl2kml -lm -lc
```

   AIX(64)の場合

```
ld -o libsub1.a sub1.o -b64 -bpT:0x100000000 -bpD:0x110000000
 -bnoentry -bM:SRE -bexpall -L/opt/HILNGcbl2k64/lib -lcbl2k64
 -lcbl2kml64 -lm -lc
```

   Linux(x86)の場合

```
ld -shared -o libsub1.so sub1.o -Bstatic -L/opt/HILNGcbl2k/lib -lcbl2kml -Bdynamic -lc
```

   Linux(x64)の場合

```
ld -shared -o libsub1.so sub1.o -Bstatic -L/opt/HILNGcbl2k64/lib -lcbl2kml -Bdynamic -lc
```

3. -L および-l オプションで共用ライブラリを指定して，実行可能ファイルを作成する。-L および-l オプションについては，システムの規則に従ってください。

   AIXの場合

```
ccbl2002 -TDInf main1.cbl -L ./ -lsub1 -OutputFile a.out
```

   Linuxの場合

```
ccbl2002 -TDInf -UniObjGen main1.cbl -L ./ -lsub1 -OutputFile a.out
```

4. -Library オプションを指定してテストデバッグを開始する。

AIX の場合

```
cbltl2k -Library libsub1.a -Execute a.out
```

Linux の場合

```
cbltl2k -Library libsub1.so -Execute a.out
```

使用例 2

ccbl2002 コマンドの-DynamicLink,Call コンパイラオプションを使用して実行可能ファイルを作成します。

1. 使用例 1 の 1.および 2.と同様の手順で，sub1.cbl に-TDInf コンパイラオプションを指定してコンパイルし，共用ライブラリを作成する。

2. -DynamicLink,Call コンパイラオプションを指定して実行可能ファイルを作成する。

AIXの場合

```
ccbl2002 -DynamicLink,Call -TDInf main1.cbl -OutputFile a.out
```

Linuxの場合

```
ccbl2002 -DynamicLink,Call -TDInf -UniObjGen main1.cbl -OutputFile a.out
```

3. 使用例 1 の 4.と同様の手順で，-Library オプションを指定してテストデバッグを開始する。

使用例 3

ccbl2002 コマンドの-DynamicLink,IdentCall コンパイラオプションを使用して実行可能ファイルを作成します。

1. 使用例 1 の 1.および 2.と同様の手順で，sub1.cbl に-TDInf コンパイラオプションを指定してコンパイルし，共用ライブラリを作成する。

2. -DynamicLink,IdentCall コンパイラオプションを指定して実行可能ファイルを作成する。

AIXの場合

```
ccbl2002 -DynamicLink,IdentCall -TDInf main1.cbl -OutputFile a.out
```

Linuxの場合

```
ccbl2002 -DynamicLink,IdentCall -TDInf -UniObjGen main1.cbl -OutputFile a.out
```

3. 使用例 1 の 4.と同様の手順で，-Library オプションを指定してテストデバッグを開始する。

(2) -TDInf および-CVInf コンパイラオプションがない COBOL プログラム，または COBOL プログラム以外から共用ライブラリを呼ぶ場合

-TDInf コンパイラオプションを指定した COBOL プログラムが含まれる共用ライブラリを，次のどちらかのプログラムが呼び出すときに，テストデバッグができない場合があります。

このような場合にテストデバッグを可能とする環境変数 CBLTDEXTARGET について説明します。

- -TDInf および-CVInf コンパイラオプションの指定がない COBOL プログラム
- COBOL プログラムを含まない実行可能ファイル

環境変数

CBLTDEXTARGET

形式

```
CBLTDEXTARGET=YES
export CBLTDEXTARGET
```

機能

-TDInf コンパイラオプションを指定した COBOL プログラムが含まれる共用ライブラリを，次のどちらかのプログラムが呼ぶ場合に，テストデバッグができます。

- -TDInf および-CVInf コンパイラオプションの指定がない COBOL プログラム
- COBOL プログラムを含まない実行可能ファイル

注意事項

1. 環境変数 CBLTDEXTARGET は，テストデバッガを使用するときに有効になります。運用環境では指定しないでください。
2. 環境変数 CBLTDEXTARGET を指定したときは，テスト対象のプロセスの情報を収集する回数が増加するため，処理速度が低下することがあります。テストデバッグ終了した部分は，-TDInf コンパイラオプションを外して再コンパイル後にテストデバッガでテストしてください。
3. 次のディレクトリに格納した共用ライブラリは，テストデバッグの対象にできません。

   AIX(32)および Linux(x86)の場合

   /usr/lib，/lib

   /opt/HILNGcbl2k/lib

   AIX(64)の場合

   /usr/lib，/lib

   /opt/HILNGcbl2k64/lib

   Linux(x64)の場合

   /usr/lib64，/lib64

   /opt/HILNGcbl2k64/lib

4. 環境変数 CBLTDEXTARGET の値に YES 以外を指定した場合，環境変数 CBLTDEXTARGET の指定は無効となります。
5. テストデバッグを実行する前に，環境変数 CBLTDEXTARGET を設定する必要があります。プログラムの連動実行時をするときは，プログラムを実行する前に環境変数 CBLTDEXTARGET を設定する必要があります。

2.4.6 ウィンドウの表示先変更

テストデバッガのウィンドウを，テストデバッガを実行しているマシンと別のマシンに表示しながらデバッグできます。プログラムからの連動実行をしたときに有効です。

表示先を変更するときは環境変数 CBLTDDISPLAY を設定します。環境変数 CBLTDDISPLAY に設定する値は環境変数 DISPLAY に設定する値と同じです。環境変数 DISPLAY については，X Window System のマニュアルを参照してください。

使用例 1

連動実行されるラインモードのテストデバッガの表示先を変更します。

- ClientX での入力コマンド

```
xhost  +ServerA                                  … 1.
rlogin  ServerA                                  … 2.
```

- rlogin した ServerA での入力コマンド

```
CBLTDDISPLAY=ClientX:0.0                         … 3.
export CBLTDDISPLAY
CBLTDEXEC='TL -Execute /tmp/a.out'               … 4.
export CBLTDEXEC
/tmp/a.out                                       … 5.
```

1. ClientX で xhost コマンドを入力してラインモードのウィンドウを表示できるようにします。
   xhost コマンドについては，X Window System のマニュアルを参照してください。
2. ClientX から ServerA にリモートログインします。
3. ServerA で環境変数 CBLTDDISPLAY を設定します。
4. ServerA で環境変数 CBLTDEXEC を設定します。
5. ServerA でプログラムを実行します。

注意事項

環境変数 CBLTDDISPLAY は，X Window System を使用している場合だけ有効となります。

2.4.7　数字項目のけた拡張機能

ここでは，数字項目のけた拡張機能を有効にしたユーザプログラムのテストデバッグについて説明します。数字項目のけた拡張機能の詳細については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

数字項目のけた拡張機能を使用したプログラムをテストデバッグ対象にした場合，次のデータ名，定数をTD コマンドに使用できます。

- 19～38 けたまで定義したデータ名
- 38 けたまでの固定小数点数字定数

上記のデータ名，定数が使用できる TD コマンドを次の表に示します。

表 2-9　数字項目のけた拡張機能が有効となる TD コマンド

項番	TD コマンド	使用可否	指定できるオペランド
1	SET BREAK	×	−
2	RESET BREAK	−	−
3	SET WATCH	○	CONDITION
4	RESET WATCH	−	−
5	GO	○	PARAMETER
6	STEP IN	○	PARAMETER
7	STEP OVER	○	PARAMETER
8	STEP TO	−	−
9	STOP	−	−
10	SET TRACE	△※1	INTERFACE
11	RESET TRACE	−	−
12	SET FLOW	×	−
13	RESET FLOW	−	−
14	DISPLAY FLOW	−	−
15	DISPLAY BREAK	−	−
16	DISPLAY POINT	−	−
17	DISPLAY DATA	○	DATA
18	DISPLAY OBJECT・DISPLAY FACTORY	×	−
19	ASSIGN DATA	○	VALUE
20	IF	○	CONDITION

項番	TD コマンド	使用可否	指定できるオペランド
21	SIMULATE MAIN	－	－
22	SIMULATE SUB	△※2	記号名
23	SIMULATE FILE	△※2	記号名
24	SELECT ACTION	－	－
25	GO END・GO INVALID・GO EOP・GO ERROR	－	－
26	レベル番号	△※2	記号名
27	REPEAT	×	－
28	SIMULATE DC	－	－
29	TEST	－	－
30	CASE	×	－
31	ASSIGN CASECODE	×	－
32	QUIT	－	－
33	DISPLAY COMMENT	－	－
34	SET QUALIFICATION	－	－
35	RESET QUALIFICATION	－	－
36	SET PRINT	－	－
37	RESET PRINT	－	－
38	SET LOG	－	－
39	RESET LOG	－	－
40	#OPTION	－	－
41	#INCLUDE	－	－
42	ASSIGN DEVICE	－	－
43	！	－	－

（凡例）

○：使用できる

×：使用できない

△：データ名または数字定数は指定できないが，影響を受ける

－：該当しない（データ名または数字定数を指定できない）

注※1

INTERFACE オペランドを指定し，かつ，プログラム引数に数字項目のけた拡張機能が有効になったデータ名が指定されていた場合でも，データ値を表示できます。

注※2

数字項目のけた拡張機能が有効になったデータ名および記号名を関連づけます。

注意事項

1. 添字，部分参照の指定では，10 けたを超える数字定数は指定できません。添字，部分参照の指定については「5.　TD コマンド」を参照してください。
2. 数字項目のけた拡張機能が有効な場合でも，16 進数字定数に 8 文字を超える文字列を指定できません。
3. 数字項目のけた拡張機能が有効な場合でも，浮動小数点数字定数の仮数部に 16 けたを超える数字定数を指定できません。

2.4.8　定数長拡張機能

英数字定数の定数長拡張機能（-LiteralExtend,Alnum オプション）を使用した場合の，デバッガの動作について説明します。定数長拡張機能については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

定数長拡張機能を有効にしたプログラムをテストデバッグ対象にした場合，デバッガでも，拡張した項目を使用できます。

ただし，#OPTION TD コマンドで 85SYNTAX を指定した場合は使用できません。また，拡張 16 進定数は拡張しません。

2.5　ソース要素・中断点・通過点の表示形式

テストデバッガの画面，およびメッセージに表示する中断点・通過点の形式を説明します。

2.5.1　ソース要素の表示形式

ソース要素は，次の形式で表示します。

(1)　プログラム定義

```
{ 外部プログラム名/外部プログラム名 }
{ 外部プログラム名/内部プログラム名 }
```

(2)　関数定義

```
関数名
```

(3)　クラス定義

```
{ クラス名/#FACTORY  }
{ クラス名/#OBJECT   }
{ クラス名/メソッド名 }
```

2.5.2　入口・出口・文

ソース要素の入口，出口，文は，次の形式で表示します。△は一つ以上の空白を表します。

(1)　ソース要素の入口

```
#ENTRY△＜ ソース要素の表示形式 ＞
```

(2)　ソース要素の出口

```
#EXIT△＜ プログラム識別子 ＞
```

(3)　文

```
文番号△＜ プログラム識別子 ＞
文番号△手続き名△＜ プログラム識別子 ＞
```

手続き名は，文番号の文が手続き名のときに表示します。

2.5.3　スレッド ID

マルチスレッドで実行するプログラムの場合は，各スレッドを識別するスレッド ID を表示します。

3 ラインモードによるテストデバッグ

ラインモードによるテストデバッグの方法について説明します。ラインモードは，TD コマンドの入力によって，対話的にテストデバッグができます。TD コマンドの入出力によって，簡単で早くテストとデバッグができます。

3.1　ラインモードによるテストデバッグの概要

TD コマンドの入力によって，対話的にテストデバッグができます。ラインの入出力だけで実行できるので，簡単で早くテストデバッグができます。UNIX COBOL2002 では，端末から起動コマンドによってテストデバッガを起動できます。

3.2　ラインモードでのテストの方法

3.2.1　テストの手順

ラインモードでプログラムをテストするときの作業の流れを説明します。

1. コンパイラオプションを指定して，コンパイルする。

   -TDInf コンパイラオプションを必ず指定してください。カバレージを使用する場合は-CVInf コンパイラオプション，主プログラムシミュレーションを使用する場合は-SimMain コンパイラオプション，副プログラムシミュレーションを使用する場合は-SimSub コンパイラオプション，-SimIdent コンパイラオプション，または-DynamicLink コンパイラオプションもあわせて指定します。

2. 環境変数を設定する。

   プログラムの実行に必要な環境変数と，テストデバッグに必要な環境変数を指定します。テストデバッグに必要な環境変数については，「3.2.2　環境変数の指定」を参照してください。プログラムの実行に必要な環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

3. cbltl2k コマンドによってテストデバッガを起動する。

   コマンドの例を次に示します。

   形式 1

   ```
   cbltl2k 〔-Include Path TDコマンド格納ファイルディレクトリ名　〕
           〔-Library 共用ライブラリ名 〔,…〕　〕
            -Execute 実行可能ファイル名
   ```

   形式 2

   ```
   cbltl2k 〔 -Help 〕
   ```

   各オプションの詳細については，「3.2.3　cbltl2k コマンド」を参照してください。

   テストデバッガが起動して，TD コマンドの入力待ちとなります。

4. プログラムの実行前の準備をする。

   テストデバッグの目的に応じて，次の操作をします。

   - プログラムを中断させるときは，SET BREAK コマンドによって中断点を設定する。
   - シミュレーションをするときは，SIMULATE MAIN，SIMULATE SUB，SIMULATE FILE，SIMULATE DC コマンドによって，手続きを指定する。

5. プログラムの実行を開始する。

   GO コマンド，STEP IN コマンド，STEP OVER コマンドによってプログラムを開始します。プログラムに渡す引数，カバレージ情報の蓄積の有無は，オペランドで指定できます。

6. プログラムが中断する。

SET BREAK コマンドによって設定された中断点で，プログラムが中断します。実行時エラーが発生したとき，割り込みの操作でもプログラムは中断します。プログラムが中断した状態で，データの値の確認や変更ができます。

7. プログラムの実行を再開する。

   中断している文から，プログラムの実行が再開します。プログラムが再開する文を，GO コマンドに STATEMENT オペランドを指定して，別の文に変更することもできます。

8. プログラムを再実行する。

   プログラムの実行が終了したあと，5 の操作をすることで，再度プログラムの実行を開始できます。

9. テストデバッガを終了する。

   QUIT コマンドで，テストデバッガを終了します。

3.2.2　環境変数の指定

テストデバッグに必要な環境変数を指定します。システム，またはラインモードを実行する端末に設定します。

表 3-1　環境変数一覧（ラインモードでのテストの方法）

環境変数名	概要
CBLPIDIR	プログラム情報ファイルおよび#INCLUDE コマンドで入力する TD コマンド格納ファイルがあるディレクトリ名を指定する。
CBLTDDISPLAY	連動実行時にテストデバッガを表示するための端末の表示先を変更する。「2.4.6　ウィンドウの表示先変更」を参照のこと。
CBLLSLIB	共用ライブラリファイルのファイル名を指定する。
CBLLPATH	共用ライブラリファイルがあるディレクトリ名を指定する。
LD_LIBRARY_PATH※	共用ライブラリファイルがあるディレクトリ名を指定する。
CBLTDEXTARGET	-TDInf および-CVInf コンパイラオプションがない COBOL プログラム，または COBOL プログラム以外から共用ライブラリが呼ばれる場合，テストデバッグを可能にする。

注※

Linux で有効

プログラムの実行に関するその他の環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注意事項

1. プログラム情報ファイルは，次に示す検索順序で検索します。検索した結果，見つからないときは，プログラム情報ファイルに該当するプログラムはテストデバッグの対象となりません。

   検索順序

   (1)環境変数 CBLPIDIR で指定したディレクトリ

   (2)実行可能ファイルに含まれるプログラムは，実行可能ファイルのあるディレクトリ

   　共用ライブラリファイルに含まれるプログラムは，共用ライブラリファイルのあるディレクトリ

   (3)カレントディレクトリ

2. #INCLUDE コマンドで入力する TD コマンド格納ファイルに，絶対パスの付かないファイル名を指定したときは，次に示す検索順序で検索します。#INCLUDE コマンドで相対パスが指定されたときは，次の各ディレクトリからの相対パスとします。

   検索順序

   (1)cbltl2k コマンドの-IncludePath オプションの指定があればそのディレクトリ

   (2)環境変数 CBLPIDIR で指定したディレクトリ

   (3)カレントディレクトリ

3. 環境変数 CBLTDDISPLAY については，「2.4.6　ウィンドウの表示先変更」を参照してください。

4. 環境変数 CBLLSLIB，環境変数 CBLLPATH，および環境変数 LD_LIBRARY_PATH については，「2.4.5　共用ライブラリ」を参照してください。

3.2.3　cbltl2k コマンド

形式 1

```
cbltl2k 〔-Include Path TDコマンド格納ファイルディレクトリ名　〕
        〔-Library 共用ライブラリ名 〔,…〕　〕
         -Execute 実行可能ファイル名
```

形式 2

```
cbltl2k 〔-Help〕
```

-IncludePath

　TD コマンド（#INCLUDE）で指定する TD コマンド格納ファイルのディレクトリを指定します。

-Library

　テストデバッグの対象とする共用ライブラリ名を指定します。

-Execute

　テストデバッグ対象のプログラムを起動する実行可能ファイル名を指定します。

-Help

　cbltl2k コマンドの構文を表示します。このオプションを指定すると，ほかのオプションをすべて無効とします。

注意事項

- -Execute オプションは，必ず最後に指定します。-Execute オプション以外のオプションの指定順序は任意です。
- -Library オプションは，複数のファイル名を指定できます。
- 起動後のカレントディレクトリは，cbltl2k コマンドを入力したディレクトリです。
- cbltl2k コマンドのメッセージは標準エラー出力へ出力します。
- cbltl2k コマンド名は，英小文字で指定します。
- cbltl2k コマンド名だけを指定した場合は，cbltl2k コマンドの構文を表示します。
- オプションは，英大文字，英小文字のどちらでも指定できます。オプションの始まりは，ハイフン（-）とします。
- オプションの区切り記号は空白文字およびタブです。空白文字およびタブを区切り記号としたくないときは，オプションをダブルコーテーション（"）で囲みます。
- 同じオプションを複数指定した場合は，最後に指定したオプションを有効とします。
- オプションにパスの付かないファイル名を指定した場合は，カレントディレクトリのファイルとなります。相対パスの付いたファイル名を指定した場合は，カレントディレクトリを起点とする相対パスのディレクトリにあるファイルとします。
- オプションに複数のファイル名を指定する場合は，コンマ（,）または空白文字で区切ります。また，アスタリスク（*）をファイル名の一部に指定すると，*以外の文字が一致するすべてのファイルを指定できます。
- cbltl2k コマンドが返す終了コードは，次のとおりです。

終了コード	内容
0	正常終了
1	エラー発生による終了
2	キー操作による割り込みによる終了

- -Help オプションによるコマンドの構文は，標準出力へ出力します。それ以外のメッセージは，標準エラー出力へ出力します。
- 次に示すどれかに該当する場合，単独で実行したユーザプログラムが異常終了すると，「セグメンテーション違反です」などのメッセージがシステムから表示されます。テストデバッガからユーザプログラムを起動すると，システムからのこのメッセージは表示されません。

  ・実行時環境変数 CBLEXCEPT に NOSIGNAL を指定した場合

  ・次に示すコンパイラオプションのどれか一つも指定しないでコンパイルしたプログラムの場合

  -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange コンパイラオプション

  ・COBOL が例外（スタックオーバーフローなど）を検出できない場合

3.3　プログラムからの連動実行による方法

3.3.1　プログラムからの連動実行

プログラムからの連動実行によるテストデバッグの方法について説明します。

1. コンパイラオプションを指定し，コンパイルする。

   -TDInf コンパイラオプションを必ず指定してください。カバレージを使用する場合は-CVInf コンパイラオプション，主プログラムシミュレーションを使用する場合は-SimMain コンパイラオプション，副プログラムシミュレーションを使用する場合は-SimSub コンパイラオプション，-SimIdent コンパイラオプション，または-DynamicLink コンパイラオプションもあわせて指定します。

2. 環境変数を設定する。

   環境変数 CBLTDEXEC を指定します。形式を次に示します。

   形式

   ```
   CBLTDEXEC=TL 〔-Include Path TDコマンド格納ファイルディレクトリ名 〕
                〔-OutFile 実行結果出力ファイル名 〕
                〔-Library 共用ライブラリ名 〔,…〕 〕
                 -Execute 実行可能ファイル名
   ```

   各オプションの詳細および注意事項については「3.3.2　環境変数の指定」の「(1)　連動実行の指定」を参照してください。

   また，プログラムの実行に必要な環境変数および，テストデバッガに必要な環境変数を指定します。テストデバッガに必要な環境変数については，「3.2.2　環境変数の指定」を参照してください。プログラムの実行に必要な環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

3. プログラムを実行する。

   テストデバッガが起動され，TD コマンドの入力待ちとなります。

4. プログラムの実行前の準備をする。

   テストデバッグの目的に応じて，次の操作をします。

   - プログラムを中断させるときは，SET BREAK コマンドによって中断点を設定する。
   - シミュレーションをするときは，SIMULATE MAIN，SIMULATE SUB，SIMULATE FILE，SIMULATE DC コマンドによって，手続きを指定する。

5. プログラムの実行を開始する。

   GO コマンド，STEP IN コマンド，STEP OVER コマンドによってプログラムが開始されます。プログラムに渡す引数，カバレージ情報の蓄積の有無は，オペランドで指定できます。

6. プログラムが中断する。

SET BREAK コマンドによって設定された中断点で，プログラムが中断します。実行時エラーが発生したとき，割り込みの操作でもプログラムは中断します。プログラムが中断した状態で，データの値の確認や変更ができます。

7. プログラムの実行を再開する。

中断している文から，プログラムの実行が再開します。実行が再開する文を，GO コマンドに STATEMENT オペランドを指定して，別の文に変更することもできます。

8. プログラムを再実行する。

プログラムの実行が終了したあと，6 の操作をすることで，再度プログラムの実行を開始できます。

9. テストデバッガを終了する。

QUIT コマンドで，テストデバッガを終了します。

注意事項

- テストデバッガを起動して，TD コマンドの入力を待つ状態になり，GO コマンドによってプログラムの実行を開始しないでテストデバッガを終了すると，プログラム実行は続行されます。
- GO コマンドによってプログラムの実行を開始したあとに，プログラムの終了前にテストデバッガを終了すると，プログラムも終了します。
- プログラムが終了すると，テストデバッガも自動的に終了します。
- AIX の場合，日本語環境でプログラムから連動実行してテストデバッガを起動すると，端末のウィンドウの日本語が文字化けする場合があります。この現象を回避するには，環境変数 LC_ALL に使用中の日本語環境と同じ値を設定する必要があります。この場合の例を使用例 1 に，OpenTP1 のユーザサービス定義の場合の例を使用例 2 に，それぞれ示します。

使用例 1

```
LC_ALL=Ja_JP
export LC_ALL
```

使用例 2

```
putenv LC_ALL Ja_JP
```

3.3.2 環境変数の指定

(1) 連動実行の指定

連動実行するための環境変数は，次のとおりです。

形式

```
CBLTDEXEC=TL 〔-Include Path TDコマンド格納ファイルディレクトリ名 〕
             〔-OutFile 実行結果出力ファイル名 〕
             〔-Library 共用ライブラリ名 〔,…〕 〕
              -Execute 実行可能ファイル名
```

-IncludePath

TD コマンド(#INCLUDE)で指定する TD コマンド格納ファイルのディレクトリを指定します。

-OutFile

テストデバッグ実行結果およびトラブルシュート情報※を出力するファイル名を指定します。ファイル名の拡張子は，「.tdo」でなければなりません。

同じ名前のファイルが存在した場合は上書きされます。

-OutFile オプション省略時は，出力しません。

注※

当社保守員が調査するときに使用する情報です。共用ライブラリがデバッグ対象にならないなど，連動実行が動作しない場合には，当社保守員に連絡してください。

-Library

テストデバッグの対象とする共用ライブラリ名を指定します。

-Execute

テストデバッグ対象のプログラムを起動する実行可能ファイル名を指定します。

注意事項

- TL の文字とオプションの間は空白またはタブで区切ります。
- 環境変数 CBLTDEXEC で指定した実行可能ファイル名と，テストデバッガを起動した実行可能ファイルが一致する必要があります。
- プログラムを起動するときのパスの名称は，次の点に注意して指定してください。次の条件に従わないパスの名称が指定された場合，テストデバッガの動作は保証しません。

  ・環境変数 CBLTDEXEC に指定した実行可能ファイル名と，プログラムを起動するときの実行可能ファイル名は，絶対パス名で指定します。

  ・プログラムを起動するときパス名は，60 バイト以下とします。60 バイトを超えたときは，61 バイト以降が切り捨てられ，60 バイトまでをパス名とみなします。

  ・パスを除いた実行可能ファイル名の長さは，14 バイト以下とします。14 バイトを超えたときは 15 バイト以降が切り捨てられ，14 バイトまでが実行可能ファイル名と見なされます。
- プログラムに渡す引数は，プログラムの起動時に指定します。環境変数 CBLTDEXEC には指定できません。
- テストデバッガが起動されると，テストデバッガを表示するための端末が表示されます。端末によって，テストデバッガの入出力および COBOL のライブラリのメッセージが出力されます。COBOL プログラムの標準入出力は，プログラムを起動した端末で実行されます。

- 環境変数 CBLTDEXEC に空白またはタブを含むパス名を指定できません。
- 拡張子が「.tdo」ではないファイルを実行結果出力ファイル名に指定した場合は，エラーになり連動実行は開始しません。
- 実行結果出力ファイルに出力できない場合は，連動実行用に表示された端末に，エラー内容を出力します。

(2)　その他の環境変数

プログラムの実行に必要な環境変数と，テストデバッガに必要な環境変数を指定します。テストデバッガに必要な環境変数については，「3.2.2　環境変数の指定」を参照してください。プログラムの実行に必要な環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注意事項

- AIX(32)の場合

  環境変数 PATH に，/opt/HILNGcbl2k/bin:/usr/bin:/usr/dt/bin を含めて指定する必要があります。
- AIX(64)の場合

  環境変数 PATH に，/opt/HILNGcbl2k64/bin:/usr/bin:/usr/dt/bin を含めて指定する必要があります。
- Linux(x86)の場合

  環境変数 PATH に，/opt/HILNGcbl2k/bin:/usr/bin:/bin を含めて指定する必要があります。
- Linux(x64)の場合

  環境変数 PATH に，/opt/HILNGcbl2k64/bin:/usr/bin:/bin を含めて指定する必要があります。
- 環境変数 CBLPIDIR は，絶対パスで指定します。
- 環境変数 CBLTDDISPLAY で，テストデバッガを実行しているマシンと別のマシンに表示しながらデバッグすることができます。「2.4.6　ウィンドウの表示先変更」を参照してください。

3.4 TD コマンド入力時の注意事項

- TD コマンドは，プロンプト=> のあとに入力します。

使用例

```
=> STEP IN
=> DISPLAY DATA(W-DATA)
```

```
名称  W-DATA
値    0007
```

- 1 行に複数の TD コマンドを入力できます。

使用例

```
=> DISPLAY DATA(W-DATA1) DISPLAY DATA(W-DATA2) STEP IN
```

```
名称  W-DATA1
値    0100
名称  W-DATA2
値    0200
```

- 一つの TD コマンドを複数行に記述するときは，行の最後に¥を付けます。¥の前には，一つ以上の空白文字，またはタブコードを記述します。

使用例

```
=> SIMULATE  FILE(FILE-IN) OPENMODE(INPUT) ¥
=>  DISPLAY COMMENT('***FILE-IN***')       ¥
=>  DISPLAY DATA(RECORD-IN)                ¥
=> ENDSIMULATE
```

3.5　cbltd コマンド(COBOL85/TD 互換)

cbltd コマンドは，UNIX COBOL85 からの COBOL2002 への移行を円滑にするために使用するコマンドです。cbltl2k コマンドと同等の機能を持つため，通常は cbltl2k コマンドを使用することをお勧めします。

指定する実行可能ファイル，共用ライブラリは COBOL2002 で生成されている必要があります。

また，TD コマンド格納ファイルの内容は，COBOL2002 の TD コマンドである必要があります。

3.5.1　cbltd コマンド

cbltd コマンドの形式を次に示します。

形式

```
cbltd〔 -hc〕〔 -i TDコマンド格納ファイルディレクトリ名〕
          〔 -d 共用ライブラリ名〔,共用ライブラリ名 …〕〕
          実行可能ファイル名
```

-h
　同じ文字を表す英数字文字の英大文字と英小文字を同じに扱います。英数字文字と拡張文字は区別します。

-c
　同じ文字を表す英大文字と英小文字，英数字文字と拡張文字を区別します。

-i
　TD コマンド（#INCLUDE）で指定する TD コマンド格納ファイルのディレクトリを指定します。

-d
　テストデバッグの対象とする共用ライブラリ名を指定します。

実行可能ファイル名
　テストデバッグ対象のプログラムを起動する実行可能ファイル名を指定します。

4 バッチモードによるテストデバッグ

バッチモードによるテストデバッグの方法について説明します。バッチモードは，テスト内容をファイルに登録しておき一括して実行させる方式です。ファイルに登録した TD コマンドを一括して実行させることで，大量のプログラムのテストが効率良く実施できます。

4.1　バッチモードによるテストデバッグの概要

あらかじめ，実行したい TD コマンドをファイルにまとめて記述しておき，そのファイルを起動することによって，一括してテストデバッグを実行する方法です。端末に，TD コマンドを記述したファイルを指定した起動コマンドを入力すると，ファイルに記述された TD コマンドを一括して実行します。一度実行を開始すれば，利用者の操作を必要としないので，大量のプログラムのテストが効率良く実施できます。

4.2　バッチモードでのテストの方法

4.2.1　テストの手順

バッチモードでプログラムをテストするときの作業の流れを説明します。

1. コンパイラオプションを指定し，テストしたいプログラムをコンパイルする。

   -TDInf コンパイラオプションを必ず指定してください。カバレージを使用する場合は-CVInf コンパイラオプション，主プログラムシミュレーションを使用する場合は-SimMain コンパイラオプション，副プログラムシミュレーションを使用する場合は-SimSub コンパイラオプション，-SimIdent コンパイラオプション，または-DynamicLink コンパイラオプションもあわせて指定します。

2. テストに必要なファイルを準備する。

   テスト内容をバッチモードの入力となる TD コマンド格納ファイルに記述します。

3. テストデバッガ，またはプログラムの実行のための環境変数を設定する。

   端末で必要な環境変数を設定します。

   環境変数については，「4.2.2　環境変数の指定」を参照してください。

4. cbltd2k コマンドを端末で指定して実行する。

   cbltd2k コマンドについては，「4.2.3　cbltd2k コマンド」を参照してください。

5. 結果出力ファイルおよび結果蓄積ファイルを開き，結果を確認する。

   カバレージ情報の蓄積の結果は，カバレージ情報の表示を使用して確認します。

   カバレージ情報の表示については，「6.2　カバレージ情報の表示と操作」を参照してください。

4.2.2　環境変数の指定

テストデバッグに必要な環境変数を指定します。

表 4-1　環境変数一覧（バッチモードでのテストの方法）

環境変数名	概要
CBLPIDIR	プログラム情報ファイルおよび#INCLUDE コマンドで入力する TD コマンド格納ファイルがあるディレクトリ名を指定する。
CBLLSLIB	共用ライブラリファイルのファイル名を指定する。
CBLLPATH	共用ライブラリファイルがあるディレクトリ名を指定する。
LD_LIBRARY_PATH※	共用ライブラリファイルがあるディレクトリ名を指定する。

環境変数名	概要
CBLTDEXTARGET	-TDInf および-CVInf コンパイラオプションがない COBOL プログラム，または COBOL プログラム以外から共用ライブラリが呼ばれる場合，テストデバッグを可能にする。

注※

Linux で有効

プログラムの実行に関するその他の環境変数は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注意事項

1. プログラム情報ファイルは，次に示す検索順序で検索します。検索した結果，見つからないときは，プログラム情報ファイルに該当するプログラムはテストデバッグの対象となりません。

   検索順序

   (1)環境変数 CBLPIDIR で指定したディレクトリ

   (2)実行可能ファイルに含まれるプログラムは，実行可能ファイルのあるディレクトリ

   　共用ライブラリファイルに含まれるプログラムは，共用ライブラリファイルのあるディレクトリ

   (3)カレントディレクトリ

2. #INCLUDE コマンドで入力する TD コマンド格納ファイルに，絶対パスの付かないファイル名を指定したときは，次に示す検索順序で検索します。#INCLUDE コマンドで相対パスが指定されたときは，次の各ディレクトリからの相対パスとします。

   検索順序

   (1)cbltd2k コマンドの-IncludePath オプションの指定があればそのディレクトリ

   　-IncludePath オプションの指定がなければ，-Input オプションのディレクトリ

   (2)環境変数 CBLPIDIR で指定したディレクトリ

   (3)カレントディレクトリ

3. 次の環境変数については，「2.4.5　共用ライブラリ」を参照してください。

   - 環境変数 CBLLSLIB
   - 環境変数 CBLLPATH
   - 環境変数 LD_LIBRARY_PATH
   - 環境変数 CBLTDEXTARGET

4.2.3　cbltd2k コマンド

cbltd2k コマンドの形式を次に示します。

形式 1

```
cbltd2k  -Input TDコマンド格納ファイル名
        〔-SyntaxOnly〕〔-Output 結果出力ファイル名 〕
        〔-IncludePath TDコマンド格納ファイルディレクトリ名 〕
        〔-Library 共用ライブラリファイル名〔,…〕〕
         -Execute 実行可能ファイル名
```

形式 2

```
cbltd2k 〔-Help〕
```

-Input

TD コマンド格納ファイル名を指定します。ファイル名の拡張子は，「.tdi」または「.tds」でなければなりません。

-SyntaxOnly

TD コマンドの構文解析だけをし，実行しません。

-Output

結果出力ファイル名（.tdl）を指定します。ファイル名の拡張子は，「.tdl」でなければなりません。省略時は，-Input で指定した TD コマンド格納ファイル名の拡張子を「.tdl」に変更したファイル名とします。

-IncludePath

#INCLUDE コマンドで指定する TD コマンド格納ファイルのディレクトリを指定します。省略時は，cbltd2k コマンドで指定した TD コマンド格納ファイルのディレクトリを仮定します。cbltd2k コマンドで指定する TD コマンド格納ファイルには適用しません。

-Library

テストデバッグ対象とする共用ライブラリ名を指定します。

-Execute

テストデバッグ対象プログラムを起動するための実行可能ファイル名を指定します。

-Help

cbltd2k コマンドの構文を表示します。このオプションの指定時は，ほかのオプションをすべて無効とします。

注意事項

- 引数の指定順序は任意です。ただし，-Execute オプションは，必ず最後に指定します。
- -Library オプションは，引数に複数のファイル名を指定できます。
- -SyntaxOnly オプションが指定されていても，#INCLUDE コマンドは TD コマンド格納ファイルの取り込みをし，取り込んだ TD コマンド格納ファイル内の TD コマンドの構文解析が行われます。また，#OPTION コマンドの指定を有効とします。
- 起動後のカレントディレクトリは，cbltd2k コマンドを入力したディレクトリです。
- cbltd2k コマンドのメッセージは標準エラー出力へ出力します。

- cbltd2k コマンド名は，英小文字で指定します。
- cbltd2k コマンド名だけを指定した場合は，cbltd2k コマンドの構文を表示します。
- オプションは，英大文字，英小文字のどちらでも指定できます。オプションの始まりは，ハイフン（-）とします。
- オプションの区切り記号は空白文字およびタブです。空白文字およびタブを区切り記号としたくないときは，オプションをダブルコーテーション（"）で囲みます。
- 同じオプションを複数指定した場合は，最後に指定したオプションを有効とします。
- オプションにパスの付かないファイル名を指定した場合は，カレントディレクトリのファイルとします。相対パスの付いたファイル名を指定した場合は，カレントディレクトリを起点とする相対パスのディレクトリにあるファイルとします。
- オプションに複数のファイル名を指定する場合は，コンマ（,）または空白文字で区切ります。また，アスタリスク（*）をファイル名の一部に指定すると，*以外の文字が一致するすべてのファイルを指定できます。
- cbltd2k コマンドが返す終了コードは，次のとおりです。

終了コード	内容
0	正常終了
1	エラー発生による終了
2	キー操作による割り込みによる終了

- -Help オプションによるコマンドの構文は，標準出力へ出力します。それ以外のメッセージは，標準エラー出力へ出力します。
- 次に示すどれかに該当する場合，単独で実行したユーザプログラムが異常終了すると，「セグメンテーション違反です」などのメッセージがシステムから表示されます。テストデバッガからユーザプログラムを起動すると，システムからのこのメッセージは表示されません。
  ・実行時環境変数 CBLEXCEPT に NOSIGNAL を指定した場合
  ・次に示すコンパイラオプションのどれか一つも指定しないでコンパイルしたプログラムの場合
  -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange コンパイラオプション
  ・COBOL が例外（スタックオーバーフローなど）を検出できない場合

4.2.4 cbltd コマンド（COBOL85/TD 互換）

cbltd コマンドは，UNIX COBOL85 から COBOL2002 への移行を円滑にするために使用するコマンドです。cbltd2k コマンドと同等の機能を持つため，通常は cbltd2k コマンドを使用することをお勧めします。

指定する実行可能ファイル・共用ライブラリは，COBOL2002 で生成されている必要があります。

また，TD コマンド格納ファイルの内容は，COBOL2002 の TD コマンドである必要があります。

cbltd コマンドについては，「3.5.1　cbltd コマンド」を参照してください。

5 TD コマンド

TD コマンドはテストデバッグの指示をするコマンドです。TD コマンドは，TD コマンド格納ファイルに定義します。

この章では，TD コマンドの指定方法，オペランドの指定方法，TD コマンドの詳細について説明します。

5.1 TD コマンドの指定方法

5.1.1 TD コマンドの形式

TD コマンドは次の形式で指定します。TD コマンド名とオペランド，オペランドとオペランドの間は，一つ以上の空白，またはタブを記述します。

形式

TD コマンド名 必須オペランド 〔 任意オペランド … 〕

指定する内容

TD コマンド名

テストデバッグの操作を示す TD コマンドを指定します。TD コマンドの種類については，「5.3 TD コマンドの一覧」を参照してください。

必須オペランド

TD コマンドの操作対象や詳細を指定します。必須オペランドは決められた順で必ず指定します。重複して指定することはできません。

任意オペランド

TD コマンドの操作対象や詳細を指定します。任意オペランドは省略できます。指定順序は任意です。同一オペランドのグループに属し，効果が背反するオペランドを重複して指定した場合は，最後に指定したものが有効になります。

5.1.2 TD コマンドの行

一つの TD コマンドを，複数の行にわたって記述できます。1 行に記述できる TD コマンドおよび注釈の長さを次に示します。

11,264 バイト以内

5.1.3 空白と括弧を含む文字列の指定方法

空白と括弧を文字列に使用する場合は，アポストロフィ（'）またはダブルコーテーション（"）で囲みます。アポストロフィ（'）またはダブルコーテーション（"）で，2 行にわたる文字列を囲むことはできません。アポストロフィ（'）またはダブルコーテーション（"）で囲んだ文字列を使用できる文字列には，次のものがあります。

- #INCLUDE コマンドのファイル名
- DISPLAY COMMENT コマンドのコメント文字列

- SET PRINT コマンドのファイル名
- SET LOG コマンドのファイル名
- GO コマンドの引数の文字列
- STEP IN コマンドの引数の文字列
- STEP OVER コマンドの引数の文字列
- 定数

アポストロフィ（'）で囲んだ文字列の中にアポストロフィ（'）を指定する場合は，連続して 2 文字のアポストロフィ（'）で指定します。ダブルコーテーション（"）で囲んだ文字列の中にダブルコーテーション（"）を指定する場合は，連続して 2 文字のダブルコーテーション（"）で指定します。

使用例

ファイル名 DON'T の TD コマンド格納ファイルを入力します。

```
#INCLUDE INFILE('DON''T')
```

5.1.4 英大文字と英小文字，英数字文字と拡張文字を区別しての文字列の指定方法

#OPTION コマンドが有効にならないように文字列を指定する場合は，アポストロフィ（' ）または引用符（"）で囲みます。アポストロフィ（' ）または引用符（"）で，2 行にわたる文字列を囲むことはできません。

アポストロフィ（' ）または引用符（"）で囲んだ文字列を使用できる文字列は，「5.1.3 空白と括弧を含む文字列の指定方法」を参照してください。

5.1.5 注釈

「*>」から行の終わりまでを注釈とします。

使用例

```
*> プログラムの実行開始前の指定
*> 中断点を設定する
SET BREAK PROCEDURE(#PROGRAM(決算処理/決算処理) #ENTRY) *> プログラムの開始で中断
```

5.1.6 等価規則

TD コマンド名とオペランドの補助語で使用する#は，等価規則に関係なく必ず英数字文字で指定しなければなりません。その他のオペランドには，等価規則が適用されます。

等価規則の詳細については，「2.2.9　テストデバッグのための設定」の「(1)　入力する文字の扱い（等価規則）」を参照してください。

5.2.1　ソース要素の名称

(1)　外部プログラム名と内部プログラム名

外部プログラム名は，プログラム定義の最外のプログラムである外部プログラムのプログラム名段落（PROGRAM-ID）で指定した名称です。内部プログラム名は，内部プログラムのプログラム名段落（PROGRAM-ID）で指定した名称です。記述規則は，COBOL の言語仕様に従って，変換規則も適用します。変換後の名称を TD コマンドのオペランドに指定できます。

(2)　クラス名とメソッド名

クラス名は，クラス定義のクラス名段落（CLASS-ID）で指定した名称です。メソッド名は，メソッド定義のメソッド名段落（METHOD-ID）で指定した名称です。記述規則は，COBOL の言語仕様に従って，変換規則も適用します。変換後の名称を TD コマンドのオペランドに指定できます。

(3)　関数名

利用者による関数定義の関数名段落（FUNCTION-ID）で指定した名称です。記述規則は，COBOL の言語仕様に従って，変換規則も適用します。変換後の名称を TD コマンドのオペランドに指定できます。

5.2.2　行番号（＝一連番号）

原始プログラムの一連番号領域に記述してある番号です。原始プログラムの一連番号領域に番号の記述がない場合，順序が昇順でない場合，または自由形式正書法の場合は，コンパイラが自動的に振り直す番号が行番号となります。

5.2.3　文番号

COBOL 言語の文を指定する番号です。行番号と位置番号をピリオドでつなぎます。行の先頭文を位置番号の 1 とします。位置番号の 1 は省略できます。

使用例

```
001000  READ A-FILE AT END GO TO P2.
          1000/1000.1      1000.2
```

文番号1000および1000.1は，READ文を表します。
文番号1000.2は，GO TO文を表します。

5.2.4　入口名

外部プログラム名および ENTRY 文で指定される入口点の名前を，入口名とします。

5.2.5　手続き名

COBOL 言語の手続き名です。手続き名には，段落名と節名があります。手続き名も一つの文として文番号が割り当てられます。

使用例

```
001000  P1.          READ
        1000/1000.1
002000                  A-FILE AT END GO TO P2.
```

文番号1000および1000.1は，手続きP1を表します。

5.2.6　データ名

テストデバッグで指定できる COBOL 言語のデータは次のとおりです。

- データ名
- 指標名
- アドレス名
- 特殊レジスタ

データ名は，COBOL 言語の仕様に準拠した形で，修飾，添字付けおよび部分参照が指定できます。COBOL 言語との指定方法の違いは次のとおりです。

(1)　表の参照

表の属性を持つデータ名を参照するときは，添字を付けて指定します。添字は，整数定数，添字の付かない数字項目，指標名の指定ができます。ただし，データ監視条件では，整数定数でなければなりません。

次の項目は，添字に指定できません。

- ALL
- 算術式（相対添字付け）
- ほかの表に割り当てた指標名

また，整数定数を，ピリオドを 2 個（..）続けてつなぎ，添字に指定することによって，表の一部の範囲を一度に参照できます。

(2) 部分参照

データの部分を参照するときは，最左端位置と長さを指定します。最左端位置と長さは，整数定数を指定します。部分参照には，次の制限があります。

- 最左端位置と長さに，算術式は指定できません。
- 添字付けの範囲指定と部分参照は同時に指定できません。

注意事項

- 報告書節のデータ項目および画面節（SCREEN SECTION）の画面名は参照できません。
- 最適化オプション（-Optimize,2）でコンパイルしたソース要素をテストデバッグするときは，最適化されたデータ項目を参照できないことがあります。
- TYPEDEF 句の付いたデータ項目およびその下位項目のデータ項目は参照できません。

  DISPLAY OBJECT コマンド，DISPLAY FACTORY コマンドによって，オブジェクトまたはファクトリオブジェクト内に定義されている表を参照するときの添字の指定は，整数定数でなければなりません。

5.2.7 定数

テストデバッガで使用できる定数には，COBOL の言語仕様に準拠した形式と，拡張 16 進定数があります。COBOL の言語仕様に準拠した形式では，次の定数を代入，比較で使用できます。

(1) COBOL 言語の仕様に準拠した定数

記号文字・定数名は使用できません。ALL は，英数字定数・ブール定数・日本語文字定数・表意定数に指定できます。

定数を囲む一対の区切り記号には，アポストロフィ（'）またはダブルコーテーション（"）が使用できます。区切り記号としてアポストロフィ（'）またはダブルコーテーション（"）を使用する場合の指定方法については，「5.1 TD コマンドの指定方法」を参照してください。

表 5-1 COBOL 言語の仕様に準拠した定数一覧

定数の種類		例
英数字定数	英数字	'ABC'
	16 進英数字	X'0AE2'
数字定数	固定小数点数字	123 123.45
	浮動小数点数字	123.E+02
	16 進数字	H'B0EF'

定数の種類		例
		長さ 8 文字まで指定できます。2 の倍数長である必要はありません。
ブール定数	ブール	B'1010'
日本語文字定数	日本語文字	N'日本語', NC'日本語', ND'日本語'
	16 進日本語文字	NX'81568134'
表意定数		ALL 'A', ALL N'日本語', ALL B'1010' , 〔ALL〕HIGH-VALUE, 〔ALL〕HIGH-VALUES, 〔ALL〕LOW-VALUE, 〔ALL〕LOW-VALUES, 〔ALL〕ZERO, 〔ALL〕ZEROS, 〔ALL〕ZEROES, 〔ALL〕SPACE, 〔ALL〕SPACES, 〔ALL〕QUOTE, 〔ALL〕QUOTES, 〔ALL〕NULL, 〔ALL〕NULLS

(2) 拡張 16 進定数

#X で始まり，一対のアポストロフィ（'）またはダブルコーテーション（"）で囲んだ 0～9，A～F の文字列です。文字数は，2 の倍数で，160 文字までの長さです。ALL による定数の繰り返しも指定できます。

拡張 16 進定数は，COBOL2002 の定数で代入，比較ができないデータ項目にも，代入，比較ができます。代入，比較の規則の詳細については，「2.3.1 データの比較・代入規則」の「(1) データの代入規則」および「(2) データの比較規則」を参照してください。

構文規則

```
〔ALL〕 #X’ ・・・・ ’
〔ALL〕 #X” ・・・・ ”
```

使用例

```
#X’313233’
ALL  #X’FF’
```

5.2.8 既定義オブジェクト名

テストデバッガで使用できる COBOL 言語の既定義オブジェクトは次のとおりです。

- SELF
- EXCEPTION-OBJECT

5.2.9　オペランドの補助語

オペランドの内容を指定するときに，補助語を使用します。補助語の先頭にはシャープ（#）が付きます。オペランドの補助語を次に示します。

表 5–2　オペランドの補助語

補助語	内容
#ENTRY	ソース要素の入口を指定します。
#EXIT	ソース要素の出口を指定します。
#PROGRAM	プログラム名を指定します。
#CLASS	クラス名を指定します。
#FACTORY	ファクトリオブジェクトを示します。
#OBJECT	インスタンスオブジェクトを示します。
#FUNCTION	利用者定義関数名を指定します。
#ALL	すべてという意味で指定します。
#>	参照するオブジェクトの特定のデータ名を示します。
#LASTCODE	直前の ASSIGN CASECODE コマンドで設定した値を示します。

5.2.10　TD 利用者定義語

利用者が任意に指定する語です。先頭が英字で始まる英数字で，長さは 31 バイト以内です。

表 5–3　TD 利用者定義語

種類	用途
カウンタ変数	TD コマンド群の実行回数をカウントします。
記号名	シミュレーションでデータ名を識別します。
監視識別子	データ条件中断の設定を識別します。
ケース識別子	テストケースを識別します。

TD 利用者定義語は，TD コマンドで使用する COBOL2002 の予約語（OF，IN，WITH など）と同じ名称を指定できません。

5.2.11　翻訳単位指定

翻訳単位を特定します。

翻訳単位指定が指定されていない場合は，次の順で翻訳単位指定を仮定します。SET QUALIFICATION コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.32　SET QUALIFICATION／RESET QUALIFICATION（翻訳単位・ソース要素の指定と解除)」の「(1)　SET QUALIFICATION（翻訳単位・ソース要素の指定)」を参照してください。

- プログラムの起動前
  1. 事前に実行された SET QUALIFICATION コマンドで指定した翻訳単位指定
  2. -Main コンパイラオプションでコンパイルされた翻訳単位

     -Main コンパイラオプションでコンパイルされた翻訳単位がない場合は，翻訳単位指定または事前に SET QUALIFICATION コマンドを実行しなければなりません。
- プログラムの中断状態
  1. 事前に実行された SET QUALIFICATION コマンドで指定した翻訳単位指定
  2. 現在中断している翻訳単位

(1)　プログラム指定

プログラムを特定するときに指定します。

形式

```
#PROGRAM（外部プログラム名）
```

(2)　クラス指定

クラスを特定するときに指定します。

形式

```
#CLASS（クラス名）
```

(3)　関数指定

関数を特定するときに指定します。

形式

```
#FUNCTION（関数名）
```

5.2.12　ソース要素指定

ソース要素を特定するときに指定します。

ソース要素指定が指定されていない場合は，次の順でソース要素指定を仮定します。

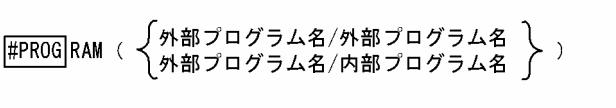
SET QUALIFICATION コマンドの詳細については，「5.4　TD コマンドの詳細」の「5.4.32　SET QUALIFICATION／RESET QUALIFICATION（翻訳単位・ソース要素の指定と解除）」の「(1)　SET QUALIFICATION（翻訳単位・ソース要素の指定）」を参照してください。

- プログラムの起動前
  1. 事前に実行された SET QUALIFICATION コマンドで指定したソース要素指定
  2. -Main コンパイラオプションでコンパイルされた翻訳単位の外部プログラム

     -Main コンパイラオプションでコンパイルされた翻訳単位がない場合は，ソース要素指定または事前に SET QUALIFICATION コマンドを指定しなければなりません。
- プログラムの中断状態
  1. 事前に実行された SET QUALIFICATION コマンドで指定したソース要素指定
  2. 中断状態のソース要素

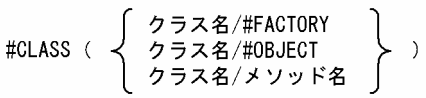
プログラム定義のソース要素を指定します。

形式



クラス定義のソース要素を特定します。

形式



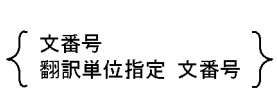
関数を指定します。

形式

5.2.13　文番号指定

文を文番号で指定します。

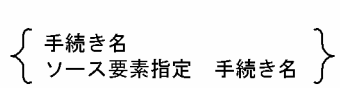
形式



5.2.14　手続き名指定

手続き名を指定します。

形式

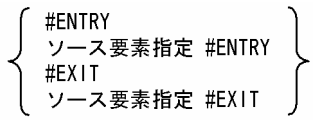


5.2.15　入口，出口指定

ソース要素の入口，出口を指定します。ソース要素の入口とはソース要素の最初の文を実行する前の状態です。ソース要素の出口とはソース要素の最後の文を実行したあとの状態です。

ソース要素の入口を#ENTRY，ソース要素の出口を#EXIT で表します。

形式



使用例

```
#ENTRY                  *> 入口
#PROGRAM(A/A)#ENTRY     *> 外部プログラムAの入口
#EXIT                   *> 出口
#PROGRAM(A/B)#EXIT      *> 外部プログラムAに属する
                        *> 内部プログラムBの出口
```

注意事項

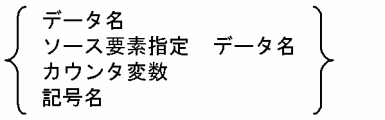
ソース要素指定をするとき，次の指定はできません。

```
#CLASS(クラス名/#FACTORY)
#CLASS(クラス名/#OBJECT)
```

5.2.16　データ名指定

操作する対象となるデータを一意にするときに指定します。

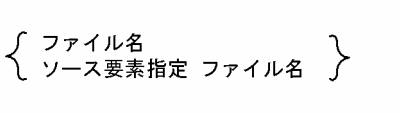
形式



5.2.17　ファイル名指定

COBOL 言語の原始プログラムのファイル記述項のファイル名を指定します。

形式



5.2.18　TD コマンド群

一つにまとまった処理をするための TD コマンドの集まりを TD コマンド群として指定できます。TD コマンド群は，次の TD コマンドで指定できます。

- SET BREAK コマンド，SET WATCH コマンドで，中断時に実行するコマンド
- IF コマンドで，成立または不成立時に実行するコマンド
- シミュレーションの手続き
- TEST コマンドのテストケース

使用例

```
SET BREAK STATEMENT(100) DO
 DISPLAY DATA(品名)
 SET BREAK STATEMENT(500) DO  *> ネスト
   DISPLAY DATA(番号)
 ENDDO
ENDDO
```

データ監視条件の設定，または中断点の設定による中断が一つの文で同時に成立したときは，次の順番で TD コマンド群が実行されます。

1.「データ監視条件の設定」の TD コマンド群

2.「中断点の設定」の TD コマンド群

さらに，データ監視条件の設定によって，一つの文で複数のデータ監視条件が同時に成立したときは，監視識別子の昇順に TD コマンド群が実行されます。

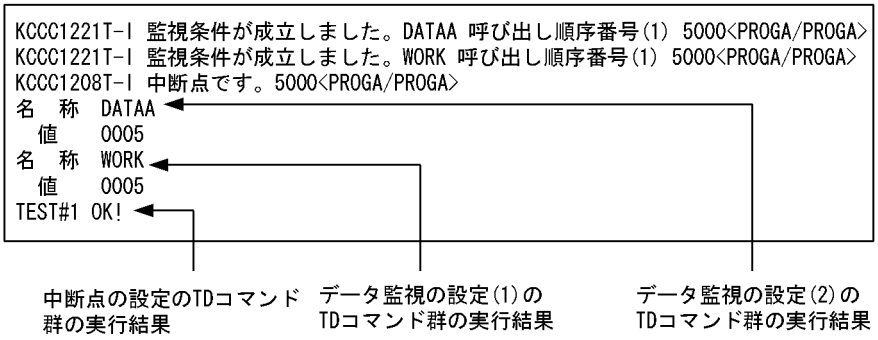
使用例

次の TD コマンドを実行し，一つの文で同時に中断が成立すると，「データ監視の設定(SET WATCH(DATAA))」「データ監視の設定(SET WATCH(WORK))」「中断点の設定」の順番で TD コマンド群が実行されます。

TD コマンド群

```
SET BREAK STATEMENT(5000)                *> 中断点の設定
DO
  DISPLAY COMMENT(”TEST#1 OK!”)
ENDDO
SET WATCH(WORK) CONDITION(WORK NOT = 0) *> データ監視の設定(1)
DO
  DISPLAY DATA(WORK)
ENDDO
SET WATCH(DATAA) CONDITION(DATAA NOT = 0) *> データ監視の設定(2)
DO
  DISPLAY DATA(DATAA)
ENDDO
```

実行結果



次のプログラムを続行する TD コマンドを TD コマンド群に指定すると，プログラムの実行が再開されてしまうために，それ以降の TD コマンドは実行されません。TD コマンド群が複数ある場合，最初に実行される TD コマンド群にプログラムを続行する TD コマンドが指定されていたときは，それ以降の TD コマンド群は実行されません。

- GO
- STEP IN
- STEP OVER
- STEP TO
- STOP

TD コマンド群が実行されなかったとき，カウンタ変数は加算されません。中断点を設定したときのスキップ回数のカウントは加算されます。

使用例

次の TD コマンドを実行した結果，文番号 6000 で比較条件式 A ＝ 50 が成立したとします。SET WATCH コマンドで指定された TD コマンド群が実行され，GO コマンドによってプログラムが続行されます。SET BREAK コマンドで指定された無条件中断の TD コマンド群の DISPLAY DATA コマンドは実行されません。カウンタ変数 C の値も変化しません。

```
SET WATCH(WATCH50)  CONDITION(A = 50)
DO
  GO
ENDDO

SET BREAK STATEMENT(6000) COUNTER(C)
DO
  DISPLAY DATA(B)
ENDDO
```

注意事項

- 「データ監視条件の設定」と「中断点の設定」が成立しても，TD コマンド群でプログラムを続行するコマンドが実行されたときは，入力待ちにならないで，実行が継続されます。
- ステップイン，ステップオーバーまたはジャンプで中断した文で，同時に「データ監視条件の設定」と「中断点の設定」が成立しても，TD コマンド群は一切実行しません。
- 中断点のスキップ回数は，TD コマンド群の実行には関係なく，中断点を通過した場合には必ず更新されます。
- 中断点に達した場合のメッセージおよびデータ監視条件が成立した場合のメッセージは，TD コマンド群の実行に関係なく表示されます。
- コマンド実行中に割り込みを指示した場合は，その時点で実行している TD コマンド以降のコマンドは実行されないで中断します。
- シミュレーション手続きの場合は，シミュレーション対象入出力文の次の文でプログラムが中断します。

5.2.19　カウンタ変数

カウンタ変数は，TD コマンド群の実行回数を参照するための変数です。カウンタ変数を定義した TD コマンド群の中だけで参照できます。2 進項目のデータとして扱われ，1 から始まり，TD コマンド群が実行されるたびに 1 ずつ加算されます。最大値は 2,147,483,647 で，最大値を超えた場合は，最大値のまま更新されません。

カウンタ変数は，表示，代入，比較に使用できます。ただし，添字・部分参照には使用できません。

- 表示

  2 進項目のデータ属性，または 16 進で表示できます。
- 代入

送り出し側作用対象に指定して，データ項目へ値を代入できます。このとき，2 進項目の代入規則に従います。カウンタ変数へ値を代入することはできません。

代入については，「2.3.1　データの比較・代入規則」の「(1)　データの代入規則」を参照してください。

- 比較

  2 進項目の比較規則に従い，データ項目，定数と比較できます。

  比較については，「2.3.1　データの比較・代入規則」の「(2)　データの比較規則」を参照してください。

TD コマンド群が実行される場合，カウンタ変数と同じ名前のデータ名が参照範囲にあるときは，カウンタ変数が指定できるオペランドであれば，カウンタ変数とみなします。カウンタ変数が指定できないオペランドであれば，データ名とみなします。

使用例

カウンタ変数によって，実行される TD コマンドを変えます。

```
SET QUALIFICATION (#PROG(給与計算))
SET BREAK  ST(1350)  COUNTER(CNT)
DO
    IF  C (CNT = 1)
      DISPLAY DATA(合計)                *> 一度目に実行される
    ENDIF
    IF  C (CNT = 2)
        DISPLAY DATA(総合計)            *> 二度目に実行される
    ENDIF
ENDDO
```

5.2.20　記号名

記号名は，副プログラムシミュレーション・ファイルシミュレーションのために定義する名前です。記号名については，「2.2.7　プログラムの単体テスト」の「(4)　記号名」を参照してください。

カウンタ変数と同じ名前は指定できません。ソース要素中のデータ名と同じ名前を記号名として定義したときは，記号名とみなします。記号名に対応するデータ名が表の場合は，添字を必要とします。部分参照できるデータ名であれば，部分参照の指定ができます。

使用例

ファイルシミュレーションのレコード定義に記号名を割り当てます。

```
    SIMULATE FILE(OUTPUT-FILE) OPENMODE(OUTPUT) RECORD(SYMBOL1)
       DEFINE
        01 SYMBOL1
          02 SYMBOL11
           03 SYMBOL111
       ENDDEFINE
          DISPLAY DATA (SYMBOL111(1, J)(1:5))
    ENDSIMULATE
```

5.2.21　繰り返し指定

次のシミュレーションで，反復回数と TD コマンド群を指定します。「5.4　TD コマンドの詳細」の「5.4.25　REPEAT（繰り返し指定）」を参照してください。

- 副プログラムシミュレーションで副プログラムが呼ばれる
- ファイルシミュレーション入出力文が実行される
- DC シミュレーションで DC 文が実行される

繰り返し指定の使用例については「2.2.7　プログラムの単体テスト」の「(3)　ファイルシミュレーション」の「(a)　ファイル入出力文のシミュレーション」の使用例 2 を参照してください。

繰り返し指定を複数記述することによって，シミュレーションが行われるごとに実行される TD コマンド群を変更できます。反復回数分のシミュレーションで TD コマンドを実行されると，次のシミュレーションから次の繰り返し指定の TD コマンド群が実行されます。すべての繰り返し指定を実行したあとに，シミュレーションが実行されるときは，最後の繰り返し指定の TD コマンド群を実行します。

5.2.22　入出力文選択指定

シミュレーションする次の入出力文に対応させて，ファイルシミュレーションの手続きを実行させます。「5.4　TD コマンドの詳細」の「5.4.22　SELECT ACTION（入出力文選択指定）」を参照してください。

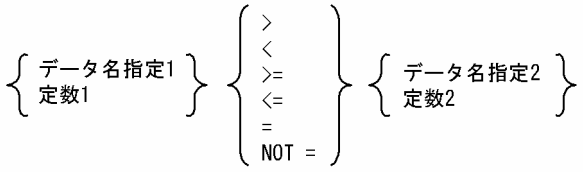
使用例については，「2.2.7　プログラムの単体テスト」の「(3)　ファイルシミュレーション」の「(a)　ファイル入出力文のシミュレーション」の使用例 3 を参照してください。

- READ
- WRITE
- REWRITE
- START
- DELETE

5.2.23　比較条件式

条件を指定します。

形式



5.3 TD コマンドの一覧

5.3.1 プログラムの実行の制御と追跡

テストプログラムの実行を制御したり，実行状態を追跡したりする TD コマンドの一覧を次に示します。

表 5-4 実行を制御したり，実行状態を追跡したりする TD コマンドの一覧

目的	コマンド名
プログラムの実行を中断させる中断点を設定する。	SET BREAK
SET BREAK コマンドで設定した中断点を解除する。	RESET BREAK
設定されている中断点の一覧を表示する。	DISPLAY BREAK
プログラムの実行を中断する条件を設定する。	SET WATCH
SET WATCH コマンドで設定したデータ監視条件を解除する。	RESET WATCH
プログラムが起動前状態のときは，実行を開始する。中断状態のときは，実行を再開する。	GO
プログラムを 1 文実行して中断する。	STEP IN
プログラムを 1 文実行して中断する。ただし，CALL 文，関数呼び出しを持つ文および INVOKE 文は 1 文として実行する。	STEP OVER
プログラムの中断時に，実行を再開する位置を変更する。	STEP TO
実行中のプログラムを強制的に終了する。	STOP
プログラムの実行を追跡し，指定した単位で通過点を表示する。	SET TRACE
SET TRACE コマンドで開始したトレース表示を中止する。	RESET TRACE
フロー情報の蓄積を開始する。	SET FLOW
フロー情報の蓄積を中止する。	RESET FLOW
蓄積されているフロー情報を表示する。	DISPLAY FLOW
プログラムが中断している現在位置を表示する。	DISPLAY POINT

5.3.2 データの操作

テストプログラムで扱われているデータの値を表示したり，任意の値を代入したりする TD コマンドの一覧を次に示します。

表 5-5　データの値を表示したり，任意の値を代入したりする TD コマンドの一覧

目的	コマンド名
データの値を指定された形式で表示する。	DISPLAY DATA
オブジェクト参照データ項目または既定義オブジェクト SELF，EXCEPTION-OBJECT の指定によって，その参照するオブジェクトのオブジェクト参照の値，クラス名，インスタンスオブジェクト・ファクトリオブジェクトの種別，データの値を表示する。	DISPLAY OBJECT
DISPLAY FACTORY コマンドは，クラス名の指定によって，ファクトリオブジェクトのデータの値を表示する。	DISPLAY FACTORY
データに値を代入する。	ASSIGN DATA
条件が成立したときは，ELSE の前の TD コマンド群を実行する。条件が成立しないときは，ELSE のあとの TD コマンド群を実行する。	IF
領域を確保して，アドレス名にアドレスを設定する。	ALLOCATE AREA
アドレス名の示す領域を解放する。	FREE AREA
アドレスを取得する。	ASSIGN ADDRESS

5.3.3　単体テスト

連動するプログラムやファイルなどの実行環境が整わない段階でこれらの処理をシミュレーションし，単体でプログラムをテストする TD コマンドの一覧を次に示します。

表 5-6　単体でテストする TD コマンドの一覧

目的	コマンド名
主プログラムシミュレーションで実行する手続きを設定する。	SIMULATE MAIN
副プログラムシミュレーションで実行する手続きを設定する。	SIMULATE SUB
ファイルのシミュレーションで実行する手続きを設定する。	SIMULATE FILE
擬似実行する入出力文に対応させてファイルシミュレーションの手続きを実行させる。	SELECT ACTION
入出力条件を擬似的に発生させる。	GO END GO EOP GO INVALID GO ERROR
TD コマンドで手続きを設定して DC シミュレーションを実行する。	SIMULATE DC
シミュレーションで一度に実行する手続きと，繰り返す回数を指定する。	REPEAT
記号名に対応するデータ名が集団項目のとき，集団項目に属するデータ名に対応する記号名を指定する。	レベル番号

5.3.4　バッチによるテスト

バッチによるテストで使用する TD コマンドの一覧を次に示します。

表 5-7　バッチによるテストで使用する TD コマンドの一覧

目的	コマンド名
複数のテストケースをまとめて定義して，その中から実行するテストケースを指定する。	TEST
テストケースの手続きを設定する。	CASE
テストケースの終了状態（ケースコード）を設定する。	ASSIGN CASECODE
テストデバッグを終了する。	QUIT※
指定した文字を表示する。	DISPLAY COMMENT※

注※

ラインモードでも使用できます。

5.3.5　テストデバッグの環境設定

入力する文字の扱いを変更したり，指定したファイルから TD コマンドを入力したりする TD コマンドの一覧を次に示します。

表 5-8　入力する文字の扱いを変更したり，指定したファイルから TD コマンドを入力したりする TD コマンドの一覧

目的	コマンド名
テストデバッグの対象とする翻訳単位またはソース要素を変更する。	SET QUALIFICATION
翻訳単位指定またはソース要素指定を解除する。	RESET QUALIFICATION
指定したファイルから TD コマンドを入力する。	#INCLUDE
入力する文字の扱い（等価規則）を変更する。	#OPTION
実行結果の出力先を指定する。	SET PRINT
実行結果の出力先の指定を解除する。	RESET PRINT
ラインモードで端末に出力されるテストデバッグの実行結果をファイルへ出力する。	SET LOG
テストデバッグの実行結果をラインモードで端末に出力するように戻す。	RESET LOG

5.3.6　その他の機能

その他の機能の TD コマンドの一覧を次に示します。

表 5-9　その他の機能の TD コマンドの一覧

目的	コマンド名
プログラムのファイル管理記述項で指定した装置名にファイルを割り当てる。	ASSIGN DEVICE
UNIX コマンドを実行する。	!※

注※

ラインモードで使用できます。

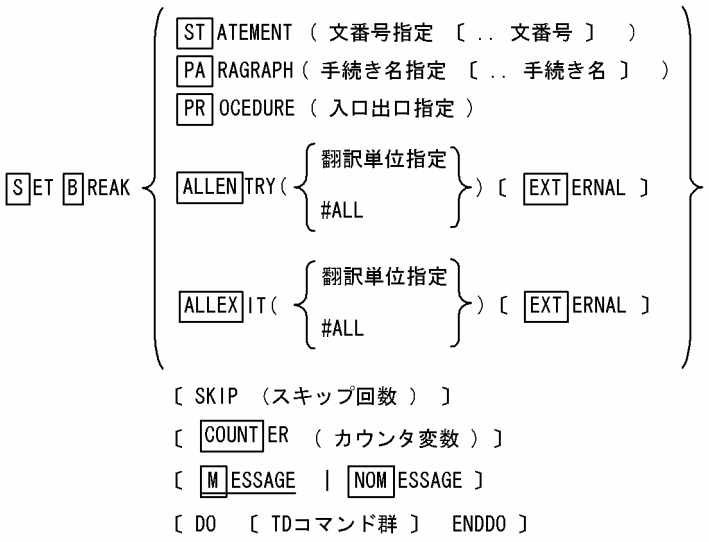
5.4 TD コマンドの詳細

5.4.1 SET BREAK／RESET BREAK（中断点の設定と解除）

(1) SET BREAK（中断点の設定）

プログラムの実行を中断させる中断点を設定します。中断したときに実行する TD コマンドの指定もできます。

形式



- STATEMENT（文番号指定）
  文へ中断点を設定します。
- STATEMENT（文番号指定.. 文番号）
  指定した範囲内のすべての文および手続き名に中断点を設定します。
- PARAGRAPH（手続き名指定）
  手続きへ中断点を設定します。
- PARAGRAPH（手続き名指定 .. 手続き名）
  指定した範囲内のすべての文および手続き名に中断点を設定します。
- PROCEDURE（入口出口指定）
  指定したソース要素の入口または出口に中断点を設定します。
- ALLENTRY（翻訳単位指定 | #ALL）
  次のソース要素の入口に中断を設定します。
  ・翻訳単位指定：指定した翻訳単位に属するソース要素
  ・#ALL：プログラムに属するすべてのソース要素

- ALLEXIT（翻訳単位指定 | #ALL）

  次のソース要素の出口に中断を設定します。

  ・翻訳単位指定：指定した翻訳単位に属するソース要素

  ・#ALL：プログラムに属するすべてのソース要素

- EXTERNAL

  ALLENTRY オペランドの指定があるときは，ほかの翻訳単位の呼び出し文から呼び出される入口だけに中断点を設定します。ALLEXIT オペランドの指定があるときは，ほかの翻訳単位に戻る出口だけに中断点を設定します。

  中断点を設定する入口と出口は，次のとおりになります。

入口または出口に中断点を設定するソース要素

- プログラム

  外部プログラム

- クラス

  メソッド

- 利用者定義関数

入口または出口に中断点を設定しないソース要素

- プログラム

  内部プログラム

- SKIP（スキップ回数）

  設定した中断点で中断する周期を指定できます。中断してから次に中断するまで，スキップ回数分，中断点で止まらずに通過させることができます。指定できるのは，1～2,147,483,647 の整数です。スキップ回数のカウントは，プログラムの開始時に 0 が設定されます。

- COUNTER（カウンタ変数）

  カウンタ変数は，プログラムの開始時に 0 が設定され，DO～ENDDO オペランドで指定した TD コマンド群が実行されるたびに 1 ずつ値が増加します。

- MESSAGE | NOMESSAGE

  中断を知らせるメッセージを表示するかどうかを指定します。

- DO 〔 TD コマンド群 〕 ENDDO

  中断時に実行する TD コマンド群を指定します。

注意事項

- 中断点が設定されている文または手続きに，再度，中断点の設定をしたときは，前の設定が解除され，新しい指定が有効となります。指定方法が異なる場合でも，同じ文に対する設定のときは，次のように，新しい指定が有効になります。

使用例

SET BREAK PARAGRAPH (P1) SKIP (10)の指定で，SET BREAK STATEMENT (100) SKIP(1)は解除されます。

```
       SET BREAK STATEMENT(100)  SKIP(1)
       SET BREAK PARAGRAPH(P1)   SKIP(10)
```

```
100  P1.
200    MOVE  A  TO  B.
```

- 中断点が設定されている入口または出口に，再度中断点を設定したときは，前の設定が解除され，新しい指定が有効となります。指定方法が異なる場合でも，同じ入口または出口に対する設定のときは，次のように，新しい指定が有効になります。

使用例

プログラム「社員」の入口に設定された中断点の SKIP 指定は 20 となります。

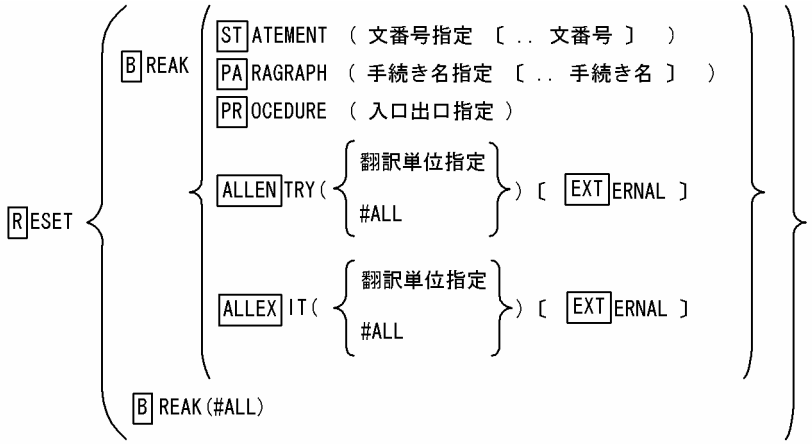
```
SET BREAK  ALLENTRY (#ALL) EXTERNAL  SKIP(10)
SET BREAK  PROCEDURE (#PROGRAM(社員/社員)  #ENTRY)  SKIP(20)
```

- STATEMENT オペランドと PARAGRAPH オペランドの範囲指定，ALLENTRY オペランド，ALLEXIT オペランドは，複数の中断点を一度に設定します。カウンタ変数とスキップ回数は，設定された個々の中断点ごとにカウントされます。
- 中断点を設定できる文が存在しない文番号を指定した場合は，その文番号よりあとで最も近い中断点を設定できる文に中断点を設定します。ただし，指定した文番号よりあとに中断点を設定できる文がないときは，翻訳単位の最後の文を仮定します。
- STATEMENT オペランドで範囲を指定したとき，最後の文番号に中断点を設定できる文がないときは，指定した文番号より前で最も近い文にまで，中断点を設定します。
- STATEMENT オペランドと PARAGRAPH オペランドの範囲指定では，終了位置が開始位置より前にあってはなりません。

(2) RESET BREAK（中断点の解除）

SET BREAK コマンドで設定した中断点を解除します。

形式



- STATEMENT（文番号指定）

文の中断点を解除します。

- STATEMENT（文番号指定 .. 文番号）
  指定した範囲内のすべての文および手続き名の中断点を解除します。
- PARAGRAPH（手続き名指定）
  手続きの中断点を解除します。
- PARAGRAPH（手続き名指定 .. 手続き名）
  指定した範囲内のすべての文および手続き名の中断点を解除します。
- PROCEDURE（入口，出口指定）
  指定したソース要素の入口または出口の中断点を解除します。
- ALLENTRY（翻訳単位指定 | #ALL）
  次のソース要素の入口の中断点を解除します。
  ・翻訳単位指定：指定した翻訳単位に属するソース要素
  ・#ALL：プログラムに属するすべてのソース要素
- ALLEXIT（翻訳単位指定 | #ALL）
  次のソース要素の出口の中断点を解除します。
  ・翻訳単位指定：指定した翻訳単位に属するソース要素
  ・#ALL：プログラムに属するすべてのソース要素
- EXTERNAL
  ALLENTRY オペランドの指定があるときは，ほかの翻訳単位の呼び出し文から呼び出される入口の中断点を解除します。ALLEXIT オペランドの指定があるときは，ほかの翻訳単位に戻る出口の中断点を解除します。
  EXTERNAL オペランド指定時に解除される入口，出口の中断点は，「(1) SET BREAK（中断点の設定)」の入口または出口に中断点を設定するソース要素と同様となります。
- BREAK（#ALL）
  すべての中断点を解除します。

注意事項

中断点を設定したときの SET BREAK コマンドの指定方法と同一の指定方法でなくても解除できます。

使用例

すべてのソース要素の入口に中断点を設定したあと，プログラム PGM01 と PGM02 の外部プログラムの入口の中断点だけを解除します。

```
SET  BREAK  ALLENTRY (#ALL)  EXTERNAL

RESET BREAK  ALLENTRY (#PROGRAM(PGM01))  EXTERNAL
RESET BREAK  ALLENTRY (#PROGRAM(PGM02))  EXTERNAL
```

5.4.2　DISPLAY BREAK（中断点の表示）

設定されている中断点の一覧を表示します。

形式

- BREAK(#ALL ¦ 翻訳単位指定)

  #ALL を指定すると，すべての中断点を表示します。翻訳単位指定を指定すると，指定した翻訳単位の中断点を表示します。

- PRINT

  SET PRINT コマンドで指定した出力先に結果を表示します。

  SET PRINT コマンドについては，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除)」の「(1)　SET PRINT（テスト結果蓄積先の設定)」を参照してください。

使用例と表示形式

使用例 1

プログラム PROGRAM_A の入口に中断点が設定されています。

```
DISPLAY BREAK(#ALL)
```

```
***** 中断点一覧 *****
#ENTRY <PROGRAM_A/PROGRAM_A> スキップ回数（0）
```

使用例 2

次の中断点が設定されています。

- プログラム PROGRAM_B の入口・出口
- 内部プログラム SUB1 の入口・出口
- プログラム PROGRAM_B の行番号 1000 の文
- 内部プログラムの手続き名 P1（行番号 1600 の文）・行番号 1800 の文

```
DISPLAY BREAK(#PROGRAM(PROGRAM_B))
```

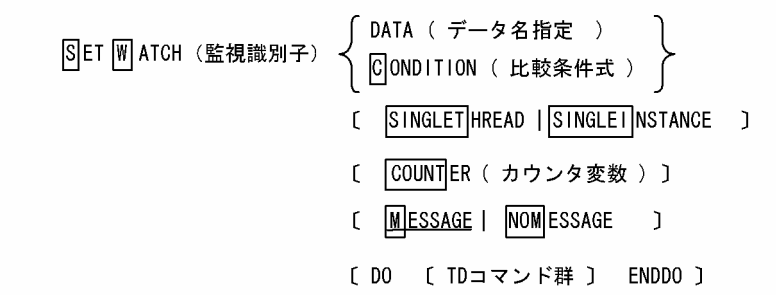
```
***** 中断点一覧 *****
#ENTRY <PROGRAM_B/PROGRAM_B> スキップ回数（0）
1000 <PROGRAM_B/PROGRAM_B> スキップ回数（0）
#ENTRY <PROGRAM_B/SUB1> スキップ回数（0）
1600 P1 <PROGRAM_B/SUB1> スキップ回数（0）
1800 <PROGRAM_B/SUB1> スキップ回数（1）
#EXIT <PROGRAM_B/SUB1> スキップ回数（0）
#EXIT <PROGRAM_B/PROGRAM_B> スキップ回数（0）
```

5.4.3　SET WATCH／RESET WATCH（データ監視条件の設定と解除）

(1)　SET WATCH（データ監視条件の設定）

プログラムの実行を中断する条件を設定します。

形式



- WATCH（監視識別子）
  データ監視条件を識別するための名称を指定します。
- DATA（データ名指定）
  監視対象とするデータ名を指定します。値の変化を監視します。詳細については，「2.2.1　プログラムの中断」の「(2)　データ監視条件の設定による中断」を参照してください。
- CONDITION（比較条件式）
  監視対象となるデータの値に対しての比較条件式を指定します。条件式の評価を監視します。
  比較条件式の指定の詳細については，「2.3.1　データの比較・代入規則」の「(2)　データの比較規則」を参照してください。
  条件式の監視については，「2.2.1　プログラムの中断」を参照してください。
- SINGLETHREAD
  単一のスレッドのデータを監視します。
- SINGLEINSTANCE
  単一のオブジェクトのデータを監視します。
- COUNTER（カウンタ変数）
  カウンタ変数は，プログラムの開始時に 0 が設定され，DO～ENDDO オペランドで指定した TD コマンド群が実行されるたびに 1 ずつ値が増加します。
- MESSAGE | NOMESSAGE
  監視条件の成立を知らせるメッセージを表示するかどうかを指定します。
- DO 〔 TD コマンド群 〕 ENDDO
  中断時に実行する TD コマンド群を指定します。

注意事項

- データ名指定または比較条件式のデータ名に，カウンタ変数は指定できません。

- データ名指定または比較条件式のデータ名に添字付きのデータ名を指定するときの添字は，整数定数でなければなりません。また，添字の範囲指定はできません。
- データ名指定または比較条件式のデータ名に部分参照のデータ名を指定するとき，最左端位置と長さ指定は整数定数でなければなりません。
- すでに設定された監視識別子を指定して，再度，データ監視条件を設定した場合は，以前の設定が無効となり，新しい設定が有効となります。
- オブジェクトに属するデータ項目の領域がオブジェクトごとに生成されるときは，生成されたオブジェクトの領域ごとに監視が行われます。カウンタ変数は，オブジェクトのデータごとにカウントされます。SINGLEINSTANCE オペランドは，その一つのデータを指定して監視できます。
- データ項目の領域が再帰によって複数生成されるときは，生成された領域ごとに監視が行われます。カウンタ変数は，データごとにカウントされます。
- マルチスレッドで実行するプログラムで，指定したデータ名の領域がスレッドごとに生成されるときは，生成されたスレッドの領域ごとに監視が行われます。カウンタ変数は，スレッドのデータごとにカウントされます。SINGLETHREAD オペランドは，その一つのデータを指定して監視できます。
- 連絡節で定義した LIMIT 指定がない動的長基本項目への部分参照指定は，プログラムの起動前には指定できません。指定すると，KCCC3434T-E のメッセージが出力されて，処理が中止されます。
- 比較条件式の両辺にデータ名を指定する場合は，データ名の定義場所や属性などによって指定できる条件が異なります。比較条件式に指定できるかどうかの組み合わせを次に示します。ここでは「NORMAL」や「RECURSIVE」などをグループ名の例として説明します。

表 5-10　監視の条件式に指定できる組み合わせ

左辺	右辺			
	NORMAL	RECURSIVE	FACTORY	INSTANCE
NORMAL	○	×	×	×
RECURSIVE	×	△	×	×
FACTORY	×	×	△	×
INSTANCE	×	×	×	△

（凡例）

○：指定できる

×：指定できない

△：同じ場所に定義されたデータ名であれば指定できる

それぞれのグループに含まれるデータ名を次に示します。

表 5-11　グループの内容

グループ名	データ名
NORMAL	• ファイル節，作業場所節，画面節，通信節で定義されたデータ名 • 再帰不可能なソース要素の連絡節で定義されたデータ名および特殊レジスタ

グループ名	データ名
RECURSIVE	再帰可能なソース要素の連絡節または局所場所節で定義されたデータ名，および特殊レジスタ
FACTORY	ファクトリデータ
INSTANCE	インスタンスデータ

(2) RESET WATCH（データ監視条件の解除）

SET WATCH コマンドで設定したデータ監視条件を解除します。

形式

```
RESET WATCH ( 監視識別子 | #ALL )
```

- WATCH（監視識別子）
  データ監視条件を識別するための名称を指定します。
- WATCH（#ALL）
  すべてのデータ監視条件を解除します。

5.4.4 GO（実行の開始／再開）

プログラムが起動前状態のときは，実行を開始します。中断状態のときは，実行を再開します。

形式

```
ユーザプログラム起動前の形式
  GO [ PARAMETER ( 引数の文字列 ) ]
     [ COVERAGE ]

ユーザプログラム中断状態の形式
  GO [ STATEMENT ( 文番号 )   ]
     [ PARAGRAPH ( 手続き名 ) ]
```

- PARAMETER（引数の文字列）
  起動時にプログラムへ渡す引数を指定します。
- COVERAGE
  カバレージ情報を蓄積します。
- STATEMENT（文番号）・PARAGRAPH（手続き名）
  プログラムの実行を再開させる位置を変更するときに，再開する文または手続きを指定します。

注意事項

- PARAMETER オペランドおよび COVERAGE オペランドは，プログラムの中断状態で指定したときは無効となります。

- PARAMETER オペランドは，次の場合は指定できません。
  ・プログラムの連動実行をしている。
  ・-SimMain コンパイラオプションでプログラムをコンパイルして主プログラムシミュレーションをしている。
  文字列に空白，括弧，アポストロフィ（'）またはダブルコーテーション（"）を使用する場合の指定方法については，「5.1　TD コマンドの指定方法」を参照してください。
- COVERAGE オペランドによってカバレージ情報を蓄積する翻訳単位は，コンパイル時に-CVInf コンパイラオプションを指定します。
- STATEMENT オペランドおよび PARAGRAPH オペランドは，プログラムの起動前に指定したときは無効となります。
- STATEMENT オペランドおよび PARAGRAPH オペランドは，中断しているソース要素の文または手続きを指定します。宣言手続きで中断しているときは，その宣言手続きの中の文または手続きを指定します。宣言手続きの外で中断しているときは，宣言手続きの外の文または手続きを指定します。
- STATEMENT オペランドに ENTRY 文の文番号は指定できません。

5.4.5　STEP IN（ステップイン実行の開始／再開）

プログラムが起動前のときは，実行を開始します。中断状態のときは，実行を再開します。プログラムを 1 文実行して中断します。

形式

```
ユーザプログラムが起動前の形式

   STEP IN 〔PARAMETER（ 引数の文字列 ）〕
           〔COVERAGE 〕

ユーザプログラムが中断状態の形式

   STEP IN
```

- PARAMETER（引数の文字列）
  起動時にプログラムへ渡す引数を指定します。
- COVERAGE
  カバレージ情報を蓄積します。

注意事項

- PARAMETER オペランドおよび COVERAGE オペランドは，プログラムの中断状態で指定したときは無効となります。
- PARAMETER オペランドは，次の場合は指定できません。
  ・プログラムの連動実行をしている。

・-SimMain コンパイラオプションでプログラムをコンパイルして主プログラムシミュレーションをしている。

文字列に空白，括弧，アポストロフィ（'）またはダブルコーテーション（"）を使用する場合の指定方法については，「5.1　TD コマンドの指定方法」を参照してください。

- COVERAGE オペランドによってカバレージ情報を蓄積する翻訳単位は，コンパイル時に-CVInf コンパイラオプションを指定します。

5.4.6　STEP OVER（ステップオーバー実行の開始／再開）

プログラムが起動前のときは，実行を開始します。中断状態のときは，実行を再開します。プログラムを 1 文実行して中断します。ただし，CALL 文，関数呼び出しを持つ文および INVOKE 文は 1 文として実行します。

形式

ユーザプログラムが起動前の形式

```
STEP OVER 〔 PARAMETER（引数の文字列 ）〕
          〔 COVERAGE 〕
```

ユーザプログラムが中断状態の形式

```
STEP OVER
```

- PARAMETER（引数の文字列）

  起動時にプログラムへ渡す引数を指定します。

- COVERAGE

  カバレージ情報を蓄積します。

注意事項

- PARAMETER オペランド，COVERAGE オペランドは，プログラムの中断状態で指定したときは無効となります。
- PARAMETER オペランドは，次の場合は指定できません。

  ・プログラムの連動実行をしている。

  ・-SimMain コンパイラオプションでプログラムをコンパイルして主プログラムシミュレーションをしている。

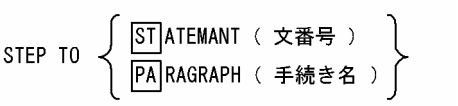
  文字列に空白，括弧，アポストロフィ（'）またはダブルコーテーション（"）を使用する場合の指定方法については，「5.1　TD コマンドの指定方法」を参照してください。

- COVERAGE オペランドによってカバレージ情報を蓄積する翻訳単位は，コンパイル時に-CVInf コンパイラオプションを指定します。

5.4.7 STEP TO（ステップツー実行の再開）

プログラムの中断時に，実行を再開する位置を変更します。

形式



- STATEMENT（文番号）・PARAGRAPH（手続き名）
  プログラムを再開させるときに，再開させる文または手続き名を指定します。

注意事項

- STATEMENT オペランド，PARAGRAPH オペランドは，中断しているソース要素の文または手続き名を指定します。宣言手続きで中断しているときは，その宣言手続きの中の文または手続きを指定します。宣言手続きの外で中断しているときは，宣言手続きの外の文または手続き名を指定します。
- STATEMENT オペランドに ENTRY 文の文番号は指定できません。

5.4.8 STOP（プログラムの強制終了）

実行中のプログラムを強制的に終了します。

形式

```
STOP
```

5.4.9 SET TRACE／RESET TRACE（トレース表示の開始と中止）

(1) SET TRACE（トレース表示の開始）

プログラムの実行を追跡し，指定した単位で通過点を表示します。

形式

```
SET TRACE
  〔 PROCEDURE | PARAGRAPH | STATEMENT 〕
  〔 INTERFACE 〕
  〔 FROMSTATEMENT(文番号指定) | FROMPARAGRAPH(手続き名指定) 〕
  〔 TOSTATEMENT(文番号指定) | TOPARAGRAPH(手続き名指定) | TOSTEP(ステップ数) 〕
  〔 PRINT 〕
```

- PROCEDURE | PARAGRAPH | STATEMENT

表示単位を指定します。

PROCEDURE：ソース単位の入口，出口を表示します。

PARAGRAPH：ソース単位の入口，出口および手続き名を表示します。

STATEMENT：ソース単位の入口，出口，手続き名および文を表示します。

- INTERFACE

  入口では引数を表示し，出口では終了コードを表示します。

- FROMSTATEMENT（文番号指定）|FROMPARAGRAPH（手続き名指定）

  表示を開始する位置を指定します。

- TOSTATEMENT（文番号指定）|TOPARAGRAPH（手続き名指定）

  表示を終了する位置を指定します。

- TOSTEP（ステップ数）

  表示を終了するステップ数を指定します。最初に表示したときのステップ数を 1 とします。指定できる範囲は，1～2,147,483,647 の整数です。

- PRINT

  SET PRINT コマンドで指定した出力先に結果を表示します。

  SET PRINT コマンドについては，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）」の「(1)　SET PRINT（テスト結果蓄積先の設定）」を参照してください。

注意事項

- プログラムを再開始した場合，FROMSTATEMENT および FROMPARAGRAPH オペランドの指定がある場合はオペランドで指定した位置を通過したときに表示が開始されます。指定がない場合は，プログラム開始時に表示が開始されます。
- FROMSTATEMENT オペランドの文番号指定に該当する文または手続き名がない場合は，指定した文番号以降の最も近い文または手続き名を仮定します。

  また，TOSTATEMENT オペランドを指定した文番号に該当する文または手続き名がない場合は，指定した文番号以前の最も近い文または手続き名を仮定します。
- FROMSTATEMENT および FROMPARAGRAPH オペランドと TOSTATEMENT および TOPARAGRAPH オペランドの指定がある場合は，開始する位置と終了する位置に同じ位置は指定できません。

(2)　RESET TRACE（トレース表示の中止）

SET TRACE コマンドで開始したトレース表示を中止します。

形式

RESET TRACE

5.4.10　SET FLOW／RESET FLOW（フロー情報の蓄積開始と終了）

(1)　SET FLOW（フロー情報の蓄積開始）

フロー情報は，プログラムの実行時に通過した文の実行経路です。SET FLOW コマンドで，フロー情報の蓄積を開始します。蓄積したフロー情報は，DISPLAY FLOW コマンドで表示できます。また，バッチモードでは，実行時エラーが発生したときに，フロー情報を表示します。

形式

```
SET FLOW 〔 PROCEDURE | PARAGRAPH | STATEMENT 〕
        〔 STACK ( 蓄積数 ) 〕
```

- PROCEDURE | PARAGRAPH | STATEMENT
  蓄積する通過点の単位を指定します。
  PROCEDURE：ソース単位の入口，出口の情報を蓄積します。
  PARAGRAPH：ソース単位の入口，出口と手続き名の情報を蓄積します。
  STATEMENT：ソース単位の入口，出口と手続き名および文の情報を蓄積します。
- STACK（蓄積数）
  蓄積する通過点の数を指定します。指定できる数は 1～500 の整数で，省略時は 99 が仮定されます。

(2)　RESET FLOW（フロー情報の蓄積中止）

フロー情報の蓄積を中止します。

形式

```
RESET FLOW
```

5.4.11　DISPLAY FLOW（フロー情報の表示）

蓄積されているフロー情報を表示します。フロー情報とは，プログラムの実行経路を示す通過点の情報です。フロー情報は，最後に通過した文から，プログラムの実行順と逆の順序に表示されます。

形式

```
DISPLAY FLOW 〔 PRINT 〕
```

- PRINT
  SET PRINT コマンドで指定した出力先に結果を表示します。
  SET PRINT コマンドについては，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）」の「(1)　SET PRINT（テスト結果蓄積先の設定）」を参照してください。

表示例

ソース要素の表示形式については，「2.5　ソース要素・中断点・通過点の表示形式」を参照してください。なお，マルチスレッドプログラムでないときは，スレッド ID は表示されません。

```
>>>>> 手続きフロー情報
スレッドID(123) #EXIT <飲み物管理/飲み物管理>
スレッドID(123) #EXIT <牛乳/牛乳>
スレッドID(123) #EXIT <牛乳/日付管理>
スレッドID(123) 41700 USE-CL1 OF SITUATION-1 <牛乳/日付管理>
スレッドID(123) 41200 USE-IN1 OF SITUATION-1 <牛乳/日付管理>
スレッドID(123) #EXIT <牛乳/製造管理>
スレッドID(123) 52600 USE-S-CL1 <牛乳/製造管理>
スレッドID(123) 52100 USE-S-IN1 <牛乳/製造管理>
スレッドID(123) #ENTRY <牛乳/製造管理 >
スレッドID(123) 40600 MAIN-START OF SITUATION-1 <牛乳/日付管理>
スレッドID(123) 40500 SITUATION-1 <牛乳/日付管理>
スレッドID(123) 40300 主処理 <牛乳/日付管理>
スレッドID(123) #ENTRY <牛乳/日付管理>
スレッドID(123) #ENTRY <牛乳/牛乳>
スレッドID(123) 4100 PARAM-GET <飲み物管理/飲み物管理>
スレッドID(123) 4000 主処理 <飲み物管理/飲み物管理>
スレッドID(123) #ENTRY <飲み物管理/飲み物管理>
<<<<<

>>>>> 文フロー情報

スレッドID(123) #EXIT <飲み物管理/飲み物管理>
スレッドID(123) 6300 <飲み物管理/飲み物管理>
スレッドID(123) #EXIT <牛乳/牛乳>
スレッドID(123) 20800 <牛乳/牛乳>
スレッドID(123) 20200 <牛乳/牛乳>
スレッドID(123) 20100 <牛乳/牛乳>
スレッドID(123) #ENTRY <牛乳/牛乳>
スレッドID(123) 4200 <飲み物管理/飲み物管理>
スレッドID(123) 4100 PARAM-GET <飲み物管理/飲み物管理>
スレッドID(123) 4000 主処理 <飲み物管理/飲み物管理>
スレッドID(123) #ENTRY <飲み物管理/飲み物管理>

<<<<<

>>>>> プログラムフロー情報

スレッドID(123) #EXIT <飲み物管理/飲み物管理>
スレッドID(123) #EXIT <牛乳/牛乳>
スレッドID(123) #EXIT <牛乳/日付管理>
スレッドID(123) #EXIT <牛乳/製造管理>
スレッドID(123) #ENTRY <牛乳/製造管理 >
スレッドID(123) #ENTRY <牛乳/日付管理>
スレッドID(123) #ENTRY <牛乳/牛乳>
スレッドID(123) #ENTRY <飲み物管理/飲み物管理>

<<<<<
```

5.4.12　DISPLAY POINT（現在位置の表示）

プログラムが中断している現在の位置を表示します。

形式

```
DISPLAY POINT [ PRINT ]
```

- PRINT

  SET PRINT コマンドで指定した出力先に結果を表示します。

  SET PRINT コマンドについては，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）」の「(1)　SET PRINT（テスト結果蓄積先の設定）」を参照してください。

使用例と表示形式

使用例 1

マルチスレッドプログラムのプログラム PROGRAM_A の入口で中断しています。

```
DISPLAY POINT
```

```
KCCC3326T-I　＜中断点＞ 呼び出し順序番号（1）　#ENTRY ＜PROGRAM_A/PROGRAM_A＞　スレッドID（20）
```

使用例 2

プログラム PROGRAM_B の内部プログラム SUB1 の 1600 行の手続き名 P1 で中断しています。

```
DISPLAY POINT
```

```
KCCC3326T-I　＜中断点＞ 呼び出し順序番号（2）　1600 P1 ＜PROGRAM_B/SUB1＞
```

5.4.13　DISPLAY DATA（データの値表示）

データの値を指定された形式で表示します。

形式

```
DISPLAY DATA ( データ名指定 | #ALL )
        [ ATTRIBUTE | HEX | ALLTYPE ]
        [ GROUP | ELEMENT           ]
        [ PRINT                     ]
```

- DATA（データ名指定 | #ALL）

  表示するデータを指定します。

  ・データ名指定：データ名を指定します。

  ・#ALL：現在中断している翻訳単位で参照できるすべてのデータの値を表示します。

- ATTRIBUTE | HEX | ALLTYPE

データの表示形式を指定します。

・ATTRIBUTE：データ名の属性で表示します。

・HEX：16 進で表示します。

・ALLTYPE：ATTRIBUTE・HEX の両方の形式で表示します。

- GROUP | ELEMENT

  集団項目の表示形式を指定します。

  ・GROUP：英数字集団項目を英数字項目とみなし全体を先頭から表示します。または，日本語集団項目を日本語項目とみなし全体を先頭から表示します。

  ・ELEMENT：基本項目単位に表示します。

- PRINT

  SET PRINT コマンドで指定した出力先に結果を表示します。

  SET PRINT コマンドについては，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）」の「(1)　SET PRINT（テスト結果蓄積先の設定）」を参照してください。

注意事項

- GROUP オペランド，および ELEMENT オペランドは，データ名が集団項目でないときは無効となります。
- #ALL を指定したとき，ELEMENT オペランドは無効となります。集団項目は GROUP オペランドの形式で表示されます。
- ELEMENT オペランドは，集団項目を部分参照で表示するときは無効となります。
- HEX オペランドおよび ALLTYPE オペランドは，データ名が内部ブール項目のときは無効となり，ATTRIBUTE オペランド指定時と同様に 0 と 1 から成る文字列が表示されます。内部ブール項目が集団項目の下位項目として存在し，集団項目を ELEMENT 指定で表示するときも HEX オペランドおよび ALLTYPE オペランドは無効となります。
- データの値がデータの属性で表示できないときは，エラーと表示した上で，値を 16 進数で表示します。
- DATA（#ALL）を指定したときは，特殊レジスタ RETURN-CODE と原始プログラムの手続き部で参照されている特殊レジスタも表示されます。

使用例と表示形式

使用例 1 について，AIX(32)および Linux(x86)の場合と，AIX(64)および Linux(x64)の場合の表示例を示します。使用例 2 以降で，明記されていない場合については，AIX(32)および Linux(x86)の場合だけを示します。

AIX(32)および Linux(x86)の場合と，AIX(64)および Linux(x64)の場合の相違は，16 進表示個所でアドレスを表示している部分が 8 けたから 16 けたになっていることです。また，表示するデータによってはデータサイズが変更になるものもあります。変更になるデータについては，「9.1.2 COBOL2002 での 64bit アプリケーション固有の言語仕様」の「(1)　アドレス系データを表現するデータ項目」を参照してください。

使用例 1

基本項目を属性と 16 進で表示します。

```
DISPLAY DATA（基本項目01） ALLTYPE
```

AIX(32)およびLinux(x86)の場合

```
名　称　　基本項目01
　値　　0123456789012345678901234567890123456789012345678901234567890123456789
　　0123456789
*001E8FAC　30313233　34353637　38393031　32333435　36373839　30313233　34353637 38393031
*001E8FCC　32333435　36373839　30313233　34353637　38393031　32333435　36373839 30313233
*001E8FEC　34353637　38393031　32333435　36373839　30313233　34353637　3839
```

AIX(64)およびLinux(x64)の場合

```
名　称　　基本項目01
　値　　0123456789012345678901234567890123456789012345678901234567890123456789
　　0123456789
*00011000001E8FAC　30313233　34353637　38393031　32333435　36373839　30313233　34353637　38393031
*00011000001E8FCC　32333435　36373839　30313233　34353637　38393031　32333435　36373839　30313233
*00011000001E8FEC　34353637　38393031　32333435　36373839　30313233　34353637　3839
```

使用例 2

表の要素の 1～20 番目を，属性と 16 進で表示します。

```
DISPLAY DATA（表基本03-1（1..20）） ALLTYPE
```

- すべての要素が正しく表示された。

```
名　称　　表基本03-1(1..9)
　値　　+0111　+0222　+0333　+0444　+0555　+0666　+0777　+0888　+0999
*00405F30　00111C　*00405F33　00222C　*00405F36　00333C　*00405F39　00444C
*00405F3C　00555C　*00405F3F　00666C　*00405F42　00777C　*00405F45　00888C
*00405F48　00999C
名　称　　表基本03-1(10..18)
　値　　+0000　+1111　+1222　+1333　+1444　+1555　+1666　+1777　+1888
*00405F4B　00000C　*00405F4E　01111C　*00405F51　01222C　*00405F54　01333C
*00405F57　01444C　*00405F5A　01555C　*00405F5D　01666C　*00405F60　01777C
*00405F63　01888C
名　称　　表基本03-1(19..20)
　値　　+1999　+1000
*00405F66　01999C　*00405F69　01000C
```

- 4～6 番目と 8 番目の要素が，データ属性で表示できない値であった。

```
名　称　　表基本03-1(1..3)
　値　　+0111　+0222　+0333
*00405FC0　00111C　*00405FC3　00222C　*00405FC6　00333C
名　称　　表基本03-1(4..6)
エラー
*00405FC9　004441　*00405FCC　414141　*00405FCF　414120
名　称　　表基本03-1(7)
　値　　+0777
*00405FD2　00777C
名　称　　表基本03-1(8)
エラー
*00405FD5　004141
名　称　　表基本03-1(9..17)
　値　　+0999　+0000　+1111　+1222　+1333　+1444　+1555　+1666　+1777
*00405FD8　00999C　*00405FDB　00000C　*00405FDE　01111C　*00405FE1　01222C
*00405FE4　01333C　*00405FE7　01444C　*00405FEA　01555C　*00405FED　01666C
*00405FF0　01777C
名　称　　表基本03-1(18..20)
　値　　+1888　+1999　+1000
*00405FF3　01888C　*00405FF6　01999C　*00405FF9　01000C
```

使用例 3～使用例 6 の TD コマンドが表示する COBOL プログラムのデータ定義

```
01 集団01.
  02 基本02-1 PIC X(5) VALUE ALL 'A'.
  02 集団02-1.
    03 基本03-1  PIC X(5) VALUE ALL 'B'.
    03 集団03-1.
      04 基本04-1  PIC X(5) VALUE ALL 'C'.
      04          PIC X(5) VALUE ALL 'D'.
    03 基本03-2  PIC X(5) VALUE ALL 'E'.
  02 集団02-2.
    03 集団03-2 OCCURS 10.
      04 基本03-3  PIC 9(2) VALUE ZERO.
  02 基本02-2 PIC X(5) VALUE ALL 'F'.
```

使用例 3

集団項目を GROUP 指定で表示します。全体が英数字項目として表示されます。GROUP オペランドは省略できます。

```
DISPLAY DATA（集団01） GROUP  ATTRIBUTE
```

```
名 称  集団01
  値    AAAAABBBBBCCCCCDDDDDEEEEE00000000000000000000FFFFF
```

使用例 4

集団項目を 16 進数で表示します。全体を英数字項目とした上で，16 進数で表示されます。

```
DISPLAY DATA（集団01） HEX
```

```
名 称     集団01
*001E8FAB  41414141  41424242  42424343  43434344  44444444  45454545  45303030  30303030
*001E8FCB  30303030  30303030  30303030  30464646  4646
```

使用例 5

集団項目を，属性と 16 進で表示します。

```
DISPLAY DATA（集団01） ALLTYPE
```

```
名 称     集団01
  値       AAAAABBBBBCCCCCDDDDDEEEEE00000000000000000000FFFFF
 *001E8FAB  41414141  41424242  42424343  43434344  44444444  45454545  45303030  30303030
 *001E8FCB  30303030  30303030  30303030  30464646  4646
```

使用例 6

集団項目を指定して，集団項目の所属するすべての基本項目を，属性と 16 進で表示します。

```
DISPLAY DATA（集団01） ELEMENT ALLTYPE
```

```
名　称       集団01
名　称       基本02-1
  値         AAAAA
*001E8FAB    41414141  41
名　称       集団02-1
名　称       基本03-1
  値         BBBBB
*001E8FAB    42424242  42
名　称       集団03-1
名　称       基本04-1
  値         CCCCC
*001E8FAB    43434343  43
名　称       FILLER
  値         DDDDD
*001E8FAB    44444444  44
名　称       基本03-2
  値         EEEEE
*001E8FAB    45454545  45
名　称       集団02-2
名　称       集団03-2
名　称       基本03-3(1..10)
  値         01     02     03     04     05     06     07     08     09     00
*001E8FAB   3031    *001E8FAB    3032    *001E8FAB    3033    *001E8FAB    3034
*001E8FAB   3035    *001E8FAB    3036    *001E8FAB    3037    *001E8FAB    3038
*001E8FAB   3039    *001E8FAB    3030

名　称       基本02-2
  値         FFFFF
*001E8FAB  46464646  46
```

使用例 7

カウンタ変数を基本項目と属性と 16 進で表示します。

```
DISPLAY DATA (Counter01)  ALLTYPE
```

AIX(32)の場合

```
名　称       #Counter01
  値         1
*--------    00000001
```

AIX(64)の場合

```
名　称       #Counter01
  値         1
*----------------    00000001
```

Linux(x86)の場合

```
名　称       #Counter01
  値         1
*--------    01000000
```

Linux(x64)の場合

```
名　称       #Counter01
  値         1
*----------------    01000000
```

使用例 8

記号名を基本項目と属性と 16 進で表示します。

```
DISPLAY DATA (Return01)  ALLTYPE
```

```
名　称<記号名>      Return01
  値         0
*001E8FAC  0000
```

使用例 9

内部プログラムで中断したときに，すべてのデータを表示します。最外レベル(01,77)単位に表示されます。

```
DISPLAY DATA (#ALL)  ALLTYPE
```

```
プログラム名：COBOL2002_P2/INTER_P2
名 称       RETURN-CODE
  値          +000000000
*00403884   00000000
          :
名 称       LOCALデータ項目
  値        abcdefjhijabcdefjhijabcdefjhijabcdefjhijabcdefjhijabcdefjhijabcdefjhij
            abcdefjhijabcdefjhij
*00403BB0  61626364  65666A68  696A6162  63646566  6A68696A  61626364  65666A68  696A6162
*00403BD0  63646566  6A68696A  61626364  65666A68  696A6162  63646566  6A68696A  61626364
*00403BF0  65666A68  696A6162  63646566  6A68696A  61626364  65666A68  696A6162  63646566
*00403C10  6A68696A

プログラム名：COBOL2002_P2/COBOL2002_P2
名 称       RETURN-CODE
  値          +000000000
*004034A4   00000000
          :
名 称       GLOBALデータ項目
  値        01234567890123456789012345678901234567890123456789012345678901234567890123456789
            0123456789012345678901234567890123456789
*00403B40  30313233  34353637  38393031  32333435  36373839  30313233  34353637  38393031
*00403B60  32333435  36373839  30313233  34353637  38393031  32333435  36373839  30313233
*00403B80  34353637  38393031  32333435  36373839  30313233  34353637  38393031  32333435
*00403BA0  36373839  30313233  34353637  3839
```

RETURN-CODEと手続き部で参照される特殊レジスタが表示される

RETURN-CODEと手続き部で参照される特殊レジスタが表示される

使用例 10

データ名整数 01，整数 02 を 16 進数で表示します。

```
DISPLAY DATA (整数01)  HEX
DISPLAY DATA (整数02)  HEX
```

AIX(32)の場合

```
名　称      整数01
*001E8FAC  01234567
名　称      整数02
*001E8FB4  0A0B0C0D
```

AIX(64)の場合

```
名　称      整数01
*00011000001E8FAC  01234567
名　称      整数02
*00011000001E8FB4  0A0B0C0D
```

Linux(x86)の場合

```
名　称      整数01
*001E8FAC  67452301
名　称      整数02
*001E8FB4  0D0C0B0A
```

Linux(x64)の場合

```
名　称      整数01
*00011000001E8FAC  67452301
名　称      整数02
*00011000001E8FB4  0D0C0B0A
```

5.4.14 DISPLAY OBJECT／DISPLAY FACTORY（オブジェクトのデータ値の表示）

DISPLAY OBJECT コマンドは，オブジェクト参照データ項目または既定義オブジェクト SELF，EXCEPTION-OBJECT の指定によって，その参照するオブジェクトの次の情報を表示します。

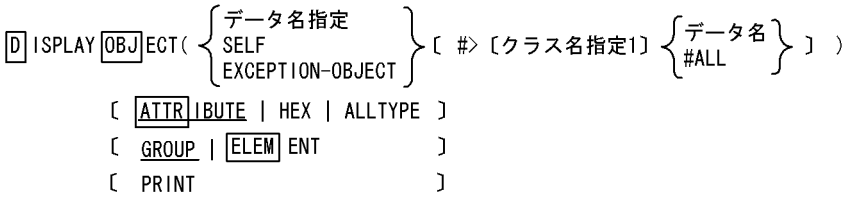
- オブジェクト参照の値
- クラス名
- インスタンスオブジェクト・ファクトリオブジェクトの種別
- データの値

DISPLAY FACTORY コマンドは，クラス名の指定によって，ファクトリオブジェクトの次の情報を表示します。

- データの値

形式

オブジェクト参照を指定してオブジェクトの情報を表示する



- データ名指定
  オブジェクトを参照するオブジェクト参照データ項目を指定します。
- SELF，EXCEPTION-OBJECT
  オブジェクトを参照する既定義オブジェクト名を指定します。
- #>
  データ名指定，SELF または EXCEPTION-OBJECT で参照するオブジェクトの特定のデータ名を指すために指定します。
- クラス名指定 1
  データ名が所属するクラス名を指定します。クラス名は，オブジェクトを定義するクラスまたはそのスーパクラスの名称です。
- データ名
  表示するデータ名を指定します。
- #ALL
  クラス名指定 1 が指定されないときは，オブジェクトを定義するクラスとそのスーパクラスに所属するデータをすべて表示します。クラス名指定 1 が指定されたときは，指定されたクラスに所属す

るデータをすべて表示します。ただし，データが定義されていないスーパクラスは何も表示されません。

- ATTRIBUTE | HEX | ALLTYPE

  データの表示形式を指定します。

  ・ATTRIBUTE：データ名の属性で表示します。

  ・HEX：16 進で表示します。

  ・ALLTYPE：ATTRIBUTE，HEX の両方の形式で表示します。

- GROUP | ELEMENT

  集団項目の表示形式を指定します。

  ・GROUP：英数字集団項目を英数字項目とみなし全体を先頭から表示します。または，日本語集団項目を日本語項目とみなし全体を先頭から表示します。

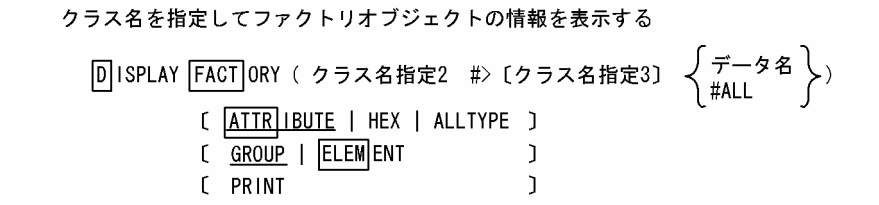
  ・ELEMENT：基本項目単位に表示します。

- PRINT

  SET PRINT コマンドで指定した出力先に結果を表示します。

  SET PRINT コマンドについては，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）」の「(1)　SET PRINT（テスト結果蓄積先の設定）」を参照してください。

形式



- クラス名指定 2

  ファクトリオブジェクトのクラス名を指定します。

- #>

  参照するオブジェクトの特定のデータ名を指すために指定します。

- クラス名指定 3

  データ名が定義されているクラス名を指定します。クラス名は，オブジェクトを作成したクラスまたはそのスーパクラスの名称です。

- データ名

  表示するデータ名を指定します。

- #ALL

  クラス名指定 3 が指定されないときは，オブジェクトを定義するクラスとそのスーパクラスに所属するデータをすべて表示します。クラス名指定 3 が指定されたときは，指定されたクラスに所属するデータをすべて表示します。ただしデータが定義されていないスーパクラスは何も表示されません。

- ATTRIBUTE | HEX | ALLTYPE

データの表示形式を指定します。

・ATTRIBUTE：データ名の属性で表示します。

・HEX：16 進で表示します。

・ALLTYPE：ATTRIBUTE，HEX の両方の形式で表示します。

- GROUP | ELEMENT

  集団項目の表示形式を指定します。

  ・GROUP：集団項目を英数字項目とみなし全体を先頭から表示します。

  ・ELEMENT：基本項目単位に表示します。

- PRINT

  SET PRINT コマンドで指定した出力先に結果を表示します。

  SET PRINT コマンドについては，「5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）」の「(1)　SET PRINT（テスト結果蓄積先の設定）」を参照してください。

注意事項

- GROUP オペランド，および ELEMENT オペランドは，データ名が集団項目でないときは無効となります。
- #ALL を指定したとき，ELEMENT オペランドは無効となります。集団項目は GROUP オペランドの形式で表示されます。
- 既定義オブジェクトの EXCEPTION-OBJECT は，COBOL プログラムの手続きで使用されているときに指定できます。既定義オブジェクトの SELF は，メソッド内では常に指定できます。
- ELEMENT オペランドは，集団項目を部分参照で表示する場合は無効となります。
- HEX | ALLTYPE オペランドは，データ名が内部ブール項目のときは無効となります。
- 表示するデータ名がオブジェクト参照データ項目の場合，データの値を表示します。オブジェクト参照データ項目が参照するオブジェクトの情報とデータ名は表示しません。
- クラス名指定 1，クラス名指定 2，クラス名指定 3 で指定するクラスは，-TDInf コンパイラオプションを指定してコンパイルする必要があります。データ名，#ALL 指定によってデータを表示するスーパクラスも，-TDInf コンパイラオプションを指定してコンパイルしたクラスだけです。
- データ名に，特殊レジスタは指定できません。
- データ名が表のとき，添字および部分参照の最左端位置と長さは整数定数を指定する必要があります。
- データ名，#ALL を指定しないとき，データ名の値は表示しません。DISPLAY OBJECT コマンドはオブジェクトの種別とクラス名だけを，DISPLAY FACTORY コマンドはクラス名だけを表示します。

使用例と表示形式

次の COBOL プログラムのクラス定義に従って，TD コマンドの使用例を示します。

クラス MILK は，クラス DRINK を継承します。

使用例 1 について，AIX(32)および Linux(x86)の場合と，AIX(64)および Linux(x64)の場合の使用例を示します。使用例 2 以降で，明記されていない場合については，AIX(32)および Linux(x86)の場合だけを示します。

AIX(32)および Linux(x86)の場合と，AIX(64)および Linux(x64)の場合の相違は，16 進表示個所でアドレスを表示している部分が 8 けたから 16 けたになっていることです。また，表示するデータによってはデータサイズが変更になるものもあります。変更になるデータについては，「9.1.2 COBOL2002 での 64bit アプリケーション固有の言語仕様」の「(1)　アドレス系データを表現するデータ項目」を参照してください。

クラス DRINK

```
IDENTIFICATION DIVISION.
CLASS-ID. DRINK INHERITS BASE.
      :
IDENTIFICATION DIVISION.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 F-DATA01 PIC S9(9) COMP.
01 F-DATA02.
 02 F-DATA03 PIC X(25).
 02 F-DATA04 PIC 9(18).
 02 F-DATA05 PIC X(5).
01 F-DATA06 PIC X(5).
 :

OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 O-DATA01 PIC S9(9) COMP.
01 O-DATA02 PIC S9(9) COMP.
 :
END CLASS DRINK.
```

クラス MILK

```
IDENTIFICATION DIVISION.
CLASS-ID. MILK INHERITS DRINK.
      :
IDENTIFICATION DIVISION.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 F-DATA01 PIC S9(9) COMP.
01 F-DATA11 PIC S9(9) COMP.
01 F-DATA12 PIC S9(9) COMP.
 :

OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 O-DATA01 PIC S9(9) COMP.
01 O-DATA11 PIC S9(9) COMP.
 :
```

```
END CLASS MILK.
```

使用例 1

既定義オブジェクト SELF の値が，クラス MILK から生成されたファクトリオブジェクトを参照するときに，クラス MILK のスーパクラス DRINK に定義されたデータ名 F-DATA01 を属性と 16 進で表示します。

```
DISPLAY OBJECT (SELF #> #CLASS(DRINK) F-DATA01)  ALLTYPE
```

AIX(32)およびLinux(x86)の場合

```
オブジェクト参照データ名 : SELF
  値        +0000112345
*001E87F0  0001B6D9
オブジェクトを定義するクラス名 : MILK
オブジェクト種別: ファクトリ
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名  称      F-DATA01
  値        +000000000
*001E88F00 00000000
```

AIX(64)およびLinux(x64)の場合

```
オブジェクト参照データ名 : SELF
  値          +00000000000000112345
*00011000001E8EAC  00000000  0001B6D9
オブジェクトを定義するクラス名 : MILK
オブジェクト種別: ファクトリ
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名  称      F-DATA01
  値        +000000000
*00011000001E8FAC  00000000
```

使用例 2

既定義オブジェクト SELF の値がクラス MILK から生成されたインスタンスオブジェクトを参照するとき，クラス MILK のスーパクラス DRINK で定義されたデータ名 O-DATA01 を属性と 16 進で表示します。

```
DISPLAY OBJECT (SELF #> #CLASS(DRINK) O-DATA01)  ALLTYPE
```

```
オブジェクト参照データ名 :  SELF
  値        +0000112345
*001E87F0  0001B6D9
オブジェクトを定義するクラス名 :  MILK
オブジェクト種別: インスタンス
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名  称      O-DATA01
  値        +000000000
*001E88F0  00000000
```

使用例 3

既定義オブジェクト SELF の値が，クラス MILK から生成されたインスタンスオブジェクトを参照するとき，クラス名を指定しないで，データ名 O-DATA01 を属性と 16 進で表示します。

```
DISPLAY OBJECT (SELF #> O-DATA01)  ALLTYPE
```

AIX(32)の場合

```
オブジェクト参照データ名 : SELF
  値         +0000112345
*001E87F0   0001B6D9
オブジェクトを定義するクラス名 : MILK
オブジェクト種別 : インスタンス
オブジェクト情報
クラスMILKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000000
*001E88F0   00000000
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000001
*001EC1F0   00000001
```

AIX(64)の場合

```
オブジェクト参照データ名 : SELF
  値         +00000000000000112345
*00000000001E87F0   00000000  0001B6D9
オブジェクトを定義するクラス名 : MILK
オブジェクト種別 : インスタンス
オブジェクト情報
クラスMILKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000000
*00000000001E88F0   00000000
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000001
*00000000001EC1F0   00000001
```

Linux(x86)の場合

```
オブジェクト参照データ名 : SELF
  値         +0000112345
*001E87F0   0001B6D9
オブジェクトを定義するクラス名 : MILK
オブジェクト種別 : インスタンス
オブジェクト情報
クラスMILKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000000
*001E88F0   00000000
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000001
*001EC1F0   01000000
```

Linux(x64)の場合

```
オブジェクト参照データ名 : SELF
  値         +00000000000000112345
*00000000001E87F0   D9B60100  00000000
オブジェクトを定義するクラス名 : MILK
オブジェクト種別 : インスタンス
オブジェクト情報
クラスMILKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000000
*00000000001E88F0   00000000
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名  称       0-DATA01
  値         +000000001
*00000000001EC1F0   01000000
```

使用例 4

既定義オブジェクト SELF の値が，クラス MILK から生成されたファクトリオブジェクトを参照するとき，クラス MILK のスーパクラス DRINK のすべてのファクトリデータを，属性と 16 進で表示します。

```
DISPLAY OBJECT (SELF #> #CLASS(DRINK) #ALL)  ALLTYPE
```

```
オブジェクト参照データ名 : SELF
  値       +0000112345
*001E87F0  0001B6D9
オブジェクトを定義するクラス名 : MILK
オブジェクト種別: ファクトリ
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名　称     F-DATA01
  値       +000000000
*001E88F0  00000000
名　称     F-DATA02
  値       AAAAABBBBBCCCCCDDDDDEEEEE0102030405060708090FFF

*001E8930   41414141  41424242  42424343 43434344 44444444 45454545  45303130  32303330
*001E8950   34303530  36303730  38303930 30464646

名　称     F-DATA06
  値       AAAAA
*001E96F0  41414141  41
```

使用例 5

オブジェクト参照データ項目 A-DRINK の値が，クラス MILK から生成されたファクトリオブジェクトを参照するとき，オブジェクトのすべてのファクトリデータを，属性で表示します。

```
DISPLAY OBJECT (A-DRINK #> #ALL)  ATTRIBUTE
```

```
オブジェクト参照データ名 : A-DRINK
  値         +0000112345
オブジェクトを定義するクラス名 : MILK
オブジェクト種別: ファクトリ
オブジェクト情報
クラスMILKで定義されたデータ項目と値 :
名　称     F-DATA01
  値       +000000001
名　称     F-DATA11
  値       +000000002
名　称     F-DATA12
  値       +000000003

オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名　称     F-DATA01
  値       +000000000
名　称     F-DATA02
  値       AAAAABBBBBCCCCCDDDDDEEEEE0102030405060708090FFF
名　称     F-DATA06
  値       AAAAA
```

使用例 6

クラス名 MILK を指定して，ファクトリオブジェクトのスーパクラス DRINK のデータ名 F-DATA01 を属性と 16 進で表示します。

```
DISPLAY FACTORY (#CLASS(MILK) #> #CLASS(DRINK) F-DATA01)  ALLTYPE
```

```
ファクトリオブジェクト名 : MILK
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名　称     F-DATA01
  値       +000000000
*001E87F0  00000000
```

使用例 7

クラス名 MILK を指定して，ファクトリオブジェクトのデータ名 F-DATA01 を属性で表示します。

ファクトリオブジェクトのスーパクラス名は指定しないで，該当するデータ名をすべて表示します。

```
DISPLAY FACTORY (#CLASS(MILK) #> F-DATA01)  ATTRIBUTE
```

```
ファクトリオブジェクト名 : MILK
オブジェクト情報
クラスMILKで定義されたデータ項目と値 :
名　称    F-DATA01
  値      +000000000
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名　称    F-DATA01
  値      +000000001
```

使用例 8

クラス名 MILK を指定して，ファクトリオブジェクトのスーパクラス DRINK のすべてのデータを属性と 16 進で表示します。

```
DISPLAY FACTORY (#CLASS(MILK) #> #CLASS(DRINK) #ALL)  ALLTYPE
```

```
ファクトリオブジェクト名 : MILK
オブジェクト情報
クラスDRINKで定義されたデータ項目と値 :
名　称      F-DATA01
  値        +000000000
*001E88F0   00000000
名　称      F-DATA02
  値        AAAAABBBBBCCCCCDDDDDEEEEE0102030405060708090FFF
*001E8930   41414141  41424242  42424343  43434344  44444444  45454545  45303130  32303330
*001E8950   34303530  36303730  38303930  30464646
名　称      F-DATA06
  値        AAAAA
*001E96F0   41414141  41
```

使用例 9

オブジェクト参照データ項目 B-DRINK がクラス MILK のインスタンスオブジェクトを参照するとき，オブジェクト参照の値，クラス名，オブジェクトの種別を表示します。

```
DISPLAY OBJECT (B-DRINK)
```

```
オブジェクト参照データ名 : B-DRINK
  値          +0000112345
オブジェクトを定義するクラス名 : MILK
オブジェクト種別: インスタンス
```

5.4.15　ASSIGN DATA（データの値代入）

データに値を代入します。

形式

```
[A]SSIGN DATA（ データ名指定1 ）[V]ALUE（ データ名指定2 | 定数 ）
```

- DATA（データ名指定 1）
  値が代入されるデータ名を指定します。
- VALUE（データ名指定 2）
  代入する値を持つデータ名を指定します。
- VALUE（定数）
  代入する定数を指定します。

注意事項

- データ名指定には，添字の範囲指定はできません。
- カウンタ変数を指定できるのは，データ名指定 2 だけです。

5.4.16　IF（データの値比較）

条件が成立したときは，ELSE の前の TD コマンド群を実行します。条件が成立しないときは，ELSE のあとの TD コマンド群を実行します。

形式

```
IF [C]ONDITION( 比較条件式 )
    〔 TDコマンド群 〕
    〔 ELSE 〔 TDコマンド群 〕〕
ENDIF
```

- CONDITION（比較条件式）

  比較については，「2.3.1　データの比較・代入規則」の「(2)　データの比較規則」を参照してください。

- TD コマンド群

  条件が成立したときと，成立しなかったときに実行する TD コマンド群を指定します。

使用例

文番号 8700 で中断し，R-CODE（I）の値によって，TD コマンドを実行します。

```
SET QUALIFICATION (#PROG(社員))
SET BREAK  ST(8700) DO
  IF CONDITION(R-CODE(I) = 0)
    DISPLAY COM('**TEST OK **')
  ELSE
    IF CONDITION(R-CODE(I) NOT= 8)
      DISPLAY COMMENT ('??? TEST NG')
      DISPLAY DATA(R-CODE(I))  HEX
      ASSIGN DATA(R-CODE(I)) VALUE(8)
    ELSE
      DISPLAY COMMENT ('**ERROR TEST OK**')
    ENDIF
  ENDIF
  GO
ENDDO
GO
```

5.4.17 ALLOCATE AREA／FREE AREA（領域の確保と解放）

(1) ALLOCATE AREA（領域の確保）

アドレス名によって参照されるデータ項目と同じサイズの領域を確保して，アドレス名にアドレスを設定します。

また，指定されたサイズの領域を確保して，アドレスデータ項目またはポインタ項目にアドレスを設定できます。

形式

ALLOCATE AREA（ データ名指定 1 ） LENGTH （ 領域長 ）

ALLOCATE AREA（ データ名指定 2 ）

- AREA（データ名指定 1）

  ポインタ項目またはアドレスデータ項目を指定します。

  LENGTH オペランドに指定されたサイズの領域を確保して，指定されたポインタ項目またはアドレスデータ項目に，確保した領域のアドレスを設定します。

- LENGTH（領域長）

  指定された領域長分の領域を確保します。指定できる長さは，1～2,147,483,647 バイトです。なお，データ名を使った指定はできません。

- AREA（データ名指定 2）

  アドレス名を指定します。アドレス名によって参照されるデータ項目と同じサイズの領域を確保して，指定されたアドレス名に確保した領域のアドレスを設定します。

注意事項

- 確保した領域は，X'00'で初期化されます。
- 確保した領域は，次の場合に解放されます。

  ・FREE AREA コマンドを実行したとき。

  ・プログラムの実行が終了したとき。
- マルチスレッドプログラムでは使用できません。
- LENGTH オペランドを指定して確保した領域を使用する場合に，確保した領域を超える部分にアクセスしたときの動作は保証しません。

  例えば，次の条件でテストした場合，ADDRESS_NAME には英数字 3 文字分の領域にしか割り当たりません。

  DATA2 へ代入するのは'ABC'の 3 文字ですが，定義は 10 文字なので残りの 7 文字分の埋め字処理で領域破壊が起きます。

  ・COBOL ソースでのデータ定義

```
01 DATA1 ADDRESSED BY ADDRESS_NAME.
```

```
  02 DATA2 PIC X(10).
01 ADDRESS_DATA USAGE ADDRESS.
```

・デバッガで実行する TD コマンド

```
ALLOCATE AREA( ADDRESS_DATA ) LENGTH( 3 )
ASSIGN DATA( ADDRESS_NAME ) VALUE( ADDRESS_DATA )
ASSIGN DATA( DATA2 ) VALUE( 'ABC' ) *> 領域破壊を起こす。
```

(2) FREE AREA（領域の解放）

アドレス名の示す領域を解放します。

また，アドレスデータ項目またはポインタ項目も指定できます。

形式

```
FREE AREA（ データ名指定 ）
```

- AREA（データ名指定）
  確保した領域を解放するアドレス名を指定します。
  または，確保した領域を解放するアドレスデータ項目，もしくはポインタ項目を指定します。

注意事項

- ALLOCATE AREA コマンドで確保された領域以外は解放できません。
- 領域の解放後，アドレス名に ZERO が設定されます。アドレスデータ項目またはポインタ項目を指定した場合も同様です。
- マルチスレッドプログラムでは使用できません。

5.4.18 ASSIGN ADDRESS（アドレスの取得）

データ名のアドレスを取得します。

形式

```
ASSIGN ADDRESS( データ名指定1 ) DATA( データ名指定2 )
```

- ADDRESS（データ名指定 1）
  アドレス名，アドレスデータ項目，またはポインタ項目を指定します。
  指定したデータ名に，DATA オペランドで指定したデータ名のアドレスが設定されます。
- DATA（データ名指定 2）
  アドレスを取得したいデータ名を指定します。

注意事項

- データ名指定 2 には，次のデータ項目は指定できません。

・指標名

・アドレス名

・動的長基本項目

- データ名指定 2 にブール項目を指定した場合，バイト境界になっていればアドレスを取得できます。バイト境界になっていない場合は，エラーになります。
- アドレスを取得したあとは，取得したアドレスが指すデータが参照できるかどうかは確認しません。参照できなくなっていた場合，そのアドレスを使ってデータ操作したときの動作は保証しません。
- データ名指定 1 にアドレス名を指定したとき，指定したアドレス名で参照されるデータとデータ名指定 2 に指定したデータの構造のチェックはしません。二つの構造が違うことによる領域破壊などの動作は保証しません。

  例えば，次の条件で，データ名指定 1 に ADDRS，データ名指定 2 に SRC_DATA1 を指定した場合，DATA1（アドレス名 ADDRS で参照されるデータ）と SRC_DATA1 は，構造およびサイズとも異なりますが，ASSIGN ADDRESS コマンドは成功します。

  ただし，DATA3 OF DATA1 の部分は SRC_DATA1 の範囲外になるため，その値を更新すると，領域破壊になります。

  ・COBOL ソースでのデータ定義

```
01 DATA1 ADDRESSED BY ADDRS.
  02 DATA2 PIC X(10).
  02 DATA3 PIC X(10).
01 SRC_DATA1 PIC X(10).
```

  ・デバッガで実行する TD コマンド

```
ASSIGN ADDRESS( ADDRS ) DATA( SRC_DATA1 )
ASSIGN DATA( DATA3 OF DATA1) VALUE( "SAMPLE" ) *> 領域破壊になる。
```

- データ名指定 2 にアドレス名で参照されたデータ名を指定した場合，アドレス名が指すアドレスが設定されます。

5.4.19 SIMULATE MAIN（主プログラムシミュレーションの設定）

主プログラムシミュレーションで実行する手続きを指定します。詳細については，「2.2.7 プログラムの単体テスト」の「(1) 主プログラムシミュレーション」を参照してください。

形式

```
SIMULATE MAIN（ プログラム指定 〔 入口名 〕）
  〔 MESSAGE | NOMESSAGE 〕
  〔 TDコマンド群 〕
ENDSIMULATE
```

- MAIN（プログラム指定）

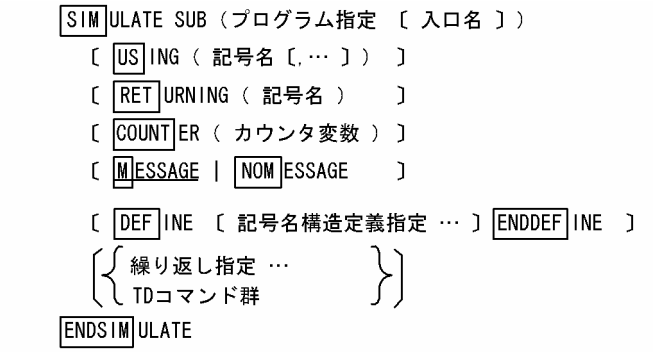
  主プログラムから呼び出される副プログラムの入口名を指定します。

- 入口名
  ENTRY 文で指定される入口点の名前を指定します。
- MESSAGE | NOMESSAGE
  シミュレーションの開始を知らせるメッセージを表示するかどうかを指定します。
- TD コマンド群
  テストデバッグ対象のプログラムへ渡すインタフェースを設定する手続きを TD コマンドで指定します。

5.4.20 SIMULATE SUB（副プログラムシミュレーションの設定）

副プログラムシミュレーションで実行する手続きを指定します。詳細については，「2.2.7 プログラムの単体テスト」の「(2) 副プログラムシミュレーション」を参照してください。

形式



- SUB（プログラム指定）
  シミュレーション対象とするプログラム定義の名前を指定します。
- 入口名
  ENTRY 文で指定される入口点の名前を指定します。
- USING（記号名〔, … 〕 ）
  CALL 文の USING で指定したデータ項目をシミュレーションの手続きで参照するための記号名を指定します。USING で指定したデータ項目の順番と USING オペランドに指定した記号名の順番が対応します。
- RETURNING（記号名）
  シミュレーション手続きで RETURNING に値を設定するための記号名を指定します。
- COUNTER（カウンタ変数）
  カウンタ変数は，プログラムの開始時に 0 が設定され，シミュレーションが実行されるたびに 1 ずつ値が増加します。
- MESSAGE | NOMESSAGE
  シミュレーションの開始を知らせるメッセージを表示するかどうかを指定します。

- DEFINE 〔 記号名構造定義指定 … 〕 ENDDEFINE

  USING または RETURNING オペランドで指定した記号名に対応するデータ名が集団項目のとき，集団項目に所属するデータ名を指定します。詳細については，「5.4.26　レベル番号（記号名構造指定）」を参照してください。

- 繰り返し指定，TD コマンド群

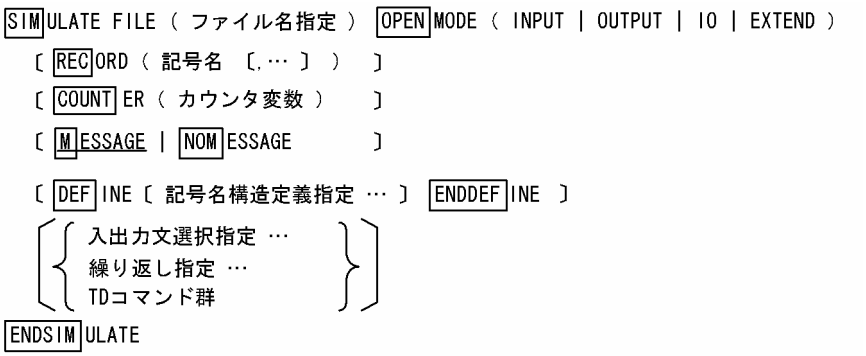
  プログラムの呼び出しをシミュレーションで実行する手続きを TD コマンドで指定します。

  繰り返し指定の詳細については，「5.4.25　REPEAT（繰り返し指定）」を参照してください。

5.4.21　SIMULATE FILE（ファイルシミュレーションの設定）

ファイルのシミュレーションで実行する手続きを指定します。詳細については，「2.2.7　プログラムの単体テスト」の「(3)　ファイルシミュレーション」を参照してください。

形式



- FILE（ファイル名指定）

  シミュレーション対象とするファイル名を指定します。

- OPENMODE（INPUT | OUTPUT | IO | EXTEND）

  ファイルのオープンモードを指定します。

- RECORD（記号名〔,… 〕 ）

  シミュレーション手続き中で使用する記号名を定義します。記号名は，該当するファイルのレコード名定義に対応します。

- COUNTER（カウンタ変数）

  カウンタ変数は，プログラムの開始時に 0 が設定され，シミュレーションが実行されるたびに 1 ずつ値が増加します。

- MESSAGE | NOMESSAGE

  シミュレーションの開始を知らせるメッセージを表示するかどうかを指定します。

- DEFINE 〔 記号名構造定義指定 … 〕 ENDDEFINE

RECORD オペランドで指定した記号名に対応するデータ名が集団項目のとき，集団項目に所属するデータ名を指定します。詳細については，「5.4.26　レベル番号（記号名構造指定）」を参照してください。

- 入出力文選択指定

  入出力文で実行する手続きを TD コマンドで指定します。詳細については，「5.4.22　SELECT ACTION（入出力文選択指定）」を参照してください。

- 繰り返し指定，TD コマンド群

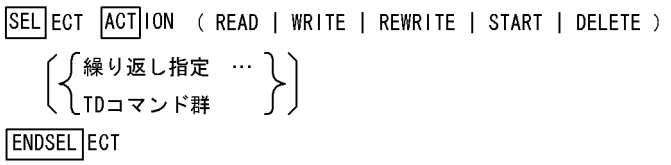
  入出力文で実行する手続きを TD コマンドで指定します。

  繰り返し指定の詳細については，「5.4.25　REPEAT（繰り返し指定）」を参照してください。

5.4.22　SELECT ACTION（入出力文選択指定）

擬似実行する入出力文に対応させて，ファイルシミュレーションの手続きを実行させます。ファイルシミュレーションについては，「2.2.7　プログラムの単体テスト」の「(3)　ファイルシミュレーション」を参照してください。

形式



- ACTION（READ | WRITE | REWRITE | START | DELETE）

  対象とする入出力文を指定します。

- 繰り返し指定・TD コマンド群

  指定した入出力文で実行するシミュレーション手続きを指定します。

  繰り返し指定の詳細については，「5.4.25　REPEAT（繰り返し指定）」を参照してください。

5.4.23　GO END・GO INVALID・GO EOP・GO ERROR（入出力条件シミュレーション）

入出力条件を擬似的に発生させます。このとき，入出力状態に値を設定します。ファイルシミュレーションの手続きにだけ指定できます。

ファイルシミュレーションについては，「2.2.7　プログラムの単体テスト」の「(3)　ファイルシミュレーション」を参照してください。

形式

```
GO END
GO INVALID
GO EOP
GO ERROR
```

注意事項

- GO END コマンド実行後のプログラムの制御は，次のような関係となります。

AT END 指定	USE 指定あり	USE 指定なし
あり	AT END 指定の無条件文に制御が移る	AT END 指定の無条件文に制御が移る
なし	USE 手続きが実行される	入出力文の終わりに制御が移る

- GO INVALID コマンド実行後のプログラムの制御は，次のような関係となります。

INVALID KEY 指定	USE 指定あり	USE 指定なし
あり	INVALID KEY 指定の無条件文に制御が移る	INVALID KEY 指定の無条件文に制御が移る
なし	USE 手続きが実行される	入出力文の終わりに制御が移る

- GO EOP コマンド実行後のプログラムの制御は，次のような関係となります。

END-OF-PAGE 指定	USE 指定あり	USE 指定なし
あり	END-OF-PAGE 指定の無条件文に制御が移る	END-OF-PAGE 指定の無条件文に制御が移る
なし	入出力文の終わりに制御が移る	入出力文の終わりに制御が移る

- GO ERROR コマンド実行後のプログラムの制御は，次のような関係となります。

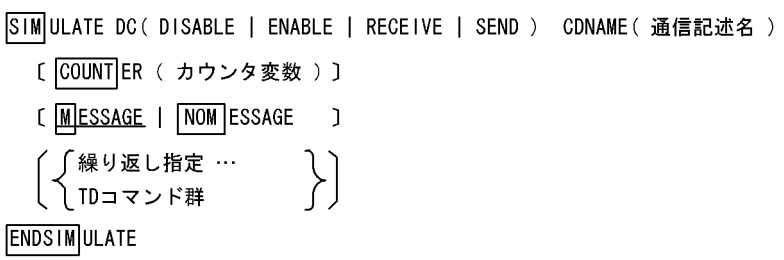
USE 指定あり	USE 指定なし
USE 手続きが実行される	入出力文の終わりに制御が移る

5.4.24　SIMULATE DC（DC シミュレーションの設定）

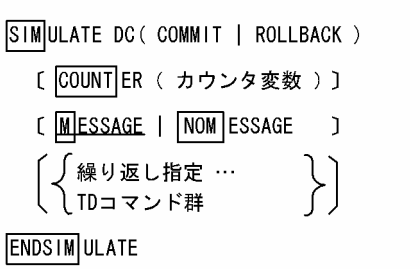
DC シミュレーションを，TD コマンドで手続きを設定して実行させます。詳細については，「2.2.7　プログラムの単体テスト」の「(5)　DC シミュレーション」を参照してください。

形式

操作文がDISABLE，ENABLE，RECEIVE，SENDのとき



操作文がCOMMIT，ROLLBACKのとき



- DC（操作文種別）

  シミュレーションの対象とするDC文を指定します。

  指定できる操作文は，COMMIT，DISABLE，ENABLE，RECEIVE，ROLLBACK，SEND文です。

- CDNAME（通信記述名）

  シミュレーションの対象とする通信記述名を指定します。操作文種別と通信記述名の組み合わせでシミュレーション対象が決まります。

- COUNTER（カウンタ変数）

  カウンタ変数は，プログラムの開始時に0が設定され，シミュレーションが実行されるたびに1ずつ値が増加します。

- MESSAGE | NOMESSAGE

  シミュレーションの開始を知らせるメッセージを表示するかどうかを指定します。

- 繰り返し指定，TDコマンド群

  DC文で実行する手続きをTDコマンドで指定します。

  繰り返し指定の詳細については，「5.4.25　REPEAT（繰り返し指定）」を参照してください。

注意事項

マルチスレッドプログラムでは，使用できません。

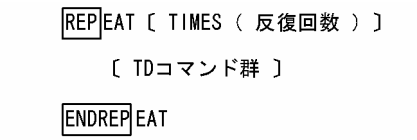
5.4.25　REPEAT（繰り返し指定）

次のシミュレーションで一度に実行する手続きと，繰り返す回数を指定します。

- 副プログラムシミュレーションで副プログラムが呼ばれる
- ファイルシミュレーション入出力文が実行される
- DC シミュレーションで DC 文が実行される

ファイルシミュレーションについては，「2.2.7　プログラムの単体テスト」の「(3)　ファイルシミュレーション」を参照してください。

形式



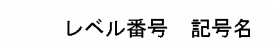
- TIMES（反復回数）
  シミュレーションが実行されることによって，指定された TD コマンド群を繰り返し実行する回数を指定します。指定できるのは 1～2,147,483,647 の整数で，省略時は 1 が仮定されます。
- TD コマンド群
  一度に実行するシミュレーション手続きを指定します。

5.4.26　レベル番号（記号名構造指定）

次のオペランドで指定する記号名に対応するデータ名が集団項目のとき，集団項目に属するデータ名に対応する記号名を指定します。指定された記号名によって，集団項目に属するデータ名をシミュレーションの手続きで参照できます。

- ファイルシミュレーションの RECORD オペランド
- 副プログラムシミュレーションの USING オペランドおよび RETURNING オペランド

形式



- レベル番号
  レベル番号を指定します。
- 記号名
  集団項目に属するデータ名に対応させる記号名を指定します。

注意事項

- DEFINE オペランドと ENDDEFINE オペランドの間でだけ指定できます。

- 対応するデータ名に TYPE 句または SAME AS 句の指定がある場合，レベル番号は展開後の構造で指定します。
- レベル番号の 01～09 の 0 は省略できます。

5.4.27　TEST（テストケースの選択）

複数のテストケースをまとめて定義して，その中から実行するテストケースを指定します。

形式

```
TEST 〔CASEID（ ケース識別子〔,…〕）〕
    〔〔 テストケース | コマンド群 〕… 〕
ENDTEST
```

- CASEID（ケース識別子）

  定義されたテストケースの中から，実行するテストケースを指定します。このオペランドを指定しなかったときは，TEST コマンドの中で定義されたすべてのテストケースを実行します。末尾に*を付けて指定すると，*より前の文字が一致するケース識別子を持つすべてのテストケースを実行します。
- テストケース

  テストケースとして実行する TD コマンドを指定します。

  テストケースの詳細については，「5.4.28　CASE（テストケースの指定）」を参照してください。
- コマンド群

  QUIT，#INCLUDE，#OPTION コマンドが指定できます。

5.4.28　CASE（テストケースの指定）

テストケースの手続きを設定します。

形式

```
CASE 〔 ID（ ケース識別子 ）〕
     〔 TOSTEP（ ステップ数 ）〕
     〔 CONDITION（ ケース選択条件〔 AND … 〕）〕
     〔 TDコマンド群 〕
ENDCASE
```

- ID（ケース識別子）

  テストケースの名称をケース識別子として定義します。英字で始まる 31 文字以内の英数字文字列で指定します。
- TOSTEP（ステップ数）

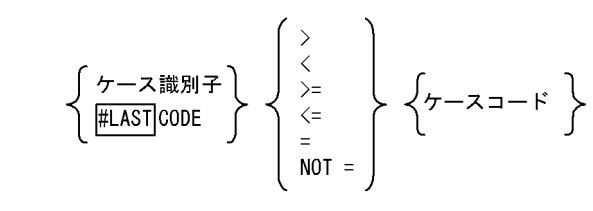
プログラムの実行を開始してから終了するまでのステップ数を指定します。最初の文を実行したときのステップ数を 1 とします。

指定できる範囲は，1～2,147,483,647 の整数です。実行した文が指定された数に達すると，実行が打ち切られます。

- CONDITION（ケース選択条件〔AND … 〕）

  テストケースを実行する条件を指定します。指定したすべての条件を満たしたときにテストケースが実行されます。

  ケース選択条件は，次の条件式で指定します。



  ケース識別子は，先に実行したテストケースのケース識別子を指定します。#LASTCODE を指定した場合は，直前のテストケースが仮定されます。

  ケースコードは，先行するテストケース内で ASSIGN CASECODE コマンドを使用して設定した値です。

- TD コマンド群

  テスト手続きを TD コマンドで指定します。

注意事項

- ステップ数は，スレッドに依存しないで，実行した文をカウントします。

5.4.29 ASSIGN CASECODE（ケースコードの設定）

テストケースの終了状態（ケースコード）を設定します。

形式

```
ASSIGN CASECODE（ ケースコード ）
```

- CASECODE（ケースコード）

  指定できる範囲は，0～2,147,483,647 の整数です。

5.4.30 QUIT（テストデバッグの終了）

テストデバッグを終了します。

形式

```
QUIT
```

5.4.31 DISPLAY COMMENT（注釈の表示）

指定した文字を表示します。

形式

```
DISPLAY COMMENT（ コメント文字列 ）[ PRINT ]
```

- COMMENT（コメント文字列）
  表示する文字列を指定します。文字列に空白，括弧，アポストロフィ（'）またはダブルコーテーション（"）を使用する場合の指定方法については，「5.1 TD コマンドの指定方法」を参照してください。
- PRINT
  SET PRINT コマンドで指定した出力先に結果を表示します。

5.4.32 SET QUALIFICATION／RESET QUALIFICATION（翻訳単位・ソース要素の指定と解除）

(1) SET QUALIFICATION（翻訳単位・ソース要素の指定）

テストデバッグの対象とする翻訳単位またはソース要素を変更します。SET QUALIFICATION コマンドで翻訳単位指定をすると，その後の文番号に翻訳単位を指定したことになります。SET QUALIFICATION コマンドでソース要素指定をすると，その後のソース要素指定ができるオペランドに指定したことになります。

形式

```
SET QUALIFICATION（ 翻訳単位指定 | ソース要素指定 ）
```

- QUALIFICATION（翻訳単位指定）
  仮定する翻訳単位を指定します。
- QUALIFICATION（ソース要素指定）
  仮定するソース要素を指定します。

注意事項

- SET QUALIFICATION コマンドは，プログラムの開始時および再開時に解除されます。また，プログラムの実行中止（STOP コマンド）でも解除されます。
- 翻訳単位指定は，文番号だけに有効です。その他にはソース要素指定をしなくてはなりません。

SET QUALIFICATION コマンド以降，文番号以外も対象とする場合は，ソース要素指定を指定します。

(2)　RESET QUALIFICATION（翻訳単位・ソース要素の指定の解除）

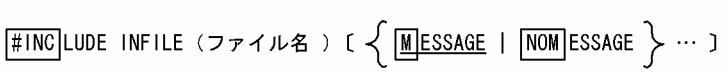
翻訳単位指定またはソース要素指定を解除します。

形式

5.4.33　#INCLUDE（TD コマンド格納ファイルの取り込み）

指定したファイルから TD コマンドを入力します。

形式



- INFILE（ファイル名）

  TD コマンドを格納しているファイル名を指定します。ファイル名に空白，括弧，アポストロフィ（'）または引用符（"）を使用する場合の指定方法については，「5.1　TD コマンドの指定方法」を参照してください。

- MESSAGE | NOMESSAGE

  入力した TD コマンドを表示するかどうかを指定します。

注意事項

- #INCLUDE コマンドで指定したファイルの中に，再度同じファイル名を指定した#INCLUDE コマンドは記述できません。
- 大文字・小文字を区別するため，#OPTION コマンドの指定がファイル名に効かないようにさせたいときは，文字列を指定します。指定方法については，「5.1　TD コマンドの指定方法」の「5.1.4 英大文字と英小文字，英数字文字と拡張文字を区別しての文字列の指定方法」を参照してください。
- ファイル名に，絶対パスの付かないファイル名を指定したときの，ファイルの検索方法は，「3.2.2 環境変数の指定」を参照してください。
- バッチモードの cbltd2k コマンドで-SyntaxOnly オプションの指定の有無に関係なく，指定された TD コマンド格納ファイルの取り込みをします。バッチモードの cbltd2k コマンドで-SyntaxOnly オプションが指定されていた場合は，取り込んだ TD コマンド格納ファイル内の TD コマンドの構文解析が行われます。
- ファイル名に正規表現を指定しても適用されません。

5.4.34　#OPTION（オプションの変更）

TD コマンドの解析方法を指定します。

形式

```
#OPTION 〔 NOCASE | CASE1 | CASE2 〕|〔2002SYNTAX | 85SYNTAX 〕
```

下記の説明で，特殊文字とは次の文字です。

＋ - * ／＝￥' " ; . , （) ＜ ＞ & :

- NOCASE

  同じ文字を表す英大文字と英小文字，英数字文字と拡張文字を同じに扱います。

  ・COBOL2002 の予約語と特殊文字は，英数字文字と拡張文字のどちらでも指定できます。

  ・データ名の拡張文字と英小文字は，英数字文字の英大文字に変換されます。

- CASE1

  同じ文字を表す英数字文字の英大文字と英小文字を同じに扱います。英数字文字と拡張文字は区別します。

  ・COBOL2002 の予約語と特殊文字は，英数字文字で指定します。

  ・データ名の英数字文字の英小文字は英大文字に変換されます。拡張文字のデータ名は変換しません。

- CASE2

  同じ文字を表す英大文字と英小文字，英数字文字と拡張文字を区別します。

  ・COBOL2002 の予約語と特殊文字は，英数字文字の英大文字で指定します。

  ・データ名は変換しません。

- 2002SYNTAX

  TD コマンドのオペランドに指定する利用者定義語の最大長を COBOL2002 の規則と同等にします。利用者定義語に関する詳細については，マニュアル「COBOL2002　言語　標準仕様編」を参照してください。

- 85SYNTAX

  TD コマンドのオペランドに指定する利用者定義語の最大長を COBOL85 の規則と同等にします。

注意事項

バッチモードの起動コマンドで-SyntaxOnly オプションの指定の有無に関係なく，指定は有効となります。

使用例

DISPLAY DATA コマンドのデータ名 data1 および data2 は，英小文字のまま扱われます。

```
#OPTION  NOCASE
SET BREAK STATEMENT(#PROGRAM( PROG ) 100) DO
      #OPTION  CASE2
      DISPLAY DATA( data1 )
ENDDO
DISPLAY DATA( data2 )
```

5.4.35　SET PRINT／RESET PRINT（テスト結果蓄積先の設定と解除）

(1)　SET PRINT（テスト結果蓄積先の設定）

PRINT オペランドが指定された TD コマンドによる実行結果の出力先を指定します。SET PRINT の指定がない場合，バッチモードでは結果出力ファイルへ，実行結果を出力します。

形式

```
SET PRINT OUTFILE（ファイル名 ）
```

- OUTFILE（ファイル名）

  実行結果を出力するファイル名を指定します。ファイル名に空白，括弧，アポストロフィ（'）または引用符（"）を使用する場合の指定方法については，「5.1　TD コマンドの指定方法」を参照してください。

注意事項

- SET PRINT でファイルがオープンされ，RESET PRINT でファイルがクローズされます。ファイルがオープンされている間は，ほかのアプリケーションからの書き込みを行うと，ファイルのオープンでエラーとなります。
- 指定されたファイルがすでにある場合は，上書きとなります。
- SET PRINT コマンドによって出力先が指定されていた場合は，先に指定されたファイルをクローズしてから指定されたファイルをオープンします。
- すでにオープンされているファイルをファイル名に指定できません。したがって，SET LOG コマンドで指定されたファイルも指定できません。
- 大文字・小文字を区別するため，#OPTION コマンドの指定がファイル名に効かないようにさせたいときは，文字列を指定します。指定方法については，「5.1　TD コマンドの指定方法」の「5.1.4 英大文字と英小文字，英数字文字と拡張文字を区別しての文字列の指定方法」を参照してください。
- ファイル名に正規表現を指定しても適用されません。

(2)　RESET PRINT（テスト結果蓄積先の解除）

実行結果の出力先を戻します。

形式

```
RESET PRINT
```

5.4.36　SET LOG／RESET LOG（ログ出力ファイルの設定／解除）

(1)　SET LOG（ログ出力ファイルの設定）

ラインモードで端末に表示されるテストデバッグの実行結果をファイルへ出力します。

形式

```
SET LOG OUTFILE（ ファイル名 ）
```

- OUTFILE（ファイル名）

  実行結果を出力するファイル名を指定します。ファイル名に空白，括弧，アポストロフィ（'）または引用符（"）を使用する場合の指定方法については，「5.1　TD コマンドの指定方法」を参照してください。

注意事項

- SET LOG でファイルがオープンされ，RESET LOG でファイルがクローズされます。ファイルがオープンされている間は，ほかのアプリケーションからの書き込みはできません。RESET LOG 以外でも，初期化，テストデバッガの終了したときにファイルがクローズされます。
- 指定されたファイルがすでにある場合は，上書きとなります。
- SET LOG によって出力先が指定されていた場合は，先に指定されたファイルをクローズしてから指定されたファイルをオープンします。
- オープンされているファイルを，ファイル名に指定することはできません。したがって，SET PRINT コマンドで指定されたファイルも，指定できません。
- 大文字・小文字を区別するため，#OPTION コマンドの指定がファイル名に効かないようにさせたいときは，文字列を指定します。指定方法については，「5.1　TD コマンドの指定方法」の「5.1.4 英大文字と英小文字，英数字文字と拡張文字を区別しての文字列の指定方法」を参照してください。
- ファイル名に正規表現を指定しても適用されません。

(2)　RESET LOG（ログ出力の解除）

テストデバッガの実行結果を端末へ出力するように戻します。

形式

```
RESET LOG
```

5.4.37　ASSIGN DEVICE（装置名へのファイルの割り当て）

プログラムのファイル管理記述項で指定した装置名にファイルを割り当てます。

形式

```
ASSIGN DEVICE（ ファイル定義 ）IOFILE（ ファイル名 ）
```

- DEVICE（ファイル定義）
  ファイル定義は，プログラムのファイル管理記述項の ASSIGN 句で定義している装置名の前に，CBL_または CBLX_を付加して指定します。
- IOFILE（ファイル名）
  プログラムで入出力をするファイル名を指定します。

注意事項

- ファイル定義に記述したハイフン（-）は，アンダーバー（_）と見なされます。
- ファイルの実体を割り当てないで OPEN 文を実行すると，実行時エラーメッセージを表示して，プログラムの実行が中断されます。AIX の場合，中断後に，ASSIGN DEVICE コマンドでファイルを割り当てたあと，ジャンプまたはジャンプ実行で OPEN 文を再び実行させることによって，プログラムを続行できます。
- プログラムの SELECT 句で割り当てた装置名に，ASSIGN DEVICE コマンドを使用すると，ASSIGN DEVICE コマンドのファイル定義で指定した装置名に置き換えられます。
- ファイル定義の長さの制限値は，CBL_または CBLX_に COBOL 利用者語の制限値を足した値となります。
- ファイル名に正規表現を指定しても適用されません。
- XMAP3 によるプリンタへの書式印刷機能を使用する場合，ファイル定義には外部装置名の前に CBLX_を付けて指定し，ファイル名には印刷サービス名称を指定します。

5.4.38 ！（UNIX コマンドの実行）

UNIX のコマンドを実行します。

形式

```
！（ UNIXコマンド〔 オペランド …〕）
```

- UNIX コマンド
  実行する UNIX コマンドを指定します。
- オペランド
  実行する UNIX コマンドのオペランドを指定します。

注意事項

- UNIX コマンドおよびオペランドの指定規則は，各 UNIX コマンドに従います。
- SET LOG コマンドで TD コマンドの実行結果をファイルへ出力することを指定しても，UNIX コマンドの実行結果だけはログ出力ファイルへ出力されません。

- UNIX コマンドとオペランドの文字列に空白，括弧，アポストロフィ（'）または引用符（"）を使用する場合の指定方法については，「5.1　TD コマンドの指定方法」を参照してください。

使用例

```
!('ls -L > out.file')
```

- !コマンドは，子プロセスを生成してコマンドを実行します。そのため，子プロセスで実行された cd，setenv などの UNIX コマンドの実行結果は無効となります。

6 カバレージ機能の概要

カバレージ機能を使うと，プログラムのテスト状況を定量的に示したり，プログラムの性能を分析したりできます。この章では，これらの情報の内容と蓄積，表示などの方法について説明します。

6.1　概要

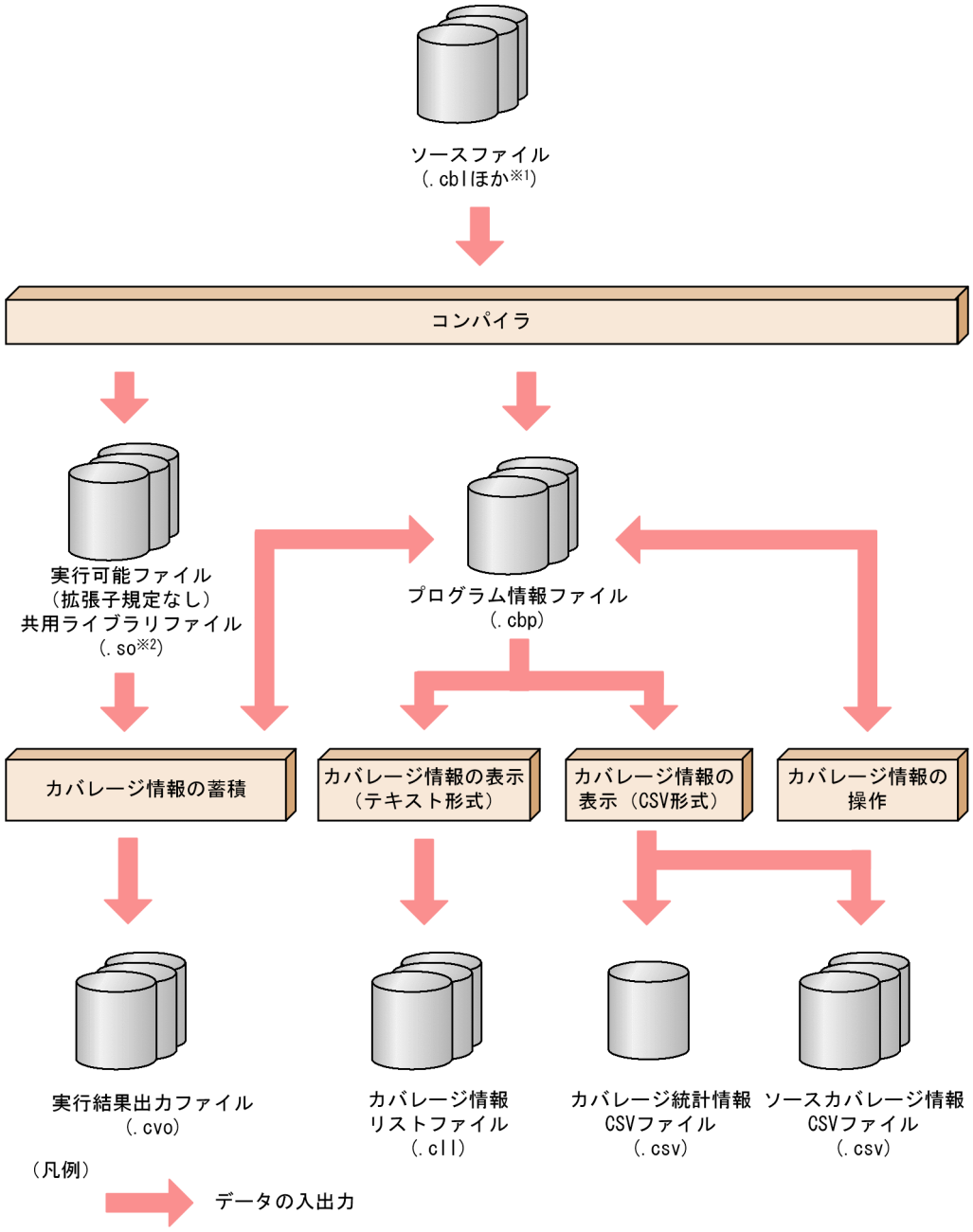
この節では，カバレージ機能の概要について説明します。

6.1.1　カバレージ機能の入出力構成と使用するファイル

(1)　入出力構成

カバレージ機能の入出力構成を図 6-1，図 6-2 に示します。

図 6-1　カバレージ情報の蓄積・表示・操作



注※1

目的に応じて次の拡張子を使用します。

・固定形式正書法で書かれた原始プログラムをコンパイルする場合

.cbl，.CBL，.cob，.ocb，または環境変数 CBLFIX で指定した拡張子

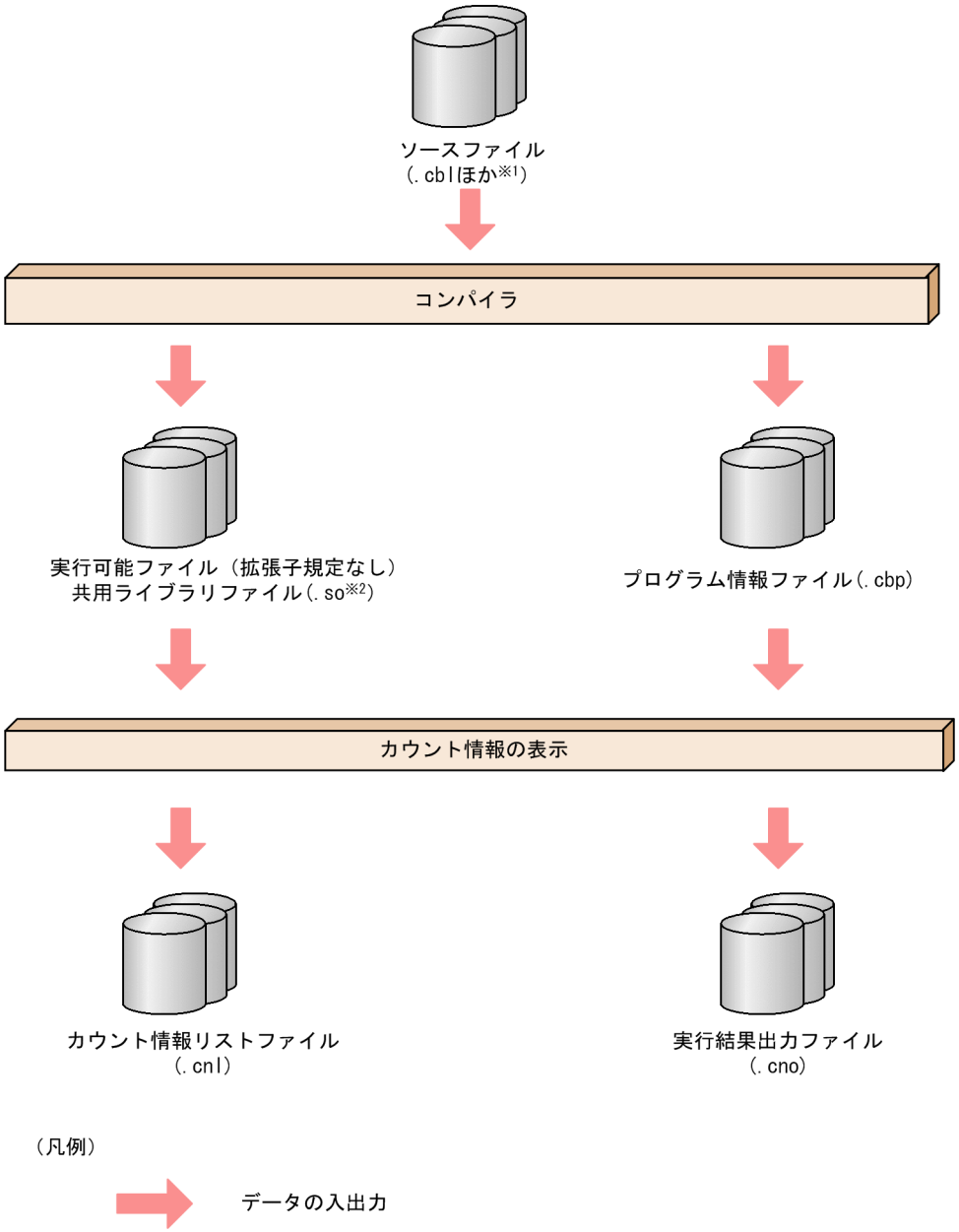
・自由形式正書法で書かれた原始プログラムをコンパイルする場合

.cbf，.ocf，または環境変数 CBLFREE で指定した拡張子

注※2

システムによって違いがあります。詳細は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

図 6-2　カウント情報の表示



注※1

目的に応じて次の拡張子を使用します。

・固定形式正書法で書かれた原始プログラムをコンパイルする場合

.cbl，.CBL，.cob，.ocb，または環境変数 CBLFIX で指定した拡張子

・自由形式正書法で書かれた原始プログラムをコンパイルする場合

.cbf，.ocf，または環境変数 CBLFREE で指定した拡張子

注※2

システムによって違いがあります。詳細は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

(2)　使用するファイル

カバレージ機能で使用するファイルを次に示します。

- 実行可能ファイル

  カバレージ情報の蓄積対象，またはカウント情報の取得対象となる実行可能ファイルです。

- 共用ライブラリファイル

  カバレージ情報の蓄積対象，またはカウント情報の取得対象となる共用ライブラリファイルです。共用ライブラリについては，「6.4.2　共用ライブラリ」を参照してください。

- プログラム情報ファイル

  カバレージ情報の蓄積，またはカウント情報の取得に必要とするプログラムの情報を格納するファイルです。また，蓄積したカバレージ情報を格納するためのファイルでもあります。-CVInf コンパイラオプションを指定してコンパイルしたときに出力されます。

  プログラム情報ファイルは，使用するカバレージ機能と同じバージョンのコンパイラで作成されたものでなければなりません。バージョンが異なる場合は，次に示すエラーメッセージが表示されます。

  KCCC4436T-E 異なるバージョンでコンパイルされたプログラム情報ファイルのため，対象としません。ファイル（xxxxx）

- カバレージ情報リストファイル（カバレージ情報表示時）

  カバレージ表示の実行結果を格納するファイルです。ファイルはテキストファイルです。

- カウント情報リストファイル（カウント情報表示時）

  カウント情報の取得結果を格納するファイルです。ファイルはテキストファイルです。

- カバレージ統計情報 CSV ファイル（カバレージ情報表示時）

  ソースファイル単位または翻訳単位に，カバレージ統計情報を出力するファイルです。複数のプログラム情報ファイルに対するカバレージ統計情報を一つのファイルにまとめて出力します。ファイルは CSV ファイルです。

- ソースカバレージ情報 CSV ファイル（カバレージ情報表示時）

  原始プログラム 1 行ごとのカバレージ情報を出力するファイルです。一つのプログラム情報ファイルに対して，ソースカバレージ情報 CSV ファイルを一つ出力します。ファイルは CSV ファイルです。

- 実行結果出力ファイル

  カバレージ情報の蓄積およびカウント情報の表示で，実行中のメッセージの出力は実行結果出力ファイルに出力されます。次の理由によって，実行結果出力ファイルに出力できない場合については，「7.2.2 プログラムからの連動実行による方法」の「(3)　実行結果出力ファイルにメッセージを出力できないときの処理」を参照してください。

  - 環境変数 CBLTDEXEC の値の指定に誤りがあった場合
  - 結果出力ファイルにエラーが発生し，実行結果出力ファイルが出力できなかった場合
  - 実行環境が整っていなかったためカバレージ情報の蓄積またはカウント情報の表示ができなかった場合

注意事項

- プログラム情報ファイルは，実行可能ファイルまたは共用ライブラリファイルのあるディレクトリを参照します。任意のディレクトリのプログラム情報ファイルを参照するときは，環境変数 CBLPIDIR にディレクトリ名を指定します。
- 環境変数 CBLPIDIR を指定した場合のプログラム情報ファイルの検索順序は，「(3)　使用する環境変数の指定」を参照してください。

(3)　使用する環境変数の指定

環境変数は，カバレージを起動する前に設定します。大文字で指定してください。

表 6-1　使用する環境変数一覧

環境変数	概要
CBLPIDIR	プログラム情報ファイルがあるディレクトリ名を指定します。
CBLTDEXEC	プログラムからの連動実行を指定します。 カバレージ情報の蓄積の場合については，「7.2.2　プログラムからの連動実行による方法」，カウント情報の表示の場合については，「7.6.2　プログラムからの連動実行による方法」を参照してください。
CBLLSLIB	共用ライブラリファイルのファイル名を指定します。
CBLLPATH	共用ライブラリファイルがあるディレクトリ名を指定します。
LD_LIBRARY_PATH※	共用ライブラリファイルがあるディレクトリ名を指定します。
CBLTDEXTARGET	-TDInf および-CVInf コンパイラオプションがない COBOL プログラム，または COBOL プログラム以外から共用ライブラリが呼ばれる場合，カバレージ情報の蓄積，カウント情報の表示を可能にします。

注※

Linux で有効

プログラムの実行に関するその他の環境変数は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注意事項

1. カバレージ情報の蓄積・カウント情報の表示のプログラム情報ファイルを次に示す検索順序で検索します。検索した結果，見つからないときは，プログラム情報ファイルに該当するプログラムはカバレージ情報の蓄積およびカウント情報の表示の対象となりません。

   検索順序

   (1)環境変数 CBLPIDIR で指定したディレクトリ

   (2)実行可能ファイルに含まれるプログラムは，実行可能ファイルのあるディレクトリ

   　共用ライブラリファイルに含まれるプログラムは，共用ライブラリファイルのあるディレクトリ

   (3)カレントディレクトリ

2. カバレージ情報の表示・カバレージ情報の操作のプログラム情報ファイルがパスの付かないファイル名または，相対パスで指定された場合，次に示すディレクトリを検索します。相対パスが指定されたときは，次に示すディレクトリからの相対パスを検索します。絶対パスが指定されたときは検索しません。

   (1)環境変数 CBLPIDIR で指定したディレクトリ

   (2)カレントディレクトリ

3. プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カバレージ情報の蓄積，表示，操作，およびカウント情報の表示の対象となりません。

   カバレージ情報の表示，操作では対象外とするエラーメッセージを出力します。

4. カバレージ情報の蓄積の対象となるプログラム情報ファイルが一つもない場合は，カバレージ情報の蓄積のためのプログラムを実行しません。

5. カバレージ情報の表示の対象となるプログラム情報ファイルが一つもない場合は，カバレージの表示をしません。カバレージ情報リストファイルやカバレージ統計情報 CSV ファイルも出力しません。

6. カバレージ情報の操作の対象となるプログラム情報ファイルが一つもない場合は，カバレージの操作をしません。

7. カウント情報の表示の対象となるプログラム情報ファイルが一つもない場合は，カウント情報の表示のためのプログラムを実行しません。カウント情報リストファイルも出力しません。

8. 環境変数 CBLTDEXEC については，「7.2.2　プログラムからの連動実行による方法」，「7.6.2　プログラムからの連動実行による方法」を参照してください。

9. 環境変数 CBLLSLIB，環境変数 CBLLPATH，および環境変数 LD_LIBRARY_PATH については，「6.4.2　共用ライブラリ」を参照してください。

6.1.2　プログラムのコンパイル

カバレージを使用するためには，プログラムのコンパイル時に-CVInf コンパイラオプションを必ず指定します。カバレージの機能を使用するために指定する COBOL2002 のコンパイラオプションを次に示します。コンパイラオプションの詳細については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

表 6-2　カバレージの機能とコンパイラオプション

カバレージの機能	コンパイラオプション
カバレージを使用する	-CVInf

ccbl2002 コマンド以外でリンクする場合に必要な，リンク時の指定を次に示します。

表 6-3　カバレージのためのリンク時の指定

対象	機能	リンク時の指定
実行可能ファイル	実行可能ファイルをカバレージの対象とする場合	指定しない

注※

-lcbl2ktd は，-lcbl2k よりも前に指定してください。

ccbl2002 を使用しないで，cc コマンドを使用する場合の使用例を次に示します。

(使用例)

実行可能ファイルを cc コマンドで作成する

AIX(32)の場合

```
cc TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -ldl -lm
```

AIX(64)の場合

```
cc -q64 TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -ldl -lm
```

Linux(x86)の場合

```
cc TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -ldl -lm
```

Linux(x64)の場合

```
cc TEST01.o TEST02.o -o TEST -L/opt/HILNGcbl2k64/lib -lcbl2k -lcbl2kml -ldl -lm
```

共用ライブラリを ld コマンドで作成するときは，「6.4.2　共用ライブラリ」を参照してください。

注意事項

-Optimize,2 コンパイラオプションによる最適化の指定をして作成したプログラムでは，文が最適化されることがあり，プログラムの動作が保証されない場合があります。

6.1.3　テストしたいプログラムの実行

カバレージの対象となるプログラムは，一つの実行可能ファイルと複数の共用ライブラリファイルで構成されます。

注意事項

- ld コマンドの-s オプションや strip コマンドなどで実行可能ファイルや共用ライブラリのシンボル情報を削除した場合，その実行可能ファイルや共用ライブラリは，カバレージの対象になりません。

  なお，Linux では，cblcv2k コマンドの-Execute オプションに指定する実行可能ファイルは，テストデバッグの対象にしない場合でもシンボル情報が必要です。

- 共用ライブラリのロードは，COBOL プログラムの実行時の動作に従います。指定した共用ライブラリが，プログラムの実行時にロードされた場合に，カバレージの対象になります。ロードされなかった場合は，カバレージの対象になりません。

- カバレージでは，共用ライブラリがロードされた場合に次のチェックをして，指定された共用ライブラリとロードされた共用ライブラリが同じであるかを判断しています。次のチェックで，指定された共用ライブラリとは異なると判断された場合は，カバレージの対象になりません。

  チェック内容

  ・ロードされた共用ライブラリのファイルが共用ライブラリを検索するパスにある。

  共用ライブラリについては，「6.4.2　共用ライブラリ」を参照してください。

- STOP RUN 文を記述しない場合は，「COBOL プログラムの実行を終了しました」のメッセージが表示されません。

- カバレージを実行するシステム以外で作成された実行可能ファイルを対象にすると，KCCC4216T-S または KCCC4412T-E のメッセージが出力され，カバレージが終了します。

- カバレージを実行するシステム以外で作成された共用ライブラリを対象にすると，KCCC4216T-S または KCCC4412T-E，および KCCC0331T-I のメッセージが出力され，カバレージの対象になりません。カバレージは続行します。

6.1.4　テストプログラムの構成

カバレージの対象となるプログラムの構成，プログラム定義，クラス定義，および関数定義の詳細については，マニュアル「COBOL2002　使用の手引　手引編」の翻訳グループについての説明を参照してください。

6.1.5　カバレージ情報の表示と操作

カバレージ情報は，テストによって実行された手続き文の割合です。計画したテスト内容に従って，プログラムに記述した手続き文がテスト実行されたかどうかを把握でき，テスト工程の進捗度を判定できます。

実行されたテストプログラムの文はカバレージ情報としてプログラム情報ファイルに記憶されます。プログラム情報ファイルに蓄積されたカバレージ情報は，次の形式で表示できます。

- 翻訳単位ごとに実行された文の割合を，文・分岐・呼び出し文ごとに示す。
- ソース原文を表示して実行が済んだ文と実行されていない文を示す。

- 原始プログラムの修正によって変更された差分に対するカバレージ情報を示す。

さらに，蓄積したカバレージ情報をクリアする，別の環境で蓄積したカバレージ情報をマージするなどの機能があります。

6.1.6　カウント情報の表示

テスト実行させたプログラムの文の実行回数を表示します。プログラムの実行性能改善の施策を検討するときに，各文の実行回数を測定し，プログラムの性能を分析するために必要な情報を取得し，表示できます。

6.1.7　その他の注意事項

(1)　ファイルの書き込み時の制約

- ccbl2002 コマンドによって，同時にプログラムがコンパイルされた場合は，コンパイラが生成する次のファイルは，あとから生成されたファイルが有効になります。
  - プログラム情報ファイル
- 同時に複数のカバレージ情報の蓄積をした場合，プログラムの終了後に，プログラム情報ファイルに書き込みを同時にすると，エラーになるときがあります。

(2)　プログラム情報ファイルのエラーについて

- COBOL2002 以外で作成されたプログラム情報ファイルをカバレージ情報の表示，またはカバレージ情報の操作の対象に指定した場合は，KCCC4416T-E のエラーメッセージを出力し，カバレージ情報の表示，またはカバレージ情報の操作の対象にしません。
- COBOL2002 ではあるが，別のシステムで作成されたプログラム情報ファイルをカバレージ情報の表示，またはカバレージ情報の操作の対象に指定した場合は，KCCC4437T-E のエラーメッセージを出力し，カバレージ情報の表示，またはカバレージ情報の操作の対象にしません。

(3)　作業用ファイルについて

COBOL2002 では，カバレージ情報の蓄積，またはカウント情報の表示で，作業用ファイルを生成して使用しています。作業用ファイルは，次の表にあるディレクトリに，先頭に「CBLTD」，その後ろに任意の文字列が付いたファイル名で生成されます。カバレージ情報の蓄積，またはカウント情報の表示の実行中に強制終了した場合など，作業用ファイルが正しく削除されないときがあります。そのときは，OS のコマンドなどを使用して削除してください。

OS	出力ディレクトリ
AIX	/tmp ディレクトリに生成されます。ただし，環境変数 TMPDIR を指定したときは，環境変数 TMPDIR に指定したディレクトリに生成されます。

OS	出力ディレクトリ
Linux	/tmp ディレクトリに生成されます。

6.2　カバレージ情報の表示と操作

この節では，カバレージ機能の表示と操作について説明します。

6.2.1　カバレージ情報

カバレージ機能が表示するカバレージ情報は，次のとおりです。どれも，翻訳単位ごとに割合を求め，小数点第 2 位以下を切り捨てた値を表示します。

C0 メジャー

実行した文の割合を表す情報です。

C0 メジャー＝（実行が済んだ実行文の数）/（実行文の数）×100（%）

C1 メジャー

処理が分岐する個所で，実行した分岐先の数の割合を表す情報です。

C1 メジャー＝（実行が済んだ分岐先の数）/（実行できる分岐先の数）×100（%）

S1 メジャー

実行した呼び出し文の数の割合を表す情報です。

S1 メジャー＝（実行が済んだ呼び出し文の数）/（実行できる呼び出し文の数）×100（%）

なお，プログラムを修正したあと，修正個所に着目してカバレージ情報を蓄積することもできます。

差分 C0 メジャー

修正個所中の，実行した文の割合を表す情報です。

差分 C0 メジャー＝（実行が済んだ実行文の数）/（実行文の数）×100（%）

差分 C1 メジャー

修正個所中の，処理が分岐する個所で実行した分岐先の数の割合を表す情報です。

差分 C1 メジャー＝（実行が済んだ分岐先の数）/（実行できる分岐先の数）×100（%）

差分 S1 メジャー

修正個所中の，実行した呼び出し文の数の割合を表す情報です。

差分 S1 メジャー＝（実行が済んだ呼び出し文の数）/（実行できる呼び出し文の数）×100（%）

注意事項

- カバレージ情報は，ブロックごとに蓄積します。カバレージのブロックは，順次実行される無条件文の集まりで，条件文の条件句の直前までが原則として一つのブロックとなります。ブロックの先頭の文を実行したときにブロック全体が実行されたと判断します。ブロック中の文で実行時エラーが発生した場合も，そのブロックは実行されたと判断します。

ブロックを分ける一般規則と例

- それ自身でブロックとなる文または句

(1)制御を 2 方向以上に分岐させる文

・IF 文・条件指定のある文

・PERFORM 文（TIMES 指定，UNTIL 指定（VARYING 指定の UNTIL を含む）の PERFORM 文）

・EVALUATE 文

・SEARCH 文

・GO TO 文（DEPENDING ON 指定付きの文，DEPENDING ON 指定はこのブロックに含まれる）

- 直前でブロックを区切る文または句

(1)2 方向以上に分岐

・THEN

・ELSE

・条件指定

・手続き名〔THRU 手続き名〕 TIMES 指定，UNTIL 指定（VARYING 指定の UNTIL を含む）の PERFORM 文

・WHEN

・WHEN OTHER

・AT END（SEARCH 文）

・手続き名（DEPENDING ON 指定付きの GO TO 文）

(2)単独の制御の開始

・手続き部分の最初の文

・宣言部分の節名

・手続き部分の節名

・段落名

・ENTRY 文

(3)制御の合流の直前

・範囲句（END－動詞）

- 直後でブロックを区切る文または句

(1)制御を 1 方向に分岐させる文

・CALL 文（条件指定がない文）

・PERFORM 文（TIMES 指定，UNTIL 指定（VARYING 指定の UNTIL を含む）がない文）

・SORT 文（INPUT PROCEDURE または OUTPUT PROCEDURE がある文）

・MERGE 文（OUTPUT PROCEDURE がある文）

・INVOKE 文（条件指定がない文）

・INITIALIZE 文（関数一意名指定あり）

・INSPECT 文（同上）

・MOVE 文（同上）

・COMPUTE 文（同上）
・STRING 文（同上）
・UNSTRING 文（同上）

(2)制御の終わり

・STOP 文
・EXIT PROGRAM 文
・GOBACK 文
・EXIT METHOD 文
・EXIT FUNCTION 文
・EXIT 文
・GO TO 文（DEPENDING ON 指定なし）
・EXIT PERFORM 文
・EXIT USE 文
・IF 文，SERCH 文の NEXT SENTENCE
・RESUME 文

(3)制御の終わりになる可能性のある文

・OPEN 文
・CLOSE 文
・READ 文（条件指定なし）
・DELETE 文(条件指定なし)
・WRITE 文（条件指定なし）
・REWRITE 文（条件指定なし）
・START 文（条件指定なし）
・SORT 文（USING または GIVING 指定がある文）
・MERGE 文（GIVING 指定がある文）
・RAISE 文

- C0 メジャーは，次の文を除く文を対象とします。

・原始文操作

COPY 文
REPLACE 文
INCLUDE 文

・宣言文

USE 文

・覚え書きの文（ENTER 文，うち PERFORM 文の外にある EXIT PERFORM 文）

- C0 メジャーは，次の文を対象とします。

・上記以外の文（データベースアクセス機能で用いる文，データコミュニケーション機能で用いる文，EXEC SQL 文も含む）

・NEXT SENTENCE 指定（IF 文，SEARCH 文）

- C1 メジャーの対象となる文は，次のとおりです。絶対に制御が渡らない分岐も対象にします。

文	分岐方向	備考
IF 文	2	・THEN ・ELSE
EVALUATE 文	WHEN の数※1	・WHEN への分岐 ・WHEN OTHER
GO TO 文（DEPENDING ON 指定つき）	手続き名の数＋ 1	・手続き名への分岐 ・手続き名への分岐なし
PERFORM 文（TIMES 指定，UNTIL（VARYING 指定の UNTIL を含む）指定）	2	・ループする ・ループしない
SEARCH 文	WHEN の数＋ 1	・WHEN への分岐 ・AT END への分岐
条件指定のある文※2	2	・条件～への分岐 ・NOT 条件～への分岐

注※1

無条件文がない WHEN 指定の直後に WHEN 指定が続く場合，明示的または暗黙的無条件文がある WHEN 指定までを 1 つの分岐方向と数えます。

（例）

```
EVALUATE  AAA
WHEN A DISPLAY A *> 分岐方向1
WHEN B *>（無条件文なし）*> 分岐方向2
WHEN C DISPLAY C
WHEN D *>（無条件文なし）*> 分岐方向3
WHEN E *>（無条件文なし）
WHEN F DISPLAY F
WHEN G DISPLAY G *> 分岐方向4
WHEN OTHER DISPLAY O *> 分岐方向5
```

この例では，EVALUATE 文は WHEN 指定を 7 つと，WHEN OTHER 指定を持ちます。しかし，無条件文を持たない WHEN 指定は，後続の無条件文を持つ WHEN 指定と合わせて 1 つと数えるため，「WHEN B」「WHEN C」で分岐方向 1 つ，「WHEN D」「WHEN E」「WHEN F」で分岐方向 1 つとなり，WHEN の数は合計 8 つありますが，分岐方向の合計は 5 つとなります。

注※2

条件指定のある文を次に示します。

条件指定	文
AT END ,NOT ～	READ,RETURN

条件指定	文
ON SIZE ERROR,NOT ～	ADD,COMPUTE,DIVIDE,MULTIPLY,SUBTRACT
ON OVERFLOW ,NOT ～	STRING,UNSTRING
ON EXCEPTION ,NOT ～	CALL,ACCEPT
AT EOP ,NOT ～	WRITE
INVALID KEY ,NOT ～	DELETE,READ,REWRITE,START,WRITE
ON OVERFLOW	CALL

- S1 メジャーの対象となる呼び出し文は，次のとおりです。

  ・CALL 文

  ・ファクトリおよびインスタンスオブジェクトのメソッド呼び起こしで指定した INVOKE 文

  GO TO 文の直後などのように制御が渡らない文も対象にします。CALL 文の条件指定の有無，呼び出し先プログラムの実行可，不可には依存しません。

6.2.2 カバレージ情報表示の手順

カバレージ情報を表示する手順を次に示します。

1. プログラムのコンパイル

   -CVInf コンパイラオプションを指定して，原始プログラムをコンパイルします。

2. プログラムの実行

   テストによるプログラムの実行を繰り返し行い，カバレージ情報を蓄積します。また，テストデバッガでも，カバレージの蓄積ができます。

3. カバレージ情報の表示

   蓄積したカバレージ情報を表示します。カバレージ情報は，テキスト形式または CSV 形式のファイルへ出力できます。

   テキスト形式の場合，カバレージ情報の表示には次の種類があります。

表 6-4 カバレージ情報の表示（テキスト形式）

表示の種別	内容
翻訳単位の一覧表示	指定した翻訳単位の C0 メジャー，C1 メジャーを一覧表示します。
まとめ表示	指定した翻訳単位の C0 メジャー，C1 メジャー，S1 メジャーを表示します。
ソース表示※	原始プログラムを表示し，変更行，変更影響行，実行が済んだ実行文（C0 メジャーの対象），実行が済んだ分岐先（C1 メジャーの対象）に印を付けます。

表示の種別	内容
差分ソース表示※	修正によって生じた原始プログラムの差分を表示し，差分に含まれる変更行，変更が影響する行，実行が済んだ実行文（差分 C0 メジャーの対象），実行が済んだ分岐先（差分 C1 メジャーの対象）に印を付けます。
未実行ソース表示※	原始プログラムの未実行の文を表示します。
差分未実行ソース表示※	修正による原始プログラムの差分に含まれる未実行の文を表示します。
呼び出し文ソース表示※	原始プログラムの S1 カバレージの対象になる呼び出し文を表示します。

注※

これらの表示を指定すると，まとめ情報を同時に表示します。

CSV 形式の場合，カバレージ情報の表示には次の種類があります。

表 6-5　カバレージ情報の表示（CSV 形式）

表示の種別	内容
カバレージ統計情報	指定したソースファイル単位または翻訳単位の C0 メジャー，C1 メジャー，S1 メジャーを出力します。
ソースカバレージ情報	原始プログラム，原始プログラムの変更による差分，C0 メジャーおよび C1 メジャーの実行が済んだ状態を出力します。また，差分ソース，未実行ソース，差分未実行ソース，または呼び出し文ソースのどれに該当する行であるかを識別できます。

4. カバレージ情報の操作

ファイルに蓄積したカバレージ情報に対して，次の操作ができます。

表 6-6　カバレージ情報の操作

操作の種別	内 容
カバレージ情報の 0%化	実行が済んだ文をすべて未実行文に戻します。すべてのカバレージ情報が 0%になります。
差分カバレージ情報の 0%化	差分の実行が済んだ文をすべて未実行文に戻します。
差分カバレージ情報のクリア	修正による原始プログラムの差分を，差分とみなさないようにします。
カバレージ情報のマージ	別々の環境で蓄積した同一翻訳単位のカバレージ情報をマージします。

6.2.3　プログラムのコンパイル

カバレージ情報を蓄積するプログラムは，-CVInf コンパイラオプションを指定してコンパイルします。コンパイラオプションについては，「2.1.2　プログラムのコンパイル」を参照してください。

コンパイラは，プログラム情報ファイルの有無によって，次のとおりにプログラム情報ファイルを出力します。

表 6-7　コンパイラによるプログラム情報ファイルの生成

プログラム情報ファイルの有無	コンパイラが出力するプログラム情報ファイル
無し	プログラム情報ファイルを新たに作成します。
有り	原始プログラムの修正部分に対する「差分」情報をプログラム情報ファイルへ出力します。

6.2.4　カバレージ情報の蓄積

次のどちらかの方法で，プログラムを実行して，カバレージ情報を，プログラム情報ファイルへ蓄積します。

- 端末から cblcv2k コマンドを指定して，プログラムを実行する。
- 「プログラムからの連動実行」の設定を行い，プログラムを実行する。

また，次の方法で，テストデバッグによるプログラムの実行時にも，カバレージ情報が蓄積できます。

- TD コマンドのプログラムの実行（GO コマンド）で，COVERAGE オペランドを指定する。

注意事項

- マルチスレッドプログラムは，COBOL プログラムの全スレッドがカバレージ情報の蓄積の対象になります。
- 実行可能ファイル・共用ライブラリに含まれる翻訳単位とプログラム情報ファイルは，コンパイルによって同時に作成されていなければなりません（同一のコンパイル時間である必要があります）。
- プログラムの実行が終了したときに，蓄積したカバレージ情報をプログラム情報ファイルへ保存します。プログラムが実行時エラーによって終了した場合も，終了までに蓄積したカバレージ情報を保存します。
- カバレージの蓄積は，ブロックごとに蓄積します。カバレージのブロックは，順次実行される無条件文の集まりで，条件文の条件句の直前までが原則として一つのブロックとなります。ブロックの先頭の文を実行したときにブロック全体が実行されたと判断します。ブロック中の文で共通例外や実行時エラーが発生した場合も，そのブロックは実行されたと判断します。

6.2.5　カバレージ情報の表示（テキスト形式）

次の方法で，プログラム情報ファイルに蓄積されたカバレージ情報を，テキスト形式のカバレージ情報リストファイルへ出力できます。

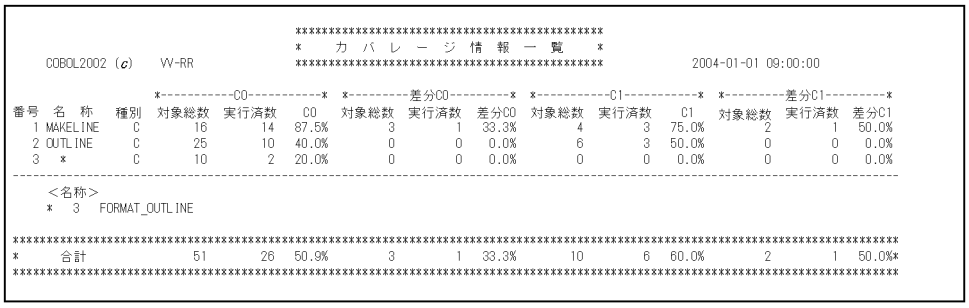
- 端末から cblcl2k コマンドを実行する。

詳細は，「7.3　カバレージ情報の表示（テキスト形式）」を参照してください。

カバレージ情報の表示による出力リストの形式は次のとおりです。表示される順序は，選択されたプログラム情報ファイルの順に依存します。スタックコンパイルされている場合は，-Main,System コンパイラオプションを指定したプログラムを，そのプログラム情報ファイル単位内の先頭に表示します。

(1)　翻訳単位の一覧表示

指定された翻訳単位の C0 メジャー，差分 C0 メジャー，C1 メジャー，差分 C1 メジャーを一覧に表示します。



```
                                        **********************************************
                                        *    カ バ レ ー ジ 情 報 一 覧    *
    COBOL2002 (c)    VV-RR              **********************************************          2004-01-01 09:00:00

                  *----------C0----------*  *---------差分C0---------*  *----------C1----------*  *---------差分C1---------*
番号  名  称   種別  対象総数  実行済数   C0   対象総数  実行済数  差分C0  対象総数  実行済数   C1   対象総数  実行済数  差分C1
   1 MAKELINE     C        16        14  87.5%         3         1   33.3%         4         3  75.0%         2         1   50.0%
   2 OUTLINE      C        25        10  40.0%         0         0    0.0%         6         3  50.0%         0         0    0.0%
   3   *          C        10         2  20.0%         0         0    0.0%         0         0   0.0%         0         0    0.0%
-------------------------------------------------------------------------------------------------------------------------------
     <名称>
     *   3   FORMAT_OUTLINE

*******************************************************************************************************************************
*      合計               51        26  50.9%         3         1   33.3%        10         6  60.0%         2         1   50.0%*
*******************************************************************************************************************************
```

表 6-8　プログラムの一覧表示

項目	内容			
翻訳単位	名称		翻訳単位の名称 9 文字以上の名称は，アスタリスク（*）を表示し，＜名称＞のあとに表示する。	
	種別		翻訳単位の種別 P：プログラム C：クラス T：利用者定義関数	
	C0	対象総数	翻訳単位	実行文の数
		実行済数		実行が済んだ実行文の数
		C0		C0 メジャー※
	差分 C0	対象総数	差分	実行文の数
		実行済数		実行が済んだ実行文の数

項目	内容			
		差分 C0		差分 C0 メジャー※
	C1	対象総数	翻訳単位	実行できる分岐先の数
		実行済数		実行が済んだ分岐先の数
		C1		C1 メジャー※
	差分 C1	対象総数	差分	実行できる分岐先の数
		実行済数		実行が済んだ分岐先の数
		差分 C1		差分 C1 メジャー※
合計	C0	対象総数	一覧の全翻訳単位	実行文の数の合計
		実行済数		実行が済んだ実行文の数の合計
		C0		C0 メジャー※
	差分 C0	対象総数	一覧の全翻訳単位の差分	実行文の数の合計
		実行済数		実行が済んだ実行文の数の合計
		差分 C0		差分 C0 メジャー※
	C1	対象総数	一覧の全翻訳単位	実行できる分岐先の数の合計
		実行済数		実行が済んだ分岐先の数の合計
		C1		C1 メジャー※
	差分 C1	対象総数	一覧の全翻訳単位の差分	実行できる分岐先の数の合計
		実行済数		実行が済んだ分岐先の数の合計
		差分 C1		差分 C1 メジャー※

注※　小数点第 2 位以下を切り捨てる。

(2)　まとめ情報の表示

指定した翻訳単位の C0 メジャー，差分 C0 メジャー，C1 メジャー，差分 C1 メジャー，S1 メジャー，差分 S1 メジャーを表示します。

```
                              ******************************
                              *       カバレージ情報       *
    COBOL2002 (c)  VV-RR      ******************************       2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名        : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                変更回数    :       1
  テスト日時      : 2004-01-01 09:15:00                              テスト回数  :       2

*****************************************************************************************************
*                    <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*      対象総数        16            3          4            2          5            0       *
*      実行済数        14            1          3            1          5            0       *
*      未実行数         2            2          1            1          0            0       *
*      カバレージ率  87.5%        33.3%      75.0%        50.0%     100.0%        0.0%       *
*****************************************************************************************************
```

表 6-9　まとめ情報の表示の説明

項目		内容	
翻訳単位名の種別		プログラム名・クラス名・利用者定義関数名	
翻訳単位名		翻訳単位の名称	
コンパイル日時		翻訳単位を含む原始プログラムをコンパイルした日時	
テスト日時		翻訳単位を含む原始プログラムのカバレージ情報を最後に蓄積した日時	
変更回数		翻訳単位を含む原始プログラムを変更した回数	
テスト回数		翻訳単位を含む原始プログラムを実行した回数 32,766 を超えた場合は*****	
＜ C0 ＞	対象総数	翻訳単位	実行文の数
	実行済数		実行が済んだ実行文の数
	未実行数		実行していない文の数
	カバレージ率		C0 メジャー※
＜差分 C0 ＞	対象総数	差分	実行文の数
	実行済数		実行が済んだ実行文の数
	未実行数		実行していない実行文の数
	カバレージ率		差分 C0 メジャー※
＜ C1 ＞	対象総数	翻訳単位	実行できる分岐の数
	実行済数		実行が済んだ分岐の数
	未実行数		実行していない分岐の数
	カバレージ率		C1 メジャー※
＜差分 C1 ＞	対象総数	差分	実行できる分岐の数
	実行済数		実行が済んだ分岐の数

項目		内容	
	未実行数		実行していない分岐の数
	カバレージ率		差分 C1 メジャー※
＜S1＞	対象総数	翻訳単位	実行できる呼び出し文の数
	実行済数		実行が済んだ呼び出し文の数
	未実行数		実行していない呼び出し文の数
	カバレージ率		S1 メジャー※
＜差分 S1＞	対象総数	差分	実行できる呼び出し文の数
	実行済数		実行が済んだ呼び出し文の数
	未実行数		実行していない呼び出し文の数
	カバレージ率		差分 S1 メジャー※

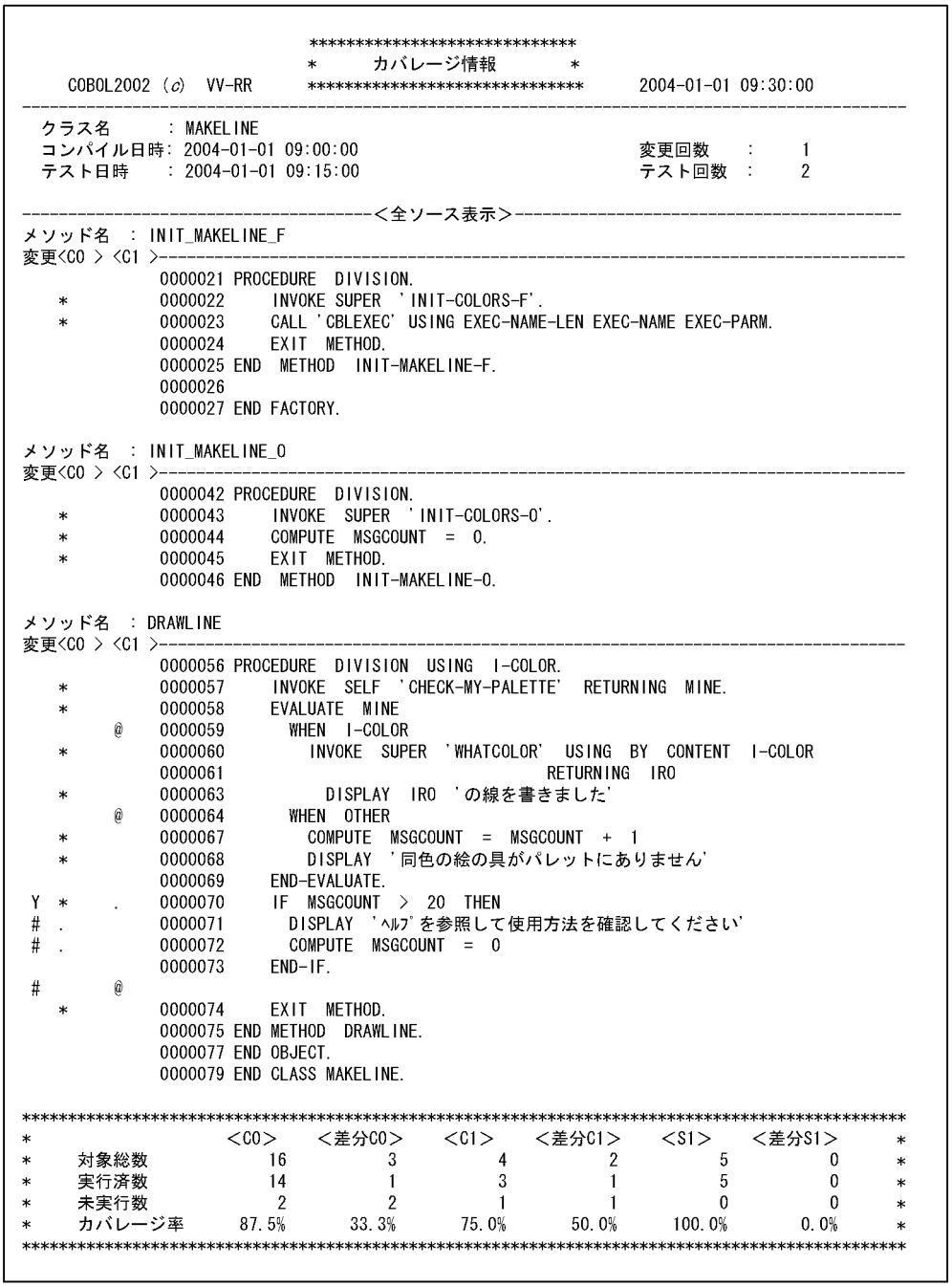
注※　小数点第 2 位以下を切り捨てる。

注意事項

C0 メジャー，C1 メジャー，差分 C0 メジャー，差分 C1 メジャー，S1 メジャーおよび差分 S1 メジャーのそれぞれの対象総数が 99,999,999 を超えた場合，表示は「********」となります。

(3)　全ソース表示

原始プログラムを表示し，原始プログラムの変更による差分と C0・C1 の実行済みの状態を表示します。



```
                              ******************************
                              *      カバレージ情報        *
     COBOL2002 (c)  VV-RR     ******************************      2004-01-01 09:30:00
-------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                変更回数    :     1
  テスト日時     : 2004-01-01 09:15:00                                テスト回数  :     2

---------------------------------------<全ソース表示>-----------------------------------------
メソッド名  : INIT_MAKELINE_F
変更<C0 > <C1 >-----------------------------------------------------------------------------
                0000021 PROCEDURE  DIVISION.
    *           0000022     INVOKE SUPER  'INIT-COLORS-F'.
    *           0000023     CALL 'CBLEXEC' USING EXEC-NAME-LEN EXEC-NAME EXEC-PARM.
                0000024     EXIT  METHOD.
                0000025 END  METHOD  INIT-MAKELINE-F.
                0000026
                0000027 END FACTORY.

メソッド名  : INIT_MAKELINE_O
変更<C0 > <C1 >-----------------------------------------------------------------------------
                0000042 PROCEDURE  DIVISION.
    *           0000043     INVOKE  SUPER  'INIT-COLORS-O'.
    *           0000044     COMPUTE  MSGCOUNT  =  0.
    *           0000045     EXIT  METHOD.
                0000046 END  METHOD  INIT-MAKELINE-O.

メソッド名  : DRAWLINE
変更<C0 > <C1 >-----------------------------------------------------------------------------
                0000056 PROCEDURE  DIVISION  USING  I-COLOR.
    *           0000057     INVOKE  SELF  'CHECK-MY-PALETTE'  RETURNING  MINE.
    *           0000058     EVALUATE  MINE
          @     0000059       WHEN  I-COLOR
    *           0000060         INVOKE  SUPER  'WHATCOLOR'  USING  BY  CONTENT  I-COLOR
                0000061                                  RETURNING  IRO
    *           0000063           DISPLAY  IRO  'の線を書きました'
          @     0000064       WHEN  OTHER
    *           0000067         COMPUTE  MSGCOUNT  =  MSGCOUNT  +  1
    *           0000068         DISPLAY  '同色の絵の具がパレットにありません'
                0000069     END-EVALUATE.
 Y  *     .     0000070     IF  MSGCOUNT  >  20  THEN
 #  .           0000071       DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .           0000072       COMPUTE  MSGCOUNT  =  0
                0000073     END-IF.
 #        @
    *           0000074     EXIT  METHOD.
                0000075 END METHOD  DRAWLINE.
                0000077 END OBJECT.
                0000079 END CLASS MAKELINE.

*******************************************************************************************
*                    <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*     対象総数         16            3          4           2          5           0       *
*     実行済数         14            1          3           1          5           0       *
*     未実行数          2            2          1           1          0           0       *
*     カバレージ率   87.5%       33.3%      75.0%       50.0%      100.0%       0.0%       *
*******************************************************************************************
```

表 6-10　全ソース表示の説明

項目	内容
翻訳単位名の種別	プログラム名・クラス名・利用者定義関数名
翻訳単位名	翻訳単位の名称
コンパイル日時	翻訳単位を含む原始プログラムをコンパイルした日時
テスト日時	翻訳単位を含む原始プログラムのカバレージ情報を最後に蓄積した日時
変更回数	翻訳単位を含む原始プログラムを変更した回数
テスト回数	翻訳単位を含む原始プログラムを実行した回数

項目	内容
ソース要素名の種別	プログラム名・メソッド名・利用者定義関数名
ソース要素名	ソース要素の名称
変更	Y：原始プログラムの変更された行
	#：原始プログラムの変更によって影響がある行
<C0>	*：実行が済んだ文
	.：実行していない文
<C1>	@：実行が済んだ分岐
	.：実行していない分岐
行番号	コンパイラが振り直した行番号
ソーステキスト	原始プログラムの行

まとめ情報の項目は，「(2)　まとめ情報の表示」を参照してください。

注意事項

- C1 に該当する明示的な文がないときは，空白行を表示して C1 の情報を表示します。

（例 1）ELSE がない IF 文

ELSE がない IF 文は，IF 文の下に空白行を示します。IF 文がネストするときは，最外の IF 文の下に C1 に該当する空白行を示します。

(a)は 111200 行の IF 文の「省略された ELSE」の C1 を示します。

(b)は 111000 行の IF 文の「省略された ELSE」の C1 を示します。

```
変更<C0 > <C1 >-----------------------------------------------------------------
    .          0111000     IF FLAG-1 = SPACE
          .    0111100       THEN
    .          0111200         IF FLAG-2 = SPACE
          .    0111300           THEN
    .          0111400             PERFORM 処理１
    .          0111500             MOVE '10' TO 番号 OF 集計レコード
    .          0111600             PERFORM 処理２
    .          0111700             MOVE 1 TO FLAG-3
               0111800         END-IF
               0111900     END-IF.
(a)        .
(b)        .
```

（例 2）うち PERFORM 文

UNTIL 指定のある「うち PERFORM 文」には「ループする」と「ループしない」の 2 方向の分岐があり，両方の C1 を表示します。

(a)は 0112000 行の PERFORM 文の「ループする」ときの分岐先の C1 の行です。

(b)は 0112000 行の PERFORM 文の「ループしない」ときの分岐先の C1 を空白行で示します。

```
変更<C0 > <C1 >-------------------------------------------------------------------------
    .          0112000     PERFORM WITH TEST BEFORE
               0112100       UNTIL リターンコード OF テーブルパラメタ NOT = ZERO
(a) .    .     0112200         MOVE  保存キー TO テーブルキー
    .          0112300         PERFORM 処理３
               0112400     END-PERFORM.
(b)        .
```

（例 3）そと PERFORM 文

UNTIL 指定のある「そと PERFORM 文」は「ループしない」ときの C1 を PERFORM 文の下に表示します（「ループする」の分岐先の C1 は PERFORM 文で実行される文に表示します）。

(a)は 0113000 行の PERFORM 文の「ループしない」ときの分岐先の C1 を示します。

```
変更<C0 > <C1 >-------------------------------------------------------------
    .    .     0113000     PERFORM 処理４ WITH TEST
(a)        .
                0113100     AFTER UNTIL (情報フラグ = C-ON) OR
                0113200                 (終了フラグ = C-ON)
```

(4)　差分ソース表示

原始プログラムの修正によって生じた差分の実行文に対して，変更行，変更影響行，C0 実行済み，C1 実行済みを明示します。

```
                              ********************************
                              *      カバレージ情報          *
    COBOL2002 (c)  VV-RR      ********************************      2004-01-01 09:30:00
-------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :     1
  テスト日時     : 2004-01-01 09:15:00                                 テスト回数  :     2

-----------------------------------------<差分表示>-----------------------------------------
メソッド名    : DRAWLINE
変更<C0 > <C1 >-------------------------------------------------------------------------------
 Y  *     .    0000070     IF  MSGCOUNT  >  20  THEN
 #  .          0000071       DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .          0000072       COMPUTE  MSGCOUNT  =  0
               0000073     END-IF.
 #       @

*******************************************************************************************
*                    <C0>    <差分C0>    <C1>    <差分C1>    <S1>    <差分S1>       *
*     対象総数         16           3          4          2          5          0      *
*     実行済数         14           1          3          1          5          0      *
*     未実行数          2           2          1          1          0          0      *
*     カバレージ率   87.5%       33.3%      75.0%      50.0%     100.0%       0.0%      *
*******************************************************************************************
```

各項目の内容については，「(3)　全ソース表示」を参照してください。

(5)　未実行表示

原始プログラムの実行していない文を表示します。

```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************        2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :       1
  テスト日時     : 2004-01-01 09:15:00                                  テスト回数  :       2

-------------------------------------<未実行表示>-----------------------------------------
メソッド名    : DRAWLINE
変更<C0 > <C1 >-------------------------------------------------------------------------------
 Y  *     .     0000070      IF  MSGCOUNT  >  20  THEN
 #  .           0000071        DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .           0000072        COMPUTE  MSGCOUNT  =  0

*************************************************************************************************
*                     <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*     対象総数          16            3          4            2          5            0       *
*     実行済数          14            1          3            1          5            0       *
*     未実行数           2            2          1            1          0            0       *
*     カバレージ率     87.5%        33.3%       75.0%        50.0%      100.0%       0.0%      *
*************************************************************************************************
```

各項目の内容については，「(3)　全ソース表示」を参照してください。

(6)　差分未実行ソース表示

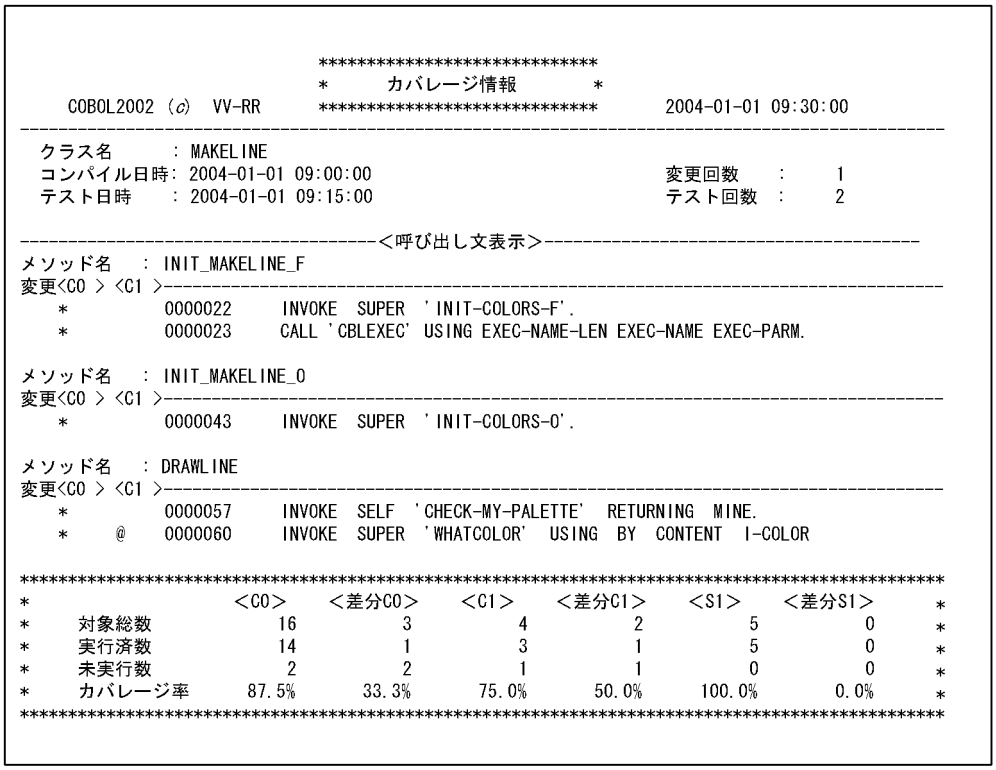
原始プログラムの差分未実行ソースを表示します。

```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************      2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :       1
  テスト日時     : 2004-01-01 09:15:00                                  テスト回数  :       2

-------------------------------------<差分未実行表示>-------------------------------------
メソッド名    : DRAWLINE
変更<C0 > <C1 >-------------------------------------------------------------------------------
 Y  *     .     0000070      IF  MSGCOUNT  >  20  THEN
 #  .           0000071        DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .           0000072        COMPUTE  MSGCOUNT  =  0

*************************************************************************************************
*                     <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*     対象総数          16            3          4            2          5            0       *
*     実行済数          14            1          3            1          5            0       *
*     未実行数           2            2          1            1          0            0       *
*     カバレージ率     87.5%        33.3%       75.0%        50.0%      100.0%       0.0%      *
*************************************************************************************************
```

各項目の内容については，「(3)　全ソース表示」を参照してください。

(7)　呼び出し文ソース表示

原始プログラムの S1 カバレージ対象の呼び出し文を表示します。



```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************       2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                変更回数    :      1
  テスト日時     : 2004-01-01 09:15:00                                テスト回数  :      2

-----------------------------------<呼び出し文表示>-----------------------------------------
メソッド名    : INIT_MAKELINE_F
変更<C0 > <C1 >-------------------------------------------------------------------------------
    *           0000022     INVOKE  SUPER  'INIT-COLORS-F'.
    *           0000023     CALL 'CBLEXEC' USING EXEC-NAME-LEN EXEC-NAME EXEC-PARM.

メソッド名    : INIT_MAKELINE_O
変更<C0 > <C1 >-------------------------------------------------------------------------------
    *           0000043     INVOKE  SUPER  'INIT-COLORS-O'.

メソッド名    : DRAWLINE
変更<C0 > <C1 >-------------------------------------------------------------------------------
    *           0000057     INVOKE  SELF  'CHECK-MY-PALETTE'  RETURNING  MINE.
    *     @     0000060     INVOKE  SUPER  'WHATCOLOR'  USING  BY  CONTENT  I-COLOR

*************************************************************************************************
*                      <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>      *
*     対象総数           16            3          4           2          5           0      *
*     実行済数           14            1          3           1          5           0      *
*     未実行数            2            2          1           1          0           0      *
*     カバレージ率     87.5%        33.3%       75.0%       50.0%      100.0%        0.0%      *
*************************************************************************************************
```

各項目の内容については，「(3)　全ソース表示」を参照してください。

6.2.6　カバレージ情報の表示（CSV 形式）

次の方法で，プログラム情報ファイルに蓄積されたカバレージ情報を，カバレージ統計情報 CSV ファイル，およびソースカバレージ情報 CSV ファイルへ出力できます。

- 端末から cblcc2k コマンドを実行する。

詳細は，「7.4　カバレージ情報の表示（CSV 形式）」を参照してください。

カバレージ情報の表示による出力ファイル（CSV）の形式は次のとおりです。

注意事項

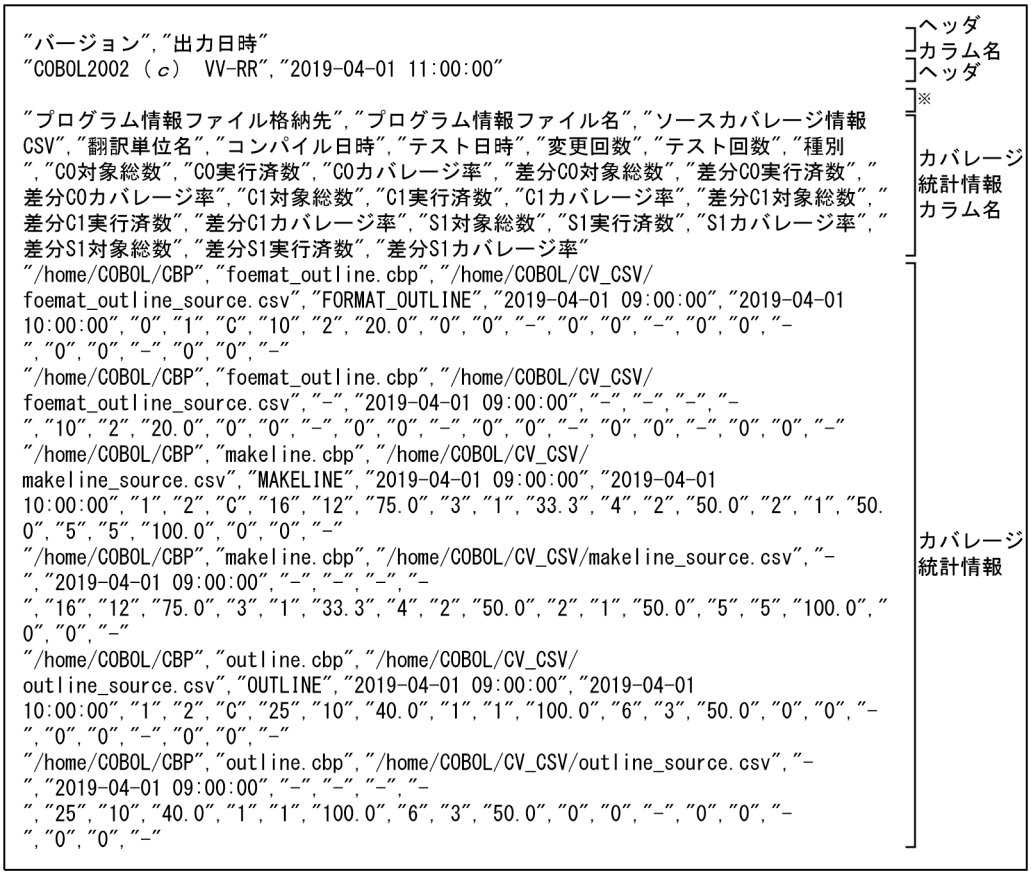
項目および出力内容は，ダブルコーテーション（"）で囲んで CSV ファイルに出力します。出力内容にダブルコーテーション（"）を含む場合は 2 個続けて（""）出力します。出力内容が何もない場合は，ダブルコーテーション（"）も出力しません。

（例）

出力内容が「XXX"YYY」の場合，CSV ファイルには「"XXX""YYY"」と出力します。

(1)　カバレージ統計情報の出力

指定されたソースファイル単位または翻訳単位の C0 メジャー，C1 メジャー，S1 メジャーをカバレージ統計情報 CSV ファイルに出力します。



注※

3 行目は空白行が出力されます。

表 6-11　カバレージ統計情報 CSV ファイルの出力内容

項目		出力内容
ヘッダ	バージョン	COBOL2002 のバージョン情報 「COBOL2002 (*c*) VV-RR」の形式で出力する。 COBOL2002：COBOL2002 であることの記述 (*c*)：COBOL2002 の識別記号 COBOL2002 の識別記号については「付録 E.2　このマニュアルでの表記」を参照 VV-RR：COBOL2002 のバージョン番号
	出力日時	カバレージ統計情報 CSV ファイルに出力した日時
カバレージ統計情報	プログラム情報ファイル格納先	プログラム情報ファイルが格納されている場所の絶対パス
	プログラム情報ファイル名	プログラム情報ファイル名
	ソースカバレージ情報 CSV	同時に出力するソースカバレージ情報 CSV ファイルのうち，該当する行のプログラム情報ファイルに対応するソースカバレージ情報 CSV ファイルの絶対パス ソースカバレージ情報 CSV ファイルを出力しない場合は-（ハイフン）

項目		出力内容
	翻訳単位名	ソースファイル単位の場合 　-（ハイフン） 翻訳単位の場合 　翻訳単位の名称
	コンパイル日時	翻訳単位を含む原始プログラムをコンパイルした日時
	テスト日時	ソースファイル単位の場合 　-（ハイフン） 翻訳単位の場合 　翻訳単位を含む原始プログラムのカバレージ情報を最後に蓄積した日時 　カバレージ情報を蓄積していない場合は-（ハイフン）
	変更回数	ソースファイル単位の場合 　-（ハイフン） 翻訳単位の場合 　翻訳単位を含む原始プログラムを変更した回数 　32,767 を超えた場合は*
	テスト回数	ソースファイル単位の場合 　-（ハイフン） 翻訳単位の場合 　翻訳単位を含む原始プログラムを実行した回数 　32,766 を超えた場合は*
	種別	ソースファイル単位の場合 　-（ハイフン） 翻訳単位の場合 　翻訳単位の種別 　P：プログラム 　C：クラス 　T：利用者定義関数
	C0 対象総数	実行文の数
	C0 実行済数	実行が済んだ実行文の数
	C0 カバレージ率	C0 メジャー※ C0 対象総数が 0 の場合は-（ハイフン）
	差分 C0 対象総数	差分に含まれる実行文の数
	差分 C0 実行済数	差分に含まれる実行が済んだ実行文の数
	差分 C0 カバレージ率	差分 C0 メジャー※ 差分 C0 対象総数が 0 の場合は-（ハイフン）
	C1 対象総数	実行できる分岐の数

項目		出力内容
	C1 実行済数	実行が済んだ分岐の数
	C1 カバレージ率	C1 メジャー※ C1 対象総数が 0 の場合は-（ハイフン）
	差分 C1 対象総数	差分に含まれる実行できる分岐の数
	差分 C1 実行済数	差分に含まれる実行が済んだ分岐の数
	差分 C1 カバレージ率	差分 C1 メジャー※ 差分 C1 対象総数が 0 の場合は-（ハイフン）
	S1 対象総数	実行できる呼び出し文の数
	S1 実行済数	実行が済んだ呼び出し文の数
	S1 カバレージ率	S1 メジャー※ S1 対象総数が 0 の場合は-（ハイフン）
	差分 S1 対象総数	差分に含まれる実行できる呼び出し文の数
	差分 S1 実行済数	差分に含まれる実行が済んだ呼び出し文の数
	差分 S1 カバレージ率	差分 S1 メジャー※ 差分 S1 対象総数が 0 の場合は-（ハイフン）

注※

小数点第 2 位以下を切り捨てます。

注意事項

- C0 メジャー，C1 メジャー，差分 C0 メジャー，差分 C1 メジャー，S1 メジャーおよび差分 S1 メジャーのそれぞれの対象総数，または実行済数が 99,999,999 を超えた場合，出力内容は「*」となります。
- プログラム情報ファイルを指定した場合，カバレージ統計情報 CSV ファイルには，指定されたプログラム情報ファイルの順に出力します。プログラム情報ファイルに複数の翻訳単位があるときは，COBOL ソースファイルに記述された順に出力します。
- プログラム情報ファイル格納ディレクトリパスを指定した場合，カバレージ統計情報 CSV ファイルには，次の順序で出力します。

  ・同一ディレクトリ内のプログラム情報ファイルおよびサブディレクトリは，ファイル名およびディレクトリ名の文字コードの昇順で出力します。

  ・同一ディレクトリにプログラム情報ファイルとサブディレクトリが混在する場合，ディレクトリ直下にあるプログラム情報ファイルを先に出力します。

  ・サブディレクトリ下にあるプログラム情報ファイルは，上位ディレクトリから順に出力します。

  ・プログラム情報ファイルに複数の翻訳単位がある場合は，COBOL ソースファイルに記述された順に出力します。

- 翻訳単位のカバレージ統計情報とソースファイル単位のカバレージ統計情報を同時に出力する場合，同一ソースファイルに対するカバレージ統計情報は，翻訳単位のカバレージ統計情報のあとにソースファイル単位のカバレージ統計情報を出力します。

(2) ソースカバレージ情報の出力

原始プログラム，原始プログラムの変更による差分，C0 と C1 の実行済みの状態をソースカバレージ情報 CSV ファイルに出力します。また，差分ソース，未実行ソース，差分未実行ソースまたは呼び出し文ソースのどれかに該当する行を識別します。

```
"バージョン","出力日時","コンパイル日時","プログラム情報ファイル格納先","プログラ
ム情報ファイル名"
"COBOL2002 (c)  VV-RR","2019-04-01 11:00:00","2019-04-01 09:00:00","/home/COBOL/
CBP","makeline.cbp"

"翻訳単位名","ソース要素種別","ソース要素名","未実行ソース","差分ソース","差分未
実行ソース","呼び出し文ソース","コメント行","変更","C0","C1","行番号","ソーステキ
スト"
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,"0000021"," PROCEDURE  DIVISION."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,"ACT",,,"*",,"0000022","     INVOKE
SUPER  'INIT-COLORS-F'."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,"ACT",,,"*",,"0000023","     CALL
'CBLEXEC' USING EXEC-NAME-LEN EXEC-NAME EXEC-PARM."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,"*",,"0000024"," EXIT  METHOD."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,"0000025"," END  METHOD  INIT-
MAKELINE-F."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,"0000026",
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,"0000027"," END FACTORY."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,,,"0000042"," PROCEDURE  DIVISION."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,"ACT",,,"*",,"0000043","     INVOKE
SUPER  'INIT-COLORS-O'."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,"*",,"0000044","     COMPUTE
MSGCOUNT  =  0."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,"*",,"0000045","     EXIT  METHOD."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,,,"0000046"," END  METHOD  INIT-
MAKELINE-O."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,"0000056"," PROCEDURE  DIVISION  USING  I-
COLOR."
"MAKELINE","METHOD","DRAWLINE",,,,"ACT",,,"*",,"0000057","     INVOKE  SELF
'CHECK-MY-PALETTE'  RETURNING  MINE."
"MAKELINE","METHOD","DRAWLINE",,,,,,"*",,"0000058","     EVALUATE  MINE"
"MAKELINE","METHOD","DRAWLINE",,,,,,,"@","0000059","       WHEN  I-COLOR"
"MAKELINE","METHOD","DRAWLINE",,,,"ACT",,,"*","@","0000060","         INVOKE
SUPER  'WHATCOLOR'  USING  BY  CONTENT  I-COLOR"
"MAKELINE","METHOD","DRAWLINE",,,,,,,,"0000061","
RETURNING  IRO"
"MAKELINE","METHOD","DRAWLINE",,,,,,"*",,"0000063","         DISPLAY  IRO  'の
線を書きました'"
"MAKELINE","METHOD","DRAWLINE","UNEX",,,,,,".","0000064","       WHEN  OTHER"
"MAKELINE","METHOD","DRAWLINE","UNEX",,,,,".",,"0000067","         COMPUTE
MSGCOUNT  =  MSGCOUNT  +  1"
"MAKELINE","METHOD","DRAWLINE","UNEX",,,,,".",,"0000068","         DISPLAY  '同
色の絵の具がパレットにありません'"
"MAKELINE","METHOD","DRAWLINE",,,,,,,,"0000069","     END-EVALUATE."
"MAKELINE","METHOD","DRAWLINE","UNEX","DIFF","UNEXDIFF",,,"Y","*",".","0000070","
IF  MSGCOUNT  >  20  THEN"
"MAKELINE","METHOD","DRAWLINE","UNEX","DIFF","UNEXDIFF",,,"#",".",,"0000071","
DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'"
"MAKELINE","METHOD","DRAWLINE","UNEX","DIFF","UNEXDIFF",,,"#",".",,"0000072","
COMPUTE  MSGCOUNT  =  0"
"MAKELINE","METHOD","DRAWLINE",,"DIFF",,,,,,"0000073","     END-IF."
"MAKELINE","METHOD","DRAWLINE",,"DIFF",,,,"#",,"@",,
"MAKELINE","METHOD","DRAWLINE",,,,,,"*",,"0000074","     EXIT  METHOD."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,"0000075"," END METHOD  DRAWLINE."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,"0000077"," END OBJECT."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,"0000079"," END CLASS MAKELINE."
```

（1～2行目：ヘッダカラム名／3～4行目：ヘッダ／5行目：※／6～8行目：ソースカバレージ情報カラム名／以降：ソースカバレージ情報）

注※

3行目は空白行が出力されます。

表 6-12　ソースカバレージ情報 CSV ファイルの各項目の出力内容

項目		出力内容
ヘッダ	バージョン	COBOL2002 のバージョン情報 「COBOL2002 (c) VV-RR」の形式で出力する。

項目		出力内容
		COBOL2002：COBOL2002 であることの記述 (*c*)：COBOL2002 の識別記号 COBOL2002 の識別記号については「付録 E.2　このマニュアルでの表記」を参照 VV-RR：COBOL2002 のバージョン番号
	出力日時	ソースカバレージ情報 CSV ファイルに出力した日時
	コンパイル日時	翻訳単位を含む原始プログラムをコンパイルした日時
	プログラム情報ファイル格納先	プログラム情報ファイルが格納されている場所の絶対パス
	プログラム情報ファイル名	プログラム情報ファイル名
ソースカバレージ情報	翻訳単位名	翻訳単位の名称
	ソース要素種別	ソース要素の種別 PPROG：いちばん外側のプログラム IPROG：入れ子プログラム METHOD：ファクトリメソッドまたはオブジェクトメソッド FUNC：利用者定義関数名
	ソース要素名	ソース要素の名称
	未実行ソース	UNEX：原始プログラムの実行していない文の行 該当しない場合は何も出力されない
	差分ソース	DIFF：原始プログラムの修正によって生じた差分の行 該当しない場合は何も出力されない
	差分未実行ソース	UNEXDIFF：原始プログラムの修正によって生じた差分に含まれる，実行していない文の行 該当しない場合は何も出力されない
	呼び出し文ソース	ACT：原始プログラムの S1 カバレージ対象の呼び出し文の行 該当しない場合は何も出力されない
	コメント行	COMMENT：コメント行（注記行） 該当しない場合は何も出力されない
	変更	Y：原始プログラムの変更された行
		#：原始プログラムの変更によって影響がある行
	C0	*：実行が済んだ文
		.：実行していない文
	C1	@：実行が済んだ分岐
		.：実行していない分岐
	行番号	コンパイラが振り直した行番号
	ソーステキスト	原始プログラムの行

注意事項

- 原始プログラムの 1 行に複数の実行可能文が含まれる場合，C0，C1 には，実行済みかどうかを示す記号を実行可能文の個数分出力します。
- 原始プログラムの末尾の半角空白文字は，ソーステキストとして出力しません。
- IF 文の ELSE 側を省略した場合など，暗黙的に生成される C1 メジャーの対象となるときは，行番号は空になります。行番号が空になる行の条件は，「6.2.5　カバレージ情報の表示（テキスト形式）の「(3)　全ソース表示」を参照してください。

6.2.7　カバレージ情報の操作

次の方法で，プログラム情報ファイルのカバレージ情報を操作できます。

- 端末から cblca2k コマンドを実行する。

カバレージ情報の操作には次の機能があります。

詳細は，「7.5　カバレージ情報の操作」を参照してください。

(1)　カバレージ情報の 0%化

実行が済んだ文を未実行文に戻します。再度のテストが必要な場合，蓄積したカバレージ情報を 0%に戻すときに使用できます。

次のカバレージ情報が 0%になります。

- C0 メジャー
- 差分 C0 メジャー
- C1 メジャー
- 差分 C1 メジャー
- S1 メジャー
- 差分 S1 メジャー

(2)　差分カバレージ情報の 0%化

差分に含まれる実行が済んだ文をすべて未実行文に戻します。次のカバレージ情報が 0%になります。

- 差分 C0 メジャー
- 差分 C1 メジャー
- 差分 S1 メジャー

(3)　差分カバレージ情報のクリア

差分の対象となる文を差分とみなさないようにします。次の情報が 0 になります。

- 差分の実行文の数
- 差分の実行できる分岐の数
- 差分の実行できる呼び出し文の数

(4)　カバレージ情報のマージ

複数のプログラム情報ファイルのカバレージ情報を，一つのプログラム情報ファイルへマージします。実行環境をコピーして複数のテスト環境を作成して別々に蓄積したカバレージ情報を，一つのプログラム情報ファイルにマージするときに使用できます。マージするプログラム情報ファイルのコンパイル時間は，すべて同じである必要があります。

注意事項

カバレージ情報の操作は，プログラム情報ファイルの単位で行います。プログラム情報ファイルに複数の翻訳単位がある場合は，すべてを対象にします。

6.3　カウント情報の表示

この節では，カウント情報の表示について説明します。

6.3.1　カウント情報

プログラムを実行させたときの各文の実行回数です。

実行回数は，ブロックごとにカウントします。カバレージのブロックは，順次実行される無条件文の集まりで，条件文の条件句の直前までが原則として一つのブロックとなります。ブロックの先頭の文を実行したときにブロック全体が実行されたと判断します。ブロック中の文で実行時エラーが発生した場合も，そのブロックは実行されたとします。

6.3.2　カウント情報表示の手順

カウント情報を表示する手順を次に示します。

1. プログラムのコンパイル

   カウント情報を表示するプログラムは，-CVInf コンパイラオプションを指定してコンパイルします。コンパイラオプションについては，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

2. プログラムのコンパイル

   カウント情報を表示するプログラムは，-CVInf コンパイラオプションを指定してコンパイルします。コンパイラオプションについては，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

3. プログラムの実行

   次の方法でプログラムを実行することによって，カウント情報をテキスト形式のカウント情報リストファイルへ出力します。

   - 端末から cblcn2k コマンドを実行する。
   - 「プログラムからの連動実行」の設定を行い，プログラムを実行する。

注意事項

- マルチスレッドプログラムは，COBOL プログラムの全スレッドがカウント情報表示の対象になります。
- 実行可能ファイル・共用ライブラリに含まれる翻訳単位とプログラム情報ファイルは，コンパイルによって同時に作成されていなければなりません（同一のコンパイル時間である必要があります）。
- プログラムの実行が終了したとき，カウントした実行回数を表示します。プログラムが実行時エラーによって終了した場合も，終了までにカウントした実行回数を表示します。

6.3.3　カウント情報の表示

カウント情報の表示によって，ファイルへ出力される出力リストを説明します。

表示される順序は，実行可能ファイル，共用ライブラリファイル（共用ライブラリファイル名でソート）の順です。実行可能ファイル，共用ライブラリファイル内の翻訳単位はソートされて表示されます。ただし，-Main コンパイラオプションで指定した翻訳単位は，最初に表示されます。翻訳単位内の表示順は，プログラムに書かれている順です。

AIX の場合

　共用ライブラリファイルがアーカイブファイルの場合，メンバ名でソートされます。

詳細は，「7.6　カウント情報の表示」を参照してください。

```
                          ******************************
                          *       カウント情報         *
    COBOL2002 (c)  VV-RR  ******************************          2004-01-01 09:30:00
------------------------------------------------------------------------------------------
--
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00
  実行日時       : 2004-01-01 09:15:00

------------------------------------------------------------------------------------------
--
メソッド名    : INIT_MAKELINE_F
  実行回数      ----------------------------------------------------------------------------
--
               0000021 PROCEDURE  DIVISION.
        1      0000022     INVOKE  SUPER  'INIT-COLORS-F'.
        1      0000023     EXIT  METHOD.
               0000024 END  METHOD  INIT-MAKELINE-F.
               0000025
               0000026 END  FACTORY.

メソッド名    : INIT_MAKELINE_O
  実行回数      ----------------------------------------------------------------------------
--
               0000042 PROCEDURE  DIVISION.
        2      0000043     INVOKE  SUPER  'INIT-COLORS-O'.
        2      0000044     COMPUTE  MSGCOUNT  =  0.
               0000045     EXIT  METHOD.
               0000046 END  METHOD  INIT-MAKELINE-O.

メソッド名    : DRAWLINE
  実行回数      ----------------------------------------------------------------------------
--
               0000056 PROCEDURE  DIVISION  USING  I-COLOR.
        6      0000057     INVOKE  SELF  'CHECK-MY-PALETTE'  RETURNING  MINE.
        6      0000058     EVALUATE  MINE
               0000059       WHEN  I-COLOR
        4      0000060         INVOKE  SUPER  'WHATCOLOR'  USING  BY  CONTENT  I-COLOR
               0000061                                    RETURNING IRO
        4      0000063         DISPLAY  IRO  'の線を書きました'
               0000064         WHEN  OTHER
        2      0000067           COMPUTE  MSGCOUNT  =  MSGCOUNT  +  1
               0000068           DISPLAY  '同色の絵の具がパレットにありません'
               0000069     END-EVALUATE.
        6      0000070     IF  MSGCOUNT  >  20  THEN
        0      0000071       DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
               0000072       COMPUTE  MSGCOUNT  =  0
               0000073     END-IF.
        6      0000074     EXIT  METHOD.
               0000075 END  METHOD  DRAWLINE.
               0000077 END  OBJECT.
               0000079 END  CLASS  MAKELINE.
```

表 6-13　カウント情報の表示

項目	備考
翻訳単位名の種別	プログラム名・クラス名・利用者定義関数名
翻訳単位名	翻訳単位の名称
コンパイル日時	翻訳単位を含む原始プログラムをコンパイルした日時
実行日時	翻訳単位を含むプログラムを実行した日時
ソース要素名の種別	プログラム名・メソッド名・利用者定義関数名
ソース要素名	ソース要素の名称
行番号	コンパイラが振り直した行番号
実行回数	文のブロックが実行された回数。0～2,147,483,647 の範囲で表示する 2,147,483,647 を超えた場合は，実行回数に代えて"**********"を表示します。
ソーステキスト	原始プログラムの行

6.4　その他の機能

6.4.1　マルチスレッド対応 COBOL プログラムのカバレージ

マルチスレッドのプログラムのための注意事項を記述します。

注意事項

- AIX の場合，マルチスレッドプログラムをカバレージ対象とする場合は，各ユーザスレッドが一つのカーネルスレッドにマップされる 1：1 モデルで動作する必要があります。1：1 モデル以外で動作した場合，カバレージの動作は保証しません。1：1 モデルの設定は，環境変数 AIXTHREAD_SCOPE に S を設定することで行えます。

  設定例

```
AIXTHREAD_SCOPE=S
export AIXTHREAD_SCOPE
```

6.4.2　共用ライブラリ

(1)　コンパイルと実行

カバレージ情報の蓄積・カウント情報の表示で共用ライブラリを使用するとき，-PIC,Std，および-CVInf コンパイラオプションを指定してプログラムをコンパイルします。コンパイラオプションについては，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

共用ライブラリファイルは，cbltd2k コマンドや，cbltl2k コマンドの-Library オプションで，パスプレフィクスが付いていないファイル名を指定した場合，環境変数の指定有無によって検索順序が決まります。

-Library オプションで指定された共用ライブラリが，環境変数 CBLLSLIB，CBLLPATH，または LD_LIBRARY_PATH（Linux の場合）に設定したパスで見つかり，そのパスがロードされた共用ライブラリのパスと一致する場合は，ローディングされた共用ライブラリをカバレージ情報の蓄積・カウント情報の表示の対象とし，KCCC1203T-I メッセージを出力します。

「KCCC1203T-I ユーザプログラムのライブラリファイルがロードされました。ファイル（*** 1 ***)」

検索した結果，共用ライブラリが見つかってもパスが一致しない場合は，カバレージ情報の蓄積・カウント情報の表示の対象としないで，KCCC1203T-I メッセージも出力しません。

検索した結果，共用ライブラリが見つからない場合は，KCCC4207T-E のエラーメッセージを出力し，カバレージ情報の蓄積・カウント情報の表示の対象となりません。

「KCCC4207T-E　ユーザプログラムの実行可能ファイル又はライブラリファイルが存在しません。ファイル（*** 1 ***)」

OS ごとに，環境変数の指定有無による検索順序を説明します。

AIX の場合

1. 環境変数 CBLLSLIB の指定がある場合

   cblcv2k コマンドや cblcn2k コマンドの-Library オプションで指定したファイル名と同一名称のファイル名が，パスプレフィクス付きで環境変数 CBLLSLIB に指定されていれば，そのパスだけが検索されます。同一名称のファイル名が指定されていない，またはパスプレフィクスなしで指定されているときは，次の項目のとおりに検索します。

2. 環境変数 CBLLPATH の指定がある場合

   指定されているパス，カレントディレクトリの順序で検索します。

3. 環境変数が何も指定されていない場合

   カレントディレクトリを検索します。

Linux の場合

1. 環境変数 CBLLSLIB の指定がある場合

   - 環境変数 CBLLSLIB にパスプレフィクス付きのファイル名の指定があり，cblcv2k コマンドや cblcn2k コマンドの-Library オプションのファイル名と一致するとき

     環境変数 CBLLSLIB に指定されているパスだけ検索します。

   - 環境変数 CBLLSLIB にパスプレフィクスなしのファイル名の指定だけがあるとき

     環境変数 CBLLSLIB に指定されているファイルを，環境変数 CBLLPATH または環境変数 LD_LIBRARY_PATH に指定されているパスから検索します。

2. 環境変数 CBLLPATH の指定がある場合

   環境変数 CBLLSLIB に指定されているファイルを，環境変数 CBLLPATH に指定されているパスから検索します。カレントディレクトリを検索対象とするときは，パスにカレントディレクトリを追加する必要があります。

3. 環境変数 LD_LIBRARY_PATH の指定がある場合

   環境変数 CBLLSLIB に指定されているファイルを，環境変数 LD_LIBRARY_PATH に指定されているパスから検索します。カレントディレクトリを検索対象とするときは，パスにカレントディレクトリを追加する必要があります。

   なお，環境変数 CBLLPATH と同時に指定されているときは，環境変数 CBLLPATH に指定されているパスを検索したあと，環境変数 LD_LIBRARY_PATH に指定されているパスを検索します。

検索についての注意事項（AIX の場合）

環境変数 LIBPATH に指定されているパスは検索されません。この環境変数で共用ライブラリの検索パスを指定している場合は，-Library オプションにパスプレフィクスを付けて指定する必要があります。

注意事項

1. AIX の場合の注意事項を次に示します。

- ar コマンドでアーカイブファイルとして作成した共用ライブラリ中に，同じ名称のメンバ（共用オブジェクト）が複数あるときは，先に配置されているメンバ（共用オブジェクト）をカバレージ対象とします。
- -L および-l オプションを指定して共用ライブラリをリンケージしたプログラムを実行し，-DynamicLink,Call または-DynamicLink,IdentCall コンパイラオプションを指定して再びリンケージした場合，この共用ライブラリをカバレージ情報の蓄積，カウント情報の表示の対象とできない場合があります。その場合は，次のどちらかの指定をします。

  ・共用ライブラリの検索ディレクトリ指定の，-L オプションと同じディレクトリを，環境変数 CBLLPATH に指定。

  ・-L オプションと同じディレクトリを指定した共用ライブラリファイルを，環境変数 CBLLSLIB に指定。
- アーカイブファイルの中に含まれるアーカイブファイルは，カバレージ情報の蓄積，カウント情報の表示の対象になりません。
- 環境変数 CBLLTAG による検索動作の指定は，デバッガでは有効になりません。

2. 遅延ロードされる共用ライブラリはカバレージ情報の蓄積，およびカウント情報の表示の対象となりません。遅延ロードについては，システムのマニュアルを参照してください。
3. ld コマンドの-s オプションや strip コマンドなどで共用ライブラリのシンボル情報を削除した場合，その共用ライブラリは，カバレージ情報の蓄積，カウント情報の表示の対象になりません。

   なお，Linux では，cblcv2k コマンドや cblcn2k コマンドの-Execute オプションに指定する実行可能ファイルは，カバレージ情報の蓄積，カウント情報の表示の対象にしない場合でもシンボル情報が必要です。

使用例 1

ccbl2002 コマンドに-L および-l オプションを使用して実行可能ファイルを作成します。

1. sub1.cbl をコンパイルする。

AIXの場合

```
ccbl2002 -PIC,Std -CVInf sub1.cbl
```

Linuxの場合

```
ccbl2002 -PIC,Std -CVInf -UniObjGen sub1.cbl
```

2. 共用ライブラリを作成する。

AIX(32)の場合

```
ld -o libsub1.a sub1.o -bpT:0x10000000 -bpD:0x20000000 -bnoentry -bM:SRE -bexpall
-L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -lm -lc
```

AIX(64)の場合

```
ld -o libsub1.a sub1.o -b64 -bpT:0x100000000 -bpD:0x110000000 -bnoentry -bM:SRE -bexpall
-L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -lm -lc
```

Linux(x86)の場合

```
ld -shared -o libsub1.so sub1.o -Bstatic -L/opt/HILNGcbl2k/lib -lcbl2kml -Bdynamic -lc
```

Linux(x64)の場合

```
ld -shared -o libsub1.so sub1.o -Bstatic -L/opt/HILNGcbl2k64/lib -lcbl2kml -Bdynamic -lc
```

3. -L および-l オプションで共用ライブラリを指定して，実行可能ファイルを作成する。-L および-l オプションについては，システムの規則に従ってください。

   AIXの場合

   ```
   ccbl2002 -CVInf main1.cbl -L ./ -lsub1 -OutputFile a.out
   ```

   Linuxの場合

   ```
   ccbl2002 -CVInf -UniObjGen main1.cbl -L ./ -lsub1 -OutputFile a.out
   ```

4. -Library オプションを指定してカバレージ情報を蓄積する。

   AIX の場合

   ```
   cblcv2k -Library libsub1.a -Execute a.out
   ```

   Linux の場合

   ```
   cblcv2k -Library libsub1.so -Execute a.out
   ```

使用例 2

ccbl2002 コマンドの-DynamicLink,Call コンパイラオプションを使用して実行可能ファイルを作成します。

1. 使用例 1 の 1.および 2.と同様の手順で，sub1.cbl に-CVInf コンパイラオプションを指定してコンパイルし，共用ライブラリを作成する。

2. -DynamicLink,Call コンパイラオプションを指定して実行可能ファイルを作成する。

   AIXの場合

   ```
   ccbl2002 -DynamicLink,Call -CVInf main1.cbl -OutputFile a.out
   ```

   Linuxの場合

   ```
   ccbl2002 -DynamicLink,Call -CVInf -UniObjGen main1.cbl -OutputFile a.out
   ```

3. 使用例 1 の 4.と同様の手順で，-Library オプションを指定してカバレージ情報を蓄積する。

使用例 3

ccbl2002 コマンドの-DynamicLink,IdentCall コンパイラオプションを使用して実行可能ファイルを作成します。

1. 使用例 1 の 1.および 2.と同様の手順で，sub1.cbl に-PIC,Std，および-CVInf コンパイラオプションを指定してコンパイルし，共用ライブラリを作成する。

2. -DynamicLink,IdentCall コンパイラオプションを指定して実行可能ファイルを作成する。

   AIXの場合

```
ccbl2002 -DynamicLink,IdentCall -CVInf main1.cbl -OutputFile a.out
```

   Linuxの場合

```
ccbl2002 -DynamicLink,IdentCall -CVInf -UniObjGen main1.cbl -OutputFile a.out
```

3. 使用例 1 の 4.と同様の手順で，-Library オプションを指定してカバレージ情報を蓄積する。

(2) -TDInf および-CVInf コンパイラオプションがない COBOL プログラム，または COBOL プログラム以外から共用ライブラリを呼ぶ場合

-CVInf コンパイラオプションを指定した COBOL プログラムが含まれる共用ライブラリを，次のどちらかのプログラムで呼び出すときに，カバレージ情報の蓄積，カウント情報の表示ができない場合があります。

このような場合にカバレージ情報の蓄積，カウント情報の表示ができる環境変数 CBLTDEXTARGET について説明します。

- -TDInf および-CVInf コンパイラオプションの指定がない COBOL プログラム
- COBOL プログラムを含まない実行可能ファイル

環境変数

CBLTDEXTARGET

形式

```
CBLTDEXTARGET=YES
export CBLTDEXTARGET
```

機能

-CVInf コンパイラオプションを指定した COBOL プログラムが含まれる共用ライブラリを，次のどちらかのプログラムを呼ぶ場合に，カバレージ情報の蓄積，カウント情報の表示ができます。

- -TDInf および-CVInf コンパイラオプションの指定がない COBOL プログラム
- COBOL プログラムを含まない実行可能ファイル

注意事項

1. 環境変数 CBLTDEXTARGET は，カバレージ情報の蓄積，カウント情報の表示を使用するときに有効になります。運用環境では指定しないでください。
2. 環境変数 CBLTDEXTARGET を指定したときは，テスト対象のプロセスの情報を収集する回数が増加するため，処理速度が低下することがあります。カバレージ情報の蓄積，カウント情報の表示

を終了した部分は，-CVInf コンパイラオプションを外して再コンパイル後に，カバレージ情報の蓄積，カウント情報の表示を実行してください。

3. 次のディレクトリに格納した共用ライブラリは，カバレージ情報の蓄積，カウント情報の表示を対象にできません。

   AIX(32)および Linux(x86)の場合

   /usr/lib，/lib

   /opt/HILNGcbl2k/lib

   AIX(64)の場合

   /usr/lib，/lib

   /opt/HILNGcbl2k64/lib

   Linux(x64)の場合

   /usr/lib64，/lib64

   /opt/HILNGcbl2k64/lib

4. 環境変数 CBLTDEXTARGET の値に YES 以外を指定した場合，環境変数 CBLTDEXTARGET の指定は無効となります。

5. カバレージ情報の蓄積，カウント情報の表示を実行する前に，環境変数 CBLTDEXTARGET を設定する必要があります。プログラムの連動実行をするときは，プログラムを実行する前に環境変数 CBLTDEXTARGET を設定する必要があります。

7 カバレージ

カバレージの方法について説明します。

カバレージは，テスト工程の管理作業をサポートします。カバレージには，テストの進捗状況を定量的に表すカバレージ情報，文の実行回数をカウントするカウント情報の表示機能があります。

一括してカバレージを実行させることができます。

7.1 概要

端末から入力するコマンドの指示によって，カバレージの機能を操作できます。プログラムからカバレージを連動実行させ，一括してカバレージの蓄積およびカウント情報の表示をすることもできます。一度実行を開始すれば，利用者の操作を必要としないので，効率良く大量のプログラムのカバレージが実行できます。

7.2　カバレージ情報の蓄積

この節では，カバレージ情報を蓄積する方法について説明します。

7.2.1　コマンドによる方法

端末からコマンドを入力して，カバレージ情報を蓄積します。

カバレージ情報については，「6.2.1　カバレージ情報」を参照してください。

(1)　カバレージ情報の蓄積の手順

バッチモードでカバレージ情報を蓄積するときの作業の流れを説明します。

1. コンパイル時にはコンパイラオプションを指定する。
   -CVInf コンパイラオプションを必ず指定してください。
2. カバレージ情報の蓄積およびカバレージ情報の蓄積対象プログラムの実行に必要な環境変数を設定する。
   プログラムの実行に必要な環境変数と，カバレージに必要な環境変数を指定します。カバレージに必要な環境変数については，「(3)　環境変数の指定」を参照してください。プログラムの実行に必要な環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。
3. cblcv2k コマンドを指定して，実行する。
   cblcv2k コマンドについては，「(2)　cblcv2k コマンド」を参照してください。
4. 実行結果を端末に出力されるメッセージで確認する。カバレージ情報の蓄積結果は，カバレージ情報の表示機能を用いて確認する。
   カバレージ情報の表示機能については，「6.2　カバレージ情報の表示と操作」を参照してください。

(2)　cblcv2k コマンド

cblcv2k コマンドの形式を次に示します。

形式 1

```
cblcv2k 〔 -NoSave 〕〔 -Library 共用ライブラリ名 〔,…〕 〕
        -Execute 実行可能ファイル名 〔 プログラムへ渡す引数 〔,…〕 〕
```

形式 2

```
cblcv2k 〔 -Help 〕
```

-NoSave

プログラム情報ファイルへ書き込みをする前にバックアップをしません。

-Library

カバレージ情報を蓄積するプログラムがある共用ライブラリ名を指定します。

-Execute

カバレージ情報を蓄積するプログラムがある実行可能ファイル名を指定します。

プログラムへ渡す引数

カバレージ情報を蓄積するプログラムへ渡す引数を指定します。

-Help

cblcv2k コマンドの構文を表示します。-Help オプションを指定すると，ほかの引数はすべて無視されます。

注意事項

- -NoSave オプションの指定がないときは，プログラム情報ファイルへ書き込みを行う前にバックアップのファイルを生成します。バックアップのファイルは，プログラム情報ファイルへの書き込みでエラーが発生したときに残り，正常に終了したときは残りません。バックアップのファイル名は，システムが生成する，ほかに合致しない名称のファイルであり，カレントディレクトリに生成します。
- -Library オプションは，複数のファイル名を指定できます。
- -Execute オプションは，必ず最後に指定します。-Execute オプション以外のオプションの指定順序は任意です。
- 実行可能ファイル名のあとにある文字列は，すべてプログラムへ渡す引数とみなします。
- 起動後のカレントディレクトリは，cblcv2k コマンドを実行したディレクトリです。
- cblcv2k コマンドのメッセージは標準エラー出力へ出力します。
- cblcv2k コマンド名は，英小文字で指定します。
- cblcv2k コマンド名だけを指定した場合は，cblcv2k コマンドの構文を表示します。
- オプションは，英大文字，英小文字のどちらでも指定できます。オプションの始まりは，ハイフン（-）とします。
- オプションの区切り記号は空白文字およびタブです。空白文字およびタブを区切り記号としたくないときは，オプションをダブルコーテーション（"）で囲みます。
- 同じオプションを複数指定した場合は，最後に指定したオプションを有効とします。
- オプションにパスの付かないファイル名を指定したときは，カレントディレクトリのファイルとします。相対パスの付いたファイル名を指定したときは，カレントディレクトリを起点とする相対パスのディレクトリにあるファイルとします。
- オプションに複数のファイル名を指定するときは，コンマ（,）または空白で区切ります。また，アスタリスク（*）をファイル名の一部に指定することによって，（*）以外の文字が一致するすべてのファイルを指定できます。
- cblcv2k コマンドが返す終了コードは，次のとおりです。

終了コード	内容
0	正常終了
1	エラー発生による終了
2	キー操作による割り込みによる終了

- -Help オプションによるコマンドの構文は，標準出力へ出力します。それ以外のメッセージは，標準エラー出力へ出力します。
- 次に示すどれかに該当する場合，単独で実行したユーザプログラムが異常終了すると，「セグメンテーション違反です」などのメッセージがシステムから表示されます。カバレージからユーザプログラムを起動すると，システムからのこのメッセージは表示されません。

  ・実行時環境変数 CBLEXCEPT に NOSIGNAL を指定した場合

  ・次に示すコンパイラオプションのどれか一つも指定しないでコンパイルしたプログラムの場合

  -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange コンパイラオプション

  ・COBOL が例外（スタックオーバーフローなど）を検出できない場合

(3) 環境変数の指定

カバレージ情報の蓄積に必要な環境変数を指定します。

表 7-1 環境変数一覧（カバレージ情報の蓄積）

環境変数名	概要
CBLPIDIR	プログラム情報ファイルがあるディレクトリ名を指定する。
CBLLSLIB	共用ライブラリファイルのファイル名を指定する。
CBLLPATH	共用ライブラリファイルがあるディレクトリ名を指定する。
LD_LIBRARY_PATH※	共用ライブラリファイルがあるディレクトリ名を指定する。

注※

Linux で有効

プログラムの実行に関するその他の環境変数は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注意事項

1. カバレージ情報の蓄積のプログラム情報ファイルを次に示す検索順序で検索します。検索した結果，見つからないときは，プログラム情報ファイルに該当するプログラムはカバレージ情報の蓄積およびカウント情報表示の対象となりません。

   検索順序

   (1)環境変数 CBLPIDIR で指定したディレクトリ

(2)実行可能ファイルに含まれるプログラムは，実行可能ファイルのあるディレクトリ
　共用ライブラリファイルに含まれるプログラムは，共用ライブラリファイルのあるディレクトリ
(3)カレントディレクトリ

2. プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カバレージ情報の蓄積の対象となりません。
3. カバレージ情報の蓄積の対象となるプログラム情報ファイルが一つもない場合は，カバレージ情報の蓄積のためのプログラムを実行しません。
4. 環境変数 CBLLSLIB，環境変数 CBLLPATH，および環境変数 LD_LIBRARY_PATH については，「6.4.2　共用ライブラリ」を参照してください。

7.2.2　プログラムからの連動実行による方法

カバレージ情報の蓄積対象プログラムからコマンドを実行し，カバレージ情報を蓄積する方法です。

(1)　カバレージ情報の蓄積の手順

連動実行でカバレージ情報を蓄積するときの作業の流れを説明します。

1. **コンパイル時にはコンパイラオプションを指定する。**
   -CVInf コンパイラオプションを必ず指定してください。

2. **連動実行のための環境変数を指定する。**
   環境変数 CBLTDEXEC を指定します。形式は次のとおりです。

   形式

   ```
   CBLTDEXEC=CV 〔-OutFile 実行結果出力ファイル名 〕〔-Add〕
                〔-NoSave〕〔 -Library 共用ライブラリ名〔,…〕〕
                 -Execute 実行可能ファイル名
   ```

   各オプションの詳細および注意事項については，「(2)　環境変数の指定」を参照してください。

3. **カバレージ情報の蓄積または，カバレージ情報の蓄積対象プログラムの実行に必要な環境変数を設定する。**
   環境変数は，「6.1.1　カバレージ機能の入出力構成と使用するファイル」の「(3)　使用する環境変数の指定」を参照してください。プログラムの実行に必要な環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

4. **カバレージ情報の蓄積対象の実行可能ファイルを実行する。**
   カバレージ情報が蓄積されます。

5. **実行結果を実行結果出力ファイルによって確認する。カバレージ情報の蓄積結果は，カバレージ情報の表示機能を用いて確認する。**

カバレージ情報の表示機能については，「7.3　カバレージ情報の表示（テキスト形式）」を参照してください。実行結果出力ファイルについては「6.1.1　カバレージ機能の入出力構成と使用するファイル」の「(2)　使用するファイル」および「(3)　実行結果出力ファイルにメッセージを出力できないときの処理」を参照してください。

(2)　環境変数の指定

連動実行するための環境変数は，次のとおりです。その他の環境変数は，「7.2.1　コマンドによる方法」の「(3)　環境変数の指定」を参照してください。

形式

```
CBLTDEXEC=CV 〔-OutFile 実行結果出力ファイル名 〕〔-Add〕
             〔-NoSave〕〔 -Library 共用ライブラリ名〔,…〕〕
              -Execute 実行可能ファイル名
```

-OutFile

カバレージ情報の蓄積の実行結果およびトラブルシュート情報※を出力するファイル名を指定します。ファイル名の拡張子は，「.cvo」でなければなりません。-OutFile オプション省略時は，実行可能ファイル名に拡張子「.cvo」を付加した名前のファイルを，カレントディレクトリに出力します。

注※

当社保守員が調査するときに使用する情報です。共用ライブラリのカバレージ情報が蓄積されないなど，連動実行が動作しない場合には，当社保守員に連絡してください。

-Add

実行結果出力ファイルに追加書きで結果を出力します。省略すると，同名の実行結果出力ファイルが存在した場合に，上書きされます。

-NoSave

プログラム情報ファイルへ書き込みをする前にバックアップをしません。

-Library

カバレージ情報の蓄積の対象とする共用ライブラリ名を指定します。

-Execute

カバレージ情報の蓄積の対象とするプログラムを起動するための実行可能ファイル名を指定します。

注意事項

- AIX(32)の場合

  環境変数 PATH に，/opt/HILNGcbl2k/bin:/usr/bin を含めて指定する必要があります。
- Linux(x86)の場合

  環境変数 PATH に，/opt/HILNGcbl2k/bin:/usr/bin:/bin を含めて指定する必要があります。
- AIX(64)の場合

  環境変数 PATH に，/opt/HILNGcbl2k64/bin:/usr/bin を含めて指定する必要があります。

- Linux(x64)の場合

  環境変数 PATH に，/opt/HILNGcbl2k64/bin:/usr/bin:/bin を含めて指定する必要があります。
- -NoSave オプションの指定がないときは，プログラム情報ファイルへ書き込みを行う前にバックアップのファイルを生成します。バックアップのファイルは，プログラム情報ファイルへの書き込みでエラーが発生したときに残り，正常に終了したときは残りません。バックアップのファイル名は，システムが生成するほかに合致しない名称のファイルであり，カレントディレクトリに生成します。
- カレントディレクトリは，プログラムが実行されたディレクトリです。
- CV の文字とオプションの間は空白またはタブで区切ります。
- 実行結果出力ファイル名・共用ライブラリ名・実行可能ファイル名は，カレントディレクトリからの相対パスが指定されたものとします。カレントディレクトリが認識できないときは，絶対パス名で指定する必要があります。
- 環境変数 CBLTDEXEC で指定した実行可能ファイル名と，カバレージを起動した実行可能ファイルが一致する必要があります。
- プログラムを起動するときのパスの名称は，次の点に注意して指定します。次の条件に従わないパスの名称が指定された場合，カバレージの動作は保証しません。

  ・環境変数 CBLTDEXEC に指定した実行可能ファイル名およびプログラムを起動するときの実行可能ファイル名は，絶対パス名で指定します。相対パス名で指定した場合，またはパスプレフィクスを指定しなかった場合は，実行可能ファイル名だけが一致しているかどうかを確認します。

  ・プログラムを起動するときパス名は，60 バイト以下とします。60 バイトを超えたときは，61 バイト以降が切り捨てられ，60 バイトまでをパス名とみなします。

  ・パスを除いた実行可能ファイル名の長さは，14 バイト以下とします。14 バイトを超えたときは 15 バイト以降が切り捨てられ，14 バイトまでが実行可能ファイル名とみなされます。
- プログラムに渡す引数は，プログラムの起動時に指定します。環境変数 CBLTDEXEC には指定できません。
- カバレージ情報の蓄積のプログラム情報ファイルを次の順序で検索します。検索した結果，見つからないときは，プログラム情報ファイルに該当するプログラムはカバレージ情報の蓄積の対象となりません。

  1.環境変数 CBLPIDIR で指定したディレクトリ

  2.実行可能ファイルに含まれるプログラムは，実行可能ファイルのあるディレクトリ

  　共用ライブラリファイルに含まれるプログラムは，共用ライブラリファイルのあるディレクトリ

  3.カレントディレクトリ
- プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カバレージ情報の蓄積の対象となりません。
- カバレージ情報の蓄積の対象となるプログラム情報ファイルが一つもない場合は，カバレージ情報の蓄積のためのプログラムを実行しません。

- 環境変数 CBLLSLIB，環境変数 CBLLPATH，および環境変数 LD_LIBRARY_PATH については，「6.4.2　共用ライブラリ」を参照してください。
- 環境変数 CBLTDEXEC に空白またはタブを含むパス名を指定できません。
- 拡張子が「.cvo」ではないファイルを実行結果出力ファイル名に指定した場合は，エラーになり連動実行は開始しません。

(3)　実行結果出力ファイルにメッセージを出力できないときの処理

実行結果出力ファイルにメッセージを出力できない場合は，次のメッセージを出力します。

- 環境変数 TMPDIR の指定がある場合は指定されたディレクトリに，環境変数 TMPDIR に指定がない場合は/tmp ディレクトリに，次のファイル名で実行結果を出力します。

  *****は，一意な文字が自動的に割り振られます。

  - カバレージ情報の蓄積では，CVO*****.tmp
  - カウント情報の表示では，CNO*****.tmp
- /tmp ディレクトリに出力された実行結果のファイルは，不要になったら削除してください。
- /tmp ディレクトリに出力できないときは，標準エラー出力に出力します。

7.3　カバレージ情報の表示（テキスト形式）

端末から，コマンドを入力してカバレージ情報をテキスト形式で表示します。

カバレージ情報の表示については，「6.2.5　カバレージ情報の表示（テキスト形式）」を参照してください。

7.3.1　カバレージ情報の表示の方法（テキスト形式）

端末から，コマンドを入力してカバレージ情報をテキスト形式で表示します。

(1)　カバレージ情報の表示の手順

バッチモードでカバレージ情報（テキスト形式）を表示するときの作業の流れを説明します。

1. カバレージ情報を蓄積する。

   カバレージ情報の蓄積については，「7.2.1　コマンドによる方法」，「7.2.2　プログラムからの連動実行による方法」を参照してください。

2. cblcl2k コマンドを指定して，実行する。

   cblcl2k コマンドについては，「(2)　cblcl2k コマンド」を参照してください。

3. コマンドに指定したカバレージ情報リストファイルをテキストエディタ（vi や FSED など）で開き，内容を確認する。

(2)　cblcl2k コマンド

cblcl2k コマンドの形式を次に示します。

形式 1

```
cblcl2k 〔-List〕 〔 -Summary〕 〔 -Source〕 〔 -Difference 〕
        〔 -Unexecuted〕 〔 -UnexecutedDifference 〕
        〔 -Activate〕 〔-All〕
        〔 -OutPut カバレージ情報リストファイル名 〕
          -Input プログラム情報ファイル名 〔,…〕
```

形式 2

```
cblcl2k 〔 -Help 〕
```

-List

すべてのプログラムのカバレージ情報一覧を表示します。

-Summary

カバレージ情報のまとめを表示します。

-Source

すべてのソース情報を表示します。

-Difference

差分ソース情報を表示します。

-Unexecuted

未実行ソース情報を表示します。

-UnexecutedDifference

差分未実行ソース情報を表示します。

-Activate

呼び出し文情報を表示します。

-All

すべての情報を表示します。次の引数をすべて指定したのと同じです。

-List，-Summary，-Source，-Difference，-Unexecuted，-UnexecutedDifference，-Activate

-All が指定されたときは，ほかの引数は無視されます。

-Output

カバレージ情報リストファイルのパス名を指定します。ファイルの拡張子は，「.cll」でなければなりません。省略時はカレントディレクトリに，最初に指定したプログラム情報ファイル名に拡張子「.cll」を付けたファイル名で出力されます。

-Input

プログラム情報ファイルのパス名を指定します。

複数指定したプログラム情報ファイルは，先頭から拡張子をチェックします。拡張子の不正を検出した場合は，それよりもあとに指定しているプログラム情報ファイルの拡張子をチェックしないで処理を終了します。

また，-Input の直後にコンマ，または末尾にコンマを指定している場合も，エラーとなり，処理を終了します。

-Help

cblcl2k コマンドの構文を表示します。-Help オプションを指定すると，ほかの引数はすべて無視されます。

出力されるカバレージ情報リストの詳細については，「6.2.5　カバレージ情報の表示（テキスト形式）」を参照してください。

注意事項

- cblcl2k コマンドのメッセージは標準エラー出力へ出力します。
- cblcl2k コマンド名は，英小文字で指定します。
- cblcl2k コマンド名だけを指定した場合は，cblcl2k コマンドの構文を表示します。
- オプションは，英大文字，英小文字のどちらでも指定できます。オプションの始まりは，ハイフン（-）とします。

- オプションの区切り記号は空白文字とタブです。空白文字およびタブを区切り記号としたくないときは，オプションをダブルコーテーション（"）で囲みます。
- 同じオプションを複数指定した場合は，最後に指定したオプションを有効とします。
- オプションにパスの付かないファイル名を指定したときは，カレントディレクトリのファイルとします。相対パスの付いたファイル名を指定したときは，カレントディレクトリを起点とする相対パスのディレクトリにあるファイルとします。
- オプションに複数のファイル名を指定するときは，コンマ（,）または空白文字で区切ります。また，アスタリスク（*）をファイル名の一部に指定することによって，（*）以外の文字が一致するすべてのファイルを指定できます。
- cblcl2k コマンドが返す終了コードは，次のとおりです。

終了コード	内容
0	正常終了
1	エラー発生による終了
2	キー操作による割り込みによる終了

- -Help オプションによるコマンドの構文は，標準出力へ出力します。それ以外のメッセージは，標準エラー出力へ出力します。

(3) 環境変数の指定

カバレージ情報の表示に必要なファイルを環境変数に指定します。

表 7-2 環境変数（カバレージ情報の表示）

環境変数名	概要
CBLPIDIR	プログラム情報ファイルの存在するディレクトリをパスで指定する。

注意事項

- カバレージ情報の表示のプログラム情報ファイルがパスの付かないファイル名または，相対パスで指定された場合，次のディレクトリを検索します。相対パスを指定したときは，次のディレクトリからの相対パスを検索します。絶対パスが指定されたとき，次の検索はしません。

  1.環境変数 CBLPIDIR で指定したディレクトリ

  2.カレントディレクトリ
- プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カバレージ情報の表示の対象となりません。

  カバレージ情報の表示では対象外とするエラーメッセージを出力します。
- カバレージ情報の表示の対象となるプログラム情報ファイルが一つもない場合は，カバレージの表示をしません。カバレージ情報リストファイルも出力しません。

7.4　カバレージ情報の表示（CSV 形式）

端末から，コマンドを入力してカバレージ情報を CSV 形式で表示します。

カバレージ情報の表示については，「6.2.6　カバレージ情報の表示（CSV 形式）」を参照してください。

7.4.1　カバレージ情報の表示の方法（CSV 形式）

端末から，コマンドを入力してカバレージ情報を CSV 形式で表示します。

(1)　カバレージ情報の表示の手順

バッチモードでカバレージ情報（CSV 形式）を表示するときの作業の流れを説明します。

1. カバレージ情報を蓄積する。

   カバレージ情報の蓄積については，「7.2.1　コマンドによる方法」，「7.2.2　プログラムからの連動実行による方法」を参照してください。

2. cblcc2k コマンドを指定して，実行する。

   cblcc2k コマンドについては，「(2)　cblcc2k コマンド」を参照してください。

   出力された CSV ファイルをエディタや Excel などの表計算プログラムで開き，内容を確認する。

(2)　cblcc2k コマンド

cblcc2k コマンドの形式を次に示します。

形式 1

```
cblcc2k　〔-List〕〔-S ou rc e〕〔-All〕〔-NoH eader〕
　　〔-ListS ourceFile〕〔-ListC ompilationUnit〕〔-ListA ll〕〔-Com ment〕
　　〔-ListF ileName カバレージ統計情報CSVファイル名〕
　　〔-Out put P ath 出力先ディレクトリ名〕〔-Rec ursive〕
　　{-In put プログラム情報ファイル名〔, …〕 |-In put P ath 入力ディレクトリ名}
```

形式 2

```
cblcc2k　〔-H elp〕
```

-List

　カバレージ統計情報 CSV ファイルを出力します。

-Source

　ソースカバレージ情報 CSV ファイルを出力します。

-All

　カバレージ統計情報 CSV ファイルとソースカバレージ情報 CSV ファイルの両方を出力します。

-List と-Source を同時に指定したのと同じです。

-NoHeader

ヘッダカラム名とヘッダを出力しません。

この引数を指定した場合，カバレージ統計情報 CSV ファイルとソースカバレージ情報 CSV ファイルには，それぞれ次の内容を出力します。

- カバレージ統計情報 CSV ファイルのとき

  カバレージ統計情報カラム名とカバレージ統計情報

- ソースカバレージ情報 CSV ファイルのとき

  ソースカバレージ情報カラム名とソースカバレージ情報

カバレージ統計情報 CSV ファイルおよびソースカバレージ情報 CSV ファイルの出力内容については，「6.2.6　カバレージ情報の表示（CSV 形式）」を参照してください。

-ListSourceFile

カバレージ統計情報 CSV ファイルにソースファイル単位の統計情報を出力します。カバレージ統計情報 CSV ファイルを出力しない場合は無視されます。

-ListCompilationUnit

カバレージ統計情報 CSV ファイルに翻訳単位の統計情報を出力します。カバレージ統計情報 CSV ファイルを出力しない場合は無視されます。

-ListAll

カバレージ統計情報 CSV ファイルに，ソースファイル単位と翻訳単位の両方の統計情報を出力します。

-ListSourceFile と-ListCompilationUnit を同時に指定したのと同じです。カバレージ統計情報 CSV ファイルを出力しない場合は無視されます。

-Comment

ソースに記述されたコメント行（注記行）をソースカバレージ情報 CSV ファイルに出力します。ソースカバレージ情報 CSV ファイルを出力しない場合は無視されます。

省略時は，ソースに記述されたコメント行（注記行）をソースカバレージ情報 CSV ファイルに出力しません。

-ListFileName

出力するカバレージ統計情報 CSV ファイルのファイル名（.csv）を指定します。ファイル名には，カバレージ統計情報 CSV ファイルを格納するディレクトリのパスを含めないで，ファイル名だけを指定してください。

省略時は，「summary_list.csv」を仮定します。カバレージ統計情報 CSV ファイルを出力しない場合は無視されます。

-OutputPath

生成する CSV ファイルの出力先ディレクトリのパスを指定します。出力先ディレクトリ名には，存在するディレクトリを指定してください。省略時は，出力先ディレクトリをカレントディレクトリとします。

- カバレージ統計情報 CSV ファイルの場合

  -OutputPath オプションで指定したディレクトリに，-ListFileName オプションで指定したファイル名で出力します。

- ソースカバレージ情報 CSV ファイルの場合

  -OutputPath オプションで指定したディレクトリに，プログラム情報ファイル名（拡張子を除く）に「_source.csv」を付けたファイル名で出力します。例えば，プログラム情報ファイル名が「makeline.cbp」の場合は，ソースカバレージ情報 CSV ファイルは「makeline_source.csv」で出力します。

-Recursive

-InputPath で指定したディレクトリと，そのサブディレクトリにあるすべてのプログラム情報ファイルを出力対象とします。また，-OutputPath に指定した出力先ディレクトリに，-InputPath で指定したディレクトリ下と同じディレクトリ構成で出力します。

省略時は，-InputPath で指定したディレクトリの直下にあるプログラム情報ファイルだけを出力対象とし，-OutputPath に指定した出力先ディレクトリの直下に出力します。

この引数は，-InputPath を指定した場合にだけ有効になります。

-Input

出力対象のプログラム情報ファイル名を指定します。

プログラム情報ファイル名は，コンマ（,）区切り，または空白区切りで複数指定できます。ファイル名が同じプログラム情報ファイル名は同時に指定できません。絶対パスや相対パスで指定した場合も，同じパスではなく，同じプログラム情報ファイル名は同時に指定できません。

複数指定したプログラム情報ファイルは，先頭から拡張子をチェックします。拡張子の不正を検出した場合は，それよりもあとに指定しているプログラム情報ファイルの拡張子をチェックしないで処理を終了します。

また，-Input の直後にコンマ，または末尾にコンマを指定している場合も，エラーとなり，処理を終了します。

-InputPath

出力対象のプログラム情報ファイルを格納しているディレクトリのパスを指定します。

-Help

cblcc2k コマンドの構文を表示します。-Help オプションを指定すると，ほかの引数はすべて無視されます。

出力されるカバレージ情報リストの詳細については，「6.2.6　カバレージ情報の表示（CSV 形式）」を参照してください。

注意事項

- -List，-Source または-All のうち何も指定がなかった場合，-All オプションを仮定します。
- -ListSourceFile，-ListCompilationUnit または-ListAll のうち何も指定がなかった場合，-ListAll オプションを仮定します。

- -ListFileName の指定によって，ソースカバレージ情報 CSV ファイルとカバレージ統計情報 CSV ファイルが同一ファイルになった場合の動作は保証しません。
- cblcc2k コマンドのメッセージは標準エラー出力へ出力します。
- 複数のファイルを出力する場合に，処理中にエラーが発生したときは，それ以降のファイルの出力を中止します。それより前に出力したファイルは削除されません。
- cblcc2k コマンド名は，英小文字で指定します。
- cblcc2k コマンド名だけを指定した場合は，cblcc2k コマンドの構文を表示します。
- オプションは，英大文字，英小文字のどちらでも指定できます。オプションの始まりは，ハイフン（-）とします。
- オプションの区切り記号は空白文字とタブです。空白文字およびタブを区切り記号としたくないときは，オプションをダブルコーテーション（"）で囲みます。
- 同じオプションを複数指定した場合は，最後に指定したオプションを有効とします。
- オプションにパスの付かないファイル名を指定したときは，カレントディレクトリのファイルとします。相対パスの付いたファイル名を指定したときは，カレントディレクトリを起点とする相対パスのディレクトリにあるファイルとします。
- オプションに複数のファイル名を指定するときは，コンマ（,）または空白文字で区切ります。また，アスタリスク（*）をファイル名の一部に指定することによって，（*）以外の文字が一致するすべてのファイルを指定できます。
- cblcc2k コマンドが返す終了コードは，次のとおりです。

終了コード	内容
0	正常終了
1	エラー発生による終了
2	キー操作による割り込みによる終了

- -Help オプションによるコマンドの構文は，標準出力へ出力します。それ以外のメッセージは，標準エラー出力へ出力します。

(3) 環境変数の指定

カバレージ情報の表示に必要なファイルを環境変数に指定します。

表 7-3 環境変数（カバレージ情報の表示）

環境変数名	概要
CBLPIDIR	プログラム情報ファイルの存在するディレクトリをパスで指定する。

注意事項

- カバレージ情報の表示のプログラム情報ファイルがパスの付かないファイル名または，相対パスで指定された場合，次のディレクトリを検索します。相対パスを指定したときは，次のディレクトリからの相対パスを検索します。絶対パスが指定されたとき，次の検索はしません。

  1.環境変数 CBLPIDIR で指定したディレクトリ

  2.カレントディレクトリ

- プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カバレージ情報の表示の対象となりません。

  カバレージ情報の表示では対象外とするエラーメッセージを出力します。

- カバレージ情報の表示の対象となるプログラム情報ファイルが一つもない場合は，カバレージの表示をしません。カバレージ統計情報 CSV ファイルも出力しません。

7.5　カバレージ情報の操作

蓄積したカバレージ情報に対して次の操作ができます。

- カバレージ全情報を 0%にする
- カバレージ差分情報を 0%にする
- ソース差分情報をクリアする
- カバレージ情報をマージする

7.5.1　コマンドによる方法

端末からコマンドを入力して，カバレージ情報を操作できます。

カバレージ情報の操作については，「6.2.7　カバレージ情報の操作」を参照してください。

(1)　カバレージ情報の操作の方法

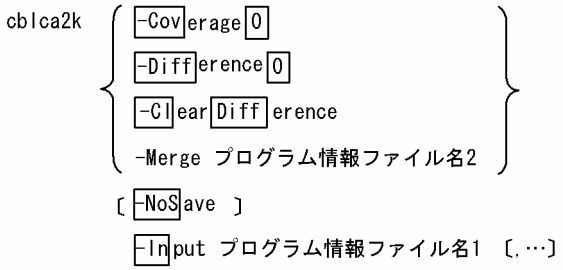
バッチモードでカバレージ情報を操作するときの作業の流れを説明します。

1. cblca2k コマンドに操作したい機能を指定して，実行する。
   cblca2k コマンドについては，「(2)　cblca2k コマンド」を参照してください。

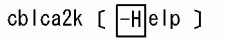
(2)　cblca2k コマンド

cblca2k コマンドの形式を次に示します。

形式 1



形式 2



-Coverage0
　カバレージ全情報を 0%にします。

-Difference0
　カバレージ差分情報を 0%にします。

-ClearDifference

ソース差分情報をクリアします。

-Merge

プログラム情報ファイル名 1 のカバレージ情報をプログラム情報ファイル名 2 へマージします。プログラム情報ファイル名 1 は，パスを除いたファイル名がプログラム情報ファイル 1 と同じでなければなりません。

-NoSave

プログラム情報ファイルへ書き込みを行う前にバックアップをしません。

-Input

プログラム情報ファイルを指定します。

-Help

cblca2k コマンドの構文を表示します。-Help オプションを指定すると，ほかの引数はすべて無視されます。

注意事項

- -Merge オプションを指定した場合，プログラム情報ファイル名 1 のパスを除いたファイル名はすべて同じでなければなりません。
- -Coverage0，-Difference0，-ClearDifference，-Merge オプションから複数のオプションを指定すると，最後に指定されたオプションが有効となります。
- -NoSave オプションの指定がない場合は，プログラム情報ファイルへ書き込みをする前にバックアップのファイルを生成します。バックアップのファイルは，プログラム情報ファイルへの書き込みでエラーが発生したときに残り，正常に終了したときは残りません。バックアップのファイル名は，システムが生成するほかに合致しない名称のファイルであり，カレントディレクトリに生成します。
- cblca2k コマンド名は，英小文字で指定します。
- cblca2k コマンド名だけを指定した場合は，cblca2k コマンドの構文を表示します。
- オプションは，英大文字，英小文字のどちらでも指定できます。オプションの始まりは，ハイフン（-）とします。
- オプションの区切り記号は空白文字およびタブです。空白文字およびタブを区切り記号としたくないときは，オプションをダブルコーテーション（"）で囲みます。
- 同じオプションを複数指定した場合は，最後に指定したオプションを有効とします。
- オプションにパスの付かないファイル名を指定したときは，カレントディレクトリのファイルとします。相対パスの付いたファイル名を指定したときは，カレントディレクトリを起点とする相対パスのディレクトリにあるファイルとします。
- オプションに複数のファイル名を指定するときは，コンマ（,）または空白で区切ります。また，アスタリスク（*）をファイル名の一部に指定することによって，（*）以外の文字が一致するすべてのファイルを指定できます。

- cblca2k コマンドが返す終了コードは，次のとおりです。

終了コード	内容
0	正常終了
1	エラー発生による終了
2	キー操作による割り込みによる終了

- -Help オプションによるコマンドの構文は，標準出力へ出力します。それ以外のメッセージは，標準エラー出力へ出力します。

(3) 環境変数の指定

カバレージ情報の操作に必要なファイルを環境変数に指定します。

表 7-4 環境変数（カバレージ情報の操作）

環境変数名	概要
CBLPIDIR	プログラム情報ファイルの存在するディレクトリをパスで指定する。

注意事項

- カバレージ情報の操作のプログラム情報ファイルがパスの付かないファイル名，または相対パスで指定された場合，次のディレクトリを検索します。相対パスを指定したときは，次のディレクトリからの相対パスを検索します。絶対パスが指定されたとき，次の検索はしません。

  1.環境変数 CBLPIDIR で指定したディレクトリ

  2.カレントディレクトリ

- プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カバレージ情報の操作の対象となりません。

  カバレージ情報の操作では対象外とするエラーメッセージを出力します。

- カバレージ情報の操作の対象となるプログラム情報ファイルが一つもない場合は，カバレージの操作をしません。

7.6　カウント情報の表示

カウント情報は，テスト実行させたプログラムの文の実行回数を表示します。COBOL プログラムの実行性能改善の施策を検討するときに，各文の実行回数を測定し，COBOL プログラムの性能を分析するために必要な情報を取得し，表示できます。

7.6.1　コマンドによる方法

端末から，コマンドを入力してカウント情報を取得します。

(1)　カウント情報の表示の手順

バッチモードでカウント情報を取得するときの作業の流れを説明します。

1. コンパイラオプションを指定して，プログラムをコンパイルする。
   -CVInf コンパイラオプションを必ず指定してください。

2. カウント情報の取得またはカウント情報の取得対象プログラムの実行に必要な環境変数を設定する。
   カウント情報の取得に必要な環境変数については，「(3)　環境変数の指定」を参照してください。プログラムの実行に必要な環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

3. cblcn2k コマンドを指定して，実行する。
   cblcn2k コマンドについては，「(2)　cblcn2k コマンド」を参照してください。

4. 実行結果を画面に出力されるメッセージから確認する。カウント情報の取得取結果は，カウント情報リストファイルをテキストエディタ（vi や FSED など）で開いて確認する。

(2)　cblcn2k コマンド

COBOL プログラムを実行してカウント情報を取得し，結果をファイルに出力します。

cblcn2k コマンドの形式を次に示します。

形式 1

```
cblcn2k 〔 -Library 共用ライブラリ名 〔,…〕 〕
        〔 -Output カウント情報リストファイル名 〕
          -Execute 実行可能ファイル名 〔 プログラムへ渡す引数 〕
```

形式 2

```
cblcn2k 〔 -Help 〕
```

-Library

カウント情報の表示を行う共用ライブラリ名を指定します。

-Output

カウント情報リストファイル名を指定します。ファイル名の拡張子は，「.cnl」でなければなりません。-Output オプション省略時は，-Execute で指定した実行可能ファイル名の拡張子を「.cnl」に変更したファイル名のファイルを，カレントディレクトリに出力します。

-Execute

実行可能ファイル名を指定します。

-Help

cblcn2k コマンドの構文を表示します。-Help オプションを指定すると，ほかの引数はすべて無視されます。

注意事項

- 実行可能ファイルのパス名のあとに指定されたものすべてをカウント情報の取得対象のプログラムへ渡す引数とみなします。
- -Library オプションは，複数のファイル名を指定できます。
- -Execute オプションは，必ず最後に指定します。-Execute オプション以外のオプションの指定順序は任意です。
- 起動後のカレントディレクトリは，cblcn2k コマンドを実行したディレクトリです。
- cblcn2k コマンドのメッセージは標準エラー出力へ出力します。
- cblcn2k コマンド名は，英小文字で指定します。
- cblcn2k コマンド名だけを指定した場合は，cblcn2k コマンドの構文を表示します。
- オプションは，英大文字，英小文字のどちらでも指定できます。オプションの始まりは，ハイフン（-）とします。
- オプションの区切り記号は空白文字およびタブです。空白文字およびタブを区切り記号としたくないときは，オプションをダブルコーテーション（"）で囲みます。
- 同じオプションを複数指定した場合は，最後に指定したオプションを有効とします。
- オプションにパスの付かないファイル名を指定したときは，カレントディレクトリのファイルとします。相対パスの付いたファイル名を指定したときは，カレントディレクトリを起点とする相対パスのディレクトリにあるファイルとします。
- オプションに複数のファイル名を指定するときは，コンマ（,）または空白文字で区切ります。また，アスタリスク（*）をファイル名の一部に指定することによって，（*）以外の文字が一致するすべてのファイルを指定できます。
- cblcn2k コマンドが返す終了コードは，次のとおりです。

終了コード	内容
0	正常終了

終了コード	内容
1	エラー発生による終了
2	キー操作による割り込みによる終了

- -Help オプションによるコマンドの構文は，標準出力へ出力します。それ以外のメッセージは，標準エラー出力へ出力します。
- 次に示すどれかに該当する場合，単独で実行したユーザプログラムが異常終了すると，「セグメンテーション違反です」などのメッセージがシステムから表示されます。カバレージからユーザプログラムを起動すると，システムからのこのメッセージは表示されません。

  ・実行時環境変数 CBLEXCEPT に NOSIGNAL を指定した場合

  ・次に示すコンパイラオプションのどれか一つも指定しないでコンパイルしたプログラムの場合

  -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange コンパイラオプション

  ・COBOL が例外（スタックオーバーフローなど）を検出できない場合

(3)　環境変数の指定

カウント情報の表示に必要な環境変数に指定します。

表 7-5　環境変数一覧（カウント情報の表示）

環境変数名	概要
CBLPIDIR	プログラム情報ファイルがあるディレクトリ名を指定する。
CBLLSLIB	共用ライブラリファイルのファイル名を指定する。
CBLLPATH	共用ライブラリファイルがあるディレクトリ名を指定する。
LD_LIBRARY_PATH※	共用ライブラリファイルがあるディレクトリ名を指定する。

注※

Linux で有効

プログラムの実行に関するその他の環境変数は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注意事項

- カウント情報の表示のプログラム情報ファイルを次の順序で検索します。検索した結果，見つからないときは，プログラム情報ファイルに該当するプログラムはカウント情報表示の対象となりません。

  1.環境変数 CBLPIDIR で指定したディレクトリ

  2.実行可能ファイルに含まれるプログラムは，実行可能ファイルのあるディレクトリ

  共用ライブラリファイルに含まれるプログラムは，共用ライブラリファイルのあるディレクトリ

  3.カレントディレクトリ

- プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カウント情報の表示の対象となりません。
- カウント情報の表示の対象となるプログラム情報ファイルが一つもない場合は，カウント情報の表示のためのプログラムを実行しません。カウント情報リストファイルも出力しません。
- 環境変数 CBLLSLIB，環境変数 CBLLPATH，および環境変数 LD_LIBRARY_PATH については，「6.4.2　共用ライブラリ」を参照してください。

7.6.2　プログラムからの連動実行による方法

カウント情報の取得対象プログラムからコマンドを実行しカウント情報を取得する方法です。

(1)　カウント情報の表示の手順

プログラムからの連動実行で，プログラムのカウント情報を表示するときの作業の流れを説明します。

1. コンパイラオプションを指定して，プログラムをコンパイルする。
   -CVInf コンパイラオプションを必ず指定してください。

2. 連動実行の環境変数を指定する。
   環境変数 CBLTDEXEC を指定します。形式は次のとおりです。

   形式

   ```
   CBLTDEXEC=CN 〔 -Output カウント情報リストファイル名 〕
                〔 -OutFile 実行結果出力ファイル名 〕 〔-Add 〕
                〔 -Library 共用ライブラリファイル名 〕 〔,…〕
                  -Execute 実行可能ファイル名
   ```

   各オプションの詳細および注意事項については，「(2)　環境変数の指定」を参照してください。

3. カウント情報の取得またはプログラムの実行に必要な環境変数を設定する。
   カウント情報の取得に必要な環境変数は，「6.1.1　カバレージ機能の入出力構成と使用するファイル」の「(3)　使用する環境変数の指定」を参照してください。プログラムの実行に必要な環境変数については，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。

4. カウント情報の取得対象の実行可能ファイルを実行する。
   カウント情報が取得されます。

5. 実行結果を実行結果出力ファイルで確認する。取得したカウント情報はカウント情報リストファイルをテキストエディタ（vi や FSED など）で開いて確認する。

(2)　環境変数の指定

連動実行するための環境変数は，次のとおりです。その他の環境変数は，「7.6.1　コマンドによる方法」の「(3)　環境変数の指定」を参照してください。

形式

```
CBLTDEXEC=CN 〔 -Output カウント情報リストファイル名 〕
             〔 -OutFile 実行結果出力ファイル名 〕 〔-Add 〕
             〔 -Library 共用ライブラリファイル名 〕 〔,…〕
               -Execute 実行可能ファイル名
```

-Output

カウント情報リストファイル名を指定します。ファイルの拡張子は，「.cnl」でなければなりません。指定しなかった場合は，カレントディレクトリに，実行可能ファイルに拡張子「.cnl」を付けた名前で出力されます。

-OutFile

カウントの実行結果およびトラブルシュート情報※を出力するファイル名を指定します。ファイルの拡張子は，「.cno」でなければなりません。指定しなかった場合は，実行可能ファイルに拡張子「.cno」を付けた名前でカレントディレクトリに出力します。

注※

当社保守員が調査するときに使用する情報です。共用ライブラリのカウント情報が表示されないなど，連動実行が動作しない場合には，当社保守員に連絡してください。

-Add

実行結果出力ファイルに追加書きで結果を出力します。指定しなかった場合，同名の実行結果出力ファイルがあった場合は，上書きされます。

-Library

カウント情報取得の対象とする共用ライブラリファイル名を指定します。

-Execute

カウント情報取得の対象とするプログラムを起動するための実行可能ファイル名を指定します。

注意事項

- AIX(32)の場合

  環境変数 PATH に，/opt/HILNGcbl2k/bin:/usr/bin を含めて指定する必要があります。

- Linux(x86)の場合

  環境変数 PATH に，/opt/HILNGcbl2k/bin:/usr/bin:/bin を含めて指定する必要があります。

- AIX(64)の場合

  環境変数 PATH に，/opt/HILNGcbl2k64/bin:/usr/bin を含めて指定する必要があります。

- Linux(x64)の場合

  環境変数 PATH に，/opt/HILNGcbl2k64/bin:/usr/bin:/bin を含めて指定する必要があります。

- カレントディレクトリは，プログラムが実行されたディレクトリです。
- ＣＮの文字とオプションの間は空白またはタブで区切ってください。
- カウント情報リストファイル名・実行結果出力ファイル名・共用ライブラリ名・実行可能ファイル名は，カレントディレクトリからの相対パスが指定されたものとします。カレントディレクトリが認識できないときは，絶対パス名で指定する必要があります。
- 環境変数 CBLTDEXEC に指定した実行可能ファイル名と，COBOL2002 カバレージを起動した実行可能ファイルが一致する必要があります。
- プログラムを起動するときのパスの名称は，次の点に注意して指定します。これらに反したパスの名称が指定された場合，カバレージの動作は保証しません。

  ・環境変数 CBLTDEXEC に指定した実行可能ファイル名と，プログラムを起動するときの実行可能ファイル名は，絶対パス名で指定します。相対パス名で指定した場合，またはパスプレフィクスを指定しなかった場合は，実行可能ファイル名だけが一致しているかどうかを確認します。

  ・プログラムを起動するときパス名は，60 バイト以下とします。60 バイトを超えたときは，61 バイト以降が切り捨てられ，60 バイトまでをパス名とみなします。

  ・パスを除いた実行可能ファイル名の長さは，14 バイト以下とします。14 バイトを超えたときは 15 バイト以降が切り捨てられ，14 バイトまでが実行可能ファイル名とみなされます。
- プログラムに渡す引数は，プログラムの起動時に指定します。環境変数 CBLTDEXEC には指定できません。
- カウント情報の表示のプログラム情報ファイルを次の順序で検索します。検索した結果，見つからないときは，プログラム情報ファイルに該当するプログラムはカウント情報表示の対象となりません。

  1.環境変数 CBLPIDIR で指定したディレクトリ

  2.実行可能ファイルに含まれるプログラムは，実行可能ファイルのあるディレクトリ

  　共用ライブラリファイルに含まれるプログラムは，共用ライブラリファイルのあるディレクトリ

  3.カレントディレクトリ
- プログラムが-CVInf コンパイラオプションでコンパイルされていない場合，カウント情報の表示の対象となりません。
- カウント情報の表示の対象となるプログラム情報ファイルが一つもない場合は，カウント情報の表示のためのプログラムを実行しません。カウント情報リストファイルも出力しません。
- 環境変数 CBLLSLIB，環境変数 CBLLPATH，および環境変数 LD_LIBRARY_PATH については，「6.4.2　共用ライブラリ」を参照してください。
- 環境変数 CBLTDEXEC に空白またはタブを含むパス名を指定できません。
- 拡張子が「.cno」ではないファイルを実行結果出力ファイル名に指定した場合は，エラーになり連動実行は開始しません。

(3)　実行結果出力ファイルにメッセージを出力できないときの処理

実行結果出力ファイルにメッセージを出力できない場合のメッセージの出力方法については，「7.2.2　プログラムからの連動実行による方法」の「(3)　実行結果出力ファイルにメッセージを出力できないときの処理」を参照してください。

7.7　cblcv コマンド（COBOL85/TD 互換）

cblcv コマンドは，UNIX COBOL85 からの COBOL2002 への移行を円滑にするために使用するコマンドです。cblcv2k コマンドと同等の機能を持つため，通常は cblcv2k コマンドを使用することをお勧めします。

指定する実行可能ファイル・共用ライブラリは COBOL2002 で生成されている必要があります。

7.7.1　cblcv コマンド

cblcv コマンドの形式を次に示します。

形式

```
cblcv 〔 -s 〕〔 -d 共用ライブラリ名〔,共用ライブラリ名 … 〕〕
                実行可能ファイル 〔 プログラムへ渡す引数 〕
```

-s

　プログラム情報ファイルへ書き込みをする前にバックアップをします。

-d

　カバレージ情報を蓄積するプログラムがある共用ライブラリ名を指定します。

実行可能ファイル名

　カバレージ情報を蓄積するプログラムがある実行可能ファイル名を指定します。

プログラムへ渡す引数

　カバレージ情報を蓄積するプログラムへ渡す引数を指定します。

8 TD コマンド生成機能

TD コマンド生成機能とは，テストデバッグで使用する TD コマンドをコンパイル時にファイルに生成する機能です。この章では，この機能によって生成される TD コマンドの内容について説明します。

8.1 概要

8.1.1 機能の概要

TD コマンド生成機能とは，COBOL ソースファイルを解析して，テストデバッガで使用する TD コマンドをファイルに生成する機能です。

生成される TD コマンドには，中断点情報，プログラムのシミュレーション情報，およびファイルのシミュレーション情報の 3 種類があります。

これらの TD コマンド群は，-TestCmd コンパイラオプションを指定してコンパイルしたときに生成されます。生成された TD コマンド群に，必要に応じて TD コマンドの修正や追加をすることで，TD コマンド格納ファイルを作成できます。

また，TD コマンド生成機能の対象となるプログラムは，コンパイルしたときすべてのプログラムで，S レベル，U レベルのコンパイルエラーのなかった原始プログラムファイルだけです。

8.1.2 TD コマンド格納ファイルの構成と出力先

(1) TD コマンド格納ファイルの構成

TD コマンド格納ファイルは，COBOL ソースファイル単位に生成されます。指定するコンパイラオプションと，生成される TD コマンド格納ファイル（.tdi または.tds）との関係を次に示します。

表 8-1 コンパイラオプションと TD コマンド格納ファイルとの関係

指定するコンパイラオプション	生成されるファイルの名称とその内容
• -TestCmd,Full • -TestCmd,Full -TestCmd,Break • -TestCmd,Full -TestCmd,Sim • -TestCmd,Full -TestCmd,Break -TestCmd,Sim	a.tdi（中断点情報，シミュレーション情報）
-TestCmd,Break	a.tdi（中断点情報）
-TestCmd,Sim	a.tds（シミュレーション情報）
-TestCmd,Break -TestCmd,Sim	a.tdi（中断点情報） a.tds（シミュレーション情報）

注 1

生成されるファイルの名称は，COBOL ソースファイル名を a.cbl としたときの例で示しています。

注 2

-TestCmd,Full コンパイラオプションと同時に指定された-TestCmd,Break，および-TestCmd,Sim コンパイラオプションは無効となります。

注 3

-TestCmd,Break，および-TestCmd,Sim コンパイラオプションを同時に指定したときに出力される.tdi ファイルには，シミュレーション情報を一括入力するための #INCLUDE コマンドが追加になります。

注 4

TD コマンドは COBOL 原始プログラム単位に作成します。

(2) TD コマンド格納ファイルの出力先

TD コマンド格納ファイルの出力先は，環境変数 CBLPIDIR で指定します。環境変数の指定がない場合は，カレントディレクトリに出力されます。

CBLPIDIR 環境変数の指定	中断点情報を含む		シミュレーション情報を含む	
	パス名	ファイル名	パス名	ファイル名
指定あり	環境変数で指定するディレクトリ	a.tdi	環境変数で指定するディレクトリ	a.tds
指定なし	カレントディレクトリ	a.tdi	カレントディレクトリ	a.tds

注

生成されるファイルの名称は，COBOL ソースファイル名を a.cbl としたときの例で示しています。

8.2　中断点情報とシミュレーション情報

TD コマンド生成機能で生成されるそれぞれの TD コマンド群について説明します。

8.2.1　中断点情報

(1)　注釈行（中断点情報の開始）

形式

```
*>*************************************************************************
*>**    SET BREAK TD COMMAND START                                       ***
*>*************************************************************************
*>*************************************************************************
*>**    TEST PROGRAM ==> プログラム名                                    ***
*>*************************************************************************
```

機能

中断点情報の TD コマンドの設定開始を表示します。

生成条件

原始プログラムごとに必ず生成されます。

(2)　SET QUALIFICATION コマンド

形式

```
SET QUALIFICATION(
      { #PROGRAM( '最外側のプログラム名'/'最外側のプログラム名' ) }
      { #CLASS(クラス名)                                         } )
      { #FUNCTION(関数名)                                        }
```

機能

この TD コマンド以降に指定する TD コマンドを最外側のプログラム名，クラス名または関数名で修飾します。内側のプログラム，メソッドについては生成されません。このため，以降の TD コマンドで内部プログラムやメソッドのデータ名や節名を参照するときは，内側のプログラム名，メソッド名による修飾が必要となります。

生成条件

- tdi ファイル上では，最外側のプログラムの SET BREAK コマンドの前ごとに生成されます。
- tds ファイル上では，複数の最外側のプログラムがあるときだけシミュレーション TD コマンドの前ごとに生成されます。

(3) #INCLUDE コマンド

形式

```
#INCLUDE INFILE('ファイル名') NOMESSAGE
```

機能

ファイル名で示すシミュレーション情報の TD コマンド格納ファイルを入力します。

生成条件

SET BREAK コマンドのあとに生成されます。環境変数 CBLPIDIR が指定されている場合，環境変数で指定したディレクトリ名がファイル名に付けられます。

(4) SET BREAK コマンド（節名）

形式

```
SET BREAK PARAGRAPH( [
   { #PROGRAM( '最外側のプログラム名'/'内側のプログラム名' )
   { #CLASS(クラス名/'メソッド名')                          } ] 節名 ) DO
   { #FUNCTION(関数名)
GO
ENDDO
```

機能

プログラムが実行中に一時停止する中断点を指定します。GO コマンドがあるので，実行のトレース情報を知るのに役立ちます。

生成条件

プログラム内で一意となる節名があるときに生成されます。

入れ子プログラムのときは内側のプログラム名で修飾します。

(5) SET BREAK コマンド（文）

形式

```
SET BREAK STATEMENT(xxxxxxx [.n] )  DO
GO
ENDDO
```

機能

プログラムが実行中に一時停止する中断点を指定します。GO コマンドによって，実行のトレース情報を参照できます。

生成条件

条件文（IF 文，EVALUATE 文，PERFORM 文，SEARCH 文，および条件指定のある文）または出口文（GOBACK 文，EXIT PROGRAM 文，EXIT METHOD 文，EXIT FUNCTION 文，および STOP RUN 文）のある行に対して生成されます。

文番号（xxxxxxx）は常に 7 けたとし，7 けたに満たない場合は前に 0 を入れます。

条件文や出口文が 1 行に 2 文以上ある場合，2 文目からは行番号に位置番号（.2，.3，……）が付きます。

(6) 注釈行（中断点情報の終了）

形式

```
*>**    SET BREAK TD COMMAND ENDED                              ***
```

機能

中断点情報の TD コマンドの設定終了を表示します。

生成条件

中断点情報 TD コマンド格納ファイルには，最外側のプログラムごとに必ず生成されます。

(7) SET BREAK コマンド（プログラム名）

形式

```
SET BREAK PROCEDURE( [
   {#PROGRAM( '最外側のプログラム名'/'最外側のプログラム名' )
   {#CLASS(クラス名/'メソッド名')                                    } ] #ENTRY ) DO
   {#FUNCTION(関数名)
  [ ASSIGN DATA(データ名) VALUE(データ値) ]
  [ GO [COVERAGE ] ]
ENDDO
```

機能

最外側のプログラム，メソッドまたは関数の実行を開始する入口点を指定します。また，USING で指定されたデータに対して ASSIGN DATA コマンドでデータ値を設定します。設定するデータ値については，「8.2.3　データ設定値」を参照してください。

生成条件

最外側のプログラム，メソッドまたは関数の PROCEDURE DIVISION に対して生成されます。

"ASSIGN DATA(データ名) VALUE(データ値)"は，USING 指定があるときに生成されます。なお，ASSIGN DATA コマンドは，USING に指定された左側の引数から生成されます。

"GO"は，最後のプログラムにだけ生成されます。また，"COVERAGE"は，-CVInf コンパイラオプションを指定してコンパイルしたときだけ生成されます。

8.2.2　シミュレーション情報

(1)　注釈行（シミュレーション情報の開始）

形式

```
*>*************************************************************************
*>**    SIMULATE TD COMMAND START                                        ***
*>*************************************************************************
```

機能

シミュレーションの TD コマンドの設定開始を表示します。

生成条件

シミュレーション情報を含む TD コマンド格納ファイルには必ず生成されます。

(2)　注釈行（主プログラム）

形式

```
*>*************************************************************************
*>**    MAIN PROGRAM SIMULATE ==> プログラム名                           ***
*>*************************************************************************
```

機能

主プログラムシミュレーションの設定開始を表示します。

生成条件

-SimMain コンパイラオプションを指定したプログラムに対して生成されます。

(3)　SIMULATE MAIN コマンド

形式

```
SIMULATE MAIN(#PROGRAM('最外側のプログラム名'))
          [ ASSIGN DATA(データ名) VALUE(データ値)  ]
                    :
ENDSIMULATE
```

機能

主プログラムをシミュレーションします。

生成条件

-SimMain コンパイラオプションを指定したプログラムに対して生成されます。USING 指定があるときは，データ値が設定されます。なお，USING 指定があるとき生成される ASSIGN DATA コマンドは，指定された左側の引数から生成されます。

設定されるデータ値については，「8.2.3　データ設定値」を参照してください。

(4) 注釈行（副プログラム）

形式

```
*>*************************************************************************
*>**    SUB PROGRAM SIMULATE ==> プログラム名                            ***
*>*************************************************************************
```

機能

副プログラムシミュレーションの設定開始を表示します。

生成条件

-SimSub コンパイラオプションを指定したプログラムを定数指定の CALL 文で呼び出しているときに生成されます。

注意事項

-SimIdent コンパイラオプションを指定した場合，一意名指定の CALL 文に対する注釈行（副プログラム）は，生成されません。

(5) SIMULATE SUB コマンド

形式

```
SIMULATE SUB(#PROGRAM('最外側のプログラム名')
    [ USING(記号名1  [,… ] ) ]
    [ RETURNING(記号名2)]
    ┌ DEFINE                  ┐
    │    01 記号名1           │
    │       :                 │
    │ ENDDEFINE               │
    │ DISPLAY DATA(記号名1)   │
    └       :                 ┘
    [ ASSIGN DATA(記号名2) VALUE(データ値)]
ENDSIMULATE
```

機能

呼び出されるプログラムをシミュレーションします。

生成条件

-SimSub コンパイラオプションを指定した最外側のプログラムを定数指定の CALL 文で呼び出しているときに生成されます。

CALL 文の USING 指定があるときは，データ名の内容を表示します。なお，USING 指定があるとき生成される DISPLAY DATA コマンドは，指定された左側の引数から生成されます。また，CALL 文の RETURNING 指定があるときは，データ値を設定します。このとき生成する記号名は，文字列"PARM"に 8 けたの 1 から始まる数字を付加したものとします。また，設定されるデータ値については，「8.2.3　データ設定値」を参照してください。

注意事項

-SimIdent コンパイラオプションを指定した場合，一意名指定の CALL 文に対する SIMULATE SUB コマンドは，生成されません。-SimIdent コンパイラオプションを指定して副プログラムシミュレー

ションをする場合は，一意名指定の CALL 文で呼び出す副プログラム名に対する SIMULATE SUB コマンドを，TD コマンド格納ファイルに定義して使用してください。

(6) 注釈行（ファイル）

形式

```
*>*****************************************************************************
*>**    FILE   SIMULATE ==> ファイル名                                       ***
*>*****************************************************************************
```

機能

ファイルシミュレーションの設定開始を表示します。

生成条件

ファイルに対して OPEN 文のあるときに生成されます。

(7) SIMULATE FILE コマンド（INPUT／I-O モード）

形式

```
SIMULATE FILE( [
  { #PROGRAM('最外側のプログラム名'/'内側のプログラム名')
    #CLASS(クラス名/#FACTORY)
    #CLASS(クラス名/#OBJECT)
    #FUNCTION(関数名)                                          } ] ファイル名 )
        OPENMODE( { INPUT } ) COUNTER(c1)
                  { IO    }

   IF CONDITION(c1=1)
     [ASSIGN DATA(データ名) VALUE(データ値)
              :                                                ]
   ELSE
        { GO END     }
        { GO INVALID }
   ENDIF
ENDSIMULATE
```

機能

OPEN INPUT または OPEN I-O モードで開かれた入出力ファイルに対するシミュレーションをします。

1 回目のシミュレーションでデータを設定し，2 回目以降のシミュレーションでファイル終了シミュレーションまたは無効キーシミュレーションをします。

生成条件

FD 項で定義したファイルが OPEN INPUT モードで開かれたときは SIMULATE FILE ～ OPENMODE(INPUT)が生成されます。OPEN I-O モードで開かれたときは SIMULATE FILE ～ OPENMODE(IO)コマンドが生成されます。

修飾対象がファクトリ定義のデータ部で定義されているときは#FACTORY で，インスタンス定義のデータ部で定義されているときは#OBJECT で，関数定義のデータ部で定義されているときは #FUNCTION で修飾されます。

また，順アクセスのファイルには GO END が，乱アクセス，動的アクセスのファイルには GO INVALID が設定されます。

(8) SIMULATE FILE コマンド（OUTPUT／EXTEND モード）

形式

```
SIMULATE FILE( [
  { #PROGRAM( '最外側のプログラム名'/'内側のプログラム名' )
    #CLASS(クラス名/#FACTORY)
    #CLASS(クラス名/#OBJECT)
    #FUNCTION(関数名)                                    } ] ファイル名 )
        OPENMODE( { OUTPUT
                  |EXTEND } )
   DISPLAY DATA(データ名)
ENDSIMULATE
```

機能

OPEN OUTPUT または OPEN EXTEND モードで開かれた出力ファイルに対するシミュレーションをします。

生成条件

FD 項で定義したファイルが OPEN OUTPUT モードで開かれたときは，SIMULATE FILE ～ OPENMODE(OUTPUT)コマンドが生成されます。OPEN EXTEND モードで開かれたときは SIMULATE FILE ～ OPENMODE(EXTEND)コマンドが生成されます。

修飾対象がファクトリ定義のデータ部で定義されているときは#FACTORY で，インスタンス定義のデータ部で定義されているときは#OBJECT で，関数定義のデータ部で定義されているときは #FUNCTION で修飾されます。

(9) 注釈行（シミュレーション情報の終了）

形式

```
>***        SIMULATE SUB COMMAND ENDED        ***
```

機能

ファイルシミュレーションの TD コマンドの終了を表示します。

生成条件

シミュレーション情報を含む TD コマンド格納ファイルには必ず生成されます。

8.2.3 データ設定値

TD コマンド格納ファイル中の ASSIGN DATA コマンドで設定されるデータ値を次に示します。

表 8-2　データ設定値

データ属性			設定内容	生成例	
				定義	設定値
英字項目			A で始まる n けたの文字を埋め込む。	PIC A(10)	'ABCDEFGHIJ'
英数字項目				PIC X(4)	'ABCD'
英数字編集項目				PIC XBXBX	'ABC'
数字項目	外部 10 進項目		n けたの数字を埋め込む。小数点の位置も合わせる。P に対しては 0 を埋め込む。	PIC S9V99	1.23
	内部 10 進項目			PIC 9V9 USAGE COMP-3	1.2
	2 進項目			PIC 99PP USAGE COMP	1200
	外部浮動小数点		仮数部に n けたの数字を埋め込む。小数点の位置も合わせる。指数部には E12 を埋め込む。	PIC +9.9E-99	1.2E12
	内部浮動小数点	COMP-1	1.2E34 を埋め込む。	USAGE COMP-1	1.2E34
		COMP-2	2.3E45 を埋め込む。	USAGE COMP-2	2.3E45
ブール項目			n けたの数字を埋め込む。	PIC 1(8) USAGE DISPLAY	B'11111111'
				PIC 1(4) USAGE BIT	B'1111'
数字編集項目			n けたの数字を埋め込む。	PIC ¥¥¥¥¥¥9	1234567
日本語項目			n けたの全角数字を埋め込む。	PIC N(4)	N'１２３４'
日本語編集項目				PIC NBN	N'１２'
英数字集団項目			半角空白を設定する。配下の集団項目，基本項目についても，規則に従って設定する。	SYU1. SYU2. KI1 PIC XX. KI2 PIC 99.	' ' ' ' 'AB' 12
日本語集団項目			全角空白を設定する。配下の集団項目，基本項目についても，規則に従って設定する。	SYU1 GROUP-USAGE NATIONAL. SYU2. KI1 PIC N. KI2 PIC NN.	N'　' N'　' N'１' N'１２'
指標データ項目			設定しない。	USAGE INDEX	-
アドレスデータ項目				USAGE ADDRESS	-

データ属性	設定内容	生成例	
		定義	設定値
ポインタ項目		USAGE POINTER	-
オブジェクト参照データ項目		USAGE OBJECT REFERENCE	-
動的長基本項目	連絡節での定義で，LIMIT 指定がある場合，LIMIT に指定されたけた数を上限値として，基本項目の規則に従って設定する。	PIC X DYNAMIC LENGTH C-STRING LIMIT 4. PIC N DYNAMIC LENGTH C-STRING LIMIT 4.	'ABCD' N'１２３４'
	連絡節での定義で，LIMIT 指定がない場合，1,024 けたが仮定されているものとして，基本項目の規則に従って設定する。※	PIC X DYNAMIC LENGTH C-STRING. PIC N DYNAMIC LENGTH C-STRING.	'ABCDE･･･' N'１２３４･･･'
その他	FILLER 項目は設定しない。	FILLER PIC X(4)	-

(凡例)

－：設定されない。

注 1

埋め込めるけた数の最大は 28 けたです。

注 2

OCCURS 句が付くときは添字(1)だけを設定します。

注※

-SimMain オプションを指定した場合の主プログラムシミュレーションのときだけ生成されます。

8.3　TD コマンドの生成例

次の「被テスト COBOL 原始プログラム」から生成される次のファイルの内容を示します。

- 中断点情報ファイル単独作成のときの中断点情報ファイル
- シミュレーション情報ファイルと同時作成時の中断点情報ファイル
- シミュレーション情報ファイル
- 中断点情報・シミュレーション情報ファイル

図 8-1　被テスト COBOL 原始プログラム（ファイル名：FLI0010.cbl）

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. FLI0010.
000300 ENVIRONMENT DIVISION.
000400 INPUT-OUTPUT SECTION.
000500 FILE-CONTROL.
000600     SELECT URIAGE-FILE
000700          ASSIGN TO SYS001.
000800     SELECT HATTCHUU-FILE
000900          ASSIGN TO SYS002.
001000 DATA DIVISION.
001100 FILE SECTION.
001200 FD  URIAGE-FILE
001300           RECORDING MODE IS F
001400           LABEL RECORD IS STANDARD.
001500 01  URIAGE-REC.
001600     03  URIAGE-CODE       PIC X(10).
001700     03  URIAGE-DATE       PIC X(6).
001800     03  URIAGE-DATA.
001900       05  URIAGE-HINMOKU    OCCURS 50.
002000         07  URIAGE-NAME       PIC X(10).
002100         07  URIAGE-SUURYO     PIC 9(07)  COMP-3.
002200         07  URIAGE-TANKA      PIC 9(07)  COMP-3.
002300         07  URIAGE-GOOKEI     PIC 9(07)  COMP-3.
002400 FD  HATTCHUU-FILE
002500           RECORDING MODE IS F
002600           LABEL RECORD IS STANDARD.
002700 01  HATTCHUU-REC.
002800     03  HATTCHUU-CODE     PIC X(10).
002900     03  HATTCHUU-DATE     PIC X(6).
003000     03  HATTCHUU-DATA.
003100       05  HATTCHUU-HINMOKU  OCCURS 50.
003200         07  HATTCHUU-NAME     PIC X(10).
003300         07  HATTCHUU-SUURYO   PIC 9(07)  COMP-3.
003400         07  HATTCHUU-GENKA    PIC 9(07)  COMP-3.
003500         07  HATTCHUU-GOOKEI   PIC 9(07)  COMP-3.
```

```
003600 WORKING-STORAGE SECTION.
003700 01  I                        PIC S9(4)   COMP.
003800 01  J                        PIC S9(4)   COMP.
003900 01  W-CODE                   PIC X(10).
004000 01  W-HATTCHUU.
004100     03  W-HINMOKU         OCCURS 50.
004200       07  W-NAME             PIC X(10).
004300       07  W-SUURYO           PIC 9(07)  COMP-3.
004400       07  W-GENKA            PIC 9(07)  COMP-3.
004500       07  W-GOOKEI           PIC 9(07)  COMP-3.
004600       07  W-FUSOKU-SW        PIC X(01).
004700 01  HATTCHUU-FLAG            PIC X(01).
004800 LINKAGE SECTION.
004900 01  W-YYMMDD                 PIC X(6).
005000 PROCEDURE DIVISION USING W-YYMMDD.
005100 MAIN      SECTION.
005200 MAIN-START.
005300     OPEN  INPUT  URIAGE-FILE.
005400     OPEN  OUTPUT HATTCHUU-FILE.
005500 URIAGE-FILE-READ.
005600     MOVE  SPACE  TO  HATTCHUU-FLAG.
005700     MOVE  ZERO   TO  J.
005800     READ  URIAGE-FILE  AT END  GO  TO  OWARI.
005900     MOVE  URIAGE-CODE  TO  W-CODE.
006000     MOVE  URIAGE-DATA  TO  W-HATTCHUU.
006100     CALL  'FLI0020'  USING  W-CODE  W-HATTCHUU.
006200     PERFORM  HATTCHUU-CHECK
006300              VARYING I FROM 1 BY 1 UNTIL (I > 50).
006400     IF  HATTCHUU-FLAG = '1'
006500         WRITE  HATTCHUU-REC.
006600     GO  TO  URIAGE-FILE-READ.
006700 OWARI.
006800     CLOSE  URIAGE-FILE.
006900     CLOSE  HATTCHUU-FILE.
007000 MAIN-EXIT.
007100     GOBACK.
007200 HATTCHUU-CHECK   SECTION.
007300 HATTCHUU-CHECK-START.
007400     IF  W-FUSOKU-SW (1) = '1'
007500         MOVE  '1'  TO  HATTCHUU-FLAG
007600         ADD   1    TO  J
007700         MOVE W-CODE     TO HATTCHUU-CODE
007800         MOVE W-YYMMDD   TO HATTCHUU-DATE
007900         MOVE W-NAME (I)   TO HATTCHUU-NAME (J)
008000         MOVE W-SUURYO (I) TO HATTCHUU-SUURYO (J)
008100         MOVE W-GENKA (I)  TO HATTCHUU-GENKA (J)
008200         MOVE W-GOOKEI (I) TO HATTCHUU-GOOKEI (J).
008300 HATTCHUU-CHECK-EXIT.
008400     EXIT.
```

図 8-2　中断点情報ファイル単独作成時の中断点情報ファイル

```
*>*************************************************************************
*>**    SET BREAK TD COMMAND START                                       ***
*>*************************************************************************
*>*************************************************************************
*>**    TEST PROGRAM ==> FLI0010                                         ***
*>*************************************************************************
SET QUALIFICATION(#PROGRAM('FLI0010'/'FLI0010'))
SET BREAK PARAGRAPH(MAIN) DO
 GO
ENDDO
SET BREAK STATEMENT(0005800) DO
 GO
ENDDO
SET BREAK STATEMENT(0006200) DO
 GO
ENDDO
SET BREAK STATEMENT(0006400) DO
 GO
ENDDO
SET BREAK STATEMENT(0007100) DO
 GO
ENDDO
SET BREAK PARAGRAPH(HATTCHUU-CHECK) DO
 GO
ENDDO
SET BREAK STATEMENT(0007400) DO
 GO
ENDDO
*>**    SET BREAK TD COMMAND ENDED                                       ***
SET BREAK PROCEDURE(#PROGRAM('FLI0010'/'FLI0010')#ENTRY) DO
    ASSIGN DATA(W-YYMMDD) VALUE('ABCDEF')
    GO
ENDDO
GO
```

図 8-3　シミュレーション情報ファイルと同時作成の中断点情報ファイル

```
*>*************************************************************************
*>**    SET BREAK TD COMMAND START                                       ***
*>*************************************************************************
*>*************************************************************************
*>**    TEST PROGRAM ==> FLI0010                                         ***
*>*************************************************************************
SET QUALIFICATION(#PROGRAM('FLI0010'/'FLI0010'))
SET BREAK PARAGRAPH(MAIN) DO
 GO
ENDDO
SET BREAK STATEMENT(0005800) DO
 GO
ENDDO
SET BREAK STATEMENT(0006200) DO
 GO
ENDDO
SET BREAK STATEMENT(0006400) DO
 GO
ENDDO
SET BREAK STATEMENT(0007100) DO
 GO
ENDDO
SET BREAK PARAGRAPH(HATTCHUU-CHECK) DO
 GO
ENDDO
SET BREAK STATEMENT(0007400) DO
 GO
ENDDO
*>**    SET BREAK TD COMMAND ENDED                                       ***
SET BREAK PROCEDURE(#PROGRAM('FLI0010'/'FLI0010')#ENTRY) DO
    ASSIGN DATA(W-YYMMDD) VALUE('ABCDEF')
    GO
ENDDO
#INCLUDE INFILE('FLI0010.tds') NOMESSAGE
GO
```

図 8-4　シミュレーション情報ファイル

```
*>*************************************************************************
*>**    SIMULATE TD COMMAND START                                        ***
*>*************************************************************************
*>*************************************************************************
*>**    MAIN PROGRAM SIMULATE ==> FLI0010                                ***
*>*************************************************************************
SIMULATE MAIN(#PROGRAM('FLI0010'))
    ASSIGN DATA(W-YYMMDD) VALUE('ABCDEF')
ENDSIMULATE
*>*************************************************************************
*>**    SUB PROGRAM SIMULATE ==> FLI0020                                 ***
*>*************************************************************************
SIMULATE SUB(#PROGRAM('FLI0020'))
         USING(PARM00000001,PARM00000002)
  DEFINE
    01 PARM00000002
     02 PARM00000003
      03 PARM00000004
      03 PARM00000005
      03 PARM00000006
      03 PARM00000007
      03 PARM00000008
  ENDDEFINE
  DISPLAY DATA(PARM00000001)
  DISPLAY DATA(PARM00000002)
  DISPLAY DATA(PARM00000003(1))
  DISPLAY DATA(PARM00000004(1))
  DISPLAY DATA(PARM00000005(1))
  DISPLAY DATA(PARM00000006(1))
  DISPLAY DATA(PARM00000007(1))
  DISPLAY DATA(PARM00000008(1))
ENDSIMULATE
*>*************************************************************************
*>**    FILE   SIMULATE ==> URIAGE-FILE                                  ***
*>*************************************************************************
SIMULATE FILE(URIAGE-FILE) OPENMODE(INPUT) COUNTER(c1)
  IF CONDITION(c1=1)
    ASSIGN DATA(URIAGE-REC) VALUE(' ')
    ASSIGN DATA(URIAGE-CODE) VALUE('ABCDEFGHIJ')
    ASSIGN DATA(URIAGE-DATE) VALUE('ABCDEF')
    ASSIGN DATA(URIAGE-DATA) VALUE(' ')
    ASSIGN DATA(URIAGE-HINMOKU(1)) VALUE(' ')
    ASSIGN DATA(URIAGE-NAME(1)) VALUE('ABCDEFGHIJ')
    ASSIGN DATA(URIAGE-SUURYO(1)) VALUE(1234567)
    ASSIGN DATA(URIAGE-TANKA(1)) VALUE(1234567)
    ASSIGN DATA(URIAGE-GOOKEI(1)) VALUE(1234567)
  ELSE
    GO END
  ENDIF
ENDSIMULATE
*>*************************************************************************
*>**    FILE   SIMULATE ==> HATTCHUU-FILE                                ***
*>*************************************************************************
SIMULATE FILE(HATTCHUU-FILE) OPENMODE(OUTPUT)
  DISPLAY DATA(HATTCHUU-REC)
ENDSIMULATE
*>**    SIMULATE TD COMMAND ENDED                                        ***
```

図 8-5　中断点情報・シミュレーション情報ファイル

```
*>*****************************************************************************
*>**    SET BREAK TD COMMAND START                                           ***
*>*****************************************************************************
*>*****************************************************************************
*>**    TEST PROGRAM ==> FLI0010                                             ***
*>*****************************************************************************
SET QUALIFICATION(#PROGRAM('FLI0010'/'FLI0010'))
SET BREAK PARAGRAPH(MAIN) DO
  GO
ENDDO
SET BREAK STATEMENT(0005800) DO
  GO
ENDDO
SET BREAK STATEMENT(0006200) DO
  GO
ENDDO
SET BREAK STATEMENT(0006400) DO
  GO
ENDDO
SET BREAK STATEMENT(0007100) DO
  GO
ENDDO
SET BREAK PARAGRAPH(HATTCHUU-CHECK) DO
  GO
ENDDO
SET BREAK STATEMENT(0007400) DO
  GO
ENDDO
*>**    SET BREAK TD COMMAND ENDED                                           ***
*>*****************************************************************************
*>**    SIMULATE TD COMMAND START                                            ***
*>*****************************************************************************
*>*****************************************************************************
*>**    MAIN PROGRAM SIMULATE ==> FLI0010                                    ***
*>*****************************************************************************
SIMULATE MAIN(#PROGRAM('FLI0010'))
    ASSIGN DATA(W-YYMMDD) VALUE('ABCDEF')
ENDSIMULATE
*>*****************************************************************************
*>**    SUB PROGRAM SIMULATE ==> FLI0020                                     ***
*>*****************************************************************************
SIMULATE SUB(#PROGRAM('FLI0020'))
         USING(PARM00000001,PARM00000002)
  DEFINE
    01 PARM00000002
     02 PARM00000003
      03 PARM00000004
      03 PARM00000005
      03 PARM00000006
      03 PARM00000007
      03 PARM00000008
  ENDDEFINE
```

```
  DISPLAY DATA(PARM00000001)
  DISPLAY DATA(PARM00000002)
  DISPLAY DATA(PARM00000003(1))
  DISPLAY DATA(PARM00000004(1))
  DISPLAY DATA(PARM00000005(1))
  DISPLAY DATA(PARM00000006(1))
  DISPLAY DATA(PARM00000007(1))
  DISPLAY DATA(PARM00000008(1))
ENDSIMULATE
*>*************************************************************************
*>**    FILE   SIMULATE ==> URIAGE-FILE                                  ***
*>*************************************************************************
SIMULATE FILE(URIAGE-FILE) OPENMODE(INPUT) COUNTER(c1)
  IF CONDITION(c1=1)
    ASSIGN DATA(URIAGE-REC) VALUE(' ')
    ASSIGN DATA(URIAGE-CODE) VALUE('ABCDEFGHIJ')
    ASSIGN DATA(URIAGE-DATE) VALUE('ABCDEF')
    ASSIGN DATA(URIAGE-DATA) VALUE(' ')
    ASSIGN DATA(URIAGE-HINMOKU(1)) VALUE(' ')
    ASSIGN DATA(URIAGE-NAME(1)) VALUE('ABCDEFGHIJ')
    ASSIGN DATA(URIAGE-SUURYO(1)) VALUE(1234567)
    ASSIGN DATA(URIAGE-TANKA(1)) VALUE(1234567)
    ASSIGN DATA(URIAGE-GOOKEI(1)) VALUE(1234567)
  ELSE
    GO END
  ENDIF
ENDSIMULATE
*>*************************************************************************
*>**    FILE   SIMULATE ==> HATTCHUU-FILE                                ***
*>*************************************************************************
SIMULATE FILE(HATTCHUU-FILE) OPENMODE(OUTPUT)
  DISPLAY DATA(HATTCHUU-REC)
ENDSIMULATE
*>**    SIMULATE TD COMMAND ENDED                                        ***
SET BREAK PROCEDURE(#PROGRAM('FLI0010'/'FLI0010')#ENTRY) DO
    ASSIGN DATA(W-YYMMDD) VALUE('ABCDEF')
  GO
ENDDO
GO
```

9 64bit アプリケーションの開発

この章では，AIX(64) COBOL2002 および Linux(x64) COBOL2002 について説明します。

9.1　COBOL2002 での 64bit アプリケーションの開発について

AIX(64) COBOL2002，Linux(x64) COBOL2002 では，ポインタサイズの 64bit 化に対応しています。

ここでは，AIX(64) COBOL2002 および Linux(x64) COBOL2002 の機能について説明します。

9.1.1　使用できない機能

AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できない機能について説明します。

(1)　機能

AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できない機能を次に示します。

表 9-1　AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できない機能

機能名	OS		説明
	AIX (64)	Linux (x64)	
画面節（SCREEN SECTION）による画面操作	○	×	画面節（SCREEN SECTION）を使用して画面の入出力をします。
画面節（WINDOW SECTION）による画面操作	○	×	画面節（WINDOW SECTION）を使用して画面の入出力をします。
XMAP3 を使用した書式印刷機能	×	×	書式と行データを重ね合わせる印刷（書式オーバレイ印刷）や，印刷制御付きの行データを印刷します。
リモートファイルアクセス機能	×	×	接続されているほかのワークステーションや PC 上にある，ISAM を使用する索引編成ファイルにアクセスします。
通信節による画面操作	×	×	ディスプレイとの間で画面データを送受信したり，プリンタに帳票データを送信したりします。
データベース操作機能（ODBC インタフェース）	×	○	SQL 埋め込み COBOL プログラムで，データベースにアクセスできます。
CGI プログラム作成支援機能	×	×	COBOL2002 で作成したプログラムを CGI プログラムとして利用するための機能です。

（凡例）

○：使用できる

×：使用できない

(2)　サービスルーチン

AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できないサービスルーチンを次に示します。

表 9–2　AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できないサービスルーチン

サービスルーチン名	OS		説明
	AIX (64)	Linux (x64)	
CBLADDPAIR	×	×	CGI リストの最後に「名前」と「値」の対を追加します。
CBLCGIINIT	×	×	受け取ったフォーム情報から CGI リストを作成します。
CBLCGITRACE	×	×	CGI プログラムの作成を支援します。サービスルーチンのトレース情報をファイルに出力します。
CBLCONVERTTEXT	×	×	テキスト文字列を実体参照形式に変換し，出力します。
CBLCREATELIST	×	×	CGI リストを新たに作成します。
CBLDELETEPAIR	×	×	CGI リストの現在のポイント位置の「名前」と「値」の対を削除します。
CBLDESTROYLIST	×	×	CGI リストを削除し，領域を解放します。
CBLDISPLAYTEXT	×	×	テキスト文字列を出力します。
CBLENDREPEAT	×	×	CGI リストに，HTML 拡張言語の REPEAT で終端を認識する終端インジケータを追加します。
CBLFILLTEMPLATE	×	×	HTML テンプレートをインタプリットし，動的な Web ページを出力します。
CBLFINDNEXTPAIR	×	×	CGI リストの，次のポイント位置から「名前」をキーにして検索し，「値」を取得します。
CBLFINDPAIR	×	×	CGI リストの先頭ポイント位置から「名前」で検索し，「値」を取得します。
CBLGETENV	×	×	環境変数の値を取得します。
CBLGETPAIR	×	×	CGI リストの現在のポイント位置から「名前」と「値」の対を取得します。
CBLGETPAIRNEXT	×	×	CGI リストの現在のポイント位置から「名前」と「値」の対を取得し，ポイント位置を進めます。
CBLHTMLBEGIN	×	×	HTML の先頭部分を出力します。
CBLHTMLEND	×	×	HTML の終端部分を出力します。
CBLLISTCOUNT	×	×	CGI リストから「名前」と「値」の対の数を取得します。
CBLPRINTENV	×	×	CGI 環境変数の値を HTML 形式で出力します。
CBLPRINTLIST	×	×	CGI リストの内容を HTML 形式で出力します。
CBLSENDERROR	×	×	エラーメッセージを HTML 形式で出力します。
JCPOPUP	○	×	表形式のデータ項目を主画面とは別の画面に表示し，選ばれたブロック番号をインタフェース領域に格納します。

（凡例）

○：使用できる

×：使用できない

9.1.2　COBOL2002 での 64bit アプリケーション固有の言語仕様

AIX(64) COBOL2002，Linux(x64) COBOL2002 では，ポインタサイズの 64bit 化に対応しました。そのため，AIX(64) COBOL2002 および Linux(x64) COBOL2002 の言語仕様にも一部変更があります。

AIX(64) COBOL2002 および Linux(x64) COBOL2002 固有の言語仕様について説明します。

(1)　アドレス系データを表現するデータ項目

長さが 8 バイトになるアドレス系データを表現するデータ項目を，次に示します。これらの項目の自然な境界は 8 バイトです。SYNCHRONIZED 句を指定してけた詰めする場合は，8 バイト境界になります。

- アドレス名
- アドレスデータ項目
- ポインタ項目
- 指標名
- 指標データ項目
- オブジェクト参照データ項目
- 既定義オブジェクト参照
- ADDRESS OF 指定で指定した一意名のアドレス
- 連絡節での BY REFERENCE 指定のデータ項目

SYNCHRONIZED 句については，マニュアル「COBOL2002　言語　標準仕様編」の「SYNCHRONIZED 句」を参照してください。

けた詰めについては，マニュアル「COBOL2002　言語　標準仕様編」の「実行用コードの効率を高めるための項目のけた詰め」を参照してください。

(2)　長さを返す組み込み関数の戻り値

返却値のサイズが 8 バイト 2 進になる組み込み関数を，次に示します。

- LENGTH 関数
- LENGTH OF 指定で生成されるデータ領域
- COUNT-CHAR 関数
- LENGTH-OF-SUBSTRING 関数

LENGTH 関数については，マニュアル「COBOL2002　言語　標準仕様編」の「LENGTH 関数」参照してください。

COUNT-CHAR 関数については，マニュアル「COBOL2002　言語　拡張仕様編」の「COUNT-CHAR 関数」を参照してください。

LENGTH-OF-SUBSTRING 関数については，マニュアル「COBOL2002　言語　拡張仕様編」の「LENGTH-OF-SUBSTRING 関数」を参照してください。

9.1.3　COBOL2002 での 64bit アプリケーション固有の機能仕様

64bit アプリケーション固有の機能仕様について説明します。

(1)　サービスルーチンの引数におけるハンドル項目

次に示すサービスルーチンについて，引数におけるハンドル項目のサイズは 8 バイト 2 進となります。

- バイトストリーム入出力サービスルーチン

(2)　インタフェース領域中のアドレス格納領域

インタフェース領域中のアドレス格納領域が 8 バイトになるサービスルーチンを，次に示します。

- COBOL 入出力サービスルーチン

なお，AIX(64) COBOL2002 および Linux(x64) COBOL2002 では，アドレス格納領域が自然な境界の 8 バイトになるように，アドレス格納領域の直前に明示的な境界の調整領域を追加しました。領域については，マニュアル「COBOL2002　使用の手引　手引編」の「インタフェース領域の形式」を参照してください。

(3)　併合処理のメモリサイズについて

併合処理のメモリサイズについては，アドレス格納領域として使うサイズが AIX(64) COBOL2002 および Linux(x64) COBOL2002 では 8 バイトになります。併合処理で使用するメモリサイズの計算式については，マニュアル「COBOL2002　使用の手引　手引編」の「併合処理のメモリサイズ」を参照してください。

9.2 COBOL ソースの作成とコンパイル

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での COBOL ソースのコンパイル方法は，AIX(32) COBOL2002 および Linux(x86) COBOL2002 と同じです。しかし，AIX(64) COBOL2002 および Linux(x64) COBOL2002 と，AIX(32) COBOL2002 および Linux(x86) COBOL2002 では使用できるコンパイラオプションの一部が違います。

AIX(64) COBOL2002 および Linux(x64) COBOL2002 が使用できるコンパイラオプションについては，マニュアル「COBOL2002　使用の手引　手引編」の「コンパイラオプションの一覧」を参照してください。

9.2.1 定義別のコンパイル方法とリポジトリファイル

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での定義別のコンパイル方法とリポジトリファイルは，AIX(32) COBOL2002 および Linux(x86) COBOL2002 と同じです。

定義別のコンパイル方法とリポジトリファイルについては，マニュアル「COBOL2002　使用の手引　手引編」のプログラム定義のほかに，関数定義，クラス定義，およびインタフェース定義など，さまざまな定義の情報を保管するリポジトリファイルや定義別のコンパイル方法について説明している「定義別のコンパイル方法とリポジトリファイル」を参照してください。

9.3　プログラムの実行

ここでは，プログラムの実行について説明します。

9.3.1　プログラムの実行環境の設定

プログラムの実行環境の設定について説明します。

(1)　実行時環境変数の設定方法

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での実行時環境変数の設定方法は，AIX(32) COBOL2002 および Linux(x86) COBOL2002 と同じです。詳細については，マニュアル「COBOL2002　使用の手引　手引編」の「実行時環境変数の設定方法」を参照してください。

(2)　実行時環境変数の一覧

AIX(64) COBOL2002 および Linux(x64) COBOL2002 と，AIX(32) COBOL2002 および Linux(x86) COBOL2002 では使用できる実行時環境変数の一部が違います。AIX(64) COBOL2002 および Linux(x64) COBOL2002 が使用できる実行時環境変数については，マニュアル「COBOL2002　使用の手引　手引編」の「実行時環境変数の一覧」を参照してください。

9.3.2　プログラムの実行時の注意事項

AIX(64) COBOL2002 および Linux(x64) COBOL2002 でのプログラム実行時の注意事項を次に示します。

AIX(64)の場合

AIX(64) COBOL2002 で作成したアプリケーションから CALL 文を使用して，AIX(32) COBOL2002 で作成した実行可能ファイルを呼び出せます※。ただし，AIX(64) COBOL2002 で作成したアプリケーションからは AIX(32) COBOL2002 で作成した共用ライブラリに含まれるプログラムは呼び出せません。

注※

AIX(64) COBOL2002 と AIX(32) COBOL2002 がインストールされた環境で実行してください。

Linux(x64)の場合

Linux(x64) COBOL2002 で作成したアプリケーションから CALL 文を使用して，Linux(x86) COBOL2002 で作成した実行可能ファイルを呼び出せます※。ただし，Linux(x64) COBOL2002 で作成したアプリケーションからは Linux(x86) COBOL2002 で作成した共用ライブラリに含まれるプログラムは呼び出せません。

注※

Linux(x64) COBOL2002 と Linux(x86) COBOL2002 がインストールされた環境で実行してください。

9.4　実行可能ファイルと共用ライブラリの作成

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での実行可能ファイルと共用ライブラリの作成の詳細については，マニュアル「COBOL2002　使用の手引　手引編」の，プログラムをコンパイルして作成したオブジェクトファイルをリンクして，実行可能ファイルや共用ライブラリを作成する方法について説明している「実行可能ファイルと共用ライブラリの作成」を参照してください。

9.4.1　実行可能ファイルと共用ライブラリの作成時の注意事項

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での実行可能ファイルと共用ライブラリの作成時の注意事項を次に示します。

- 他システムの COBOL2002 で作成したファイル（オブジェクト，共用ライブラリ，アーカイブ）をリンクすることはできません。

10 COBOL2002 での UTF-8 ロケールの対応

この章では，Linux(x86) COBOL2002，Linux(x64) COBOL2002 を使用する場合の注意事項について説明します。

10.1　UTF-8 環境での COBOL2002 について

ここでは，Linux(x86) COBOL2002，Linux(x64) COBOL2002 の動作環境，使用できる機能，使用できるサービスルーチンについて説明します。

10.1.1　動作環境

Linux(x86) COBOL2002，Linux(x64) COBOL2002 では，UTF-8 ロケールだけをサポートしています。

UTF-8 ロケールで動作する場合，Unicode 機能を前提機能としています。このため，使用できる機能，日本語項目，および多バイト文字の使用範囲に制限があります。

UTF-8 ロケールの詳細については，マニュアル「COBOL2002　使用の手引　手引編」の付録「COBOL で使用する文字集合」の「Unicode の場合」の「UTF-8 の動作環境」を参照してください。

Unicode 機能の詳細については，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」を参照してください。

10.1.2　使用できる機能

Linux(x86) COBOL2002，Linux(x64) COBOL2002 で使用できる機能については，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」の「Unicode に対応する機能」を参照してください。

10.1.3　使用できるサービスルーチン

使用できるサービスルーチンに制限があります。

使用できるサービスルーチンについては，マニュアル「COBOL2002　使用の手引　手引編」の「サービスルーチン」を参照してください。

10.2　コンパイル時の注意事項

Linux(x86) COBOL2002，Linux(x64) COBOL2002 でコンパイルするときの注意事項を次に示します。

- シフト JIS で記述された COBOL ソースプログラムだけがコンパイルできます。UTF-8 で記述された COBOL ソースはコンパイルできません。
- コンパイル時には，-UniObjGen コンパイラオプションおよび環境変数 CBLSRCENCODING に SJIS を指定しなければなりません。

  -UniObjGen コンパイラオプションについては，マニュアル「COBOL2002　使用の手引　手引編」の「コンパイラオプション」の「その他の設定」の「-UniObjGen オプション」を参照してください。コンパイラ環境変数 CBLSRCENCODING については，マニュアル「COBOL2002　使用の手引　手引編」の「コンパイラ環境変数」の「コンパイラ環境変数の詳細」の「CBLSRCENCODING」を参照してください。
- ccbl コマンドは使用できません。ccbl2002 コマンドを使用してください。
- 日本語文字，半角かたかななど，UTF-8 で多バイトとなる文字は，言語仕様上，使用できる個所に制限があります。詳細は，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」の「Unicode 機能での制限事項」の「コンパイル時の制限事項」を参照してください。
- 環境変数名，環境変数に設定する値，コマンドライン引数，コンパイル時に使用するファイル名，ディレクトリの絶対パス名，およびコンパイルを実行するカレントディレクトリの絶対パス名には，UTF-8 で多バイトとなる文字は使用できません。詳細は，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」の「Unicode 機能での制限事項」の「コンパイル時の制限事項」を参照してください。
- Linux(x86) COBOL2002，Linux(x64) COBOL2002 で使用できるコンパイラオプションについては，マニュアル「COBOL2002　使用の手引　手引編」の「コンパイラオプション」の「コンパイラオプションの一覧」を参照してください。また，-UniObjGen コンパイラオプションと同時に指定した場合に無効となるオプションについては，マニュアル「COBOL2002　使用の手引　手引編」の「コンパイラオプション」の「コンパイラオプションの優先順位」の「オプション間の優先順位」の「オプションを指定すると，ほかのオプションが無効となる場合」を参照してください。
- Linux(x86) COBOL2002，Linux(x64) COBOL2002 で使用できるコンパイラ環境変数については，マニュアル「COBOL2002　使用の手引　手引編」の「コンパイラ環境変数」の「コンパイラ環境変数の一覧」を参照してください。
- コンパイルリストはシフト JIS で出力されます。詳細は，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」の「入出力情報と取り扱う文字コード」を参照してください。
- コンソールに出力するエラーメッセージは UTF-8 で出力されます。詳細は，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」の「入出力情報と取り扱う文字コード」を参照してください。

10.3　実行可能ファイルと共用ライブラリの作成時の注意事項

Linux(x86) COBOL2002，Linux(x64) COBOL2002 で，実行可能ファイルと共用ライブラリを作成するときの注意事項を次に示します。

- Linux(x86) COBOL85 で作成したファイル（オブジェクト，共用ライブラリ，アーカイブ）をリンクすることはできません。COBOL2002 環境下で再コンパイルが必要です。
- 実行可能ファイル名および共用ライブラリ名に UTF-8 で多バイトとなる文字を使用しないでください。使用した場合の動作は保証しません。

10.4　実行時の注意事項

Linux(x86) COBOL2002，Linux(x64) COBOL2002 の実行時の注意事項を次に示します。

- Linux(x86) COBOL85 で作成した実行可能ファイル，または共用ライブラリは，COBOL2002 環境下で実行および呼び出すことはできません。COBOL2002 環境下で，COBOL85 で作成したプログラムが実行または呼び出された場合，KCCC0117R-S のメッセージが出力されます。
- 実行時には，環境変数 CBLLANG に UNICODE を指定しなければなりません。環境変数 CBLLANG などの Unicode 機能で使用する環境変数については，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」の「Unicode 機能の詳細」の「実行」を参照してください。
- 日本語文字，半角かたかななど，UTF-8 で多バイトとなる文字は，ファイル名に使用できないなどの制限があります。詳細は，マニュアル「COBOL2002　使用の手引　手引編」の「Unicode 機能」の「Unicode 機能での制限事項」の「実行時の制限事項」を参照してください。

10.5　テストデバッガ，およびカバレージ使用時の注意事項

Linux(x86) COBOL2002，Linux(x64) COBOL2002 のテストデバッガ，およびカバレージ使用時の注意事項を次に示します。

- テストデバッガ，またはカバレージを使用するときは，環境変数 CBLLANG に UNICODE，および CBLSRCENCODING に SJIS を指定しなければなりません。これらの環境変数の詳細については，「1.4.1　機能の概要」の「(2)　コンパイラオプションと環境変数について」を参照してください。
- 日本語文字，半角かたかななど，UTF-8 で多バイトとなる文字は，ファイル名に使用できないなどの制限があります。詳細は，「1.4.3　その他の注意事項」の「(4)　UTF-8 環境下での注意事項（Linux の場合）」を参照してください。
- 日本語文字など，シフト JIS と UTF-8 の間で変換できない文字は，TD コマンドで使用できないなどの制限があります。詳細は，「1.4.3　その他の注意事項」の「(4)　UTF-8 環境下での注意事項（Linux の場合）」を参照してください。

付録

付録 A　TD コマンドとサブコマンドの違い

COBOL2002 の TD コマンドと，COBOL85 のサブコマンドの違いを表 A-1 に示します。

表 A-1　TD コマンドとサブコマンドの違い

機能	COBOL2002 TD コマンド名	COBOL85 サブコマンド名
中断点の設定	SET BREAK	sb
中断点の解除	RESET BREAK	rb
中断点の表示	DISPLAY BREAK	db
データ値の監視の開始	SET WATCH	sw
データ値の監視の終了	RESET WATCH	rw
連続実行	GO	g
ステップイン	STEP IN	gs
ステップオーバー	STEP OVER	gp
ジャンプ	STEP TO	gs/gp
実行の強制終了	STOP	stop
トレース表示の設定	SET TRACE	st/mt
トレース表示の解除	RESET TRACE	st/mt
フロー情報の蓄積	SET FLOW	flow
フロー情報の蓄積中止	RESET FLOW	flow
フロー情報の表示	DISPLAY FLOW	df
現在位置の表示	DISPLAY POINT	dp
データ値の表示	DISPLAY DATA	dv
オブジェクトデータ値の表示	DISPLAY OBJEECT	dcl
	DISPLAY FACTORY	
データ値の代入	ASSIGN DATA	sv
データ値の比較	IF	if/else
領域の確保	ALLOCATE AREA	geta
領域の解放	FREE AREA	freea
主プログラムシミュレーション	SIMULATE MAIN	simm～esim
副プログラムシミュレーション	SIMULATE SUB	sims～esim
ファイルシミュレーション	SIMULATE FILE	simf～esim

機能		COBOL2002 TD コマンド名	COBOL85 サブコマンド名
入出力文選択指定		SELECT ACTION	－
ファイル入出力条件のシミュレーション	ファイル終了条件	GO END	ge
	ページ終了条件	GO EOP	geop
	無効キー	GO INVALID	gi
	入出力誤り	GO ERROR	gue
DC シミュレーション		SIMULATE DC	simdc～esim
繰り返し指定		REPEAT	rpt～erpt
実行するテストケースの指定		TEST	test～et
テストケースの手続きの設定		CASE	case～ec
ケースコードの指定		ASSIGN CASECODE	ccd
テストデバッガの終了		QUIT	q
テスト結果への注釈の表示		DISPLAY COMMENT	dc
翻訳単位，ソース要素の変更		SET QUALIFICATION	sq
翻訳単位指定，ソース要素指定の解除		RESET QUALIFICATION	rq
TD コマンド格納ファイルの TD コマンドの実行		#INCLUDE	i
等価規則の変更		#OPTION	－
結果出力先の指定		SET PRINT	spr
結果出力先の指定の解除		RESET PRINT	spr
ログ出力ファイルの設定		SET LOG	log
ログ出力ファイルの解除		RESET LOG	log
装置名へのファイルの割り当て		ASSIGN DEVICE	asgn
UNIX コマンドの実行		!	!
アドレスの取得		ASSIGN ADDRESS	－

付録 B　制限値，限界値

付録 B.1　テストデバッガの制限値，限界値

テストデバッガの制限値，限界値を次に示します。

表 B-1　テストデバッガの制限値，限界値

分類	項目	制限値，限界値
テストデバッガ	表示するデータのサイズ	32,767 バイト
	1 行に入力できる文字列長※	511 バイト
TD コマンド	1 行に記述できる TD コマンド文字列長	11,264 バイト以内
	文番号指定の行番号に指定できる整数値	7 けた以内
	文番号指定の位置番号に指定できる整数値	2 けた以内
	添字に指定する次元数	7
	添字に指定する整数定数	10 けた以内
	添字の範囲指定で指定する整数定数	10 けた以内
	部分参照の最左端位置に指定する整数定数	10 けた以内
	部分参照の長さに指定する整数定数	10 けた以内
	拡張 16 進定数の文字列長	160 文字
	TD 利用者定義語であるカウンタ変数名，記号名，監視識別子，ケース識別子の文字列長	31 文字
	カウンタ変数に設定される値	1～2,147,483,647
	SET BREAK コマンドの SKIP オペランドに指定する整数値	1～2,147,483,647
	SET TRACE コマンドの TOSTEP オペランドに指定する整数値	1～2,147,483,647
	SET FLOW コマンドの STACK オペランドに指定する整数値	1～500
	データの値表示の表示形式	見出し域 12 バイト 値域 80 バイト
	記号名構造指定のレベル番号に指定する整数値	2 けた以内
	REPEAT コマンドの TIMES オペランドに指定する整数値	1～2,147,483,647
	CASE コマンドの TOSTEP オペランドに指定する整数値	1～2,147,483,647
	ASSIGN CASECODE コマンドで設定できる値	0～2,147,483,647
	TD コマンド群のネスト数	128 以内

分類	項目	制限値，限界値
カバレージ	カバレージ情報の表示形式：対象総数，実行済数，未実行数	8 けた
	カバレージ情報の表示形式：変更回数	0～32,767
	カバレージ情報の表示形式：テスト回数	0～32,766
	カバレージ情報の表示形式：実行回数	0～2,147,483,647

注※

ラインモードの場合です。記載の値を超える文字列を端末から入力したときの動作は保証しません。

付録 C　テストデバッガの例題

付録 C.1　例題プログラム

(1)　主プログラム（AVERAGE）

入力ファイル IN-FILE から English，Japanese，Mathematics の得点を入力し，3 教科の平均点を表示するプログラムです。

```
000010 IDENTIFICATION  DIVISION.
000020 PROGRAM-ID.     AVERAGE.
000030
000040 ENVIRONMENT     DIVISION.
000050 INPUT-OUTPUT    SECTION.
000060 FILE-CONTROL.
000070      SELECT     IN-FILE  ASSIGN TO SYS001
000080        ORGANIZATION IS LINE SEQUENTIAL.
000090      SELECT     OUT-FILE ASSIGN TO SYS002.
000100 DATA            DIVISION.
000110 FILE            SECTION.
000120 FD   IN-FILE.
000130 01   IN-REC.
000140  02   IN-POINT  PIC 9(9).
000150  02   FILLER    PIC X.
000160 FD   OUT-FILE   LABEL RECORD OMITTED.
000170 01   OUT-REC    PIC X(80).
000180 WORKING-STORAGE SECTION.
000190 01  PR-REC.
000200  02  FILLER     PIC X(13) VALUE SPACE.
000210  02  E-POINT    PIC Z(3)9.
000220  02  FILLER     PIC X(6)  VALUE SPACE.
000230  02  K-POINT    PIC Z(3)9.
000240  02  FILLER     PIC X(8)  VALUE SPACE.
000250  02  S-POINT    PIC Z(3)9.
000260  02  FILLER     PIC X(5)  VALUE SPACE.
000270  02  A-POINT    PIC Z(3)9.
000280  02  FILLER     PIC X(31) VALUE SPACE.
000290 01  TITLE.
000300  02  FILLER     PIC X(10) VALUE SPACE.
000310  02  T-REC      PIC X(40)
000320              VALUE 'English  Japanese  Mathematics  Average'.
000330  02  FILLER     PIC X(30) VALUE SPACE.
000340 01  ERR-MSG.
000350  02  FILLER     PIC X(10)  VALUE SPACE.
000360  02  ERMSG      PIC X(34)
000370              VALUE '** Error:  Input data is mistaken.'.
000380  02  ERMSG2     PIC X(18) VALUE ' ==> error data:'.
000390  02  ER-REC.
000400   03  ER-DATA   PIC 9(9).
000410  02  FILLER     PIC X(9)  VALUE SPACE.
000420 01  NO-MSG.
000430  02  FILLER     PIC X(10) VALUE SPACE.
```

```
000440  02  NMSG       PIC X(40)
000450              VALUE '** Information:  There is no input data.'.
000460  02  FILLER     PIC X(30) VALUE SPACE.
000470 77  ENDM       PIC X(3).
000480 88  END-OF-FIL  VALUE 'EOF'.
000490 01  WORK-TBL.
000500  02  W-TBL      OCCURS 20.
000510   03  W-REC.
000520    04  WE-P     PIC 9(3).
000530    04  WK-P     PIC 9(3).
000540    04  WS-P     PIC 9(3).
000550   03  WA-P      PIC 9(4).
000560   03  R-CODE    PIC 9.
000570 77  I           PIC 9(3).
000580 77  J           PIC 9(3).
000590*
000600 PROCEDURE       DIVISION.
000610 MAIN-PROCESS    SECTION.
000620 OPEN-FILE.
000630        OPEN INPUT  IN-FILE
000640             OUTPUT OUT-FILE.
000650 MAKING-WTBL.
000660        READ IN-FILE
000670             AT END MOVE 'EOF' TO ENDM.
000680        PERFORM VARYING I FROM 1 BY 1 UNTIL END-OF-FIL
000690          MOVE IN-POINT TO W-REC(I)
000700          READ IN-FILE AT END MOVE 'EOF' TO ENDM
000710          END-READ
000720         END-PERFORM.
000730 ZERO-CHECK.
000740        COMPUTE I = I - 1.
000750        IF I = 0
000760          WRITE OUT-REC FROM NO-MSG AFTER 1
000770          GO TO CLOSE-FILE
000780        ELSE
000790          CONTINUE
000800        END-IF.
000810 CALCULATE.
000820        CALL 'SUBAV' USING WORK-TBL I.
000830 PRINTING-DATA.
000840        PERFORM  DATA-PUT.
000850 CLOSE-FILE.
000860       CLOSE IN-FILE OUT-FILE.
000870       STOP RUN.
000880*
000890 DATA-PUT        SECTION.
000900 PRINTING-TITLE.
000910        WRITE OUT-REC FROM TITLE AFTER 1.
000920 PRINTING-AVERAGE.
000930        PERFORM VARYING J FROM 1 BY 1 UNTIL J > I
000940          IF R-CODE(J) = 8
000950            MOVE W-REC(J) TO ER-REC
000960            WRITE OUT-REC FROM ERR-MSG AFTER 1
000970          ELSE
000980            MOVE WE-P(J) TO E-POINT
000990            MOVE WK-P(J) TO K-POINT
001000            MOVE WS-P(J) TO S-POINT
001010            MOVE WA-P(J) TO A-POINT
```

```
001020            WRITE OUT-REC FROM PR-REC AFTER 1
001030          END-IF
001040        END-PERFORM.
001050 END PROGRAM AVERAGE.
```

(2) 副プログラム（SUBAV）

平均点を計算するプログラムです。

```
000010 IDENTIFICATION  DIVISION.
000020 PROGRAM-ID.     SUBAV.
000030 ENVIRONMENT     DIVISION.
000040 DATA            DIVISION.
000050 WORKING-STORAGE SECTION.
000060 77  I            PIC S9(8) COMP.
000070 77  J            PIC S9(8) COMP.
000080 77  K            PIC S9(8) COMP.
000090 77  L            PIC S9(8) COMP.
000100 77  M            PIC S9(8) COMP.
000110 77  MM           PIC S9(8) COMP.
000120 77  CHGSW        PIC X     VALUE LOW-VALUE.
000130 88  NO-CHANGE    VALUE HIGH-VALUE.
000140 88  ALL-CHANGE   VALUE LOW-VALUE.
000150 01  SORT-TBL.
000160  02  S-TBL       OCCURS 20 INDEXED BY DATA-INDEX
000170                                 COMP-INDEX1
000180                                 COMP-INDEX2.
000190   03  S-P.
000200    04  FILLER    PIC 9(9).
000210   03  SA-P        PIC 9(4).
000220   03  FILLER     PIC 9.
000230 01  SW-TBL.
000240  02  SORT-ITEM   INDEX  OCCURS 20 INDEXED BY SORT-INDEX.
000250 LINKAGE         SECTION.
000260 01  WORK-TBL.
000270  02  W-TBL       OCCURS 20.
000280   03  W-REC.
000290    04  WE-P      PIC 9(3).
000300    04  WK-P      PIC 9(3).
000310    04  WS-P      PIC 9(3).
000320   03  WA-P        PIC 9(4).
000330   03  R-CODE     PIC 9.
000340 77  KOSU         PIC 9(3).
000350*
000360 PROCEDURE       DIVISION USING WORK-TBL KOSU.
000370 MAIN-PROCESS    SECTION.
000380 DATA-CHEK-AND-COMPUTE.
000390        PERFORM VARYING I FROM 1 BY 1 UNTIL I > KOSU
000400          MOVE ZERO TO WA-P(I)
000410          IF WE-P(I) > 100 OR
000420             WK-P(I) > 100 OR
000430             WS-P(I) > 100
000440            MOVE 8 TO R-CODE(I)
000450          ELSE
000460            MOVE 0 TO R-CODE(I)
000470            COMPUTE WA-P(I) ROUNDED
```

```
000480             = (WE-P(I) + WK-P(I) + WS-P(I)) / 3
000490           END-IF
000500           MOVE W-TBL(I) TO S-TBL(I)
000510         END-PERFORM.
000520         PERFORM DATA-SORT.
000530         EXIT PROGRAM.
000540*
000550 DATA-SORT       SECTION.
000560         IF KOSU = 1 THEN
000570           GO TO DATA-SORT-EXIT
000580         ELSE
000590           CONTINUE
000600         END-IF.
000610         PERFORM VARYING I FROM 1 BY 1 UNTIL I > 20
000620           SET DATA-INDEX TO I
000630           SET SORT-ITEM(I) TO DATA-INDEX
000640         END-PERFORM.
000650         MOVE 2 TO M.
000660         PERFORM UNTIL M > KOSU
000670           COMPUTE M = M * 2
000680         END-PERFORM.
000690         COMPUTE M = M / 2 - 1.
000700         PERFORM UNTIL M < 1
000710           PERFORM VARYING I FROM 1 BY 1 UNTIL I > M
000720             PERFORM VARYING J FROM I BY M UNTIL J + M > KOSU
000730               COMPUTE K = J + M
000740               SET COMP-INDEX1 TO SORT-ITEM(J)
000750               SET COMP-INDEX2 TO SORT-ITEM(K)
000760               IF SA-P(COMP-INDEX1) < SA-P(COMP-INDEX2) THEN
000770                 SET SORT-ITEM(K) TO COMP-INDEX1
000780                 MOVE LOW-VALUE TO CHGSW
000790                 COMPUTE MM = - M
000800                 PERFORM VARYING K FROM J BY MM
000810                         UNTIL K - M < I OR NO-CHANGE
000820                   COMPUTE L = K - M
000830                   SET COMP-INDEX1 TO SORT-ITEM(L)
000840                   IF SA-P(COMP-INDEX1) < SA-P(COMP-INDEX2) THEN
000850                     SET SORT-ITEM(K) TO COMP-INDEX1
000860                   ELSE
000870                     SET SORT-ITEM(K) TO COMP-INDEX2
000880                     MOVE HIGH-VALUE TO CHGSW
000890                   END-IF
000900                 END-PERFORM
000910                 IF ALL-CHANGE THEN
000920                   IF J = I THEN
000930                     SET SORT-ITEM(J) TO COMP-INDEX2
000940                   ELSE
000950                     SET SORT-ITEM(L) TO COMP-INDEX2
000960                   END-IF
000970                 ELSE
000980                   CONTINUE
000990                 END-IF
001000               ELSE
001010                 CONTINUE
001020               END-IF
001030             END-PERFORM
001040           END-PERFORM
001050           COMPUTE M = M / 2
```

```
001060         END-PERFORM.
001070         PERFORM VARYING J FROM 1 BY 1 UNTIL J > KOSU
001080           SET DATA-INDEX TO SORT-ITEM(J)
001090           MOVE S-TBL(DATA-INDEX) TO W-TBL(J)
001100         END-PERFORM.
001110 DATA-SORT-EXIT.
001120         EXIT.
001130 END PROGRAM SUBAV.
```

(3) 実行結果

```
 English  Japanese  Mathematics  Average
     100       100          100      100
      78        89           99       89
      90        90           80       87
      74        85           96       85
      75        48           86       70
     100         0          100       67
      32        65           98       65
      30        70           80       60
      45        56           78       60
      68        58           27       51
      46        85           21       51
      40        50           60       50
      12        45           78       45
      14        25           36       25
      11        22           33       22
      10        20           30       20
** Error:  Input data is mistaken. ==> error data:  200400300
** Error:  Input data is mistaken. ==> error data:  110450900
** Error:  Input data is mistaken. ==> error data:  111222333
       0         0            0        0
```

付録 C.2 TD コマンドとその結果

(1) TD コマンドの例

入力ファイル（IN-FILE）と副プログラム（SUBAV）が未完成な場合，擬似実行によって主プログラムをテストします。

図 C-1　テストケース 1（TD コマンドの例）

```
KCCC0300T-I テストデバッグを開始します。 YYYY-MM-DD HH:MM:SS
#INCLUDE INFILE(INCAV1)
*>*************************************************************************
*>      Input File 及び SUBAV が未作成                                     *
*>*************************************************************************
*>-----------------------------------------------------------------------*
*>      入力データがない場合     テスト NO  --> 001                       *
*>-----------------------------------------------------------------------*
SET QUALIFICATION(#PROGRAM(AVERAGE/AVERAGE))
SIMULATE FILE(IN-FILE) OPENMODE(INPUT)          *> READ文の擬似実行手続き
   GO END
ENDSIMULATE
SIMULATE FILE(OUT-FILE) OPENMODE(OUTPUT)        *> WRITE文の擬似実行手続き
ENDSIMULATE
GO COVERAGE
KCCC1200T-I ユーザプログラムの実行を開始しました。
KCCC1205T-I COBOLプログラムの実行を開始しました。
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名 (IN-FILE) 660 <AVERAGE/AVERAGE>
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名 (OUT-FILE) 760 <AVERAGE/AVERAGE>
KCCC1206T-I COBOLプログラムの実行を終了しました。
KCCC1201T-I ユーザプログラムの実行を終了しました。
KCCC3405T-I カバレージ情報の蓄積を行いました。翻訳単位名(AVERAGE) ファイル (/home/cobol2002/TP/AVERAGE.cbp)
KCCC0301T-I テストデバッグを終了します。 YYYY-MM-DD HH:MM:SS
```

※

注※

テストケース 1 の TD コマンド群

図 C-2　テストケース 2（TD コマンドの例）

```
KCGC0300T-I テストデバッグを開始します。 YYYY-MM-DD HH:MM:SS
#INCLUDE INFILE(INCAV2)
*>*************************************************************************
*>      Input File 及び SUBAV が未作成                                     *
*>*************************************************************************
*>-----------------------------------------------------------------------*
*>      入力データが全て正しい場合     テスト NO  --> 002                  *
*>-----------------------------------------------------------------------*
SET QUALIFICATION(#PROGRAM(AVERAGE/AVERAGE))
SIMULATE FILE(IN-FILE) OPENMODE(INPUT)          *> READ文の擬似実行手続き
    REPEAT
        ASSIGN DATA(IN-REC) VALUE(080100090)
        DISPLAY DATA(IN-REC)
    ENDREPEAT
    REPEAT
        ASSIGN DATA(IN-REC) VALUE(050060070)
        DISPLAY DATA(IN-REC)
    ENDREPEAT
    REPEAT
        ASSIGN DATA(IN-REC) VALUE(010050060)
        DISPLAY DATA(IN-REC)
    ENDREPEAT
    REPEAT                                                              ※
        GO END
    ENDREPEAT
ENDSIMULATE
SIMULATE FILE(OUT-FILE) OPENMODE(OUTPUT)        *> WRITE文の擬似実行手続き
ENDSIMULATE
SIMULATE SUB(#PROGRAM(SUBAV))                   *> サブプログラムの擬似実行手続き
    ASSIGN DATA(WA-P(1)) VALUE(90)
    ASSIGN DATA(R-CODE(1)) VALUE(0)
    ASSIGN DATA(WA-P(2)) VALUE(60)
    ASSIGN DATA(R-CODE(2)) VALUE(0)
    ASSIGN DATA(WA-P(3)) VALUE(40)
    ASSIGN DATA(R-CODE(3)) VALUE(0)
    DISPLAY DATA(WA-P(1..3))
    DISPLAY DATA(R-CODE(1..3))
ENDSIMULATE
GO COVERAGE
KCGC1200T-I ユーザプログラムの実行を開始しました。
KCGC1205T-I COBOLプログラムの実行を開始しました。
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (IN-FILE) 660 <AVERAGE/AVERAGE>
名　称      IN-REC
  値        080100090
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (IN-FILE) 700 <AVERAGE/AVERAGE>
名　称      IN-REC
  値        050060070
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (IN-FILE) 700 <AVERAGE/AVERAGE>
名　称      IN-REC
  値        010050060
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (IN-FILE) 700 <AVERAGE/AVERAGE>
KCGC1250T-I 副プログラムシミュレーションを開始します。 プログラム名 (SUBAV) 820 <AVERAGE/AVERAGE>
名　称      WA-P(1..3)
  値        0090    0060    0040
名　称      R-CODE(1..3)
  値        0    0    0
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (OUT-FILE) 910 <AVERAGE/AVERAGE>
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (OUT-FILE) 1020 <AVERAGE/AVERAGE>
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (OUT-FILE) 1020 <AVERAGE/AVERAGE>
KCGC1232T-I ファイルシミュレーションを開始します。ファイル名 (OUT-FILE) 1020 <AVERAGE/AVERAGE>
KCGC1206T-I COBOLプログラムの実行を終了しました。
KCGC1201T-I ユーザプログラムの実行を終了しました。
KCGC3405T-I カバレージ情報の蓄積を行いました。翻訳単位名(AVERAGE) ファイル (/home/cobol2002/TP/AVERAGE.cbp)
KCGC0301T-I テストデバッグを終了します。 YYYY-MM-DD HH:MM:SS
```

注※

テストケース 2 の TD コマンド群

図 C-3　テストケース 3（TD コマンドの例）

```
KCCC0300T-I テストデバッグを開始します。 YYYY-MM-DD HH:MM:SS
#INCLUDE INFILE(INCAV3)
*>*************************************************************************
*>      Input File 及び SUBAV が未作成                                     *
*>*************************************************************************
*>-----------------------------------------------------------------------*
*>      入力データにエラーがある場合     テスト NO  --> 003                *
*>-----------------------------------------------------------------------*
SET QUALIFICATION(#PROGRAM(AVERAGE/AVERAGE))
SIMULATE FILE(IN-FILE) OPENMODE(INPUT)          *> READ文の擬似実行手続き
    REPEAT
        ASSIGN DATA(IN-REC) VALUE(080060100)
        DISPLAY DATA(IN-REC)
    ENDREPEAT
    REPEAT
        ASSIGN DATA(IN-REC) VALUE(110020300)
        DISPLAY DATA(IN-REC)
    ENDREPEAT
    REPEAT
        ASSIGN DATA(IN-REC) VALUE(060080040)
        DISPLAY DATA(IN-REC)
    ENDREPEAT
    REPEAT
        GO END
    ENDREPEAT
ENDSIMULATE
SIMULATE FILE(OUT-FILE) OPENMODE(OUTPUT)        *> WRITE文の擬似実行手続き
ENDSIMULATE
SIMULATE SUB(#PROGRAM(SUBAV))                   *> サブプログラムの擬似実行手続き
    ASSIGN DATA(WA-P(1)) VALUE(80)
    ASSIGN DATA(R-CODE(1)) VALUE(0)
    ASSIGN DATA(WA-P(2)) VALUE(0)
    ASSIGN DATA(R-CODE(2)) VALUE(8)
    ASSIGN DATA(WA-P(3)) VALUE(60)
    ASSIGN DATA(R-CODE(3)) VALUE(0)
    DISPLAY DATA(WA-P(1..3))
    DISPLAY DATA(R-CODE(1..3))
ENDSIMULATE
GO COVERAGE
KCCC1200T-I ユーザプログラムの実行を開始しました。
KCCC1205T-I COBOLプログラムの実行を開始しました。
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（IN-FILE）660 <AVERAGE/AVERAGE>
名　称      IN-REC
  値        080060100
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（IN-FILE）700 <AVERAGE/AVERAGE>
名　称      IN-REC
  値        110020300
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（IN-FILE）700 <AVERAGE/AVERAGE>
名　称      IN-REC
  値        060080040
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（IN-FILE）700 <AVERAGE/AVERAGE>
KCCC1250T-I 副プログラムシミュレーションを開始します。 プログラム名（SUBAV）820 <AVERAGE/AVERAGE>
名　称      WA-P(1..3)
  値        0080    0000    0060
名　称      R-CODE(1..3)
  値        0    8    0
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（OUT-FILE）910 <AVERAGE/AVERAGE>
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（OUT-FILE）1020 <AVERAGE/AVERAGE>
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（OUT-FILE）960 <AVERAGE/AVERAGE>
KCCC1232T-I ファイルシミュレーションを開始します。ファイル名（OUT-FILE）1020 <AVERAGE/AVERAGE>
KCCC1206T-I COBOLプログラムの実行を終了しました。
KCCC1201T-I ユーザプログラムの実行を終了しました。
KCCC3405T-I カバレージ情報の蓄積を行いました。翻訳単位名(AVERAGE) ファイル（/home/cobol2002/TP/AVERAGE.cbp）
KCCC0301T-I テストデバッグを終了します。 YYYY-MM-DD HH:MM:SS
```

※

注※

テストケース 3 の TD コマンド群

(2)　カバレージ情報

付録 C.2(1)に示した TD コマンド群（テストケース 1～3）で，主プログラム（AVERAGE）の単体テストをした結果のカバレージ情報を次に示します。

図 C-4　テストケース 1（カバレージ情報）

```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************              YYYY-MM-DD HH:MM:SS
-------------------------------------------------------------------------------------------------
  プログラム名   : AVERAGE
  コンパイル日時: YYYY-MM-DD HH:MM:SS                                   変更回数     :     0
  テスト日時     : YYYY-MM-DD HH:MM:SS                                   テスト回数   :     1

-----------------------------------<全ソース表示>----------------------------------------
プログラム名 : AVERAGE
変更<C0 > <C1 >-------------------------------------------------------------------------------
              0000600 PROCEDURE       DIVISION.
              0000610 MAIN-PROCESS    SECTION.
              0000620 OPEN-FILE.
    *         0000630         OPEN INPUT  IN-FILE
              0000640              OUTPUT OUT-FILE.
              0000650 MAKING-WTBL.
    *         0000660         READ IN-FILE
    *     @   0000670              AT END MOVE 'EOF' TO ENDM.
              .
    *         0000680         PERFORM VARYING I FROM 1 BY 1 UNTIL END-OF-FIL
    .     .   0000690           MOVE IN-POINT TO W-REC(I)
    ..    .   0000700           READ IN-FILE AT END MOVE 'EOF' TO ENDM
              0000710           END-READ
              0000720          END-PERFORM.
              .
              @
              0000730 ZERO-CHECK.
    *         0000740         COMPUTE I = I - 1.
    *         0000750         IF I = 0
    *     @   0000760           WRITE OUT-REC FROM NO-MSG AFTER 1
    *         0000770           GO TO CLOSE-FILE
              .   0000780         ELSE
    .         0000790           CONTINUE
              0000800         END-IF.
              0000810 CALCULATE.
    .         0000820         CALL 'SUBAV' USING WORK-TBL I.
              0000830 PRINTING-DATA.
    .         0000840         PERFORM  DATA-PUT.
              0000850 CLOSE-FILE.
    *         0000860        CLOSE IN-FILE OUT-FILE.
    *         0000870        STOP RUN.
              0000890 DATA-PUT        SECTION.
              0000900 PRINTING-TITLE.
    .         0000910         WRITE OUT-REC FROM TITLE AFTER 1.
              0000920 PRINTING-AVERAGE.
    .         0000930         PERFORM VARYING J FROM 1 BY 1 UNTIL J > I
              .
    .         0000940           IF R-CODE(J) = 8
    .     .   0000950             MOVE W-REC(J) TO ER-REC
    .         0000960             WRITE OUT-REC FROM ERR-MSG AFTER 1
              .   0000970           ELSE
    .         0000980             MOVE WE-P(J) TO E-POINT
    .         0000990             MOVE WK-P(J) TO K-POINT
    .         0001000             MOVE WS-P(J) TO S-POINT
    .         0001010             MOVE WA-P(J) TO A-POINT
    .         0001020             WRITE OUT-REC FROM PR-REC AFTER 1
              0001030           END-IF
              0001040         END-PERFORM.
              0001050 END PROGRAM AVERAGE.
              .

*************************************************************************************************
*                      <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>     *
*     対象総数           26            0          12            0          1            0     *
*     実行済数           10            0           3            0          0            0     *
*     未実行数           16            0           9            0          1            0     *
*     カバレージ率    38.4%         0.0%       25.0%         0.0%       0.0%         0.0%     *
*************************************************************************************************
```

図 C-5　テストケース 2（カバレージ情報）

```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************              YYYY-MM-DD HH:MM:SS
------------------------------------------------------------------------------------------------
  プログラム名   : AVERAGE
  コンパイル日時: YYYY-MM-DD HH:MM:SS                                   変更回数     :     0
  テスト日時     : YYYY-MM-DD HH:MM:SS                                   テスト回数   :     2

--------------------------------<全ソース表示>----------------------------------------
プログラム名 : AVERAGE
変更<C0 > <C1 >-------------------------------------------------------------------------
              0000600 PROCEDURE       DIVISION.
              0000610 MAIN-PROCESS    SECTION.
              0000620 OPEN-FILE.
    *         0000630         OPEN INPUT  IN-FILE
              0000640              OUTPUT OUT-FILE.
              0000650 MAKING-WTBL.
    *         0000660         READ IN-FILE
    *     @   0000670              AT END MOVE 'EOF' TO ENDM.
          @
    *         0000680         PERFORM VARYING I FROM 1 BY 1 UNTIL END-OF-FIL
    *     @   0000690           MOVE IN-POINT TO W-REC(I)
    **    @   0000700           READ IN-FILE AT END MOVE 'EOF' TO ENDM
              0000710           END-READ
              0000720         END-PERFORM.
          @
          @
              0000730 ZERO-CHECK.
    *         0000740         COMPUTE I = I - 1.
    *         0000750         IF I = 0
    *     @   0000760           WRITE OUT-REC FROM NO-MSG AFTER 1
    *         0000770           GO TO CLOSE-FILE
          @   0000780         ELSE
    *         0000790           CONTINUE
              0000800         END-IF.
              0000810 CALCULATE.
    *         0000820         CALL 'SUBAV' USING WORK-TBL I.
              0000830 PRINTING-DATA.
    *         0000840         PERFORM  DATA-PUT.
              0000850 CLOSE-FILE.
    *         0000860        CLOSE IN-FILE OUT-FILE.
    *         0000870        STOP RUN.
              0000890 DATA-PUT        SECTION.
              0000900 PRINTING-TITLE.
    *         0000910         WRITE OUT-REC FROM TITLE AFTER 1.
              0000920 PRINTING-AVERAGE.
    *         0000930         PERFORM VARYING J FROM 1 BY 1 UNTIL J > I
          @
    *         0000940           IF R-CODE(J) = 8
    .     .   0000950             MOVE W-REC(J) TO ER-REC
    .         0000960             WRITE OUT-REC FROM ERR-MSG AFTER 1
          @   0000970           ELSE
    *         0000980             MOVE WE-P(J) TO E-POINT
    *         0000990             MOVE WK-P(J) TO K-POINT
    *         0001000             MOVE WS-P(J) TO S-POINT
    *         0001010             MOVE WA-P(J) TO A-POINT
    *         0001020             WRITE OUT-REC FROM PR-REC AFTER 1
              0001030           END-IF
              0001040         END-PERFORM.
              0001050 END PROGRAM AVERAGE.
          @

*************************************************************************************************
*                     <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>      *
*     対象総数          26            0          12            0          1            0      *
*     実行済数          24            0          11            0          1            0      *
*     未実行数           2            0           1            0          0            0      *
*     カバレージ率    92.3%        0.0%       91.6%        0.0%     100.0%        0.0%      *
*************************************************************************************************
```

図 C-6　テストケース 3（カバレージ情報）

```
                          ******************************
                          *      カバレージ情報        *
     COBOL2002 (c)  VV-RR ******************************              YYYY-MM-DD HH:MM:SS
------------------------------------------------------------------------------------------
  プログラム名   : AVERAGE
  コンパイル日時: YYYY-MM-DD HH:MM:SS                                  変更回数    :     0
  テスト日時     : YYYY-MM-DD HH:MM:SS                                  テスト回数  :     3

-----------------------------------<全ソース表示>-----------------------------------------
プログラム名 : AVERAGE
変更<C0 > <C1 >-----------------------------------------------------------------------------
              0000600 PROCEDURE       DIVISION.
              0000610 MAIN-PROCESS    SECTION.
              0000620 OPEN-FILE.
    *         0000630         OPEN INPUT  IN-FILE
              0000640             OUTPUT OUT-FILE.
              0000650 MAKING-WTBL.
    *         0000660         READ IN-FILE
    *     @   0000670              AT END MOVE 'EOF' TO ENDM.
          @
    *         0000680         PERFORM VARYING I FROM 1 BY 1 UNTIL END-OF-FIL
    *     @   0000690           MOVE IN-POINT TO W-REC(I)
    **    @   0000700           READ IN-FILE AT END MOVE 'EOF' TO ENDM
              0000710           END-READ
              0000720          END-PERFORM.
          @
          @
              0000730 ZERO-CHECK.
    *         0000740         COMPUTE I = I - 1.
    *         0000750         IF I = 0
    *     @   0000760           WRITE OUT-REC FROM NO-MSG AFTER 1
    *         0000770           GO TO CLOSE-FILE
          @   0000780         ELSE
    *         0000790           CONTINUE
              0000800         END-IF.
              0000810 CALCULATE.
    *         0000820         CALL 'SUBAV' USING WORK-TBL I.
              0000830 PRINTING-DATA.
    *         0000840         PERFORM  DATA-PUT.
              0000850 CLOSE-FILE.
    *         0000860        CLOSE IN-FILE OUT-FILE.
    *         0000870        STOP RUN.
              0000890 DATA-PUT        SECTION.
              0000900 PRINTING-TITLE.
    *         0000910         WRITE OUT-REC FROM TITLE AFTER 1.
              0000920 PRINTING-AVERAGE.
    *         0000930         PERFORM VARYING J FROM 1 BY 1 UNTIL J > I
          @
    *         0000940           IF R-CODE(J) = 8
    *     @   0000950             MOVE W-REC(J) TO ER-REC
    *         0000960             WRITE OUT-REC FROM ERR-MSG AFTER 1
          @   0000970           ELSE
    *         0000980             MOVE WE-P(J) TO E-POINT
    *         0000990             MOVE WK-P(J) TO K-POINT
    *         0001000             MOVE WS-P(J) TO S-POINT
    *         0001010             MOVE WA-P(J) TO A-POINT
    *         0001020             WRITE OUT-REC FROM PR-REC AFTER 1
              0001030           END-IF
              0001040         END-PERFORM.
              0001050 END PROGRAM AVERAGE.
          @

******************************************************************************************
*                    <C0>    <差分C0>    <C1>    <差分C1>    <S1>    <差分S1>     *
*     対象総数         26           0         12           0          1           0     *
*     実行済数         26           0         12           0          1           0     *
*     未実行数          0           0          0           0          0           0     *
*     カバレージ率   100.0%        0.0%     100.0%        0.0%     100.0%        0.0%     *
******************************************************************************************
```

付録 C.3　カバレージ情報の操作

(1)　プログラム情報ファイルのマージ

次の例では，プログラム情報ファイルをマスタ用ファイルとテスト用ファイルに分けて使用します。カバレージ情報はテスト用ファイルに蓄積し，テスト後にマスタ用ファイルにマージします。このようにプロ

グラム情報ファイルを使用すると，ファイルを保護したり，同一のテストプログラムを複数の使用者でテストしたりできます。

(使用例)

プログラム情報ファイル/usr/test/master/testa.cbp をマスタ用ファイルとして使用する。

テスト方法

```
cp /usr/test/master/testa.cbp /usr/tmp/testa.cbp  …1.
CBLPIDIR=/usr/tmp                                 …2.
export CBLPIDIR
cbltd2k -input /usr/batch/testa01.tdi -Execute /usr/test/testa    …3.
```

1. マスタ用のプログラム情報ファイルをコピーする。
2. 蓄積したカバレージ情報を格納するディレクトリを環境変数 CBLPIDIR で設定する。
3. cbltd2k コマンドでプログラムのテストを実行し，カバレージ情報を蓄積する。TD コマンド格納ファイル中の GO TD コマンドには，COVERAGE オペランドを指定しておく。

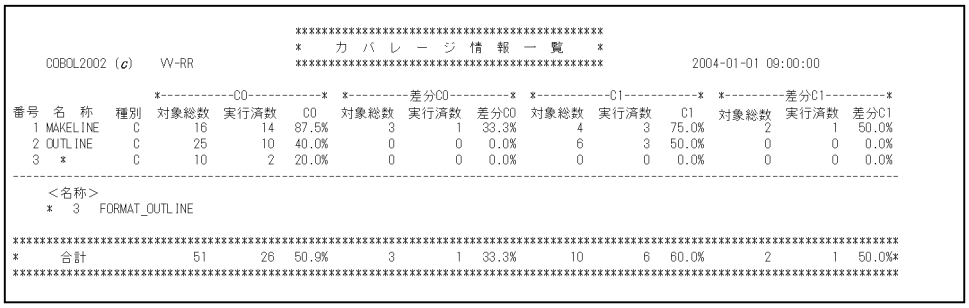
マージ方法

```
cblca2k -merge /usr/test/master/testa.cbp -input /usr/tmp/testa.cbp
```

付録 D　カバレージ情報の表示例

付録 D.1　カバレージ情報の表示例（テキスト形式）

(1)　翻訳単位の一覧表示

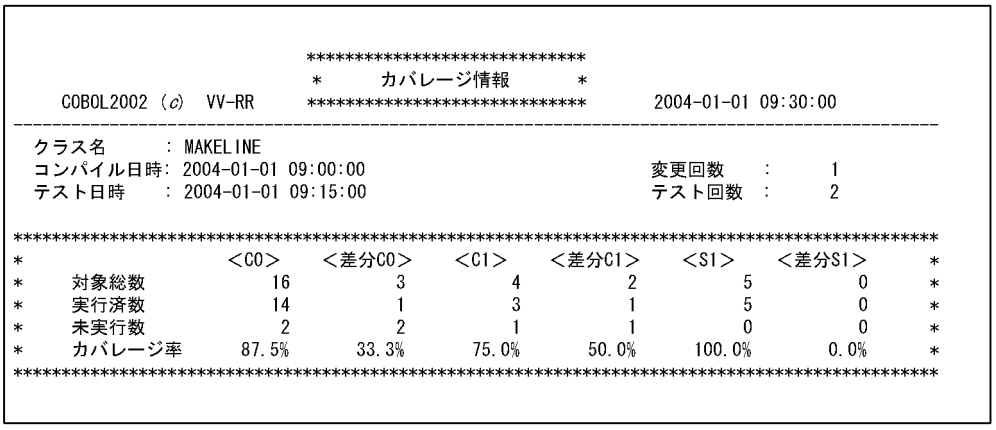


```
                                  **************************************************
                                  *    カ バ レ ー ジ 情 報 一 覧    *
    COBOL2002 (c)    VV-RR        **************************************************          2004-01-01 09:00:00

                  *----------C0----------*  *--------差分C0--------*  *----------C1----------*  *--------差分C1--------*
番号  名　称    種別 対象総数  実行済数    C0   対象総数  実行済数  差分C0 対象総数  実行済数    C1   対象総数  実行済数  差分C1
   1 MAKELINE     C       16        14  87.5%         3         1  33.3%         4         3  75.0%         2         1  50.0%
   2 OUTLINE      C       25        10  40.0%         0         0   0.0%         6         3  50.0%         0         0   0.0%
   3  *           C       10         2  20.0%         0         0   0.0%         0         0   0.0%         0         0   0.0%
-------------------------------------------------------------------------------------------------------------------------------
     <名称>
     *  3   FORMAT_OUTLINE

*******************************************************************************************************************************
*      合計              51        26  50.9%         3         1  33.3%        10         6  60.0%         2         1  50.0%*
*******************************************************************************************************************************
```

名称欄には，プログラム名，またはクラス名が最大８文字で表示されます。名称が９文字以上の場合，名称欄には記号（＊）が表示され，正しい名称は〈名称〉の欄に表示されます。

種別欄には，プログラムの場合は「P」が，クラスの場合は「C」，利用者定義関数の場合は「T」が表示されます。

(2)　まとめ情報の表示



```
                                  ******************************
                                  *       カバレージ情報       *
    COBOL2002 (c)  VV-RR          ******************************       2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :       1
  テスト日時     : 2004-01-01 09:15:00                                 テスト回数  :       2

*************************************************************************************************
*                    <C0>    <差分C0>    <C1>    <差分C1>    <S1>    <差分S1>       *
*      対象総数        16           3         4           2         5           0       *
*      実行済数        14           1         3           1         5           0       *
*      未実行数         2           2         1           1         0           0       *
*      カバレージ率  87.5%       33.3%     75.0%       50.0%     100.0%       0.0%      *
*************************************************************************************************
```

- 変更回数：最初のコンパイル時からの COBOL ソースファイルの変更回数を示します。
- テスト回数：カバレージ情報を蓄積した回数を示します。
- テスト日時：カバレージ情報を蓄積した最新の日時を表示します。

(3)　全ソース表示

```
                              ******************************
                              *      カバレージ情報        *
     COBOL2002 (σ)  VV-RR     ******************************      2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :     1
  テスト日時     : 2004-01-01 09:15:00                                  テスト回数  :     2

-----------------------------------<全ソース表示>-----------------------------------------
メソッド名  : INIT_MAKELINE_F
変更<C0 > <C1 >-------------------------------------------------------------------------------
               0000021 PROCEDURE  DIVISION.
    *          0000022     INVOKE SUPER  'INIT-COLORS-F'.
    *          0000023     CALL 'CBLEXEC' USING EXEC-NAME-LEN EXEC-NAME EXEC-PARM.
               0000024     EXIT  METHOD.
               0000025 END  METHOD  INIT-MAKELINE-F.
               0000026
               0000027 END FACTORY.

メソッド名  : INIT_MAKELINE_O
変更<C0 > <C1 >-------------------------------------------------------------------------------
               0000042 PROCEDURE  DIVISION.
    *          0000043     INVOKE  SUPER  'INIT-COLORS-O'.
    *          0000044     COMPUTE  MSGCOUNT  =  0.
    *          0000045     EXIT  METHOD.
               0000046 END  METHOD  INIT-MAKELINE-O.

メソッド名  : DRAWLINE
変更<C0 > <C1 >-------------------------------------------------------------------------------
               0000056 PROCEDURE  DIVISION  USING  I-COLOR.
    *          0000057     INVOKE  SELF  'CHECK-MY-PALETTE'  RETURNING  MINE.
    *          0000058     EVALUATE  MINE
          @    0000059       WHEN  I-COLOR
    *          0000060         INVOKE  SUPER  'WHATCOLOR'  USING  BY  CONTENT  I-COLOR
               0000061                                    RETURNING  IRO
    *          0000063           DISPLAY  IRO  'の線を書きました'
          @    0000064       WHEN  OTHER
    *          0000067         COMPUTE  MSGCOUNT  =  MSGCOUNT  +  1
    *          0000068         DISPLAY  '同色の絵の具がパレットにありません'
               0000069     END-EVALUATE.
 Y  *     .    0000070     IF  MSGCOUNT  >  20  THEN
 #  .          0000071       DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .          0000072       COMPUTE  MSGCOUNT  =  0
               0000073     END-IF.
 #        @
    *          0000074     EXIT  METHOD.
               0000075 END METHOD  DRAWLINE.
               0000077 END OBJECT.
               0000079 END CLASS MAKELINE.

*************************************************************************************************
*                    <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*     対象総数         16            3           4           2           5           0       *
*     実行済数         14            1           3           1           5           0       *
*     未実行数          2            2           1           1           0           0       *
*     カバレージ率    87.5%        33.3%       75.0%       50.0%      100.0%       0.0%       *
*************************************************************************************************
```

表 D-1　全ソース表示の説明

項目	内容
翻訳単位名の種別	プログラム名・クラス名・利用者定義関数名
翻訳単位名	翻訳単位の名称
コンパイル日時	翻訳単位を含む原始プログラムをコンパイルした日時
テスト日時	翻訳単位を含む原始プログラムのカバレージ情報を最後に蓄積した日時
変更回数	翻訳単位を含む原始プログラムを変更した回数

項目	内容
テスト回数	翻訳単位を含む原始プログラムを実行した回数
ソース要素名の種別	プログラム名・メソッド名・利用者定義関数名
ソース要素名	ソース要素の名称
変更	Y：原始プログラムの変更された行
	#：原始プログラムの変更により影響がある行
<C0 >	*：実行が済んだ文
	.：実行していない文
<C1 >	@：実行が済んだ分岐
	.：実行していない分岐
行番号	コンパイラが振り直した行番号
ソーステキスト	原始プログラムの行

(4)　差分ソース表示

```
                              ********************************
                              *       カバレージ情報         *
    COBOL2002 (c)  VV-RR      ********************************      2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :     1
  テスト日時     : 2004-01-01 09:15:00                                  テスト回数  :     2

---------------------------------------<差分表示>---------------------------------------------
メソッド名    : DRAWLINE
変更<C0 > <C1 >-------------------------------------------------------------------------------
 Y  *     .     0000070     IF  MSGCOUNT  >  20  THEN
 #  .           0000071       DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .           0000072       COMPUTE  MSGCOUNT  =  0
                0000073     END-IF.
 #        @

*************************************************************************************************
*                    <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*     対象総数         16            3          4           2          5           0       *
*     実行済数         14            1          3           1          5           0       *
*     未実行数          2            2          1           1          0           0       *
*     カバレージ率    87.5%        33.3%       75.0%       50.0%      100.0%       0.0%       *
*************************************************************************************************
```

(5)　未実行表示

```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************        2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :       1
  テスト日時     : 2004-01-01 09:15:00                                  テスト回数  :       2

-------------------------------------<未実行表示>-------------------------------------------
メソッド名    : DRAWLINE
変更<C0 > <C1 >-----------------------------------------------------------------------------
 Y  *     .     0000070      IF  MSGCOUNT  >  20  THEN
 #  .           0000071        DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .           0000072        COMPUTE  MSGCOUNT  =  0

*****************************************************************************************************
*                     <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*     対象総数          16            3          4            2          5            0       *
*     実行済数          14            1          3            1          5            0       *
*     未実行数           2            2          1            1          0            0       *
*     カバレージ率     87.5%        33.3%       75.0%        50.0%       100.0%         0.0%       *
*****************************************************************************************************
```

(6)　差分未実行ソース表示

```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************        2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                  変更回数    :       1
  テスト日時     : 2004-01-01 09:15:00                                  テスト回数  :       2

-------------------------------------<差分未実行表示>-------------------------------------------
メソッド名    : DRAWLINE
変更<C0 > <C1 >-----------------------------------------------------------------------------
 Y  *     .     0000070      IF  MSGCOUNT  >  20  THEN
 #  .           0000071        DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
 #  .           0000072        COMPUTE  MSGCOUNT  =  0

*****************************************************************************************************
*                     <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>       *
*     対象総数          16            3          4            2          5            0       *
*     実行済数          14            1          3            1          5            0       *
*     未実行数           2            2          1            1          0            0       *
*     カバレージ率     87.5%        33.3%       75.0%        50.0%       100.0%         0.0%       *
*****************************************************************************************************
```

(7)　呼び出し文ソース表示

```
                              ******************************
                              *       カバレージ情報       *
     COBOL2002 (c)  VV-RR     ******************************       2004-01-01 09:30:00
-------------------------------------------------------------------------------------------------
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00                                変更回数    :     1
  テスト日時     : 2004-01-01 09:15:00                                テスト回数  :     2

-----------------------------------<呼び出し文表示>-----------------------------------------
メソッド名    : INIT_MAKELINE_F
変更<C0 > <C1 >-------------------------------------------------------------------------------
    *          0000022     INVOKE  SUPER  'INIT-COLORS-F'.
    *          0000023     CALL 'CBLEXEC' USING EXEC-NAME-LEN EXEC-NAME EXEC-PARM.

メソッド名    : INIT_MAKELINE_O
変更<C0 > <C1 >-------------------------------------------------------------------------------
    *          0000043     INVOKE  SUPER  'INIT-COLORS-O'.

メソッド名    : DRAWLINE
変更<C0 > <C1 >-------------------------------------------------------------------------------
    *          0000057     INVOKE  SELF  'CHECK-MY-PALETTE'  RETURNING  MINE.
    *     @    0000060     INVOKE  SUPER  'WHATCOLOR'  USING  BY  CONTENT  I-COLOR

*************************************************************************************************
*                      <C0>     <差分C0>     <C1>     <差分C1>     <S1>     <差分S1>      *
*     対象総数           16            3          4            2          5            0      *
*     実行済数           14            1          3            1          5            0      *
*     未実行数            2            2          1            1          0            0      *
*     カバレージ率      87.5%        33.3%       75.0%        50.0%      100.0%         0.0%      *
*************************************************************************************************
```

付録 D.2　カバレージ情報の表示例（CSV 形式）

(1)　カバレージ統計情報の出力

AIX(32) COBOL2002 で実行したときの表示例を次に示します。

```
”バージョン”,”出力日時”
”COBOL2002 (A)  04-10”,”2019-04-01 11:00:00”

”プログラム情報ファイル格納先”,”プログラム情報ファイル名”,”ソースカバレージ情報CSV”,”翻訳単位名”,”コンパイル日時”,”テスト日時”,”変更回数”,”テスト回数”,”種別”,”C0対象総数”,”C0実行済数”,”C0カバレージ率”,”差分C0対象総数”,”差分C0実行済数”,”差分C0カバレージ率”,”C1対象総数”,”C1実行済数”,”C1カバレージ率”,”差分C1対象総数”,”差分C1実行済数”,”差分C1カバレージ率”,”S1対象総数”,”S1実行済数”,”S1カバレージ率”,”差分S1対象総数”,”差分S1実行済数”,”差分S1カバレージ率”
”/home/COBOL/CBP”,”foemat_outline.cbp”,”/home/COBOL/CV_CSV/foemat_outline_source.csv”,”FORMAT_OUTLINE”,”2019-04-01 09:00:00”,”2019-04-01 10:00:00”,”0”,”1”,”C”,”10”,”2”,”20.0”,”0”,”0”,”-”,”0”,”0”,”-”,”0”,”0”,”-”,”0”,”0”,”-”,”0”,”0”,”-”
”/home/COBOL/CBP”,”foemat_outline.cbp”,”/home/COBOL/CV_CSV/foemat_outline_source.csv”,”-”,”2019-04-01 09:00:00”,”-”,”-”,”-”,”-”,”10”,”2”,”20.0”,”0”,”0”,”-”,”0”,”0”,”-”,”0”,”0”,”-”,”0”,”0”,”-”
”/home/COBOL/CBP”,”makeline.cbp”,”/home/COBOL/CV_CSV/makeline_source.csv”,”MAKELINE”,”2019-04-01 09:00:00”,”2019-04-01 10:00:00”,”1”,”2”,”C”,”16”,”12”,”75.0”,”3”,”1”,”33.3”,”4”,”2”,”50.0”,”2”,”1”,”50.0”,”5”,”5”,”100.0”,”0”,”0”,”-”
”/home/COBOL/CBP”,”makeline.cbp”,”/home/COBOL/CV_CSV/makeline_source.csv”,”-”,”2019-04-01 09:00:00”,”-”,”-”,”-”,”-”,”16”,”12”,”75.0”,”3”,”1”,”33.3”,”4”,”2”,”50.0”,”2”,”1”,”50.0”,”5”,”5”,”100.0”,”0”,”0”,”-”
```

```
"/home/COBOL/CBP","outline.cbp","/home/COBOL/CV_CSV/outline_source.csv","OUTLINE","2019-04-0
1 09:00:00","2019-04-01 10:00:00","1","2","C","25","10","40.0","1","1","100.0","6","3","50.0
","0","0","-","0","0","-","0","0","-"
"/home/COBOL/CBP","outline.cbp","/home/COBOL/CV_CSV/outline_source.csv","-","2019-04-01 09:0
0:00","-","-","-","-","-","25","10","40.0","1","1","100.0","6","3","50.0","0","0","-","0","0","-
","0","0","-"
```

出力内容については，「6.2.6　カバレージ情報の表示（CSV 形式）」の「(1)　カバレージ統計情報の出力」を参照してください。

(2)　ソースカバレージ情報の出力

AIX(32) COBOL2002 で実行したときの表示例を次に示します。

```
"バージョン","出力日時","コンパイル日時","プログラム情報ファイル格納先","プログラム情報ファ
イル名"
"COBOL2002 (A)  04-10","2019-04-01 11:00:00","2019-04-01 09:00:00","/home/COBOL/CBP","makeli
ne.cbp"

"翻訳単位名","ソース要素種別","ソース要素名","未実行ソース","差分ソース","差分未実行ソース",
"呼び出し文ソース","コメント行","変更","C0","C1","行番号","ソーステキスト"
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,,"0000021"," PROCEDURE  DIVISION."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,"ACT",,,"*",,"0000022","     INVOKE SUPER  'INIT-CO
LORS-F'."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,"ACT",,,"*",,"0000023","     CALL 'CBLEXEC' USING E
XEC-NAME-LEN EXEC-NAME EXEC-PARM."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,"*",,"0000024"," EXIT  METHOD."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,,"0000025"," END  METHOD  INIT-MAKELINE-F."
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,,"0000026",
"MAKELINE","METHOD","INIT_MAKELINE_F",,,,,,,,,"0000027"," END FACTORY."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,,,,"0000042"," PROCEDURE  DIVISION."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,"ACT",,,"*",,"0000043","     INVOKE  SUPER  'INIT-C
OLORS-O'."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,,"*",,"0000044","     COMPUTE  MSGCOUNT  =  0."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,,"*",,"0000045","     EXIT  METHOD."
"MAKELINE","METHOD","INIT_MAKELINE_O",,,,,,,,,"0000046"," END  METHOD  INIT-MAKELINE-O."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,,"0000056"," PROCEDURE  DIVISION  USING  I-COLOR."
"MAKELINE","METHOD","DRAWLINE",,,,"ACT",,,"*",,"0000057","     INVOKE  SELF  'CHECK-MY-PALET
TE'  RETURNING  MINE."
"MAKELINE","METHOD","DRAWLINE",,,,,,,"*",,"0000058","     EVALUATE  MINE"
"MAKELINE","METHOD","DRAWLINE",,,,,,,,,"@","0000059","       WHEN  I-COLOR"
"MAKELINE","METHOD","DRAWLINE",,,,"ACT",,,"*","@","0000060","         INVOKE  SUPER  'WHATCO
LOR'  USING  BY  CONTENT  I-COLOR"
"MAKELINE","METHOD","DRAWLINE",,,,,,,,,"0000061","                                  RETURNI
NG  IRO"
"MAKELINE","METHOD","DRAWLINE",,,,,,,"*",,"0000063","           DISPLAY  IRO  'の線を書きま
した'"
"MAKELINE","METHOD","DRAWLINE","UNEX",,,,,,,".","0000064","       WHEN  OTHER"
"MAKELINE","METHOD","DRAWLINE","UNEX",,,,,,".",,"0000067","         COMPUTE  MSGCOUNT  =  MS
GCOUNT  +  1"
"MAKELINE","METHOD","DRAWLINE","UNEX",,,,,,".",,"0000068","         DISPLAY  '同色の絵の具が
パレットにありません'"
"MAKELINE","METHOD","DRAWLINE",,,,,,,,,"0000069","     END-EVALUATE."
"MAKELINE","METHOD","DRAWLINE","UNEX","DIFF","UNEXDIFF",,,"Y","*",".","0000070","     IF  MS
GCOUNT  >  20  THEN"
```

```
"MAKELINE","METHOD","DRAWLINE","UNEX","DIFF","UNEXDIFF",,,"#",".",,"0000071","       DISPLAY
  'ﾍﾙﾌﾟを参照して使用方法を確認してください'"
"MAKELINE","METHOD","DRAWLINE","UNEX","DIFF","UNEXDIFF",,,"#",".",,"0000072","       COMPUTE
  MSGCOUNT  =  0"
"MAKELINE","METHOD","DRAWLINE",,"DIFF",,,,,,,"0000073","     END-IF."
"MAKELINE","METHOD","DRAWLINE",,"DIFF",,,,"#",,"@",,
"MAKELINE","METHOD","DRAWLINE",,,,,,,"*",,"0000074","     EXIT  METHOD."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,,"0000075"," END METHOD  DRAWLINE."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,,"0000077"," END OBJECT."
"MAKELINE","METHOD","DRAWLINE",,,,,,,,,"0000079"," END CLASS MAKELINE."
```

出力内容については，「6.2.6　カバレージ情報の表示（CSV 形式）」の「(2)　ソースカバレージ情報の出力」を参照してください。

付録 D.3　カウント情報の表示例

```
                          ******************************
                          *       カウント情報         *
     COBOL2002 (c)  VV-RR ******************************          2004-01-01 09:30:00
------------------------------------------------------------------------------------------
--
  クラス名       : MAKELINE
  コンパイル日時: 2004-01-01 09:00:00
  実行日時       : 2004-01-01 09:15:00

------------------------------------------------------------------------------------------
--
メソッド名    : INIT_MAKELINE_F
  実行回数      ----------------------------------------------------------------------------
--
                0000021 PROCEDURE  DIVISION.
         1      0000022     INVOKE  SUPER  'INIT-COLORS-F'.
         1      0000023     EXIT  METHOD.
                0000024 END  METHOD  INIT-MAKELINE-F.
                0000025
                0000026 END  FACTORY.

メソッド名    : INIT_MAKELINE_O
  実行回数      ----------------------------------------------------------------------------
--
                0000042 PROCEDURE  DIVISION.
         2      0000043     INVOKE  SUPER  'INIT-COLORS-O'.
         2      0000044     COMPUTE  MSGCOUNT  =  0.
                0000045     EXIT  METHOD.
                0000046 END  METHOD  INIT-MAKELINE-O.

メソッド名    : DRAWLINE
  実行回数      ----------------------------------------------------------------------------
--
                0000056 PROCEDURE  DIVISION  USING  I-COLOR.
         6      0000057     INVOKE  SELF  'CHECK-MY-PALETTE'  RETURNING  MINE.
         6      0000058     EVALUATE  MINE
                0000059       WHEN  I-COLOR
         4      0000060         INVOKE  SUPER  'WHATCOLOR'  USING  BY  CONTENT  I-COLOR
                0000061                                    RETURNING IRO
         4      0000063         DISPLAY  IRO  'の線を書きました'
                0000064         WHEN  OTHER
         2      0000067           COMPUTE  MSGCOUNT  =  MSGCOUNT  +  1
                0000068           DISPLAY  '同色の絵の具がパレットにありません'
                0000069     END-EVALUATE.
         6      0000070     IF  MSGCOUNT  >  20  THEN
         0      0000071       DISPLAY  'ﾍﾙﾌﾟを参照して使用方法を確認してください'
                0000072       COMPUTE  MSGCOUNT  =  0
                0000073     END-IF.
         6      0000074     EXIT  METHOD.
                0000075 END  METHOD  DRAWLINE.
                0000077 END  OBJECT.
                0000079 END  CLASS  MAKELINE.
```

付録 E　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 E.1　関連マニュアル

このマニュアルは次のマニュアルと関連があります。必要に応じてお読みください。

- COBOL2002　使用の手引　手引編　（4010-1J-802）
- COBOL2002　言語　標準仕様編　（4010-1J-804）
- COBOL2002　言語　拡張仕様編　（4010-1J-805）
- COBOL2002　Cosminexus 連携機能ガイド　（4010-1J-806）
- COBOL2002　XML 連携機能ガイド　（4010-1J-808）
- COBOL2002　メッセージ　（4010-1J-809）
- Hitachi Code Converter（UNIX 編）（3020-7-358）

付録 E.2　このマニュアルでの表記

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

マニュアルでの表記	製品名
Excel	Microsoft Excel

このマニュアル中では，「このシステム」と表現している場合，「COBOL2002」を示しています。

また，このマニュアルは，製品種別によって相違点があります。本文中での製品種別ごとの表記を次に示します。

マニュアルでの表記			該当する製品の形名
UNIX または UNIX COBOL2002	AIX	AIX(32)または AIX(32) COBOL2002	－※
		AIX(64)または AIX(64) COBOL2002	－※
	Linux	Linux(x86)または Linux(x86) COBOL2002	－※
		Linux(x64)または Linux(x64) COBOL2002	P-9W36-1251 P-9W36-2251

注※

該当する製品の形名の詳細は，「リリースノート」でご確認ください。

また，このマニュアルでは各製品を次のように表記しています。

マニュアルでの表記			製品名
UNIX	AIX		AIX V7.1
			AIX V7.2
			AIX 7.3
	Linux	Linux Server 7 (64-bit x86_64)	Red Hat Enterprise Linux Server 7 (64-bit x86_64)
		Linux Server 8 (64-bit x86_64)	Red Hat Enterprise Linux Server 8 (64-bit x86_64)
		Linux Server 9 (64-bit x86_64)	Red Hat Enterprise Linux Server 9 (64-bit x86_64)
XMAP3			XMAP3 Server
			XMAP3 Developer Version 5
			XMAP3 Server Runtime Version 5
			XMAP3 Client Runtime Version 5

- XMAP3 の製品を区別する必要がある場合は，それぞれの製品名称を表記しています。
- 日立 COBOL2002 のことを日立 COBOL2002，または単に COBOL2002 と表記しています。
- 日立 COBOL85 のことを日立 COBOL85，または単に COBOL85 と表記しています。また，プラットフォームを明確にする必要がある場合は，「PC COBOL85」「VOS3 COBOL85」のように表記しています。
- リストのヘッダ部に表示される COBOL2002 の識別記号を次のように表記しています。識別記号以外については，マニュアル「COBOL2002　使用の手引　手引編」のコンパイルリストを参照してください。

```
COBOL2002 (c)  VV-RR  *** CCC･･･CCC *** YYYY-MM-DD HH:MM:SS
        |
識別記号
```

識別記号（c）	内容
A と表示された場合	AIX(32)であることを示します。
B と表示された場合	AIX(64)であることを示します。
J と表示された場合	Linux(x86)であることを示します。
K と表示された場合	Linux(x64)であることを示します。

- CSV ファイルのヘッダ部の「バージョン」に表示される COBOL2002 の識別記号を（*c*）と表記しています。識別記号以外については，「6.2.6　カバレージ情報の表示（CSV 形式）」を参照してください。

識別記号（*c*）	内容
A と表示された場合	AIX(32)であることを示します。
B と表示された場合	AIX(64)であることを示します。
J と表示された場合	Linux(x86)であることを示します。
K と表示された場合	Linux(x64)であることを示します。

- コンパイラオプションの説明では，次の表記を使用します。

  「XXX オプション」，または単に「XXX」とオプション名が表記されている場合

  　XXX オプションについて，サブオプションの組み合わせを含む，すべての場合を意味します。

  「XXX,YYY オプション」，または単に「XXX,YYY」とサブオプションを含めたオプション名が表記されている場合

  　XXX,YYY オプションだけの場合を意味します。

  「XXX コンパイラオプション」と表記されている場合

  　リンカオプションなど，ほかのオプションと明確に区別する必要がある場合を意味します。

  （例 1）

  　「-Compile オプション」または「-Compile」と記載している場合，-Compile オプションのサブオプションの組み合わせすべて（-Compile,CheckOnly／-Compile,NoLink）を意味します。

  （例 2）

  　「-Compile,CheckOnly オプション」または「-Compile,CheckOnly」と記載している場合，-Compile,CheckOnly だけを意味します。

付録 E.3　KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

付録 F　用語解説

(英字)

BOM（バイトオーダマーク）

ファイルの先頭に付加された，Unicode の表現形式を表す情報。

COBOL2002 では，テキスト編成ファイルに対してこの情報を付加する。本文中では，Unicode シグニチャと表記する。

C0 メジャー情報

全体の実行文の数に対する実行済み文の数の割合を示す情報。

C1 メジャー情報

全体の分岐の数に対する実行済みの分岐の数の割合を示す情報。

IVS（Ideographic Variation Sequence/Selector）

漢字を表す Unicode の直後に Variation Selector と呼ばれるコードを付加し，漢字の「異体字」を表現する方法のこと。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で使用する。

S1 メジャー情報

全体の呼び出し文（CALL 文，INVOKE 文）の数に対する実行済みの呼び出し文の数の割合を示す情報。

SELF

メッセージの受け手のオブジェクトを参照するための名前。SELF が参照するオブジェクトを SELF オブジェクトという。メソッド中で使用される SELF は，そのメソッドの呼び起こし対象のオブジェクト自身のことを表す。

TD コマンド

テストデバッグで使用するコマンド。

UCS-2（Universal multi-octet Character Set 2）

符号化文字集合の一つの形式。1 文字を 2 バイトで表現する。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目でサポートする。

UCS-4（Universal multi-octet Character Set 4）

符号化文字集合の一つの形式。1 文字を 4 バイトで表現する。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で UCS-4 の範囲をサポートする。

Unicode

1 バイトでは表現できない文字セットを含めあらゆる文字セットをサポートした文字コード。代表的な符号化文字集合として UCS-2，UCS-4 がある。代表的なエンコーディングスキーマとして UTF-8，UTF-16 がある。

COBOL2002 では，UCS-4 の範囲（UCS-2 の範囲を含む）をサポートする。

本文中では，符号化文字集合，およびエンコーディングスキーマを文字コードと表記する。

UTF-16(16-bit UCS Transformation Format)

エンコーディングスキーマの一つの形式。1 つのコード単位を 2 バイトとし，1 文字を 1 コード単位（2 バイト），または 2 コード単位（4 バイト）で表現する。UTF-16 では 2 バイトのコード単位の 1 バイト目を先に書くビッグエンディアン形式（UTF-16BE）と 1 バイト目を後に書くリトルエンディアン形式（UTF-16LE）がある。

COBOL2002 では，用途が NATIONAL の項目で UCS-4 の範囲（UCS-2 の範囲を含む）をサポートする。

UTF-8(8-bit UCS Transformation Format)

エンコーディングスキーマの一つの形式。ASCII 文字を 1 バイト，日本語文字を 3~8 バイト，半角かたかなを 3 バイトで表現する。

COBOL2002 では，用途が DISPLAY の項目で使用する。

（ア行）

入れ子プログラム

原始プログラムが入れ子構造になっている場合の，内側のプログラムのこと。内側のプログラム，または内部プログラムともいう。

インスタンスオブジェクト（instance object）

あるクラスに属するオブジェクトのこと。「BASE クラス」のファクトリオブジェクトのメソッド「NEW クラス」の呼び起こしによって生成され，実行単位の終了か，ガーベジコレクションによって消滅させられる。

インタフェース（interface）

メッセージの送り手から見えるオブジェクトの部分のこと。

例えば，現金支払い機を一つのオブジェクトとすると，「引き出し」「預け入れ」などのメニューがインタフェースに当たり，そのサービスを実現するための内部的な仕組みが実装に当たる。

インタフェースは，メソッド原型の集合である。つまり，どのようなメソッド原型を持つかによって，オブジェクトが持つインタフェースの型が決まる。インタフェースは，クラスとは別に定義できる。

内側のプログラム

原始プログラムが入れ子構造になっている場合の，内側のプログラムのこと。入れ子プログラム，または内部プログラムともいう。

オブジェクト（object）

データとそのデータを操作するメソッド（手続き）を一体化させたモジュール。オブジェクトは，ある特定の仕事を受け持ち，所定のメッセージに対してメソッドを動作させることで問題を処理する。オブジェクト指向のプログラムでは，オブジェクト同士がメッセージのやり取りをすることによって，問題が処理される。

オブジェクト参照データ項目

オブジェクトを参照するためのデータ。オブジェクト参照は，データ項目に，USAGE IS OBJECT REFERENCE 句を指定したもの。

オブジェクト指向（Object Orientation）

現実にある「もの」（オブジェクト）同士のやり取りをモデル化し，それをそのままコンピュータの世界でも実現しようとする考え方のこと。

オプション

コンパイラやコマンドへの動作を指示するためのもの。

（カ行）

外部プログラム

原始プログラムが入れ子構造になっている場合の，最も外側のプログラムのこと。このマニュアルでは，外部プログラムのことを最外側のプログラムと呼ぶ。

カウント情報

プログラム中の文の実行回数をカウントして数値で表したもの。

型（type）

ある集合の中の個々を，何によって識別するかを明示するもの。例えば，プログラムを構成する要素をデータごとに識別する場合は，データ型となる。

活性化されるプログラム

CALL 文の対象となるプログラムで，実行時に呼び出し元プログラムと組み合わされて 1 個の実行単位となる。このマニュアルでは，活性化されるプログラムのことを呼び出し先のプログラムと呼ぶ。

カバレージ情報

テストの進捗状況を数値で表したもので，C0 メジャー情報，C1 メジャー情報，S1 メジャー情報，およびこれらの差分情報がある。

環境変数

実行可能ファイルおよびその実行環境へのオプションを変数として設定しておくもの。

クラス（class）

ある共通の性質を持つオブジェクトを一つのグループにまとめて定義したもの。クラスでは，それに属するオブジェクトのデータおよびメソッドを定義することで，そのクラスに属するオブジェクトの型を規定する。

継承（inheritance）

スーパクラスから，その性質（メソッド）をサブクラスが受け継ぐ仕組みのこと。COBOL2002 のオブジェクト指向機能では，INHERITS 句を指定することでスーパクラスから，その性質を継承できる。一つのクラスの性質を継承することを単純継承という。これに対して，複数のスーパクラスの性質を継承することを多重継承という。また，多重継承の結果，同じクラスを重複して継承することをダイヤモンド継承という。

コンパイル

原始プログラムを翻訳すること。

COBOL プログラムのコンパイルは，ccbl2002 コマンドで実行する。

（サ行）

最外側のプログラム

原始プログラムが入れ子構造になっている場合の，最も外側のプログラムのこと。外部プログラムともいう。

サブクラス（subclass）

クラスの階層関係の中の継承する側のクラス。

差分カバレージ情報

原始プログラムの修正によって影響を受けた部分に対するカバレージ情報。

サロゲート

UTF-16 の拡張で，2 つのコード単位（4 バイト）で 1 文字を表す機能。この 2 つのコード単位の組み合わせのことをサロゲートペアと呼ぶ。

COBOL2002 では，用途が NATIONAL の項目で使用する。

実行可能プログラム

呼び出し元のプログラムと呼び出し先のプログラムをコンパイル，リンクし，一つの実行できるプログラムにしたもの。

実行時要素

プログラム定義，メソッド定義，および関数定義の総称。このマニュアルでは，実行時要素のことをプログラムと呼ぶ。

スーパクラス（superclass）

クラスの階層関係の中の継承される側のクラス。

（タ行）

単体テスト

プログラム（コンパイル）単位のテスト。

定数長拡張機能

英数字定数の定数長を拡張できるようにする機能。英数字定数の長さの限界値を 160 文字（バイト）から 8,191 文字（バイト）に拡張できる。

定数長を拡張するプログラムのコンパイル時には，-LiteralExtend,Alnum コンパイラオプションを指定する。

動的長基本項目（dynamic-length elementary item）

実行時に長さを変更できるデータ項目（英数字項目または日本語項目）のこと。DYNAMIC LENGTH 句で定義する。動的長基本項目に新しい値が格納されると，項目の長さは自動的に調整される。

動的長基本項目は格納する値によって長さが変わる。想定以上にデータ項目の長さが長くなることを防ぐため，データ項目の最大長は LIMIT で指定できる。

（ナ行）

内部プログラム

原始プログラムが入れ子構造になっている場合の，内側のプログラムのこと。このマニュアルでは，内部プログラムのことを内側のプログラム，または入れ子プログラムと呼ぶ。

日本語集団項目（national group item）

用途が NATIONAL で字類および項類が日本語となる集団項目。日本語集団項目となるデータ項目には，明示的にまたは暗黙的に GROUP-USAGE IS NATIONAL の指定が必要。

日本語集団項目の従属項目は，日本語項目，日本語編集項目および日本語集団項目だけとなる。

（ハ行）

バイトオーダ

2 バイト以上のデータの記録を行う順序のこと。例えば，0x1234 のデータを 0x1234 のように最上位のバイトから順番に記録する方式をビッグエンディアン，0x3412 のように最下位のバイトから順番に記録する方式をリトルエンディアンという。2 バイトの UTF-16 は，バイトオーダを意識する。

ファクトリオブジェクト

一つのクラス中のすべてのオブジェクトが共有するデータ（ファクトリデータ）とそれを操作するためのメソッド（ファクトリメソッド）から成るオブジェクト。ファクトリオブジェクトは，実行単位の開始によって生成され，実行単位が終了すると消滅させられる。

ファクトリデータ

一つのクラス中のすべてのオブジェクトが共有するデータのこと。ファクトリメソッドによって操作される。

ファクトリメソッド

ファクトリデータを操作するためのメソッド。

プログラム

プログラム定義，メソッド定義，および関数定義の総称。実行時要素ともいう。

（マ行）

マルチスレッド

プログラム内の仕事を，スレッドという単位に分けて実行する方式。複数のプログラムを並列に実行できる。

見た目幅

Unicode 機能の組み込み関数で使用する，文字の見た目の幅。半角文字の幅は 1，全角文字の幅は 2 として扱う。

Unicode 機能の組み込み関数で，文字を見た目幅で数える場合に使用する。

メソッド（method）

オブジェクトが提供するサービス（メソッド原型という）とそのサービスの実装を定義した手続き。オブジェクト間のやり取りは，すべてメソッドを通して行われる。メソッドには，ファクトリメソッドおよびインスタンスメソッドの 2 種類がある。

メソッド原型（method prototype）

メソッドを定義する際，メソッドが提供するインタフェースを定義した部分。メソッド原型は，メソッド名と，パラメタおよび終了コードから成る。

メソッドの呼び起こし（method invocation）

オブジェクトに対してメソッドを呼び起こすこと。COBOL2002 では，INVOKE 文によってメソッドを呼び起こす。

（ヤ行）

呼び出し先プログラム（called program，subprogram）

CALL 文の対象となるプログラムであって，実行時に呼び出し元プログラムと組み合わされて 1 個の実行単位となる。

呼び出し元プログラム（calling program）

CALL 文を実行してほかのプログラムを呼ぶプログラム。

索引

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

は

ひ

ふ

ほ

ま

み

め

も

よ

ら

り

れ

ろ