HITACHI
Inspire the Next

JP1 Version 11

JP1/IT Desktop Management 2 - Asset Console アクセス定義ファイル作成ガイド

3021-3-B57-30

前書き

■ 対象製品

●JP1/IT Desktop Management 2 - Manager

P-2A42-78BL JP1/IT Desktop Management 2 - Manager 11-50

製品構成一覧および内訳形名

P-CC2A42-7ABL JP1/IT Desktop Management 2 - Manager 11-50（適用 OS：Windows Server 2016、Windows Server 2012、Windows Server 2008 R2）

P-CC2A42-7BBL JP1/IT Desktop Management 2 - Agent 11-50（適用 OS：Windows Server 2016、Windows 10、Windows 8.1、Windows 8、Windows Server 2012、Windows 7、Windows Server 2008 R2）

P-CC2A42-7CBL JP1/IT Desktop Management 2 - Network Monitor 11-50（適用 OS：Windows Server 2016、Windows 10、Windows 8.1 Enterprise、Windows 8.1 Pro、Windows 8 Enterprise、Windows 8 Pro、Windows Server 2012、Windows 7 Enterprise、Windows 7 Professional、Windows 7 Ultimate、Windows Server 2008 R2）

P-CC2A42-7DBL JP1/IT Desktop Management 2 - Asset Console 11-50（適用 OS：Windows Server 2016、Windows Server 2012、Windows Server 2008 R2）

■ 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

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

This product includes software developed by Ben Laurie for use in the Apache-SSL HTTP server project.

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

This product includes software developed by the University of California, Berkeley and its contributors.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. The original software is available from ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/

This product includes software developed by Ralf S. Engelschall <rse@engelschall.com> for use in the mod_ssl project (http://www.modssl.org/).

This product includes software developed by IAIK of Graz University of Technology.

This product includes software developed by Daisuke Okajima and Kohsuke Kawaguchi (http://relaxngcc.sf.net/).

This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (http://java.apache.org/).

This product includes software developed by Andy Clark.

本製品は、米国 EMC コーポレーションの RSA BSAFE(R)ソフトウェアを搭載しています。

■ 発行

2017 年 11 月　3021-3-B57-30

■ 著作権

変更内容

変更内容(3021-3-B57-30)　JP1/IT Desktop Management 2　11-50

追加・変更内容	変更個所
なし。	－

(凡例)

－：該当なし

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは、JP1/IT Desktop Management 2 - Asset Console（以降、Asset Console と略します）のスクリプトを使用して、独自の処理を追加する方法について説明したものです。

■ 対象読者

このマニュアルは、次の方にお読みいただくことを前提に説明しています。

- Asset Console を使用した資産管理システムを構築するシステム管理者の方
- 資産情報を管理する資産管理者の方
- オブジェクト指向に関する基本的な知識をお持ちの方

■ マニュアルの構成

このマニュアルは、次に示す章と付録から構成されています。

第 1 章　概要

アクセス定義ファイルを作成する目的、およびアクセス定義ファイルを作成する作業の流れについて説明しています。

第 2 章　アクセス定義ファイルの作成

アクセス定義ファイルの作成方法について説明しています。また、アクセス定義ファイルの記述例についても紹介しています。

第 3 章　アクセス定義ファイルの実行

アクセス定義ファイルを実行する方法について説明しています。

第 4 章　アクセス定義ファイルに記述するタグ

アクセス定義ファイルに記述する各タグについて説明しています。

第 5 章　アクセス定義ファイルに記述する組み込み関数

アクセス定義ファイルに記述する各組み込み関数について説明しています。

付録 A　各バージョンの変更内容

各バージョンの変更内容について説明しています。

付録 B　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について説明しています。

目次

1 概要

この章では、アクセス定義ファイルを作成する目的、およびアクセス定義ファイルを作成する作業の流れを説明します。

1.1　アクセス定義ファイルを作成する目的

この節では、Asset Console を利用した資産管理システムで、アクセス定義ファイルを作成する目的について説明します。

1.1.1　自由なフォーマットでの入出力が可能になります

Asset Console で構築する資産管理システムとは、ネットワーク機器を含めたハードウェア資産情報、ソフトウェア資産情報、保守契約情報などを**資産管理データベース**で一元管理するシステムです。資産情報をデータベースで一元管理することで、棚卸や機器増設・移設に伴う IT 資産管理業務の合理化および管理コストの低減を支援します。

資産管理システムの資産管理データベースに登録されているこれらの情報の入出力は、通常 Asset Console が提供するコマンド（jamimport および jamexport)、または Asset Console の業務メニュー（「インポート」および「エクスポート」）を使用します。コマンドでの入出力ではフォーマット固定の一括処理、業務メニューでの入出力では項目指定での一括処理となります。

これに対して、Asset Console が提供する専用のスクリプト（**アクセス定義ファイル**）を使用して処理を定義すると、フォーマットを気にせずに、詳細な条件を設定して資産管理データベースの情報を入出力できるようになります。

1.1.2　さまざまな業務に資産情報を活用できます

アクセス定義ファイルの実行結果をそのまま利用するだけでなく、アクセス定義ファイルの実行結果と Windows Script のメール送信処理を組み合わせることで、資産情報が一定の基準を超えたら資産管理者にメールを送付するといった業務も作成できます。

また、Windows のタスクスケジューラに、作成した処理を実行するタスクを登録することで、定期的に資産の情報を監視することができ、資産管理者の負荷を大幅に軽減できます。

1.1.3　ディレクトリサービスの情報にもアクセスできます

Asset Console が提供する専用のスクリプトでは、LDAP（TCP/IP 上で動作する解放型 DAP を提供し、X.500 のデータモデルを保持するもの）を使用したディレクトリサービスで管理される情報（**ディレクトリ情報**）にアクセスする処理を定義できます。そのため、ユーザや組織の情報を、ディレクトリ情報から取得するといった業務も可能になります。

1.2　アクセス定義ファイルとは

アクセス定義ファイルとは、資産管理データベースへのデータの取り込み方法や、データの更新方法を定義したファイルです。

アクセス定義ファイルは、テキスト形式のファイルです。アクセス定義ファイルを作成するための特別なアプリケーションは必要ありません。また、ファイル名も任意に設定できます。

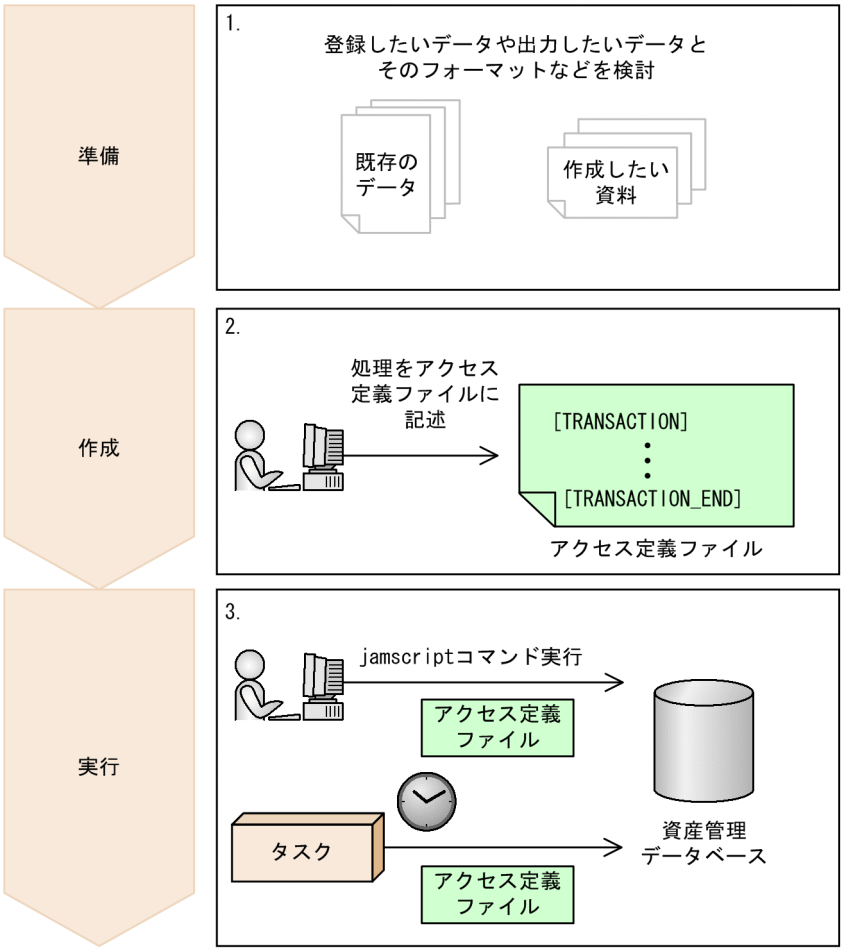
アクセス定義ファイルには、Asset Console の提供する専用のタグおよび組み込み関数と、変数、演算子などを組み合わせて処理を定義します。

アクセス定義ファイルに定義した処理は、jamscript コマンドまたはタスクによって実行されます。

1.3　アクセス定義ファイルを作成して機能を拡張する作業の流れ

この節では、アクセス定義ファイルを作成して、独自の処理を追加する作業の流れについて説明します。

図 1-1　アクセス定義を作成して、独自の処理を追加する作業の流れ



1. 資産管理データベースに登録する内容や、資産管理データベースの情報を出力して作成したい一覧など、どのような処理を追加するかを検討します。資産管理データベースで管理する情報のクラスおよびプロパティについては、マニュアル「JP1/IT Desktop Management 2 - Asset Console 構築・運用ガイド」を参照してください。

   また、あわせて、処理に必要なデータも準備します。アクセス定義ファイルで入出力できるのは、CSV 形式のファイルです。

2. 資産管理データベースに情報を登録・更新する手順や、必要な情報を CSV ファイルに入出力する手順などを、アクセス定義ファイルに記述します。

   アクセス定義ファイルの作成方法については、「2.　アクセス定義ファイルの作成」を参照してください。

3. **jamscript（アクセス定義ファイルの実行）コマンド**を実行します。アクセス定義ファイルで指定された条件で、処理が実行されます。

   なお、アクセス定義ファイルに記述した処理は、Windows のタスクスケジューラに登録して、定期的に実行させることもできます。処理の内容に応じて使い分けてください。

jamscript コマンドの実行方法については「3.1　コマンドでの実行」を、タスクの登録については「3.2　タスクでの実行」を参照してください。

2 アクセス定義ファイルの作成

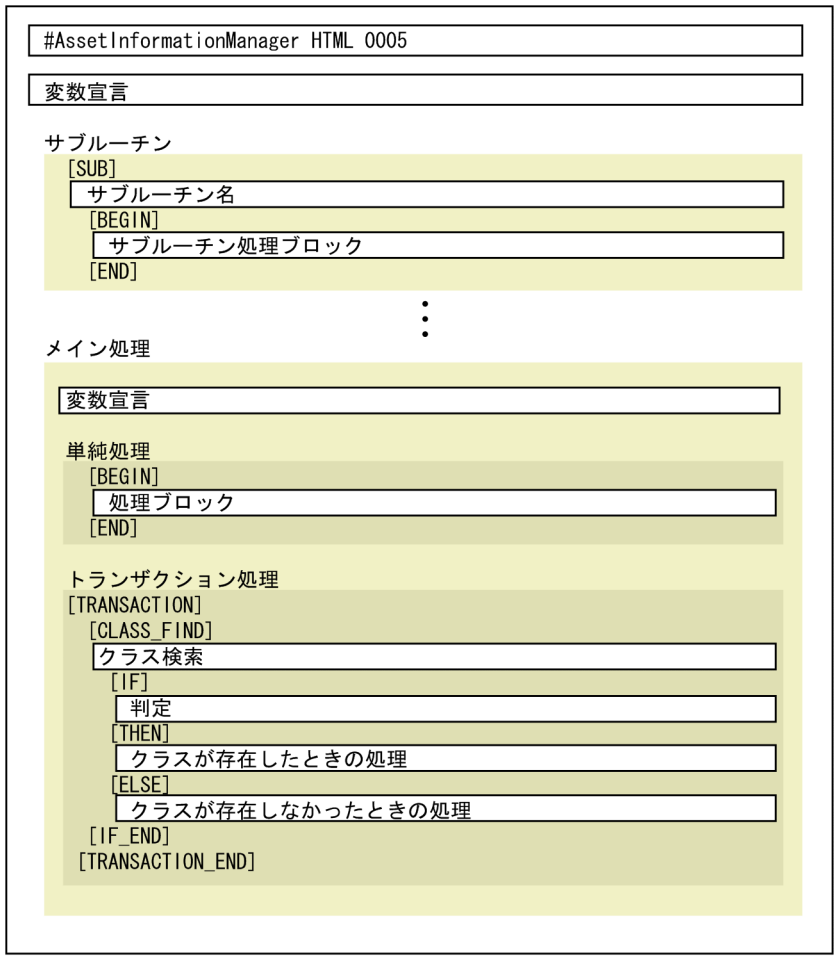
この章では、アクセス定義ファイルの作成方法について説明します。また、アクセス定義ファイルの記述例についても紹介します。

2.1　アクセス定義ファイルの記述形式

アクセス定義ファイルは、タグ、変数、演算子および組み込み関数を使用して作成します。

アクセス定義ファイルの記述形式を次の図に示します。サブルーチンは、必ずメイン処理の前に記述してください。

図 2-1　アクセス定義ファイルの記述形式



2.2　記述方法

この節では、アクセス定義ファイルの記述規則について説明します。また、アクセス定義ファイルに記述できる処理として、ディレクトリ情報の操作についても説明します。

2.2.1　記述規則

ここでは、アクセス定義ファイルの記述規則を示します。

- アクセス定義ファイルは、最大 2,097,152 バイト（2 メガバイト）以内で作成してください。
- アクセス定義ファイルを作成する場合は、シフト JIS コードで記述してください。
- 改行コードは、CRLF（¥r¥n）を使用してください。
- 1 行目には、スクリプトヘッダを記述します。行頭・行末のスペースおよびタブは指定できません。
- 2 行目以降では、行頭・行末のスペースおよびタブを使用できます。また、「#（シャープ）」に続けてコメントを記述できます。
- 定数は、「'（シングルクォーテーション）」で囲んで指定してください。
- 「'（シングルクォーテーション）」を文字定数として指定する場合は、「'」を 2 つ指定してください。
- 2 行目以降で「#%include」に続けてファイル名を指定すると、そのファイルの内容を取り込むことができます。ただし、取り込めるファイルは、アクセス定義ファイルだけです。

注意事項

jamscript コマンドを複数の PC から同一のデータベースに対して実行すると、処理の内容によっては、データの不整合が発生する場合がありますので、アクセス定義ファイルの記述にはご注意ください。

2.2.2　スクリプトヘッダの設定

アクセス定義ファイルの先頭行に、スクリプトヘッダを設定します。スクリプトヘッダは、必ず設定してください。スクリプトヘッダを設定しないと、jamscript コマンド実行時にアクセス定義ファイルとして見なされないため、処理が実行されませんので注意してください。

スクリプトヘッダは、1 カラム目から次のように記述します。

```
#AssetInformationManager△HTML△0005
```

スクリプトヘッダの 4 けたの数字「0005」は、スクリプトのバージョンを示しています。

2.2.3　変数の記述規則

ここでは、アクセス定義ファイルで使用する変数について説明します。アクセス定義ファイルで使用できる変数は、次の種類です。

- 変数
- 配列変数

なお、上記の変数を区別する必要がない場合は、これらの変数を総称して「変数」と呼びます。

(1)　変数に使用できる文字

変数には、半角英数字および「_（アンダーバー）」を使用します。ただし、変数の先頭に半角数字は使用できません。また、変数は、英文字の大小文字を区別して使用してください。

(2)　変数に使用できない文字列

次の予約文字列は、変数に使用できません。

NORMAL、ERROR、NODATA、MULTI、FLUSH、RENEW、NEW、ADD、CRLF

(3)　変数の宣言

アクセス定義ファイルで変数および配列変数を使用するには、あらかじめその変数および配列変数を宣言しておく必要があります。変数の宣言には[VAR]タグ、配列変数の宣言には[ARRAY]タグを使用します。[VAR]タグ、および[ARRAY]タグの使用方法については、それぞれ「4.　アクセス定義ファイルに記述するタグ」の[VAR]（変数宣言）および[ARRAY]（配列変数宣言）を参照してください。

(4)　変数への代入

変数への代入は、「＝」演算子を使用します。

(a)　変数の場合

変数へ代入できるのは次のものです。

- 文字列
- 変数の値
- クラス.プロパティの値

変数の代入文を記述できるのは、次のどちらかのタグです。ただし、クラス.プロパティの値を代入できるのは、[GET_VALUE]タグ内だけです。

- [SET_VALUE]タグ
- [CLASS_FIND]、[ASSOC_FIND]および[JOIN_FIND]タグ内の[GET_VALUE]タグ

(b)　配列変数の場合

配列変数は、組み込み関数$SETARRAY を使用して代入文を記述します。

(5)　変数の値の参照

配列変数の値を参照する場合は、次の形式で記述します。

```
DATA=$GETARRAY(配列変数の値1,配列変数の値2)
```

(6)　変数の有効範囲

変数、リスト変数、および配列変数は、その変数が宣言されている[BEGIN]タグ～[END]タグの範囲内で有効です。

2.2.4　演算子

ここでは、使用できる演算子を一覧で説明します。また、クラス.プロパティの値をあいまい検索する場合の指定方法についても説明します。

(1)　使用できる演算子

使用できる演算子には、文字列の結合など、2 つの文字列に対して操作を実行する二項演算子、右辺の演算結果を左辺に代入する代入演算子、クラス.プロパティの値や変数を左辺と右辺で比較する比較演算子の 3 種類があります。使用できる演算子をそれぞれ次の表に示します。

表 2-1　使用できる演算子

演算子		説明
二項演算子	＋	「＋」の直前の文字列に、直後の文字列を結合します。文字列には、定数、変数または組み込み関数を指定します。組み込み関数を指定した場合、その戻り値の文字列を結合します。「A ＋ B ＋ C」のように「＋」を複数指定して文字列を結合することもできます。
代入演算子	＝	右辺の定数や演算結果を左辺に代入します。右辺には、定数、変数または組み込み関数を指定します。組み込み関数を指定した場合は、その戻り値を代入します。
比較演算子	＝	左辺と右辺を比較し、値が等しい場合に真となります。
	!=	左辺と右辺を比較し、値が異なる場合に真となります。
	＞	左辺と右辺を比較し、左辺が右辺より大きい場合に真となります。
	>=	左辺と右辺を比較し、左辺が右辺以上の場合に真となります。
	＜	左辺と右辺を比較し、左辺が右辺より小さい場合に真となります。
	<=	左辺と右辺を比較し、左辺が右辺以下の場合に真となります。
	<>	「!=」と同じ判定内容の演算子です。左辺と右辺を比較し、値が異なる場合に真となります。

演算子		説明
比較演算子	LIKE※	あいまい検索の条件式を記述するための演算子です。左辺のクラス.プロパティの値と右辺のワイルドキャラクタを含む文字列を比較し、一致する場合に真となります。指定できるワイルドキャラクタについては、「(2)　あいまい検索の条件式に指定するワイルドキャラクタ」を参照してください。
	NOT_LIKE※	あいまい検索の条件式を記述するための演算子です。左辺のクラス.プロパティの値と右辺のワイルドキャラクタを含む文字列を比較し、一致しない場合に真となります。指定できるワイルドキャラクタについては、「(2)　あいまい検索の条件式に指定するワイルドキャラクタ」を参照してください。

注※

[IF]タグ内では使用できません。

(2)　あいまい検索の条件式に指定するワイルドキャラクタ

比較演算子「LIKE」または「NOT_LIKE」を使用して、クラス.プロパティの値をあいまい検索する場合、右辺の文字列に、ワイルドキャラクタである「%（パーセント）」または「_（アンダーバー）」を指定します。

それぞれのワイルドキャラクタの使用方法について次に説明します。

- %（パーセント）

  指定された位置に 0 文字以上の任意の文字が含まれるものが検索されます。指定例を次に示します。

  - クラス.プロパティの値の文字列が「ABC」で始まるものを検索したい場合（前方一致検索）、「ABC%」と指定します。
  - クラス.プロパティの値の文字列が「ABC」で終わるものを検索したい場合（後方一致検索）、「%ABC」と指定します。
  - クラス.プロパティの値の文字列に「ABC」が含まれるものを検索したい場合、「%ABC%」と指定します。この場合、任意の位置に「ABC」が含まれる文字列が検索されます。

- _（アンダーバー）

  指定された位置に任意の 1 文字が含まれるものが検索されます。例えば、「AAABC」のように 3 文字目以降が「ABC」である 5 文字のものを検索したい場合、「__ABC」のように「_」を連続して 2 つ指定します。この場合、「BBABC」、「CCABC」なども検索されます。

なお、ワイルドキャラクタ（「%」および「_」）を使用しない場合、検索は完全一致で判定されます。「LIKE」は「=」、「NOT_LIKE」は「!=」と同じ判定結果となります。

メモ

「%」および「_」を含んだ文字列を検索する場合は、「/%」、「/_」のように「/（スラッシュ）」を前に付けて指定します。また、「/」を含んだ文字列を検索する場合は、「//」のように「/（スラッシュ）」を 2 つ連続して指定します。

2.2.5　組み込み関数の記述規則

組み込み関数で、関数の引数として指定できる書式を次に示します。

- **変数**
  変数、配列変数など、アクセス定義ファイル内や外部で定義した名称を指定します。
- **定数**
  定数には、「'（シングルクォーテーション）」で囲まれた文字列を指定します。
- **数字**
  半角の 0～9 で構成される文字列を指定します。

2.2.6　ディレクトリ情報の操作

組み込み関数$LDAPACS を使用すると、ディレクトリサービスへの接続認証、検索、エントリの取得、属性の取得などができます。

ここでは、組み込み関数$LDAPACS で使用できる関数およびオブジェクト操作規則について説明します。

(1)　組み込み関数$LDAPACS で使用できる関数

組み込み関数$LDAPACS で使用できる関数と機能を次の表に示します。

表 2-2　組み込み関数$LDAPACS で使用できる関数一覧

関数名	機能
CONNECT	ディレクトリサービスへの接続認証
CONVERT	ディレクトリ情報の検索用文字列への変換
DISCONNECT	ディレクトリサービスとの接続解除
FIRSTENTRY	最初に検索されたエントリの取得
FREEENTRY	エントリの解放
FREERESULT	検索結果の解放
GETDN	エントリの DN の取得
NEXTENTRY	2 番目以降に検索されたエントリの取得
SEARCH	ディレクトリサービスの検索
SELECTVALUE	属性値の取得

各関数の機能、記述形式、パラメーター、および戻り値については、「5.　アクセス定義ファイルに記述する組み込み関数」の$LDAPACS（ディレクトリ情報へのアクセス関数）を参照してください。

(2) オブジェクトの操作規則

エントリの取得

- 最初に検索されたエントリを取得しないで、2 番目以降のエントリは取得できません。
- 同じエントリは取得できません。
- エントリの内容はスクリプトから変更、削除、および追加できません。

属性値の取得

- 属性名称を指定して該当する属性値を取得できます。
- 属性値はスクリプトから変更、削除、および追加できません。
- 同一属性名称に対して複数の値がある場合には、SELECTVALUE を複数記述することで、値を順次指定できます。

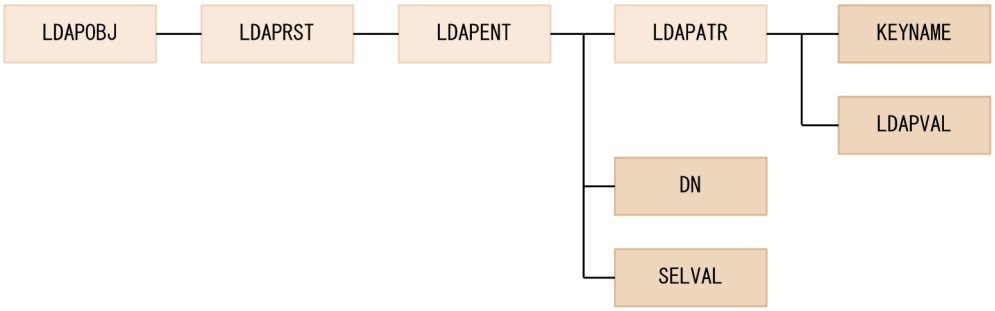
オブジェクト変数の使用と参照

- 一度解放したオブジェクト変数を、再度使用および参照することはできません。

(3) オブジェクトのメモリ管理構造

オブジェクトのメモリ管理構造を次の図に示します。

図 2-2 オブジェクトのメモリ管理構造



LDAPOBJ、LDAPRST、LDAPENT、および LDAPATR は、その下位すべてのオブジェクトを管理します。そのため、LDAPOBJ、LDAPRST、LDAPENT、および LDAPATR を解放すると、その下位のすべてのオブジェクトも解放されます。なお、KEYNAME、LDAPVAL、DN、SELVAL などの最下位文字列は、オブジェクト自身の解放はできません。

(4) 記述例

組み込み関数$LDAPACS の記述例を次に示します。

```
[VAR]
STATUS
MSG
HOST
PORT
FILTER
```

```
BASE
SCOPE
FIRST
LDOBJ
LDRST
LDENT
DN
NAME

[SET_VALUE]
  HOST = 'localhost'
  PORT = '389'
  BASE = 'ou=people,o=xxxxxxx.co.jp'
  SCOPE= 'LDAP_SCOPE_ONELEVEL'

[SET_VALUE]
  $LDAPACS('CONNECT',LDOBJ,HOST,PORT,'','')           # CONNECT
  STATUS = $GETSTATUS()

  [SET_VALUE]
    FILTER = '(&(objectclass=*)(title;lang-ja='
    FILTER = FILTER+$LDAPACS('CONVERT','主任')        # CONVERT
    FILTER = FILTER+'))'
    # FILETER=(&(objectclass=*)(title;lang-ja=¥E4¥B8¥BB¥E4¥BB¥BB))

    $LDAPACS('SEARCH',LDRST,LDOBJ,BASE,FILTER,SCOPE)  # SEARCH
    FIRST = 1

    [DO]
      [IF]
        FIRST = 1
        [THEN]
          [SET_VALUE]
            $LDAPACS('FIRSTENTRY',LDENT,LDRST)        # GET FIRST ENTRY
            STATUS = $GETSTATUS()
            FIRST = 0
        [ELSE]
          [SET_VALUE]
            $LDAPACS('NEXTENTRY',LDENT,LDRST)         # GET NEXT ENTRY
            STATUS = $GETSTATUS()
      [IF_END]

      [IF]
        STATUS = NORMAL
        [THEN]
          [SET_VALUE]
            $LDAPACS('GETDN',DN,LDENT)                # GET DN
            $LDAPACS('SELECTVALUE',NAME,LDENT,'cn')   # GET VALUE OF CN
            MSG='DN ['+DN+'] is '+NAME
            $ECHO(MSG)
            $LDAPACS('FREEENTRY',LDENT)               # FREE ENTRY OBJECT
        [ELSE]
          [SET_VALUE]
            $BREAK()
        [IF_END]
    [DO_END]

  [SET_VALUE]
```

```
    $LDAPACS('FREERESULT',LDRST)                    # FREE SEARCH OBJECT

[SET_VALUE]
  $LDAPACS('DISCONNECT',LDOBJ)                      # FREE LDAP OBJECT
```

2.3　アクセス定義ファイルの記述例

この節では、次の 2 つの場合を例に、アクセス定義ファイルの記述例を紹介します。

- 資産の状態に応じて資産情報を更新・削除する場合
- 許可外ソフトウェアがインストールされている資産をリストアップする場合

2.3.1　状態に応じて資産情報を更新・削除する場合の記述例

資産の状態を判定し、各状態に応じて資産情報を次のように更新・削除する場合の例を示します。

- 状態が「抹消」のハードウェア資産、および関連するすべての情報を削除します。
- 状態が「廃棄」（コードの範囲が 500～719）の状態に対して次の更新・削除をします。
  - ネットワーク情報の IP アドレス項目をクリアする。
  - インストールソフトウェア情報を削除する。

```
#AssetInformationManager HTML 0005

#==============================================================================
# 変数定義
#==============================================================================
[VAR]
  STATUS
  DUMMY
  ECHOMSG
  MSG
  ASSET_ID
  ASSET_NO
  ASSET_ST
  NETWORK_ID
  IPADDR
  IPKIND
  WORK

#==============================================================================
# メッセージ出力ルーチン
#==============================================================================
[SUB]
  ECHO

  [IF]
    MSG = '1'
    [THEN]
    [SET_VALUE]
      $ECHO(ECHOMSG)
  [IF_END]

[SUB_END]

#==============================================================================
```

```
# ハードウェア資産情報の削除（メイン）
#==========================================================================
[BEGIN]

  [SET_VALUE]
  MSG = $GETSESSION('MSG')                          # メッセージ出力要否の取得

  [TRANSACTION]
    [ASSET_ITEM_LOOP]
      [CLASS_FIND]
        AssetInfo
      [FIND_DATA]
        (AssetInfo.AssetKind   = '001') and      # ハードウェア資産
        (AssetInfo.AssetStatus >= '500')         # 資産状態が「運用」ではない資産
      [GET_VALUE]
        ASSET_ID = AssetInfo.AssetID
        ASSET_NO = AssetInfo.AssetNo
        ASSET_ST = AssetInfo.AssetStatus

      [IF]
        (ASSET_ST = '999')
        [THEN]                                   # 資産状態が「抹消」の資産情報を削除
          [DELETE]
            AssetInfo
          [DATA]
            AssetInfo.AssetID           = ASSET_ID

          [SET_VALUE]
            ECHOMSG = '資産番号['+ASSET_NO+' (ID:'+ASSET_ID+')]を削除しました'
            $GOSUB(ECHO)

        [ELSE]                           # 資産状態が「抹消」でない場合（「運用」でもない）

          [ASSET_ITEM_LOOP]          # ネットワーク情報のIPアドレス項目をクリア
            [CLASS_FIND]
              NetworkInfo
            [FIND_DATA]
              (NetworkInfo.AssetID = ASSET_ID)
            [GET_VALUE]
              NETWORK_ID = NetworkInfo.NetworkID
              IPADDR     = NetworkInfo.IPAddress
              IPKIND     = NetworkInfo.IPAddressKind

            [UPDATE]
              NetworkInfo
            [DATA]
              NetworkInfo.AssetID       = ASSET_ID
              NetworkInfo.NetworkID     = NETWORK_ID
              NetworkInfo.IPAddressKind = '002'
              NetworkInfo.IPAddress     = ''
            [SET_VALUE]
              ECHOMSG = '['+ASSET_ID+']ネットワーク情報['+NETWORK_ID+' '+IPADDR+']をクリ
アしました'
              $GOSUB(ECHO)

              $SETSTATUS('NORMAL')
          [ASSET_ITEM_LOOP_END]
```

```
          [ASSET_ITEM_LOOP]          # インストールソフトウェア情報をすべて削除
            [CLASS_FIND]
              InstalledInfo
            [FIND_DATA]
              (InstalledInfo.AssetID = ASSET_ID)
            [GET_VALUE]
              WORK = InstalledInfo.InstalledID
            [DELETE]
              InstalledInfo
            [DATA]
              InstalledInfo.AssetID           = ASSET_ID
              InstalledInfo.InstalledID       = WORK
              InstalledInfo.CreationClassName = 'InstalledInfo'
            [SET_VALUE]
              ECHOMSG = '['+ASSET_ID+']インストール情報['+WORK+']を削除しました'
              $GOSUB(ECHO)

              $SETSTATUS('NORMAL')
          [ASSET_ITEM_LOOP_END]

      [IF_END]

    [SET_VALUE]
      $SETSTATUS('NORMAL')
    [ASSET_ITEM_LOOP_END]

  [SET_VALUE]
    $SETSTATUS('NORMAL')
  [TRANSACTION_END]

[END]
#================================================================================
```

2.3.2 許可外ソフトウェアがインストールされている資産をリストアップする場合の記述例

許可外ソフトウェアがインストールされている資産と、そのソフトウェアをリストアップしてファイルに出力する場合の記述例を次に示します。この例では、リストを作成するために、[JOIN_FIND]タグを使って複数クラスを結合して検索しています。

- 出力ファイルの内容例

  この記述例の処理で出力されるファイルの内容例を次に示します。

```
"ソフトウェア名","バージョン","資産番号","部署","利用者"
"SoftwareA","0100","0000000001","本社/営業部/営業1課","ユーザ1"
"SoftwareA","0101","0000000001","本社/営業部/営業1課","ユーザ1"
"SoftwareB","0100","0000000001","本社/営業部/営業1課","ユーザ1"
"SoftwareA","0100","0000000002","本社/営業部/営業2課","ユーザ2"
```

```
#AssetInformationManager HTML 0005

#================================================================================
```

```
# 変数定義
#================================================================================
[VAR]
  STATUS
  DUMMY
  WORK
  ECHOMSG
  MSG
  FILENAME

[SET_VALUE]
  MSG = $GETSESSION('MSG')

#================================================================================
# 許可外ソフトウェアのインストール情報をCSV出力（メイン）
#================================================================================
[BEGIN]

  [VAR]
    ASSET_NO
    GROUP
    USER
    SOFTNAME
    SOFTVR
    FH
    LINE
  [ARRAY]
    OUTLINE

  [SET_VALUE]
    FILENAME = $GETSESSION('CSV')
    FH = $FILEOPEN(FILENAME, RENEW)

    $SETARRAY(OUTLINE,'ソフトウェア名')
    $SETARRAY(OUTLINE,'バージョン')
    $SETARRAY(OUTLINE,'資産番号')
    $SETARRAY(OUTLINE,'部署')
    $SETARRAY(OUTLINE,'利用者')

    $FILEARRAY(FH, OUTLINE)
    $CLEARARRAY(OUTLINE)

    LINE = 0

  [TRANSACTION]
    [ASSET_ITEM_LOOP]
      [JOIN_FIND]
      [JOIN]
        joinassoc;InstalledListLink;
        joinfrom;InstalledList,InstalledInfo;
        jointype;INNER;
      [JOIN]
        joinassoc;InstalledLink;
        joinfrom;InstalledInfo,AssetInfo;
        jointype;INNER;
      [JOIN]
        joinfrom;AssetInfo,GroupInfo;
        jointype;OUTER;
```

```
        joinkey;AssetInfo.GroupID,GroupInfo.GroupID;
      [FIND_DATA]
        (InstalledList.InstalledPermit = '2')
      [GET_VALUE]
        SOFTNAME = InstalledList.InstalledName
        SOFTVR   = InstalledList.InstalledVersion
        ASSET_NO = AssetInfo.AssetNo
        GROUP    = GroupInfo.FullPathName
        USER     = AssetInfo.UserName

      [SET_VALUE]
        $SETARRAY(OUTLINE,SOFTNAME)
        $SETARRAY(OUTLINE,SOFTVR)
        $SETARRAY(OUTLINE,ASSET_NO)
        $SETARRAY(OUTLINE,GROUP)
        $SETARRAY(OUTLINE,USER)

        $FILEARRAY(FH, OUTLINE)
        $CLEARARRAY(OUTLINE)

        LINE = $ADD(LINE, 1)
    [ASSET_ITEM_LOOP_END]

  [SET_VALUE]
    $SETSTATUS('NORMAL')
  [TRANSACTION_END]

  [SET_VALUE]
    $FILECLOSE(FH)

  [IF]
    (LINE = 0)
    [THEN]
      [SET_VALUE]
        ECHOMSG = 'データはありませんでした'
        $ECHO(ECHOMSG)
        $EXIT(1)
    [ELSE]
      [SET_VALUE]
        ECHOMSG = LINE+'件のデータを出力しました'
        $ECHO(ECHOMSG)
  [IF_END]

[END]
```

3 アクセス定義ファイルの実行

この章では、作成したアクセス定義ファイルを実行する方法について説明します。

3.1　コマンドでの実行

この節では、アクセス定義ファイルに定義した処理を、コマンドで実行する方法について説明します。

3.1.1　コマンドを実行する前に

ここでは、資産管理サーバの運用コマンドを使う前に知っておきたいことについて説明します。

(1)　コマンドの実行手順

コマンドは、次の方法で実行してください。

1. Administrators 権限を持つユーザでログオンする。
2. コマンドプロンプトを開き、コマンドを入力して［Enter］を押す。
   コマンドが実行されます。

(2)　コマンド実行時の注意事項

コマンドのオプションの引数で、スペースを含む文字列を指定する場合、「"（ダブルクォーテーション）」で囲む必要があります。

（例）

```
jamscript -f ”c:¥example¥accessdef.txt” -s ”CSV =c:¥temp¥data.csv”
```

(3)　コマンドの実行ファイルの格納先

コマンドの実行ファイルの格納先を次に示します。

Asset Console のインストール先フォルダ¥exe

(4)　処理が中断される場合

次のエラーが発生した場合、アクセス定義ファイルの処理を中断します。

- アクセス定義ファイルの構文に誤りがある。
- クラスの新規登録・削除で、代入文にキープロパティが設定されていない。
- クラスの新規登録・削除で、代入文に新規登録時に必ず指定するプロパティ※が設定されていない。
- 代入文で指定した値が、プロパティの属性の範囲外。
- 資産管理データベースにアクセスするタグに対して、[TRANSACTION]タグが定義されていない。
- 組み込み関数の引数の形式に誤りがある。

- 組み込み関数の引数に指定できない変数の種類や、指定できる範囲外の文字や数値が指定されている。
- 組み込み関数の実行時にメモリ不足が発生した。
- そのほかの資産管理データベースへのアクセスエラー。

注※

新規登録時に必ず指定するプロパティについては、マニュアル「JP1/IT Desktop Management 2 - Asset Console 構築・運用ガイド」を参照してください。

3.1.2 jamscript（アクセス定義ファイルの実行）コマンド

(1) 機能

アクセス定義ファイルの定義内容に従って、資産情報を一括で登録、更新および削除します。アクセス定義ファイルの定義内容に従って、CSV ファイルの情報を資産管理データベースに取り込んで処理したり、資産管理データベースの情報を検索して CSV ファイルに出力したりできます。

(2) 形式

```
jamscript -f アクセス定義ファイル
          (-s 変数名＝値（-s 変数名＝値）)
          (-bp ベースパス名)
          (-c)
```

(3) オプション

-f アクセス定義ファイル

目的に応じたアクセス定義ファイルのパスを指定します。フルパスと相対パスのどちらで指定してもかまいません。相対パスの場合は、ベースパス名が付いたフォルダを基点にして指定してください。このオプションは必ず指定します。

-s 変数名＝値

セッション情報として使用する変数名と変数の値を指定します。このほか、このオプションは、アクセス定義ファイルに定義された処理の条件（検索条件など）を変更したい場合に、値を指定するときに使用します。また、アクセス定義ファイルで、組み込み関数$GETSESSION を使用すると変数に値を代入できます。

組み込み関数$GETSESSION の詳細については、「5. アクセス定義ファイルに記述する組み込み関数」の$GETSESSION（セッション情報取得関数）を参照してください。

-bp ベースパス名

アクセス定義ファイルの基点となるパス名を指定します。ベースパス名はフルパスで指定します。このオプションは省略できます。このオプションを省略した場合は、Asset Console のインストール先フォルダ¥scriptwork が設定されます。

-c

jamscript コマンドを実行しないで、コマンドの構文解析だけを行います。

(4) 戻り値

次の戻り値を返します。

戻り値	内容
0	正常終了。
11	コマンドオプションの書式に誤りがあります。
21	アクセス定義ファイルが存在しません。
31	メモリが不足しています。
32	アクセス定義ファイルの実行に必要な環境が設定されていません。
34	アクセス定義ファイルに誤りがあります。
52	ユーザによってキャンセルされました。
101 以上	そのほかのエラーでコマンドが終了しました。

(5) 実行例

```
jamscript -f ”c:¥example¥accessdef.txt” -s ”CSV =c:¥temp¥data.csv”
```

(6) 注意事項

-bp オプションでベースパス名を指定した場合は、指定したフォルダの直下に「CSV」フォルダを作成してください。指定したフォルダの直下に「CSV」フォルダがないと、コマンド実行中にエラーになる場合があります。

3.2　タスクでの実行

この節では、アクセス定義ファイルに定義した処理を、Windows のタスクスケジューラのタスクによって実行する方法を説明します。

Windows Server 2008 R2 の場合を例に、タスクの登録手順を次に示します。

1. Windows のタスクスケジューラで、「基本タスクの作成」をダブルクリックする。
2. 表示されたタスクウィザードに従って、各項目を指定する。
   実行するプログラムには、jamscript コマンドの実行ファイル（jamscript.exe）を指定してください。
   jamscript コマンドの実行ファイルの格納先を次に示します。
   Asset Console のインストール先フォルダ¥exe
   また、ユーザ名には、Administrators 権限を持つユーザ名を指定してください。
3. 「[完了] をクリックしたときに、このタスクの［プロパティ］ダイアログを開く」チェックボックスをチェックして、[完了] ボタンをクリックする。
4. 表示されたダイアログで、「タスク」タブを選択する。
5. 「実行するファイル名」に、-f オプションでアクセス定義ファイルの格納場所を追加する。
   そのほかのオプションについては、「3.1.2　jamscript（アクセス定義ファイルの実行）コマンド」を参照してください。
6. [OK] ボタンをクリックする。
   ダイアログが閉じて、指定したアクセス定義ファイルを実行するタスクが登録されます。

4 アクセス定義ファイルに記述するタグ

この章では、アクセス定義ファイルに記述するタグの詳細について説明します。

アクセス定義ファイルに記述するタグ一覧

ここでは、アクセス定義ファイルに記述するタグの内容および種類について次の表に示します。

表 4-1　タグ一覧

種類	内容	記述するタグ
処理の流れの制御	CSV ファイル情報取得開始、終了	[CSV_FILE_LOOP] [CSV_COLUMN_NAME] [CSV_FILE_LOOP_END]
	ブロック	[BEGIN] [END]
	条件分岐（IF）	[IF] [THEN] ([ELSEIF][THEN]) ([ELSE]) [IF_END]
	条件分岐（SWITCH）	[SWITCH] [CASE] ([DEFAULT]) [SWITCH_END]
	クラス検索ループ	[ASSET_ITEM_LOOP] [ASSET_ITEM_LOOP_END]
	無限ループ	[DO][DO_END]
	サブルーチン	[SUB][SUB_END]
	解析再実行（処理の動的生成）	[EVALUATE][EVALUATE_END]
トランザクション定義	トランザクション範囲定義	[TRANSACTION] [TRANSACTION_END]
変数およびカウンタ	変数宣言	[VAR]
	配列変数宣言	[ARRAY]
	変数代入	[SET_VALUE]
クラスの検索およびプロパティに設定した値の取得	オブジェクトクラスの検索	[CLASS_FIND] [FIND_DATA][GET_VALUE] ([ORDER_ASC][ORDER_DESC])
	アソシエーションクラスの検索	[ASSOC_FIND] [CLASS1] [CLASS2] [FIND_DATA][GET_VALUE] ([ORDER_ASC][ORDER_DESC])

種類	内容	記述するタグ
クラスの検索およびプロパティに設定した値の取得	複数クラスの結合検索	[JOIN_FIND] [JOIN][FIND_DATA][GET_VALUE]（[ORDER_ASC][ORDER_DESC]） （[DUPLICATE]）
基点となるクラスの検索結果に対するデータ作成処理	オブジェクトクラスの新規作成	[APPEND] [DATA]
	オブジェクトクラスの更新	[UPDATE] [DATA]
	オブジェクトクラスの削除	[DELETE] [DATA]
	アソシエーションクラスの新規作成	[APPEND_ASSOC] [CLASS1] [DATA] [CLASS2] [DATA]
	アソシエーションクラスの削除	[DELETE_ASSOC] [CLASS1] [DATA] [CLASS2] [DATA]

（凡例）

（）内のタグは省略可

アクセス定義ファイルに記述するタグの詳細

アクセス定義ファイルに記述するタグの詳細は、これ以降、基本的には次に示す形式で説明していきます。なお、タグの説明は、タグ名のアルファベット順になっています。

タグ

記述するタグとタグの機能を説明しています。

形式

タグの記述形式を説明しています。

指定する値

タグに指定できる値を説明しています。

終了状態

タグで記述した処理の終了状態と、各終了状態の表す内容を説明しています。

記述例

タグの記述例を示しています。

[APPEND]（オブジェクトクラスの新規作成）

オブジェクトクラスを新規作成します。

形式

```
[APPEND]
  オブジェクトクラス名
  [DATA]
    代入文
```

指定する値

- **オブジェクトクラス名**

  新規で作成するオブジェクトクラス名を記述します。

- **代入文**

  プロパティに代入する値を記述します。

  新規登録時に指定が必要なプロパティおよびキープロパティは、必ず設定します。ただし、「CreationClassName」は省略できます。

  新規登録時に指定が必要なプロパティについては、マニュアル「JP1/IT Desktop Management 2 - Asset Console 構築・運用ガイド」を参照してください。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	キーがすでに登録されている
MULTI	－

（凡例）

－：該当しない

記述例

オブジェクトクラス「AssetInfo」を新規作成する場合の記述例を次に示します。

```
[APPEND]
  AssetInfo
[DATA]
  AssetInfo.AssetID       = '10000'
  AssetInfo.AssetNo       = 'R11111'
  AssetInfo.AssetWorkKind = '001'
```

```
  AssetInfo.AssetStatus   = '002'
  AssetInfo.AssetKind     = '001'
  AssetInfo.AssetBranchNo = '0'

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  (STATUS != NORMAL)
  [THEN]
    [SET_VALUE]
      MSG = 'APPEND(' +STATUS+ ')'
      $ECHO(MSG)
[IF_END]
```

[APPEND_ASSOC]（アソシエーションクラスの新規作成）

アソシエーションクラスを使って、オブジェクトクラス同士を関連づけます。

形式

```
[APPEND_ASSOC]
  アソシエーションクラス名
  [CLASS1]
    オブジェクトクラス名
    [DATA]
      代入文
  [CLASS2]
    もう一方のオブジェクトクラス名
    [DATA]
      代入文
```

指定する値

- **アソシエーションクラス名**

  新規で作成するアソシエーションクラス名を記述します。

- **オブジェクトクラス名**

  アソシエーションクラスで関連づけられている、オブジェクトクラス名を記述します。

- **代入文**

  オブジェクトクラス同士を関連づけるための情報を、プロパティへの代入文で記述します。

- **もう一方のオブジェクトクラス名**

  アソシエーションクラスで関連づけられている、もう一方のオブジェクトクラス名を記述します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	次のどちらかを示す • キーがすでに登録されている • 関係を結ぶオブジェクトクラスのデータが存在しない
MULTI	－

（凡例）

－：該当しない

注意事項

関連を設定するためには、対象となるオブジェクトクラスが登録されている必要があります。

記述例

アソシエーションクラス「MemberLink」を使用して、オブジェクトクラス「UserInfo」と「GroupInfo」を関連づけて、部署にユーザを追加する場合の記述例を次に示します。

```
[APPEND_ASSOC]
  MemberLink
[CLASS1]
  UserInfo
[DATA]
  UserInfo.UserID = 'user1'
[CLASS2]
  GroupInfo
[DATA]
  GroupInfo.GroupID = '11000000'

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS != NORMAL
[THEN]
  [SET_VALUE]
    MSG = 'APPEND_ASSOC(' +STATUS+ ')'
    $ECHO(MSG)
[IF_END]
```

[ARRAY]（配列変数宣言）

配列の変数名を宣言します。

形式

```
[ARRAY]
  変数名
```

指定する値

- 変数名

  宣言する配列の変数名を指定します。

記述例

配列の変数「ARY」を宣言する場合の記述例を次に示します。

```
[ARRAY]
  ARY
```

[ASSET_ITEM_LOOP]（クラス検索ループ）

クラス検索ループを指定します。クラス検索ループは、検索する情報 1 件につき 1 つ指定します。クラス検索ループでは、指定した条件に一致するすべての件数分ループします。

クラスの検索条件は、[CLASS_FIND]タグ、[ASSOC_FIND]タグおよび[JOIN_FIND]タグを組み合わせて使用します。これらのタグの詳細については、[CLASS_FIND]（オブジェクトクラスの検索）、[ASSOC_FIND]（アソシエーションクラスの検索）および[JOIN_FIND]（複数クラスの結合検索）を参照してください。

形式

```
[ASSET_ITEM_LOOP]
  [CLASS_FIND]
  クラスに対する検索条件
   ・・・
  [BEGIN]
   検索した結果に対する処理内容
   ・・・
  [END]
[ASSET_ITEM_LOOP_END]
```

指定する値

- **クラスに対する検索条件**

  クラスを検索するための条件を記述します。

- **検索した各クラスに対する処理内容**

  検索条件に一致したクラスに対する処理内容を記述します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	－
NODATA	データ終了
ERROR	－
MULTI	－

（凡例）

－：該当しない

注意事項

[ASSET_ITEM_LOOP]タグの処理を終了するかどうかは、ループ処理の終了タグである[ASSET_ITEM_LOOP_END]で参照した終了状態で判定されます。終了状態が「NORMAL」の場合は処

理を継続して、「NORMAL」以外のときは処理を終了します。これは、処理の途中でエラーが発生したら、その時点で処理を中断する場合に有効です。

データがなくなるまで処理を継続させる場合は、[ASSET_ITEM_LOOP_END]タグで終了するかどうかを判定する前に、組み込み関数$SETSTATUS で「NORMAL」を明示的に指定することをお勧めします。

アクセス定義ファイルの終了状態は、タグや組み込み関数の実行によって随時更新されます。
[ASSET_ITEM_LOOP]タグ中に終了状態の更新を伴う処理を複数指定した場合、最後に指定した終了状態を基に処理を終了するかどうかが判定されます。

記述例

資産 ID「10000」のインストールソフトウェア名一覧を出力する場合の記述例を次に示します。

```
[ASSET_ITEM_LOOP]
  [ASSOC_FIND]
    InstalledListLink
  [FIND_DATA]
    InstalledInfo.AssetID = '10000'
  [CLASS1]
    InstalledInfo
  [CLASS2]
    InstalledList
  [GET_VALUE]
    INSTNAME = InstalledList.InstalledName
  [SET_VALUE]
    MSG = 'Installed Software Name :' +INSTNAME
    $ECHO(MSG)
    $SETSTATUS('NORMAL')
[ASSET_ITEM_LOOP_END]
```

実行結果

Installed Software Name : InstalledSoftware A

Installed Software Name : InstalledSoftware B

[ASSOC_FIND]（アソシエーションクラスの検索）

アソシエーションクラスを使用してクラスを検索します。

検索対象のクラスと一致した場合、そのプロパティ値を変数に代入します。また、基点クラスを見つけるために、クラスの検索を複数回実施する必要がある場合は、実行する回数分[ASSOC_FIND]タグを記述します。

形式

```
[ASSOC_FIND]
  アソシエーションクラス名
   ([FIND_DATA])
     検索条件式
  [CLASS1]
     オブジェクトクラス名
  [CLASS2]
     もう一方のオブジェクトクラス名
  [GET_VALUE]
     変数への代入文
   ([ORDER_ASC]または[ORDER_DESC])
     検索結果をソートするためのキー
```

指定する値

- **アソシエーションクラス名**

  検索対象となるアソシエーションクラス名を記述します。

- **オブジェクトクラス名**

  アソシエーションで関連づけられている、オブジェクトクラス名を記述します。

- **もう一方のオブジェクトクラス名**

  アソシエーションクラスで関連づけられている、もう一方のオブジェクトクラス名を記述します。

- **検索条件式**

  検索条件式を記述します。複数の検索条件を記述する場合は、演算子で結合します。検索条件式に使用できる演算子については、「2.2.4　演算子」を参照してください。

  なお、検索条件式を使用しない場合は、対応する[FIND_DATA]タグを省略できます。

- **変数への代入文**

  検索したクラスのプロパティ情報を、変数へ代入するための代入文を記述します。表示名を取得する場合は、代入文のクラス.プロパティに「@（単価記号）」を付けてください。

- **検索結果をソートするためのキー**

  検索結果をソートする場合は、ソートするためのキーをクラス.プロパティの形式で指定します。[ORDER_ASC]タグは昇順、[ORDER_DESC]タグは降順にソートします。省略した場合、[GET_VALUE]タグで指定したプロパティの昇順になります。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	検索条件を満たすデータがない
ERROR	－
MULTI	－

（凡例）

－：該当しない

記述例

ユーザ ID「user1」の所属している部署を取得する場合の記述例を次に示します。

```
[ASSOC_FIND]
  MemberLink
[FIND_DATA]
  UserInfo.UserID = 'user1'
[CLASS1]
  UserInfo
[CLASS2]
  GroupInfo
[GET_VALUE]
  FULLPATH = GroupInfo.FullPathName

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS = NORMAL
[THEN]
  [SET_VALUE]
    MSG = 'FullPathName :' +FULLPATH
    $ECHO(MSG)
[ELSE]
  [SET_VALUE]
    MSG = 'ASSOC_FIND (' +STATUS+ ')'
    $ECHO(MSG)
[IF_END]
```

[BEGIN]（ブロック）

ブロックを指定します。

形式

```
[BEGIN]
   処理
[END]
```

指定する値

- 処理

  ブロックでの処理を定義します。

記述例

ブロックの記述例を次に示します。

```
[BEGIN]
  [SET_VALUE]
    $ECHO('Hello world')
[END]
```

[CLASS_FIND]（オブジェクトクラスの検索）

オブジェクトクラスを検索します。

検索対象のクラスと一致した場合、そのプロパティ値を変数に代入できます。また、基点クラスを見つけるために、クラスの検索を複数回実施する必要がある場合は、実行する回数分[CLASS_FIND]タグを記述します。

形式

```
[CLASS_FIND]
  検索するオブジェクトクラス名
   ([FIND_DATA])
     検索条件式
  [GET_VALUE]
     変数への代入文
   ([ORDER_ASC]または[ORDER_DESC])
     検索結果をソートするためのキー
```

指定する値

- **検索するオブジェクトクラス名**

  検索対象となるオブジェクトクラス名を記述します。

- **検索条件式**

  検索条件式を記述します。複数の検索条件を記述する場合は、演算子で結合します。

  検索条件式に使用できる演算子については、「2.2.4　演算子」を参照してください。

  なお、検索条件式を使用しない場合は、対応する[FIND_DATA]タグを省略できます。

- **変数への代入文**

  変数への代入文を記述します。表示名を取得する場合は、代入文のクラス.プロパティに「@（単価記号）」を付けてください。

- **検索結果をソートするためのキー**

  検索結果をソートする場合は、ソートするためのキーをクラス.プロパティの形式で指定します。[ORDER_ASC]タグは昇順、[ORDER_DESC]タグは降順にソートします。省略した場合、[GET_VALUE]タグで指定したプロパティの昇順になります。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	検索条件を満たすデータがない
ERROR	−

終了状態	内容
MULTI	－

（凡例）

－：該当しない

記述例

資産 ID「10000」の資産番号を出力する場合の記述例を次に示します。

```
[CLASS_FIND]
  AssetInfo
[FIND_DATA]
  (AssetInfo.AssetID  = '10000')
[GET_VALUE]
  ASSETNO     = AssetInfo.AssetNo
  ASSETSTATUS = AssetInfo.AssetStatus@

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS = NORMAL
  [THEN]
    [SET_VALUE]
      MSG = 'ASSETNO = '+ASSETNO+'('+ASSETSTATUS+')'
      $ECHO(MSG)
  [ELSE]
    [SET_VALUE]
      MSG = 'CLASS_FIND ('+STATUS+')'
      $ECHO(MSG)
[IF_END]
```

実行結果

ASSETNO = R11111(運用)

[CSV_FILE_LOOP]（CSV ファイルのデータ取得）

CSV ファイルからデータを取り込む処理の開始と終了を指定します。

形式

```
[CSV_FILE_LOOP]
  CSVファイル名
  [CSV_COLUMN_NAME]
    列名=列番号
  [BEGIN]
    取得したデータの処理
  [END]
[CSV_FILE_LOOP_END]
```

指定する値

- CSV ファイル名

  データを取得する CSV ファイルのファイル名を変数で指定します。ファイル名は、jamscript コマンドの-bp オプションで指定したベースパスを基点にした相対パスで指定します。-bp オプションを省略した場合は、Asset Console のインストール先フォルダ¥scriptwork が基点となります。

- 列名=列番号

  データを参照するための名称と CSV ファイルの列の番号を対応づけます。列番号は、先頭を 1 として指定します。

- 取得したデータの処理

  CSV ファイルのデータに対する処理を記述します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	−
NODATA	データ終了
ERROR	1 カラムのデータが 32 キロバイトを超える行がある
MULTI	−

（凡例）

−：該当しない

注意事項

指定した CSV ファイルが存在しない場合、処理は中断されます。

CSV ファイルからデータを取り込む処理を終了するかどうかは、ループ処理の終了タグである[CSV_FILE_LOOP_END]で参照した終了状態で判定されます。終了状態が「NORMAL」の場合は処理を継続して、「NORMAL」以外のときは処理を終了します。これは、処理の途中でエラーが発生したら、その時点で処理を中断する場合に有効です。

データがなくなるまで処理を継続させる場合は、[CSV_FILE_LOOP_END]タグで終了するかどうかを判定する前に、組み込み関数$SETSTATUS で「NORMAL」を明示的に指定することをお勧めします。

アクセス定義ファイルの終了状態は、タグや組み込み関数の実行によって随時更新されます。
[CSV_FILE_LOOP]タグ中に終了状態の更新を伴う処理を複数指定した場合、最後に指定した終了状態を基に処理を終了するか判定されます。

記述例

CSV ファイル「input.csv」の各行からデータを取得して、行番号と取得したデータの内容を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  FILENAME = 'input.csv'
  CNT = 1
[CSV_FILE_LOOP]
  FILENAME
  [CSV_COLUMN_NAME]
    COLUMN1 = 1
    COLUMN2 = 2
    COLUMN3 = 3
    [SET_VALUE]
      MSG = 'LINE('+CNT+') ['+COLUMN1+']['+COLUMN2+']['+COLUMN3+']'
      $ECHO(MSG)
      CNT = $ADD(CNT,1)

  [SET_VALUE]
    $SETSTATUS('NORMAL')
[CSV_FILE_LOOP_END]
```

実行結果

LINE(1) [0000000001][R11111][運用]

LINE(2) [0000000002][R22222][返却]

LINE(3) [0000000003][R33333][廃棄]

[DELETE]（オブジェクトクラスの削除）

オブジェクトクラスを削除します。該当するクラスの情報が複数検索された場合は、すべてのクラスを削除します。

形式

```
[DELETE]
  オブジェクトクラス名
  [DATA]
    プロパティ代入文
```

指定する値

- **オブジェクトクラス名**
  削除するオブジェクトクラス名を記述します。
- **プロパティ代入文**
  キープロパティについてのプロパティ代入文を記述します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	該当するオブジェクトクラスのデータがない
MULTI	－

（凡例）
－：該当しない

記述例

資産 ID「10000」の資産情報を削除する場合の記述例を次に示します。

```
[DELETE]
  AssetInfo
[DATA]
  AssetInfo.AssetID = '10000'

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS != NORMAL
[THEN]
  [SET_VALUE]
```

```
    MSG = 'DELETE (' +STATUS+ ')'
    $ECHO(MSG)
[IF_END]
```

[DELETE_ASSOC]（アソシエーションクラスの削除）

オブジェクトクラス同士の関連づけを解除します。

形式

```
[DELETE_ASSOC]
  アソシエーションクラス名
  [CLASS1]
    オブジェクトクラス名
    [DATA]
      プロパティ代入文
  [CLASS2]
    もう一方のオブジェクトクラス名
    [DATA]
      プロパティ代入文
```

指定する値

- アソシエーションクラス名

  関連づけを解除するためのアソシエーションクラス名を記述します。

- オブジェクトクラス名

  削除するアソシエーションクラスで関連づけられている、オブジェクトクラス名を記述します。

- プロパティ代入文

  アソシエーションでキーとして関連づけられている、すべてのプロパティの代入文を記述します。

- もう一方のオブジェクトクラス名

  削除するアソシエーションクラスで関連づけられている、もう一方のオブジェクトクラス名を記述します。

記述例

ユーザ ID「user1」と部署 ID「11000000」のアソシエーションを削除する場合の記述例を次に示します。

```
[DELETE_ASSOC]
  MemberLink
[CLASS1]
  UserInfo
[DATA]
  UserInfo.UserID = 'user1'
[CLASS2]
  GroupInfo
[DATA]
  GroupInfo.GroupID = '11000000'

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS != NORMAL
```

```
[THEN]
  [SET_VALUE]
    MSG = ’DELETE_ASSOC (’ +STATUS+ ’)’
    $ECHO(MSG)
[IF_END]
```

[DO]（無限ループ）

特定の情報に依存しない処理を繰り返します。ループを終了するには、組み込み関数$BREAK を指定します。

形式

```
[DO]
  データの処理内容
  $BREAK()
[DO_END]
```

指定する値

- データの処理内容

  各種データに対する処理を記述します。この記述内容に基づいて、処理を実行します。

記述例

無限ループの記述例を次に示します。この例では、無限ループ内で 10 回 COUNT の出力処理を繰り返したら終了することを示しています。

```
[SET_VALUE]
  CNT = 1
[DO]
  [IF]
    CNT > 10
    [THEN]
      [SET_VALUE]
        $BREAK()
  [IF_END]

  [SET_VALUE]
    MSG = 'COUNT =' +CNT
    $ECHO(MSG)
    CNT = $ADD(CNT,1)
[DO_END]
```

実行結果

COUNT = 1

COUNT = 2

COUNT = 3

COUNT = 4

COUNT = 5

COUNT = 6

COUNT = 7

COUNT = 8

COUNT = 9
COUNT = 10

[EVALUATE]（解析再実行）

処理ブロックを定義します。この処理ブロックでは、記述した内容の構文を処理実行時に解析します。変数を「%（パーセント）」で囲むと、実行時に変数の保持している内容を展開して実行できます。検索条件などを動的に生成するときに使用します。

形式

```
[EVALUATE]
  処理ブロック
[EVALUATE_END]
```

指定する値

- 処理ブロック

  データを処理する処理ブロックを記述します。処理ブロック内で各タグの構文は完結している必要があります。また、「%（パーセント）」で囲んだ変数にはタグを含んだ構文も指定できます。

記述例

例 1

検索条件部分を変数化することで、実行時に検索条件を決定する場合の記述例を次に示します。この例では、資産番号「R11111」および資産状態「002」の資産 ID を検索することを示しています。

```
[SET_VALUE]
  STATEMENT = '(AssetInfo.AssetNo =' 'R11111' ') and' +CRLF
  STATEMENT = STATEMENT + '(AssetInfo.AssetStatus =' '002' ')'

[EVALUATE]
  [CLASS_FIND]
    AssetInfo
  [FIND_DATA]
    %STATEMENT%
  [GET_VALUE]
    ASSETID = AssetInfo.AssetID

  [SET_VALUE]
    STATUS = $GETSTATUS()
[EVALUATE_END]
  [IF]
    (STATUS = NORMAL)
    [THEN]
      [SET_VALUE]
        MSG = 'ASSETID =' +ASSETID
        $ECHO(MSG)
    [ELSE]
      [SET_VALUE]
        MSG = 'CLASS_FIND (' +STATUS+ ')'
        $ECHO(MSG)
  [IF_END]
```

実行結果

ASSETID = 10000

例 2

この例では、ユーザ ID からユーザの名前を取得する処理を、変数に格納して検索することを示しています。

```
[SET_VALUE]
  STATEMENT =             '[CLASS_FIND]' + CRLF
  STATEMENT = STATEMENT + 'UserEntry'    + CRLF
  STATEMENT = STATEMENT + '[FIND_DATA]'  + CRLF
  STATEMENT = STATEMENT + 'UserID ='     + USERID + CRLF
  STATEMENT = STATEMENT + '[GET_VALUE]'  + CRLF
  STATEMENT = STATEMENT + 'USERNAME = UserEntry.UserName' + CRLF
[EVALUATE]
  %STATEMENT%
[EVALUATE_END]
```

[IF]（条件分岐）

条件を指定して処理を分岐します。

条件が真の場合は[THEN]以降の処理をします。条件が偽の場合は、[ELSEIF]タグを使用して、さらに処理を分岐することもできます。すべての条件が偽の場合は[ELSE]以降の処理を実行し、[ELSE]が定義されていない場合は、処理を実行しません。

[ELSEIF]タグはいくつでも指定できますが、条件値を複数の定数と比較する場合は、[SWITCH]タグを使用して値となる定数を指定する方が、条件分岐の階層が深くならないため見やすく記述できます。[SWITCH]タグの記述例については、[SWITCH]（条件分岐）を参照してください。

形式

```
[IF]
  条件1
  [THEN]
    条件1が真の場合の処理内容
 ([ELSEIF])
    条件2
   ([THEN])
    条件1が偽で、条件2が真の場合の処理内容
 ([ELSE])
    すべての条件が偽の場合の処理内容
[IF_END]
```

指定する値

- **条件 1、条件 2**

  処理を分岐するための条件を記述します。

- **条件 1 が真の場合の処理内容**

  条件 1 が真の場合の処理内容を記述します。

- **条件 1 が偽で、条件 2 が真の場合の処理内容**

  条件 1 が偽で、条件 2 が真の場合の処理内容を記述します。この記述は省略できます。記述を省略する場合は、対になっている条件の記述も省略してください。

- **すべての条件が偽の場合の処理内容**

  すべての条件が偽の場合の処理内容を記述します。この記述は省略できます。

記述例

例 1

資産番号「10000」の資産情報を検索して、データがあった場合（[THEN]）そのデータを削除し、データがなかった場合（[ELSE]）終了状態を表示する記述例を次に示します。

```
[CLASS_FIND]
  AssetInfo
```

```
[FIND_DATA]
  (AssetInfo.AssetNo = '10000')
[GET_VALUE]
  ASSETID = AssetInfo.AssetID

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS = NORMAL
  [THEN]
    [DELETE]
      AssetInfo
    [DATA]
      AssetInfo.AssetID = ASSETID
    [SET_VALUE]
      MSG = 'CLASS_FIND (' +STATUS+ ')'
      $ECHO(MSG)

  [ELSE]
    [SET_VALUE]
      MSG = 'CLASS_FIND (' +STATUS+ ')'
      $ECHO(MSG)
[IF_END]
```

例 2

次の処理を実行する記述例を示します。

- データを正常に更新できた場合（[THEN]）終了状態を表示する。
- すでにほかの制御で更新されている場合（[ELSEIF][THEN]）メッセージを出力する。
- データがない場合（[ELSE]）データを追加する。

```
[UPDATE]
  AssetInfo
[DATA]
  AssetInfo.AssetID = '10000'
  AssetInfo.AssetNo = '10000'
  AssetInfo.AssetKind = '001'
  AssetInfo.AssetBranchNo = 0
  AssetInfo.UpdateTime = _UpdateTime
[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS = NORMAL
    [THEN]
      [SET_VALUE]
        MSG = 'UPDATE (' +STATUS+ ')'
        $ECHO(MSG)
    [ELSEIF]
      STATUS = MULTI
        [THEN]
          [SET_VALUE]
            MSG = 'Asset number [10000] is updated already.'
            $ECHO(MSG)
    [ELSE]
      [APPEND]
        AssetInfo
```

```
      [DATA]
        AssetInfo.AssetID = '10000'
        AssetInfo.AssetNo = '10000'
        AssetInfo.AssetKind = '001'
        AssetInfo.AssetBranchNo = 0
      [SET_VALUE]
        MSG = 'UPDATE (' +STATUS+ ')'
        $ECHO(MSG)
[IF_END]
```

[JOIN_FIND]（複数クラスの結合検索）

複数のオブジェクトクラスを結合して検索します。

形式

- アソシエーションクラスを使用した結合

```
[JOIN_FIND]
  [JOIN]
  joinassoc;アソシエーションクラス名;
  joinfrom;オブジェクトクラス名,もう一方のオブジェクトクラス名;
  jointype;(OUTER|INNER)
  ([FIND_DATA])
  検索条件式
[GET_VALUE]
  変数への代入文
 ([ORDER_ASC]または[ORDER_DESC])
    検索結果をソートするためのキー
 ([DUPLICATE]※)
```

- 任意のプロパティをキーにした結合

```
[JOIN_FIND]
  [JOIN]
  joinfrom;オブジェクトクラス名,もう一方のオブジェクトクラス名;
  jointype;(OUTER|INNER);
  joinkey;結合キーとなるクラスのプロパティ名
  ([FIND_DATA])
  検索条件式
[GET_VALUE]
  変数への代入文
 ([ORDER_ASC]または[ORDER_DESC])
    検索結果をソートするためのキー
 ([DUPLICATE]※)
```

注※

[DUPLICATE]タグは、検索結果ですべてのプロパティが同一の場合、2つ目以降の結果は出力しない場合に指定します。

指定する値

- **結合情報**

  アソシエーションクラスを使用するか、または任意のキーとなるプロパティを使用して、結合するクラスを記述します。

- **検索条件式**

  検索条件式を記述します。複数の検索条件を記述する場合は、演算子で結合します。検索条件式に使用できる演算子については、「2.2.4　演算子」を参照してください。

  なお、検索条件式を使用しない場合は、対応する[FIND_DATA]タグを省略できます。

- **変数への代入文**

  検索したクラスのプロパティ情報を、変数へ代入するための代入文を記述します。

  コードテーブルで定義したプロパティを取得する際には、コードテーブルの値または表示名を取得できます。表示名を取得する場合は、代入文のクラス.プロパティに「@（単価記号）」を付けてください。

- **検索結果をソートするためのキー**

  検索結果をソートする場合は、ソートするためのキーをクラス.プロパティの形式で指定します。[ORDER_ASC]タグは昇順、[ORDER_DESC]タグは降順にソートします。省略した場合、[GET_VALUE]タグで指定したプロパティの昇順になります。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	検索条件を満たすデータがない
ERROR	－
MULTI	－

（凡例）

－：該当しない

記述例

複数クラスを検索する記述例を次に示します。

例 1

資産情報、インストールソフトウェア情報、インストールソフトウェアリストを結合し、資産番号「1000」のインストールソフトウェア情報の一覧を出力する場合の記述例を次に示します。

```
[ASSET_ITEM_LOOP]
  [JOIN_FIND]
  [JOIN]
    joinassoc;InstalledLink;
    joinfrom;AssetInfo,InstalledInfo;
    jointype;INNER;
  [JOIN]
    joinassoc;InstalledListLink;
    joinfrom;InstalledInfo,InstalledList;
    jointype;INNER;
  [FIND_DATA]
    (AssetInfo.AssetNo = '1000')
  [GET_VALUE]
    INSTALLNAME = InstalledList.InstalledName
    VERSION     = InstalledList.InstalledVersion
    PERMIT      = InstalledList.InstalledPermit@

  [SET_VALUE]
```

```
    MSG = INSTALLNAME+'['+VERSION+'] ('+PERMIT+')'
    $ECHO(MSG)

  [SET_VALUE]
    $SETSTATUS('NORMAL')
[ASSET_ITEM_LOOP_END]
```

実行結果

SoftwareA[0100] (許可する)

SoftwareA[0101] (許可する)

SoftwareB[0000] (許可する)

SoftwareC[0000] (許可しない)

例 2

資産情報と部署情報を結合し、資産 ID「10000」の部署名を出力する場合の記述例を次に示します。この例では、外部結合（[OUTER]）しているので、部署 ID が設定されていない場合は「" "」が出力されます。

```
[JOIN_FIND]
[JOIN]
  joinfrom;AssetInfo,GroupInfo;
  jointype;OUTER;
  joinkey;AssetInfo.GroupID,GroupInfo.GroupID;
[FIND_DATA]
  (AssetInfo.AssetID = '10000')
[GET_VALUE]
  GROUP_NAME = GroupInfo.FullPathName

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS = NORMAL
  [THEN]
    [SET_VALUE]
      MSG = 'GROUP =' +GROUP_NAME
      $ECHO(MSG)
  [ELSE]
    [SET_VALUE]
      MSG = 'JOIN (' +STATUS+ ')'
      $ECHO(MSG)
[IF_END]
```

実行結果

GROUP ＝ 本社/営業部/営業 1 課

[SET_VALUE]（変数代入）

変数代入文を列挙します。組み込み関数単体での呼び出しができます。

形式

```
[SET_VALUE]
  変数代入文1
  変数代入文2・・・
```

指定する値

- 変数代入文

  [VAR]タグで宣言した変数を使用して、変数代入文を記述します。

記述例

変数「MSG」に「Hello world」を設定して出力する場合の記述例を次に示します。

```
[SET_VALUE]
  MSG = 'Hello world'
  $ECHO(MSG)
```

[SUB]（サブルーチン）

同一処理を処理ブロックとして定義します。サブルーチンは、処理パスが通過しても実行されないで、組み込み関数$GOSUB で実行されます。そのため、サブルーチンは組み込み関数$GOSUB より前で定義している必要があります。

形式

```
[SUB]
  サブルーチン名
  ・・・
[SUB_END]
[BEGIN]
  ・・・
  [SET_VALUE]
  $GOSUB(サブルーチン名)
[END]
```

指定する値

- サブルーチン名

  サブルーチン名を指定します。サブルーチン名には、半角英数字と「_（アンダーバー）」が使用できます。ただし、サブルーチン名の先頭に、半角数字は使用できません。また、サブルーチン名は、英文字の大小文字を区別して使用してください。

記述例

コマンドプロンプト画面への出力をサブルーチン化する場合の記述例を次に示します。この例では、jamscript コマンドでオプションに「-s MSG=1」を指定すると、「ECHOMSG」に設定したテキストがコマンドプロンプト画面に出力されることを示しています。

```
[SUB]
  ECHO
  [IF]
    MSG = '1'
  [THEN]
    [SET_VALUE]
      $ECHO(ECHOMSG)
  [IF_END]
[SUB_END]

[BEGIN]
  [SET_VALUE]
    MSG = $GETSESSION('MSG')

    ECHOMSG = 'Hello world'
    $GOSUB(ECHO)
[END]
```

実行するコマンド

jamscript -f Example.txt -s MSG=1

実行結果

Hello world

[SWITCH]（条件分岐）

条件値が指定した定数と一致した場合、処理を実行します。

条件値を複数の定数と比較することが多い場合に使用すると[IF]タグを使用するよりも階層を深くしないで記述できるため便利です。条件値が定数と一致したときは[CASE]以降の処理をします。どの定数とも一致しなかったときは[DEFAULT]以降の処理を実行し、[DEFAULT]が定義されていないときは、処理を実行しません。

複数の[CASE]タグに同じ定数を指定した場合は、記述の上から（1 行で記述してある場合は左から）処理を実行し、条件値が定数と一致したときの処理内容をすべて実行します。

形式

```
[SWITCH]
  条件値
  [CASE]
    定数
      条件値が定数と一致した場合の処理内容
 ([CASE])
    定数[,定数[,定数・・・]]
      条件値がどれかの定数と一致した場合の処理内容
 ([DEFAULT])
    どの定数とも一致しなかった場合の処理内容
[SWITCH_END]
```

指定する値

- **条件値**

  条件値を記述します。

- **定数**

  処理を実行する場合の条件値の定数を記述します。定数をコンマで区切ると複数記述できます。

- **条件値が定数と一致した場合の処理内容**

  条件値が定数と一致した場合の処理内容を記述します。条件値が最初に定数と一致した場合の処理だけを実行したいときは、[CASE]タグの記述の最後に「$BREAK()」を指定します。

- **条件値がどれかの定数と一致した場合の処理内容**

  条件値がどれかの定数と一致した場合の処理内容を記述します。条件値が最初に定数と一致した場合の処理だけを実行したいときは、[CASE]タグの記述の最後に「$BREAK()」を指定します。この記述は省略できます。

- **どの定数とも一致しなかった場合の処理内容**

  どの定数とも一致しなかった場合の処理内容を記述します。[DEFAULT]タグは、[CASE]タグより前に指定してもかまいません。この記述は省略できます。

記述例

次の処理を実行する記述例を示します。

- データを正常に更新できた場合（STATUS ＝ 'NORMAL'）終了状態を表示する。
- すでにほかの制御で更新されている場合（STATUS ＝ 'MULTI'）メッセージを出力する。
- データがない場合（[DEFAULT]）データを追加する。

```
[UPDATE]
  AssetInfo
[DATA]
  AssetInfo.AssetID = ’10000’
  AssetInfo.AssetNo = ’10000’
  AssetInfo.AssetKind = ’001’
  AssetInfo.AssetBranchNo = ’0’
  AssetInfo.UpdateTime = _UpdateTime
[SET_VALUE]
  STATUS = $GETSTATUS()
[SWITCH]
  STATUS
  [CASE]
    ’NORMAL’
    [SET_VALUE]
      MSG = ’UPDATE (’ +STATUS+ ’)’
      $ECHO(MSG)
      $BREAK()
  [CASE]
    ’MULTI’
    [SET_VALUE]
      MSG = ’Asset number [10000] is updated already.’
      $ECHO(MSG)
      $BREAK()
  [DEFAULT]
    [APPEND]
      AssetInfo
    [DATA]
      AssetInfo.AssetID = ’10000’
      AssetInfo.AssetNo = ’10000’
      AssetInfo.AssetKind = ’001’
      AssetInfo.AssetBranchNo = ’0’
    [SET_VALUE]
      MSG = ’UPDATE (’ +STATUS+ ’)’
      $ECHO(MSG)
      $BREAK()
[SWITCH_END]
```

[TRANSACTION]（トランザクション範囲定義）

アクセス定義ファイルを使用して操作画面をカスタマイズしている場合、トランザクションとして扱う範囲を指定します。

トランザクションとして定義した範囲の処理を終了するときに、処理状態が正常（NORMAL）の場合はトランザクションのコミットを、処理状態が正常以外の場合はトランザクションのロールバックを行います。

形式

```
[TRANSACTION]
   ・・・
  [BEGIN]
     ・・・
  [END]
[TRANSACTION_END]
```

指定する値

トランザクションの処理を記述します。

注意事項

トランザクション処理を終了するかどうかは、トランザクション処理を終了するタグである[TRANSACTION_END]、またはトランザクション処理を途中で抜ける組み込み関数$BREAK で、参照した終了状態によって判定されます。終了状態が「NORMAL」の場合は直前にコミットします。「NORMAL」以外のときはコミットしません（ロールバックします）。

アクセス定義ファイルの終了状態は、タグや組み込み関数の実行によって随時更新されます。トランザクション中に終了状態の更新を伴う処理を複数指定した場合、最後に指定した終了状態を基に、データベースのコミットまたはロールバックが実行されます。

そのため、[TRANSACTION_END]タグおよび組み込み関数$BREAK でその直前にコミットする場合は、$SETSTATUS で「NORMAL」を明示的に指定してください。また、コミットしない（ロールバックする）場合は、「NORMAL」以外を明示的に指定してください。

なお、トランザクション処理は階層化できません。

記述例

トランザクション処理の記述例を次に示します。

```
[TRANSACTION]
  [APPEND]
    AssetInfo
  [DATA]
    AssetInfo.AssetID = '10000'
    AssetInfo.AssetNo = '10000'
    AssetInfo.AssetWorkKind = '001'
```

```
    AssetInfo.AssetKind = '001'
    AssetInfo.AssetBranchNo = '0'
  [SET_VALUE]
    STATUS = $GETSTATUS()
  [IF]
    STATUS = NORMAL
  [THEN]
    ##処理1
    [APPEND]
      InstalledInfo
    [DATA]
      InstalledInfo.InstalledID = '10000'
      InstalledInfo.AssetID = '10000'
    [SET_VALUE]
      STATUS = $GETSTATUS()
    [IF]
      STATUS = NORMAL
    [THEN]
      ##処理2
      [SET_VALUE]
        MSG = 'TRANSACTION : COMMIT'
        $ECHO(MSG)
    [ELSE]
      ##処理3
      [SET_VALUE]
        MSG = 'TRANSACTION : ROLLBACK'
        $ECHO(MSG)
    [IF_END]
  [ELSE]
    ##処理4
    [SET_VALUE]
      MSG = 'TRANSACTION : ROLLBACK'
      $ECHO(MSG)
  [IF_END]
[TRANSACTION_END]
```

上記の処理では、次のように例外処理が実行されます。

- 資産 ID「10000」の追加に失敗した場合

  コメントの「##処理 4」に遷移します。[APPEND]の終了状態が正常以外のため、[TRANSACTION]～[TRANSACTION_END]のトランザクションをロールバックして終了します。このため、すべてのデータベースへの要求が無効となります。

- 資産 ID「10000」の追加に成功した場合

  コメントの「##処理 1」に遷移します。インストールソフトウェア情報に、資産 ID「10000」、インストール ID「10000」を登録します。

- 資産 ID「10000」の追加に成功し、インストール ID「10000」の追加に失敗した場合

  コメントの「##処理 3」に遷移します。[APPEND]の終了状態が正常以外のため、[TRANSACTION]～[TRANSACTION_END]のトランザクションをロールバックして終了します。このため、すべてのデータベースへの要求が無効となります。

- 資産 ID「10000」、インストール ID「10000」の追加に成功した場合

コメントの「##処理 2」に遷移します。[APPEND]の終了状態が正常のため、すべてのデータベースへの要求が有効となります。

[UPDATE]（オブジェクトクラスの更新）

オブジェクトクラスを検索して更新します。複数のオブジェクトクラスが検索された場合は、最初に見つかったオブジェクトクラスだけを更新します。指定したオブジェクトクラスが見つからなかった場合は、何も実行されません。

形式

```
[UPDATE]
  オブジェクトクラス名
  [DATA]
    プロパティ代入文
```

指定する値

- **オブジェクトクラス名**

  更新するオブジェクトクラス名を記述します。

- **プロパティ代入文**

  更新するすべてのキープロパティを代入文として記述します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	更新するオブジェクトクラスのデータが存在しない
MULTI	他の制御ですでに更新されている

（凡例）

－：該当しない

記述例

資産 ID「10000」の資産状態を「301」に更新する場合の記述例を次に示します。

```
[UPDATE]
  AssetInfo
[DATA]
  AssetInfo.AssetID     = '10000'
  AssetInfo.AssetStatus = '301'

[SET_VALUE]
  STATUS = $GETSTATUS()
[IF]
  STATUS != NORMAL
```

```
  [THEN]
    [SET_VALUE]
      MSG = 'UPDATE (' +STATUS+ ')'
      $ECHO(MSG)
[IF_END]
```

[VAR]（変数宣言）

変数宣言をします。

形式

```
[VAR]
  変数名
```

指定する値

- 変数名

  宣言する変数名を記述します。

記述例

変数「STATUS」および「MSG」を宣言する記述例を次に示します。

```
[VAR]
STATUS
MSG
```

5 アクセス定義ファイルに記述する組み込み関数

この章では、アクセス定義ファイルのタグで使用する組み込み関数について説明します。

組み込み関数一覧

ここでは、変数代入文に指定できる組み込み関数について説明します。

使用目的別に、アクセス定義ファイルに記述する組み込み関数の一覧を次に示します。

表 5-1　組み込み関数一覧

組み込み関数		機能
使用目的	組み込み関数名	
配列の操作	$GETARRAY	配列からのデータ取得
	$CLEARARRAY	配列の初期化
	$SETARRAY	配列へのデータ設定
	$SETARRAYBYKEY	配列へのキー付きデータの設定
	$GETARRAYBYKEY	配列からのキー対応データ取得
	$GETKEYFROMARRAY	配列から対応するキーの取得
	$UPDARRAY	配列データの更新
	$UPDARRAYBYKEY	キー付き配列データの更新
ブロック処理制御	$BREAK	処理ブロックの中断
	$GETSTATUS	処理ブロックの終了状態取得
	$SETSTATUS	処理ブロックの終了状態設定
セッション情報の操作	$GETSESSION	セッション情報の取得
	$GETTEMPNAME	セッション単位に作成するダウンロード用のワークファイル名の指定
	$SETSESSION	セッション情報の設定
ファイル操作	$FILEARRAY	配列に格納してある情報を CSV ファイルに出力
	$FILECLOSE	ファイルの編集終了
	$FILECOPY	ファイルのコピー（常に上書きしてコピー）
	$FILEDEL	ファイルの削除
	$FILEOPEN	ファイルの編集開始
	$FILEPUT	ファイルへのデータ出力
	$FILEPUTLN	ファイルへのデータ出力時、最後に復帰改行を追加
	$FINDFILE	ファイルの検索
ディレクトリ情報の操作	$LDAPACS	ディレクトリサービスへの接続認証、検索、エントリの取得、属性の取得など

組み込み関数		機能
使用目的	組み込み関数名	
ディレクトリ情報の操作	$GETROLE	ユーザの持つ権限を配列に格納
サブルーチン	$GOSUB	サブルーチンの実行
演算	$ADD	加算結果の取得
	$CALCDATE	日時の計算
	$DIV	除算結果の取得
	$MOD	除算結果の余りの取得
	$MUL	乗算結果の取得
	$SUB	減算結果の取得
情報の抽出	$SUBSTR	部分文字列抽出
	$TOKEN	トークン抽出
情報の取得	$DATACOUNT	直前の検索結果の件数取得
	$DATETIME	日付の取得
採番	$NUMBER	データベースを利用した通番採番
サーバの環境設定情報の取得	$ENVIRONMENT	サーバの環境設定情報の取得
	$GETREGVALUE	サーバ環境のレジストリ情報取得
処理終了	$EXIT	アクセス定義ファイルの処理終了
	$SETOPTION	エラー発生時の終了オプションの設定
メッセージ出力	$ECHO	標準出力へのメッセージ出力
	$LOGMSG	ログファイルへのメッセージ出力
	$FORMATMSG	メッセージ文字列の書式化
変換	$LOWER	英文字列を小文字に変換
	$UPPER	英文字列を大文字に変換
初期設定ファイルの操作	$GETPROFILEDATA	初期設定ファイルのキーと値の取得
NULL 判定	$ISNULL	NULL かどうかの判定
文字列の操作	$LENGTH	文字列長の取得
	$MATCH	文字列の評価
	$STRCMP	文字列の比較
DLL の操作※	$DLLLOAD	DLL のロード
	$DLLEXEC2	ユーザが用意した DLL に実装される、ユーザ関数の実行

組み込み関数		機能
使用目的	組み込み関数名	
DLL の操作※	$DLLFREE	DLL の解放
	$DLLMSG	DLL のメッセージ取得

注※

DLL の操作に関する組み込み関数で実行する DLL は、必ず 32 ビットアプリケーションとして作成したものを指定してください。使用している OS が、Windows Server 2008 R2 の場合でも、64 ビットアプリケーションとして作成した DLL は使用しないでください。

各組み込み関数の詳細

組み込み関数の詳細は、これ以降、基本的には次に示す形式で説明していきます。なお、組み込み関数の説明は、組み込み関数名のアルファベット順になっています。

組み込み関数

記述する組み込み関数とその機能を説明しています。

形式

組み込み関数の記述形式を説明しています。

指定する値

組み込み関数に指定できる値を説明しています。

使用する DLL のインターフェース

DLL を使用する組み込み関数を記述する場合の、DLL とのインターフェースを説明しています。

終了状態

組み込み関数で記述した処理の終了状態と、各終了状態の表す内容を説明しています。

記述例

組み込み関数の記述例を示しています。

$ADD（加算結果の取得関数）

文字列を数値と見なして加算を実行し、その演算結果を返します。

形式

```
返却値=$ADD(文字列,数字)
```

指定する値

- **返却値**
  演算結果を設定する変数名を指定します。結果として取得できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。
- **文字列**
  加算する数値を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。
- **数字**
  加算する数値を定数または変数で指定します。小数点を含む数値を定数で指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 文字列、数値に指定した値の誤り • 演算結果が表現できる値の範囲外
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

注意事項

「文字列」、「数字」に指定できない値が記述されている場合、および演算結果が表現できる範囲外となった場合は、返却値に 0 が返されます。

記述例

「10 ＋ 5」を演算して結果を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL1 = 10
  VAL2 = $ADD(VAL1, 5)

MSG = ’ADD: ’ +VAL1+ ’+ 5 =’ +VAL2
$ECHO(MSG)
```

実行結果

ADD: 10 ＋ 5 ＝ 15

$BREAK（処理ブロックの中断関数）

処理ブロックを中断します。また、処理中の処理ブロックから抜けます。

クラス作成時の例外処理では、各要求の終了状態を判定文で判定し、組み込み関数$BREAK を使って異常時の後処理をしたり、処理を中断したりします。

形式

```
$BREAK()
```

指定する値

指定する値はありません。

記述例

記述例については、「4. アクセス定義ファイルに記述するタグ」の[TRANSACTION]（トランザクション範囲定義）の記述例を参照してください。

$CALCDATE（日時の計算関数）

日時に日数、時間数などを指定して加算し、その演算結果を返します。指定した日時から二日前、3 時間後といった相対日時を、月、日、時間などの繰り越しを考慮することなく取得できます。

形式

```
返却値=$CALCDATE(日時,単位,加算値,出力形式)
```

指定する値

- 返却値
  取得した日時を設定する変数名を指定します。
- 日時
  定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。
  指定できる形式を次に示します。
  - yyyy/mm/dd hh:mm:ss または yyyy-mm-dd hh:mm:ss
    日時を指定します。「hh:mm:ss」は省略できます。
  - 何も指定しない
    この組み込み関数を実行する日時が指定されます。
- 単位
  定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。
  指定できる変数を次に示します。
  - d
    日数を加算する場合に指定します。
  - h
    時間数を加算する場合に指定します。
  - n
    分数を加算する場合に指定します。
  - s
    秒数を加算する場合に指定します。
- 加算値
  定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。
  値には、整数を指定します。
- 出力形式
  日時の出力形式に、定数または変数を指定します。出力形式の詳細については、$DATETIME（日付の取得関数）を参照してください。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
スクリプト中断	次のどちらかを示す • 加算値に数字以外を指定するなど不正な値を指定した • 引数の誤り、またはそのほかのエラー

記述例

2015/04/11 から 10 日前の日にちを出力する場合の記述例を次に示します。

```
DATE = $CALCDATE('2015/04/11','d','-10','%Y/%m/%d')
$ECHO(DATE)
```

実行結果

2015/04/01

$CLEARARRAY（配列初期化関数）

配列に格納されている情報をすべて削除して、配列を初期化します。

形式

```
$CLEARARRAY(配列変数名)
```

指定する値

- 配列変数名
  初期化する配列変数名を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

$SETARRAY（配列へのデータ設定関数）の記述例を参照してください。

$DATACOUNT（直前の検索結果の件数取得関数）

直前に実行した検索で見つかった情報の件数を取得します。

[ASSET_ITEM_LOOP]、[ASSET_ITEM_LOOP_END]タグ内に検索を定義している場合、[ASSET_ITEM_LOOP_END]後の終了状態は必ず「NODATA」になります。該当するデータがないかどうかを判定するには、組み込み関数$DATACOUNT で検索結果の件数を取得して判定する必要があります。

形式

```
返却値=$DATACOUNT()
```

指定する値

- 返却値

  検索結果の件数を設定する変数名を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

機器状態が運用（002）の資産情報のデータ件数（100 件の場合）を、出力する記述例を次に示します。

```
[CLASS_FIND]
  AssetInfo
[FIND_DATA]
  (AssetInfo.AssetStatus = '002')AND
  (AssetInfo.AssetKind = '001')
[GET_VALUE]
  WORK = AssetInfo.AssetNo

[SET_VALUE]
  STATUS = $GETSTATUS()
  TOTAL = $DATACOUNT()
[IF]
```

```
  STATUS = NORMAL
[THEN]
  [SET_VALUE]
    MSG = 'DataCount :' +TOTAL
    $ECHO(MSG)
[ELSE]
  [SET_VALUE]
    MSG = 'CLASS_FIND (' +STATUS+ ')'
    $ECHO(MSG)
[IF_END]
```

実行結果

DataCount : 100

$DATETIME（日付の取得関数）

指定した出力形式に従って、日時を取得します。

形式

```
返却値=$DATETIME(出力形式)
```

指定する値

- 返却値
  取得した日時を設定する変数名を指定します。
- 出力形式
  日時の出力形式を、定数または変数を指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。
  出力形式は、**フォーマット指示子**と文字定数を組み合わせて指定します。
  フォーマット指示子は、「%（パーセント）」と 1 文字の英字の組で表した取得情報を示す記号です。フォーマット指示子で指定する英字には大文字と小文字の区別があり、それぞれ取得できる情報が異なります。「%（パーセント）」に続く英字がフォーマット指示子以外の英字の場合は、指定した英字がそのまま取得されますのでご注意ください。
  なお、文字定数として「"（ダブルクォーテーション）」を使用する場合は、「¥"」と指定してください。
  指定できるフォーマット指示子を次に示します。
  - %a
    実行日の曜日の英略語を次に示す形式で取得します。
    Sun Mon Tue Wed Thu Fri Sat
  - %A
    実行日の曜日を次に示す英語表記で取得します。
    Sunday Monday Tuesday Wednesday
    Thursday Friday Saturday
  - %b
    実行月の英略語を次に示す形式で取得します。
    Jan Feb Mar Apr May Jun
    Jul Aug Sep Oct Nov Dec
  - %B
    実行月を次に示す英語表記で取得します。
    January February March April May June
    July August September October November December
  - %d

日付を 01～31 の数値で取得します。

- %H
  時刻を 24 時間制の 00～23 の数値で取得します。
- %I
  時刻を 12 時間制の 01～12 の数値で取得します。
- %j
  1 月 1 日からの通算日を 001～366 の数値で取得します。
- %m
  月を 01～12 の数値で取得します。
- %M
  分を 00～59 の数値で取得します。
- %p
  AM（午前）または PM（午後）を文字列で取得します。
- %S
  秒を 00～59 の数値で取得します。
- %w
  日曜日を 0 として、1 けたの曜日番号を 0～6 の数値で取得します。
  日曜日：0 月曜日：1 火曜日：2 水曜日：3
  木曜日：4 金曜日：5 土曜日：6
- %Y
  年を 4 けたの西暦で取得します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

日時（2015 年 04 月 01 日 1 時 5 分 36 秒の場合）を「年/月/日 時:分:秒」のフォーマットで取得する場合の記述例を次に示します。

```
[SET_VALUE]
  DATE = $DATETIME('%Y/%m/%d %H:%M:%S')
  MSG = 'DATE =' +DATE
  $ECHO(MSG)
```

実行結果

DATE = 2015/04/01 01:05:36

$DIV（除算結果の取得関数）

文字列を数値と見なして除算を実行し、その演算結果を返します。

例えば、Asset Console の標準のメモリやディスクサイズは、メガバイト単位で管理されていますが、ギガバイト単位で管理している情報を資産管理データベースに登録する際、単位をメガバイトに合わせる必要があります。このような場合、組み込み関数$DIV を使用して、「ギガバイト」を「メガバイト」に変換できます。

形式

```
返却値=$DIV(文字列,数字)
```

指定する値

- **返却値**

  演算結果を設定する変数名を指定します。結果として取得できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

- **文字列**

  被除数を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

- **数字**

  除数を定数または変数で指定します。小数点を含む数値を定数で指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 文字列、数値に指定した値の誤り • 演算結果が表現できる値の範囲外
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

注意事項

「文字列」、「数字」に指定できない値が記述されている場合、および演算結果が表現できる範囲外となった場合は、返却値に 0 が返されます。

記述例

「10÷5」を演算し、結果を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL1 = 10
  VAL2 = $DIV(VAL1, 5)

  MSG = 'DIV: ' +VAL1+ ' / 5 = ' +VAL2
  $ECHO(MSG)
```

実行結果

DIV: 10 / 5 = 2

$DLLEXEC2（DLL 実行関数）

ユーザが作成したユーザ関数を実行します。$DLLEXEC2 で実行するユーザ関数とアクセス定義ファイルとの間のデータの受け渡しは、配列変数を使用します。ユーザ関数内で配列変数にアクセスするためには、Asset Console が提供するマクロを使用します。

形式

```
$DLLEXEC2(DLLオブジェクト,関数名,配列変数名1（,配列変数名2（,…））)
```

指定する値

- DLL オブジェクト

  $DLLLOAD で求めた DLL オブジェクトの変数名を指定します。

- 関数名

  実行する関数の名称を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- 配列変数名

  関数に渡す情報を設定した配列変数名、または関数で実行した結果を取得する配列変数名を指定します。

使用する DLL のインターフェース

$DLLEXEC2 で呼び出す関数の形式を次に示します。

```
int  関数名(int  argc          //スクリプトで指定した配列変数の数
           ,void**  argv       //スクリプトで指定した配列変数群の先頭アドレス
          )
```

スクリプトで指定した配列変数は、指定した順（左から）で、argv の 0 番配列から順番に格納されています。そのため、配列変数名 1 は argv[0]、配列変数名 2 は argv[1]として関数内で処理します。

関数の戻り値が負数の場合には、同じ DLL にユーザが用意した aim_getmessage 関数を実行し、メッセージが書き込まれていたら、その内容を Asset Console のログに書き込み、スクリプトの実行を中断します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了（関数の戻り値が 0 の場合）
NODATA	警告付き終了（関数の戻り値が 1 の場合）
ERROR	異常終了（関数の戻り値が正数の場合）
スクリプト中断	次のどちらかを示す

終了状態	内容
スクリプト中断	• 異常終了（関数の戻り値が負数の場合）※ • 引数の誤り、またはそのほかのエラー

注※

指定した DLL オブジェクトの変数名が、組み込み関数$DLLLOAD で求めた DLL オブジェクトではないことを示しています。

注意事項

組み込み関数$DLLEXEC2 で呼び出すユーザ関数を実装した DLL を、あらかじめ用意しておく必要があります。

記述例

「sample.dll」をロードし、集計関数「FunctionSum」を実行する場合の記述例を次に示します。この例では、ロードした「sample.dll」に引数「10」、「20」、「30」を設定して「FunctionSum」を実行し、その結果を出力することを示しています。

```
[SET_VALUE]
  DLLOBJ = $DLLLOAD('sample.dll')
  STATUS = $GETSTATUS()
[IF]
  STATUS != NORMAL
  [THEN]
    [SET_VALUE]
      $ECHO('DLL LOAD ERROR')
      $EXIT(3)
[IF_END]

[SET_VALUE]
  $SETARRAY(INPUT, '10')
  $SETARRAY(INPUT, '20')
  $SETARRAY(INPUT, '30')
  $DLLEXEC2(DLLOBJ,'FunctionSum',INPUT,OUTPUT)
  STATUS = $GETSTATUS()
[IF]
  STATUS = NORMAL
  [THEN]
    [SET_VALUE]
      VAL = $GETARRAY(OUTPUT, 1)
      MSG = 'OUTPUT = ' +VAL
      $ECHO(MSG)
  [ELSE]
    [SET_VALUE]
      VAL = $DLLMSG(DLLOBJ)
      MSG ='DLLEXEC FunctionSum ERROR (' +VAL+ ')'
      $ECHO(MSG)
[IF_END]

[SET_VALUE]
  $DLLFREE(DLLOBJ)
```

$DLLEXEC2 で実行可能なユーザ関数の作成

$DLLEXEC2 で実行するユーザ関数が、スクリプトとの間でデータの受け渡しが必要ない場合は、一般的な手順で DLL を作成できます。

ユーザ関数がスクリプトから情報を受け取ったり、ユーザ関数の処理結果をスクリプトの配列に設定したりする場合には、スクリプトの配列変数にアクセスするために、Asset Console が提供する専用の関数を使用する必要があります。Asset Console では、これらの配列変数へのアクセス用関数をマクロとして提供しています。

マクロは C および C++言語用のヘッダファイル「jamScriptAPI.h」に定義されています。このヘッダファイルをインクルードすることで、アクセス定義ファイル中の組み込み関数と同じインターフェースで配列を操作できます。

なお、DLL に実装された、配列操作のマクロを使用した関数を実行できるのは、$DLLEXEC2 からだけです。

jamScriptAPI.h の格納先を次に示します。

```
Asset Consoleのインストール先フォルダ¥sdk¥include
```

jamScriptAPI.h に定義されたマクロの一覧を次に示します。

表 5-2　DLL で使用できるマクロ一覧

マクロ		機能
使用目的	マクロ名	
配列の操作	$GETARRAY	配列からのデータ取得
	$CLEARARRAY	配列の初期化
	$SETARRAY	配列へのデータ設定
	$SETARRAYBYKEY	配列へのキー付きデータ設定
	$GETARRAYBYKEY	配列からキー付きデータ取得
	$GETKEYFROMARRAY	配列から対応するキー取得
	$GETARRAYLENGTH	配列の要素数の取得
	$GETARRAYNAME	配列名の取得
	$UPDARRAY	配列データの更新
	$UPDARRAYBYKEY	キー付き配列データの更新
インスタンスの取得	$GETINITAREA	aim_init 関数の戻り値の取得
状態の取得	$GETSTATUS	関数の終了状態の詳細を取得

DLL 作成時に使用できる各マクロについて説明します。

■$CLEARARRAY（配列初期化マクロ）

指定した配列に格納されている情報をすべて削除して、配列を初期化します。

形式（相当関数形式）

```
int $CREARARRAY(void** argv,void* array);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（ユーザ関数の引数）

戻り値

戻り値	内容
0	正常終了
-1	エラー※

注※
詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した、1 番目の配列を初期化するコーディング例を次に示します。

```
int DllFunc8(int argc,void** argv){
  int rc;
  rc = $CLEARARRAY(argv,argv[0]);
  if(rc)return-1;
  return 0;
}
```

■$GETARRAY（配列からのデータ取得マクロ）

配列に格納されている情報から、指定した位置番号の情報を取得します。

形式

```
char* $GETARRAY(void** argv,void* array,int pos);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（ユーザ関数の引数）
- pos

情報を取得する配列内の位置番号（「1」から開始）

戻り値

戻り値	内容
指定した配列の情報（文字列）	正常終了
NULL	エラー※、または指定した位置番号の情報が存在しない

注※

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 1 番目の配列の、1 番目の情報を取得するコーディング例を次に示します。

```
int DllFunc3(int argc,void** argv){
  void *data;
  data = $GETARRAY(argv,argv[0],1);
  if(!data) if($GETSTATUS(argv) != JAM_SCRIPTAPI_NORMAL &&
  $GETSTATUS(argv) != JAM_SCRIPTAPI_NODATA) return -1;
  return 0;
}
```

■$GETARRAYBYKEY（キー指定による配列からのデータ取得マクロ）

スクリプト側で組み込み関数$SETARRAYBYKEY を使用して作成した配列や、ユーザ関数内で$SETARRAYBYKEY マクロを使用して作成したキー付き配列から、指定したキーを持つ情報を取得します。同一のキーを持つ要素が存在する場合、同一キーの何番目の要素を取り出すかを示す、キー内配列番号を指定します。

形式

```
char* $GETARRAYBYKEY(void** argv,void* array,char* key,int pos);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（ユーザ関数の引数）
- key
  キー
- pos
  キー内配列番号（「1」から開始）

戻り値

戻り値	内容
キーで指定した配列の情報（文字列）	正常終了

戻り値	内容
NULL	エラー※、または指定したキー内配列番号の情報が存在しない

注※

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 1 番目の配列から、キー「key1」で格納された 1 番目のキー内配列番号の情報を取得するコーディング例を次に示します。

```
int DllFunc4(int argc,void** argv){
  void *data;
  data = $GETARRAYBYKEY(argv,argv[0],"key1",1);
  if(!data) if($GETSTATUS(argv) != JAM_SCRIPTAPI_NORMAL &&
  $GETSTATUS(argv) != JAM_SCRIPTAPI_NODATA) return -1;
  return 0;
}
```

■$GETARRAYLENGTH（配列の要素数の取得マクロ）

指定した配列の要素数を取得します。

形式

```
int $GETARRAYLENGTH(void** argv,void* array);
```

引数

- argv

  スクリプトで指定した配列変数群の先頭アドレス

- array

  配列変数群の要素の一つ（ユーザ関数の引数）

戻り値

戻り値	内容
配列数（0 以上の数値）	正常終了
-1	エラー※

注※

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 1 番目の配列の配列要素数を取得するコーディング例を次に示します。

```
int DllFunc10(void* obj,void* functbl,int argc,void** argv){
  int len;
  len = $GETARRAYLENGTH(argv,argv[0]);
  if (len<0) return -1;
```

```
  return 0;
}
```

■$GETARRAYNAME（配列名の取得マクロ）

指定した配列の配列変数名を取得します。

形式

```
char* $GETARRAYNAME(void** argv,void* array);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（ユーザ関数の引数）

戻り値

戻り値	内容
配列変数名（文字列）	正常終了
NULL	エラー※

注※
詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 1 番目の配列の配列変数名を取得するコーディング例を次に示します。

```
int DllFunc9(int argc,void** argv){
  char* name;
  name = $GETARRAYNAME(argv,argv[0]);
  if(!name) return -1;
  return 0;
}
```

■$GETINITAREA（aim_init 関数の戻り値（インスタンス）の取得マクロ）

aim_init 関数の戻り値であるインスタンスを取得します。

形式

```
void * $GETINITAREA(void** argv);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス

戻り値

aim_init 関数の戻り値（インスタンス）です。

コーディング例

aim_init 関数の戻り値を取得するコーディング例を次に示します。

```
int DllFunc(int argc, void** argv){
  USER_HANDLE*  obj;
  obj = $GETINITAREA(argv);
  return 0;
}
```

■$GETKEYFROMARRAY（キー情報の取得マクロ）

キー付きで配列に格納されている情報から、指定した配列番号の情報が持つキー情報を取得します。

形式

```
char* $GETKEYFROMARRAY(void** argv,void* array,int pos);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（ユーザ関数の引数）
- pos
  情報を取得する配列要素の配列番号（「1」から開始）

戻り値

戻り値	内容
キー情報（文字列）	正常終了※1
NULL	エラー※2、または指定した配列番号の値が存在しない

注※1

指定した配列要素にキーが付いていない場合、0 バイトの文字列が返されます。

注※2

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 3 番目の配列の、1 番目の配列要素に格納されているキー情報を取得するコーディング例を次に示します。

```
int DllFunc7(void* obj,void* functbl,int argc,void** argv){
  void *key;
  key = $GETKEYFROMARRAY(argv,argv[2],1);
  if(!key) if($GETSTATUS(argv) != JAM_SCRIPTAPI_NORMAL &&
  $GETSTATUS(argv) != JAM_SCRIPTAPI_NODATA) return -1;
```

```
  return 0;
}
```

■$GETSTATUS（マクロの終了状態の取得マクロ）

処理の終了状態を取得します。

形式

```
int $GETSTATUS(void** argv);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス

戻り値

戻り値	内容
JAM_SCRIPTAPI_NORMAL	正常終了
JAM_SCRIPTAPI_DUPLICATE	キー値の重複
JAM_SCRIPTAPI_NODATA	該当配列データなし
JAM_SCRIPTAPI_INSUFFICIENTMEMORY	処理に必要なメモリの不足
JAM_SCRIPTAPI_ILLEGAL	呼び出しインターフェース不正
JAM_SCRIPTAPI_ERROR	上記以外の内部エラーの発生

コーディング例

$GETARRAY（配列からのデータ取得マクロ）のコーディング例を参照してください。

■$SETARRAY（配列へのデータ設定マクロ）

配列に情報を追加します。

形式

```
int $SETARRAY(void** argv,void* array,char* value);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（情報を追加するユーザ関数の引数）
- value
  追加する情報（文字列）

戻り値

戻り値	内容
0	正常終了
-1	エラー※

注※

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 2 番目の配列に、情報「data1」で配列要素を追加するコーディング例を次に示します。

```
int DllFunc1(int argc, void** argv){
  int rc;
  rc = $SETARRAY(argv,argv[1],"data1");
  if(rc) return -1;
  return 0;
}
```

■$SETARRAYBYKEY（配列へのキー付きデータ設定マクロ）

キー付きで配列に情報を追加します。

形式

```
int $SETARRAYBYKEY(void** argv,void* array,char* key,char* value);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（情報を追加するユーザ関数の引数）
- key
  キー
- value
  追加する情報（文字列）

戻り値

戻り値	内容
0	正常終了
-1	エラー※

注※

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 2 番目の配列に、キー値「key1」、情報「data1」で配列要素を追加するコーディング例を次に示します。

```
int DllFunc2(int argc,void** argv){
  int rc;
  rc = $SETARRAYBYKEY(argv,argv[1],"key1","data1");
  if(rc) return -1;
  return 0;
}
```

■$UPDARRAY（配列データの更新マクロ）

配列変数の配列要素の値を更新します。配列要素は、配列番号で指定します。

形式

```
int $UPDARRAY(void** argv,void* array,int pos,char* value);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（配列要素を更新するユーザ関数の引数）
- pos
  情報を更新する配列要素の配列番号（「1」から開始）
- value
  更新する情報（文字列）

戻り値

戻り値	内容
0	正常終了
-1	エラー※

注※

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 2 番目の配列の、2 番目の配列要素の値を「data2」に更新するコーディング例を次に示します。

```
int DllFunc5(int argc,void** argv){
  int rc;
  rc = $UPDARRAY(argv,argv[1],2,"data2");
  if(rc) if($GETSTATUS(argv) != JAM_SCRIPTAPI_NORMAL &&
  $GETSTATUS(argv) != JAM_SCRIPTAPI_NODATA) return -1;
```

```
  return 0;
}
```

■$UPDARRAYBYKEY（キー付き配列データの更新マクロ）

配列変数の配列要素の値を更新します。配列要素は、キーおよびキー内の配列番号で指定します。

形式

```
int $UPDARRAYBYKEY(void ** argv,void* array,char* key,int pos,char* value);
```

引数

- argv
  スクリプトで指定した配列変数群の先頭アドレス
- array
  配列変数群の要素の一つ（配列要素を更新するユーザ関数の引数）
- key
  キー
- pos
  キー内配列番号（「1」から開始）
- value
  更新する情報（文字列）

戻り値

戻り値	内容
0	正常終了
-1	エラー※

注※

詳細なエラー情報は、$GETSTATUS（マクロの終了状態の取得マクロ）で取得できます。

コーディング例

スクリプトで指定した 2 番目の配列で、キー「key1」で格納された 1 番目の配列要素の値を「data1」に更新するコーディングの例を次に示します。

```
int DllFunc6(int argc,void** argv){
  int rc;
  rc = $UPDARRAYBYKEY(argv,argv[1],"key1",1,"data1");
  if(rc) if($GETSTATUS(argv) != JAM_SCRIPTAPI_NORMAL &&
  $GETSTATUS(argv) != JAM_SCRIPTAPI_NODATA) return -1;
  return 0;
}
```

$DLLFREE（DLL 解放関数）

組み込み関数$DLLLOAD でロードした DLL の使用を終了（DLL を解放）します。

形式

```
$DLLFREE(DLLオブジェクト)
```

指定する値

- DLL オブジェクト
  組み込み関数$DLLLOAD で求めた DLL オブジェクトの変数名を指定します。

使用する DLL のインターフェース

$DLLFREE で呼び出す関数の形式を次に示します。

```
void aim_free(void* object)
```

組み込み関数$DLLFREE では、DLL オブジェクトを引数として aim_free 関数を実行し、実行完了後に、DLL をアンロードします。aim_free 関数では、aim_init 関数や組み込み関数$DLLEXEC2 で呼び出した関数で使用した共有メモリやそのほかのリソースを解放します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどちらかを示す • 指定した DLL オブジェクトの変数名が、$DLLLOAD で求めた DLL オブジェクトではない • 引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

$DLLEXEC2（DLL 実行関数）の記述例を参照してください。

$DLLLOAD（DLL ロード関数）

ユーザ関数を実装した DLL をロードし、オブジェクトを取得します。

ロードに成功すると、取得した DLL オブジェクトを返します。ロードに失敗した場合、0 バイトの文字列を返します。

形式

```
DLLオブジェクト＝$DLLLOAD（DLL名）
```

指定する値

- DLL オブジェクト
  DLL オブジェクトを設定する変数名を指定します。
- DLL 名
  ロードする DLL の名称を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。
  DLL の名称は、jamscript コマンドの-bp オプションで指定したベースパスを基点にした相対パスで指定します。-bp オプションを省略した場合は、Asset Console のインストール先フォルダ¥scriptwork が基点となります。

使用する DLL のインターフェース

呼び出される DLL に、次の 3 つの実行制御関数をエクスポートしておく必要があります。

- インスタンス初期化のための aim_init 関数
- エラーメッセージ応答のための aim_getmessage 関数
- インスタンス解放のための aim_free 関数

各実行制御関数の形式を次に示します。

```
void* aim_init()
void aim_free(void* dllobj)
int aim_getmessage(void* dllobj, char** msg)
```

DLL のロード成功時は、インスタンス生成のために aim_init 関数を呼び出します。aim_init 関数のリターンには、エラーがありません。エラー発生時に、詳細なエラー情報をログファイルに出力するには、エラーメッセージのアドレスなどを aim_init 関数のリターン情報としてください。また、エラーメッセージを Asset Console のログファイルに出力するには、aim_init 関数のあとに呼び出される aim_getmessage 関数でメッセージを設定してください。

スクリプト側では、エラーを組み込み関数$GETSTATUS で検出したら、組み込み関数$DLLFREE を実行し、DLL を解放してください。$DLLFREE の実行によって aim_free 関数が呼び出されるので、aim_init 関数のリターン情報の情報域を開放してください。

なお、aim_init 関数が正常終了した場合には、必ず aim_getmessage 関数は 0 をリターンしてください。これで組み込み関数$DLLEXEC2 が実行可能な状態となります。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	−
ERROR	次のどれかを示す • 指定した DLL が存在しない • DLL のロードに失敗した • DLL 内に必要な関数のエントリが見つからない
スクリプト中断	次のどれかを示す • 指定した DLL が存在しない • DLL のロードに失敗した • 引数の誤り、またはそのほかのエラー

(凡例)

−：該当しない

注意事項

組み込み関数$DLLLOAD でロードする DLL は、あらかじめ用意しておく必要があります。コンパイル時には、Asset Console が提供しているヘッダーファイルをソースファイルと一緒にコンパイル環境に格納して、コンパイルしてください。また、コンパイルする際は、オプションに「/MT」を指定してください。

ヘッダーファイルの格納先を次に示します。

```
Asset Consoleのインストール先フォルダ¥sdk¥include
```

記述例

$DLLEXEC2（DLL 実行関数）の記述例を参照してください。

実行制御関数の作成

Asset Console のスクリプトから DLL のユーザ関数を呼び出すためには、次の 3 つの実行制御関数が必要です。

- aim_init

- aim_free
- aim_getmessage

■aim_init

機能

インスタンスを作成し、戻り値にそのアドレスを設定します。これによって、それ以降ほかの関数では、aim_init 関数で作成したアドレスを取得できます。インスタンス内に値を保持することで、スレッド間の競合を避けられます。

形式

```
void* aim_init()
```

戻り値

- 関数が成功した場合、この DLL で使用するハンドルのアドレスを返します。アドレスは NULL でもかまいません。リターン直後、aim_getmessage 関数が呼ばれるので、必ず 0 をリターンしてください。
- 継続可能なエラーや、エラーの内容を Asset Console のログファイルに出力する場合、エラー情報を設定したハンドルのアドレスを返して、aim_getmessage 関数でメッセージを取得できるようにします。また、同時にスクリプトの終了状態を決定するリターンコードを返すようにします。リターンコードについては、aim_getmessage 関数の戻り値を参照してください。

■aim_free

機能

aim_init 関数で作成したインスタンスを解放します。

形式

```
void aim_free(void* dllobj)
```

引数

- dllobj（入力情報）
  aim_init 関数で作成したインスタンス

■aim_getmessage

機能

作成する DLL 内のメッセージを、Asset Console のログに出力したり、スクリプトで容易に取得したりできます。aim_getmessage 関数は、次に示すタイミングで呼び出されます。

- aim_init 関数の実行後
  msg の内容を Asset Console のログに出力します。
- 作成した関数の実行後

関数の戻り値が負数（スクリプト中断エラー発生時）の場合に実行し、必要に応じて msg にアドレスを設定します。メッセージを設定した場合は、msg の内容が Asset Console のログに出力されます。

- 組み込み関数$DLLMSG 実行時

  組み込み関数$DLLMSG で実行され、メッセージを変数で取得します。

形式

```
int aim_getmessage(void* dllobj, char** msg)
```

引数

- dllobj（入力情報）

  aim_init 関数で作成したインスタンス

- msg（出力情報）

  メッセージが格納されたアドレス

戻り値

戻り値と内容、および戻り値に応じて実行される処理を次に示します。

戻り値	内容	実行できる処理
0	正常終了	スクリプトの状態を NORMAL に変更
1	Warning	スクリプトの状態を NODATA に変更
正数	Error	スクリプトの状態を ERROR に変更
負数	強制終了	スクリプトを強制終了

コーディング例

DLL のソースファイルのコーディング例を示したあと、アクセス定義ファイルからの呼び出しと関連づけについて説明します。

- DLL のソースファイル（C++）

```
#include <stdio.h>
#include <windows.h>

#include "jamScriptAPI.h"
extern "C" __declspec(dllexport) void*  aim_init();
extern "C" __declspec(dllexport) void   aim_free(void*);
extern "C" __declspec(dllexport) int    aim_getmessage(void*, char**);
extern "C" __declspec(dllexport) int    DllFunc1(int, void**);
extern "C" __declspec(dllexport) int    DllFunc2(int, void**);

typedef struct aimsample{
    int         status;
    int         datalen;
    char**      data;
    char        message[256];
}AIMSAMPLE;
```

```
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    return TRUE;
}
void* aim_init()
{
    AIMSAMPLE* dllobj = NULL;
    dllobj = (AIMSAMPLE*)LocalAlloc(LMEM_FIXED, sizeof(AIMSAMPLE));
    if(!dllobj) {
        /* エラー処理 */
        return NULL;
    }
    dllobj->status = 0;
    dllobj->datalen = 0;
    dllobj->data = NULL;
    *(dllobj->message) = '¥0';
    return dllobj;
}
void aim_free(void* dllobj)
{
    int i;
    if(dllobj){
        if(dllobj->data){
            for(i=0;i<dllobj->datalen;i++){
                if(*(dllobj->data+i)){
                    LocalFree(*(dllobj->data+i));
                    *(dllobj->data+i) = NULL;
                }
            }
            LocalFree(dllobj->data);
            dllobj->data = NULL;
        }
        LocalFree(dllobj);
    }
}
int aim_getmessage(void* dllobj, char** message)
{
    if(dllobj){
        *message = ((AIMSAMPLE*)dllobj)->message;
    }
    return 0;
}
int DllFunc1(int argc, void** argv)
{
    AIMSAMPLE* dllobj = NULL;
    int i,status,length;
    char* data;
    if(argc != 1){
       /* エラー処理 */
        return -1;
    }
    /*aim_initで返却したエリアを取得*/
    dllobj = (AIMSAMPLE*)$GETINITAREA(argv);
    if(!dllobj){
        /* エラー処理 */
```

```
        strcpy(dllobj->message, "引数の指定に誤りがあります。");
        return -1;
    }
    if(dllobj->status != 0){
        strcpy(dllobj->message, "呼び出し順序に誤りがあります。");
        return -1;
    }
    length = $GETARRAYLENGTH(argv, argv[0]);
    status = $GETSTATUS(argv);
    if(status != JAM_SCRIPTAPI_NORMAL){
        /* エラー処理 */
        strcpy(dllobj->message, "$GETARRAYLENGTH関数でエラーを検知しました。");
        return -1;
    }
    dllobj->datalen = length;
    dllobj->data = (char**)LocalAlloc(LMEM_FIXED, sizeof(char*)*length);
    if(!dllobj->data){
        /* エラー処理 */
        strcpy(dllobj->message, "メモリ確保に失敗しました。");
        return -1;
    }
    ZeroMemory(dllobj->data, sizeof(char*)*length);
    for(i=0;i<length;i++){
        data = (char*)$GETARRAY(argv, argv[0], i+1);
        status = $GETSTATUS(argv);
        if(status != JAM_SCRIPTAPI_NORMAL){
            /* エラー処理 */
            strcpy(dllobj->message, "$GETARRAY関数でエラーを検知しました。");
            return -1;
        }
        *(dllobj->data+length-(i+1)) = (char*)LocalAlloc(LMEM_FIXED, strlen(data)+1);
        if(!*(dllobj->data+length-(i+1))){
            /* エラー処理 */
            strcpy(dllobj->message, "メモリ確保に失敗しました。");
            return -1;
        }
        strcpy(*(dllobj->data+length-(i+1)), data);
    }
    /* 固有の処理 */
    dllobj->status = 1;
    *(dllobj->message) = '¥0';
    return 0;
}
int DllFunc2(int argc, void** argv)
{
    AIMSAMPLE* dllobj = NULL;
    int i;
    if(argc != 1){
        /* エラー処理 */
        return -1;
    }
    /*aim_initで返却したエリアを取得*/
    dllobj = (AIMSAMPLE*)$GETINITAREA(argv);
    if(!dllobj){
        /* エラー処理 */
        return -1;
    }
    if(dllobj->status != 1){
```

```
        strcpy(dllobj->message, "呼び出し順序に誤りがあります。");
        return -1;
    }
    if(dllobj->data){
        for(i=0;i<dllobj->datalen;i++){
            if(*(dllobj->data+i)){
                $SETARRAY(argv, argv[0], *(dllobj->data+i));
            }
        }
    }
    dllobj->status = 2;
    *(dllobj->message) = '¥0';
    return 0;
}
```

- アクセス定義ファイルからの呼び出しと関連づけ

  この例では、「DllFunc1」で取得した配列データの逆順を「DllFunc2」で出力します。また、「DllFunc1」、「DllFunc2」の順序で実行されない場合には、スクリプトを中断します。

```
#AssetInformationManager HTML 0005

[VAR]
  DLLOBJ
  DATA
  STATUS[ARRAY]
  ARY1
  ARY2
[SET_VALUE]
  DLLOBJ = $DLLLOAD('jamsample.dll')
  STATUS = $GETSTATUS()
[IF]
  STATUS != NORMAL
[THEN]
  [SET_VALUE]
      #エラー処理
      $EXIT(1)
[IF_END]
[SET_VALUE]
  $SETARRAY(ARY1, 1)
  $SETARRAY(ARY1, 2)
  $SETARRAY(ARY1, 3)
  $DLLEXEC2(DLLOBJ,'DllFunc1',ARY1)
  $DLLEXEC2(DLLOBJ,'DllFunc2',ARY2)
  DATA = $GETARRAY(ARY2,1)
  $ECHO(DATA)
  DATA = $GETARRAY(ARY2,2)
  $ECHO(DATA)
  DATA = $GETARRAY(ARY2,3)
  $ECHO(DATA)
  $DLLFREE(DLLOBJ)
```

出力結果

3

2

1

$DLLMSG（DLL メッセージ取得関数）

直前に実行した組み込み関数$DLLEXEC2 および$DLLLOAD のメッセージを取得します。組み込み関数$DLLMSG でメッセージを取得するには、呼び出す DLL の aim_getmessage 関数でメッセージのアドレスを設定してください。

形式

```
変数名＝$DLLMSG（DLLオブジェクト）
```

指定する値

- DLL オブジェクト

  組み込み関数$DLLLOAD で求めた DLL オブジェクトの変数名を指定します。

使用する DLL のインターフェース

組み込み関数$DLLMSG で呼び出す関数の形式を次に示します。

```
int aim_getmessage( void* object      //aim_init関数の戻り値
                  ,char** message     //メッセージ参照ポインタ
                  )
```

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了（関数の戻り値が 0 の場合）
NODATA	警告付き終了（関数の戻り値が 1 の場合）
ERROR	異常終了（関数の戻り値が正数の場合）
スクリプト中断	次のどちらかを示す • 異常終了（関数の戻り値が負数の場合）※ • 引数の誤り、またはそのほかのエラー

注※

指定した DLL オブジェクトの変数名が、組み込み関数$DLLLOAD で求めた DLL オブジェクトではないことを示しています。

記述例

$DLLEXEC2（DLL 実行関数）の記述例を参照してください。

$ECHO（標準出力へのメッセージ出力関数）

コマンドプロンプトにメッセージを出力します。

形式

```
$ECHO(メッセージ)
```

指定する値

- メッセージ

  メッセージを定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

記述例

「Hello world」のメッセージが出力される場合の記述例を次に示します。

```
[SET_VALUE]
  $ECHO('Hello world')

[SET_VALUE]
  MSG = 'Hello world'
  $ECHO(MSG)
```

実行結果

```
Hello world
Hello world
```

$ENVIRONMENT（サーバ環境情報の取得関数）

資産管理サーバの環境設定の情報を取得します。

形式

```
返却値=$ENVIRONMENT(セクション名,キー名)
```

指定する値

- 返却値

  取得した環境情報を設定する変数名を指定します。

- セクション名

  資産管理サーバの環境設定のカテゴリに対応したセクション名を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- キー名

  環境設定項目の値に対応したキー名を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

環境設定項目と値に対応したセクション名およびキー名については、マニュアル「JP1/IT Desktop Management 2 - Asset Console 構築・運用ガイド」を参照してください。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	指定したセクション名またはキー名が存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

環境設定で指定した、契約履歴の取得設定を取得する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL= $ENVIRONMENT('BASE','CONTRACT_HISTORY')

  MSG = 'BASE CONTRACT_HISTORY = ' +VAL
  $ECHO(MSG)
```

実行結果

BASE CONTRACT_HISTORY = YES

$EXIT（処理終了関数）

アクセス定義ファイルの処理を終了します。

形式

```
$EXIT(終了コード)
```

指定する値

- 終了コード

  アクセス定義ファイルの終了コードの数値を、定数または変数で指定します。指定できる値の範囲は0～9です。

注意事項

終了コードに指定できる範囲外の値を指定した場合は、処理が中断されます。

記述例

機器状態が運用（002）の資産情報のデータ件数を出力し、データがなかった場合、終了コード「3」で処理を終了する記述例を次に示します。

```
[CLASS_FIND]
  AssetInfo
[FIND_DATA]
  (AssetInfo.AssetStatus = '002')AND
  (AssetInfo.AssetKind = '001')
[GET_VALUE]
  WORK = AssetInfo.AssetNo

[SET_VALUE]
  STATUS = $GETSTATUS()
  TOTAL = $DATACOUNT()
[IF]
  STATUS = NORMAL
[THEN]
  [SET_VALUE]
    MSG = 'DataCount : ' +TOTAL
    $ECHO(MSG)
[ELSE]
  [SET_VALUE]
    MSG = 'EXIT : 3'
    $ECHO(MSG)
    $EXIT(3)
[IF_END]
```

実行結果

```
EXIT：3
```

$FILEARRAY（配列データの CSV 出力関数）

配列に格納されているデータを、CSV ファイルの 1 行として出力します。

形式

```
$FILEARRAY(ファイルオブジェクト,配列変数名)
```

指定する値

- ファイルオブジェクト
  組み込み関数$FILEOPEN で求めたファイルオブジェクトの変数名を指定します。
- 配列変数名
  CSV ファイルのレコードとして出力したい配列変数名を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどちらかを示す • $FILEOPEN で求めたファイルオブジェクトではない • 引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

$FILEOPEN（ファイル編集開始関数）の記述例を参照してください。

$FILECLOSE（ファイル編集終了関数）

データを出力するためのファイルの編集完了を宣言します。

形式

```
$FILECLOSE(ファイルオブジェクト)
```

指定する値

- ファイルオブジェクト

  編集を終了するファイルのファイルオブジェクトを指定します。ここには、組み込み関数$FILEOPENで求めたファイルオブジェクトの変数名を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどれかを示す • $FILEOPEN で求めたファイルオブジェクトではない • ファイルを閉じる際にエラーが発生した • 引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

$FILEOPEN（ファイル編集開始関数）の記述例を参照してください。

$FILECOPY（ファイルコピー関数）

ファイルをコピーします。

形式

```
$FILECOPY(コピー元ファイル名,コピー先ファイル名)
```

指定する値

- コピー元ファイル名、コピー先ファイル名

  ファイル名を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  ファイル名は、jamscript コマンドの-bp オプションで指定したベースパスを基点にした相対パスで指定します。-bp オプションを省略した場合は、Asset Console のインストール先フォルダ¥scriptwork が基点となります。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	次のどれかを示す • コピー元のファイルが存在しない • 出力先のパスが存在しない • ファイル名の指定に誤りがある
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

ファイル「input.csv」を「output.csv」にコピーする場合の記述例を次に示します。

```
[SET_VALUE]
  SRCFILE = 'input.csv'
  OUTFILE = 'output.csv'
  $FILECOPY(SRCFILE, OUTFILE)
```

$FILEDEL（ファイル削除関数）

ファイルを削除します。

形式

```
$FILEDEL(ファイル名)
```

指定する値

- ファイル名

  削除するファイルのファイル名を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  ファイル名は、jamscript コマンドの-bp オプションで指定したベースパスを基点にした相対パスで指定します。-bp オプションを省略した場合は、Asset Console のインストール先フォルダ¥scriptwork が基点となります。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどれかを示す • ファイル名に指定したファイルが存在しない • ファイルを削除する際にエラー（ファイル排他など）が発生した • 引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

ファイル「input.csv」を「output.csv」としてコピーし、コピー元の「input.csv」を削除する場合の記述例を次に示します。

```
[SET_VALUE]
  SRCFILE = 'input.csv'
  OUTFILE = 'output.csv'
  $FILECOPY(SRCFILE, OUTFILE)
  $FILEDEL(SRCFILE)
```

$FILEOPEN（ファイル編集開始関数）

データを出力するためのファイルの編集開始を宣言します。

形式

```
ファイルオブジェクト=$FILEOPEN(ファイル名,モード)
```

指定する値

- ファイルオブジェクト

  編集を開始するファイルを設定する変数名を指定します。

- ファイル名

  ファイル名を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  ファイル名は、jamscript コマンドの-bp オプションで指定したベースパスを基点にした相対パスで指定します。-bp オプションを省略した場合は、Asset Console のインストール先フォルダ¥scriptwork が基点となります。

- モード

  編集する方法を、次のモードから 1 つ指定します。

  - NEW：新規にファイルを作成する。既存のファイルがある場合は指定しない。
  - RENEW：既存のファイルに上書きする。既存のファイルがない場合は、新規にファイルを作成する。
  - ADD：既存のファイルに追加書き込みする。新規にファイルを作成する場合は指定しない。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどれかを示す • ファイルオブジェクトに指定したファイルが存在しない • モードの指定に誤りがある • ファイルのオープンでエラー（ファイル排他など）が発生した • 引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

ファイル「output.csv」を作成し、「"資産 ID","資産番号","資産状態","資産種別"」を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  FH = $FILEOPEN('output.csv', RENEW)

  $SETARRAY(OUTLINE,'資産ID')
  $SETARRAY(OUTLINE,'資産番号')
  $SETARRAY(OUTLINE,'資産状態')
  $SETARRAY(OUTLINE,'資産種別')

  $FILEARRAY(FH, OUTLINE)
  $CLEARARRAY(OUTLINE)

  $FILECLOSE(FH)
```

実行結果

「output.csv」に次のように出力されます。

"資産 ID","資産番号","資産状態","資産種別"

$FILEPUT（ファイルへのデータ出力関数）

データをファイルに出力します。

形式

```
$FILEPUT(ファイルオブジェクト,文字列)
```

指定する値

- ファイルオブジェクト
  出力するファイルのファイルオブジェクトを指定します。ここには、組み込み関数$FILEOPEN で求めたファイルオブジェクトの変数名を指定します。
- 文字列
  ファイルに出力する文字列を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどれかを示す • $FILEOPEN で求めたファイルオブジェクトではない • ファイルへの書き込みでエラーが発生した • 引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

ファイル「output.csv」を作成し、「ABCDEFGHI」を 1 行に出力する場合の記述例を次に示します。

```
[SET_VALUE]
  FH = $FILEOPEN('output.csv', RENEW)

  $FILEPUT(FH, 'ABC')
  $FILEPUT(FH, 'DEF')
  $FILEPUT(FH, 'GHI')

  $FILECLOSE(FH)
```

実行結果

「output.csv」に次のように出力されます。

ABCDEFGHI

$FILEPUTLN（ファイルへのデータ出力時復帰改行の追加関数）

文字列をファイルに出力し、最後に復帰改行を追加します。

形式

```
$FILEPUTLN(ファイルオブジェクト,文字列)
```

指定する値

- ファイルオブジェクト

  出力するファイルのファイルオブジェクトを指定します。ここには、組み込み関数$FILEOPEN で求めたファイルオブジェクトの変数名を指定します。

- 文字列

  ファイルに出力する文字列を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどれかを示す • $FILEOPEN で求めたファイルオブジェクトではない • ファイルを閉じる際にエラーが発生した • 引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

ファイル「output.csv」を作成し、「ABC」、「DEF」、「GHI」を３行に分けて出力する場合の記述例を次に示します。

```
[SET_VALUE]
  FH = $FILEOPEN('output.csv', RENEW)

  $FILEPUTLN(FH, 'ABC')
  $FILEPUTLN(FH, 'DEF')
  $FILEPUTLN(FH, 'GHI')
  $FILECLOSE(FH)
```

実行結果

「output.csv」に次のように出力されます。

ABC

DEF

GHI

$FINDFILE（ファイル検索関数）

指定したフォルダ内のファイルを検索します。

形式

```
$FINDFILE(フォルダ名,配列変数名)
```

指定する値

- フォルダ名

  ベースパスからの相対パスを、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- 配列変数名

  フォルダ内のファイル名とフォルダ名が格納される配列名を、変数で指定します。

  この配列には、ファイルの場合は「File」、フォルダの場合は「Directory」がキー値で指定されます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
ERROR	指定したフォルダが有効ではない
スクリプト中断	次のどちらかを示す • ワイルドカードや「..」または「.」などの文字列が含まれている • 引数の誤り、またはそのほかのエラー

記述例

Asset Console のインストール先フォルダ¥wwwroot¥bin のファイルを検索して、1 番目のファイル名を出力する記述例を次に示します。

```
$FINDFILE('bin', FileData)
FILE=$GETARRAY(FileData, 1)
$ECHO(FILE)
```

実行結果

```
bin/default.htm
```

$FORMATMSG（メッセージの書式設定関数）

メッセージに出力する文字列の書式を設定します。

形式

```
メッセージ=$FORMATMSG(メッセージテキスト,文字列1(,文字列2(,…)))
```

指定する値

- メッセージ

  メッセージを変数で指定します。

- メッセージテキスト

  書式を設定する文字列を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- 文字列

  メッセージテキストに挿入される文字列を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  文字列は、最大 99 個まで指定できます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
ERROR	－
スクリプト中断	次のどちらかを示す • 引数が 99 個を超えた • 引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

「資産番号」と「処理内容」を挿入したメッセージを出力する場合の記述例を次に示します。

```
MSG = $FORMATMSG('資産番号%1の機器%2に失敗しました。','1001','削除')
$ECHO(MSG)
```

実行結果

資産番号 1001 の機器削除に失敗しました。

$GETARRAY（配列からのデータ取得関数）

配列に格納されている情報から、指定した配列番号の情報を取得します。

形式

```
返却値=$GETARRAY(配列変数名,配列番号)
```

指定する値

- **返却値**
  取得した情報を設定する変数名を指定します。
- **配列変数名**
  情報を取り出す対象の配列の変数名を指定します。
- **配列番号**
  情報を取り出す配列要素の配列番号を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は 1～2,147,483,647 です。
  指定した配列番号の配列要素がない場合は、0 バイトの文字列を返します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	指定した配列番号の情報が存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

$SETARRAY（配列へのデータ設定関数）の記述例を参照してください。

$GETARRAYBYKEY（キー指定による配列からのデータ取得関数）

キー付きで配列に格納されている情報から、指定したキーの情報を取得します。

形式

```
返却値=$GETARRAYBYKEY(配列変数名,キー値（,キー内配列番号））
```

指定する値

- **返却値**
  取得した値を設定する変数名を指定します。
- **配列変数名**
  情報を取り出す配列変数の変数名を指定します。
- **キー値**
  取得する情報のキーを指定します。指定したキー内の配列要素がない場合は、0バイトの文字列を返します。
- **キー内配列番号**
  指定したキーに対応するデータが複数ある場合、キー内の配列番号を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、1～2,147,483,647です。
  指定したキー内の配列番号の配列要素がない場合は、0バイトの文字列を返します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 指定したキーが存在しない • 指定したキー内の配列番号の情報が存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

$SETARRAYBYKEY（配列へのキー付きデータ設定関数）の記述例を参照してください。

$GETKEYFROMARRAY（キー情報の取得関数）

キー付きで配列に格納されている情報から、指定した配列番号のキー情報を取得します。

形式

```
返却値=$GETKEYFROMARRAY(配列変数名,配列番号)
```

指定する値

- **返却値**

  取得したキーを設定する変数名を指定します。

- **配列変数名**

  キー情報を取り出す配列変数の変数名を指定します。

- **配列番号**

  数字、定数または変数を指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。数字に指定できる範囲は 1～2,147,483,647 です。

  指定した配列番号の配列要素がない場合は、0 バイトの文字列を返します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	指定した配列番号の情報が存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

配列変数「ARY」の 5 番目のキー情報および値を取得する場合の記述例を次に示します。

```
[SET_VALUE]
  $CLEARARRAY(ARY)

  $SETARRAYBYKEY(ARY,'CPU','100')    # ARY[1] CPU[1]
  $SETARRAYBYKEY(ARY,'CPU','200')    # ARY[2] CPU[2]
  $SETARRAYBYKEY(ARY,'HD' ,'40')     # ARY[3] HD[1]
  $SETARRAYBYKEY(ARY,'HD' ,'20')     # ARY[4] HD[2]
  $SETARRAYBYKEY(ARY,'MEM','128')    # ARY[5] MEM[1]
  $SETARRAYBYKEY(ARY,'MEM','256')    # ARY[6] MEM[2]
```

```
KEY = $GETKEYFROMARRAY(ARY,5)
VAL = $GETARRAY(ARY, 5)
MSG = 'ARY[5]: KEY=' +KEY+ ' VAL=' +VAL
$ECHO(MSG)
```

実行結果

ARY[5]: KEY=MEM VAL=128

$GETPROFILEDATA（初期設定ファイル情報の取得関数）

指定された初期設定ファイル名に指定されているセクション内の、すべてのキーと値を配列変数に取得します。その際、セクション内のキーを配列のキー値として格納します。

形式

```
$GETPROFILEDATA(初期設定ファイル名,セクション名,配列変数名)
```

指定する値

- 初期設定ファイル名

  入力する初期設定ファイルのファイル名を指定します。ファイル名は、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  初期設定ファイルの格納場所は、Asset Console のインストール先フォルダ¥env です。

- セクション名

  取得するセクション名を指定します。セクション名は、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  セクション名が 0 バイトの文字列の場合、初期設定ファイルからすべてのセクションの名前をキー配列に格納します。このとき、配列にキー値は設定されません。

- 配列変数名

  初期設定ファイルの情報を取得する配列の変数名を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 初期設定ファイルが存在しない • 指定したセクションが存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

初期設定ファイル「Sample.ini」から、セクション「TITLE」、キー名「HardwareInfo」の値を取得する場合の記述例を次に示します。

```
[SET_VALUE]
  FILENAME = 'Sample.ini'
  SECTION  = 'TITLE'

  $GETPROFILEDATA(FILENAME, SECTION, ARY)
  VAL=$GETARRAYBYKEY(ARY, 'HardwareInfo')
  MSG = 'HardwareInfo = ' +VAL
  $ECHO(MSG)
```

「Sample.ini」ファイル

```
[TITLE]
AssetInfo    = 資産情報
HardwareInfo = ハードウェア情報
SoftwareInfo = ソフトウェア情報
```

実行結果

HardwareInfo = ハードウェア情報

$GETREGVALUE（レジストリ値の取得関数）

指定したレジストリの値を取得します。

形式

```
$GETREGVALUE('レジストリ名')
```

指定する値

- レジストリ名
  レジストリ名の文字列を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
ERROR	レジストリ取得失敗
スクリプト中断	次のどちらかを示す • 指定したレジストリの属性が文字列または「DWORD」以外 • 引数の誤り、またはそのほかのエラー

記述例

「KEYVERSION」の値（1050）を取得する場合の記述例を次に示します。

```
[SET_VALUE]
  AIMVERSION = $GETREGVALUE('KEYVERSION')
  STATUS = $GETSTATUS()
[IF]
  STATUS = NORMAL
[THEN]
  [SET_VALUE]
    MSG = 'AIMVERSION = ' +AIMVERSION
    $ECHO(MSG)
[ELSE]
  [SET_VALUE]
    MSG = '$GETREGVALUE (' +STATUS+ ')'
    $ECHO(MSG)
[IF_END]
```

実行結果

AIMVERSION = 1050

$GETROLE（ユーザ権限代入関数）

操作画面で資産管理業務を実行中のユーザの権限一覧を、指定された配列に取得します。

形式

```
$GETROLE(配列変数名)
```

指定する値

- 配列変数名

  配列の変数名を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどちらかを示す • データベースアクセスエラー • 引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

注意事項

ユーザに何も権限が設定されていない場合、処理は正常終了しますが、配列変数には情報が設定されません。

記述例

ユーザ権限を取得し、配列番号の 1 番目の権限（administrator）を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  $GETROLE(ARY)
  VAL = $GETARRAY(ARY, 1)
  MSG = 'ROLE = ' +VAL
  $ECHO(MSG)
```

実行結果

ROLE = administrator

$GETSESSION（セッション情報取得関数）

jamscript コマンドの-s オプションで指定した値を取得します。-s オプションの記述形式は、「-s セッション変数名=値」です。

形式

```
返却値=$GETSESSION(セッション変数名)
```

指定する値

- 返却値

  取得したセッション情報を設定する変数名を指定します。

- セッション変数名

  取得したセッションの変数名を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	指定したセッション変数が存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

注意事項

指定したセッション変数名の情報がない場合は、0 バイトの文字列が返されます。

記述例

例 1

jamscript コマンドの-s オプションを指定して、情報（aaaa）を取得する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL = $GETSESSION('OPTION')
  MSG = 'OPTION = ' +VAL
  $ECHO(MSG)
```

実行するコマンド

jamscript -f C:¥Test.txt -s OPTION=aaaa

実行結果

OPTION = aaaa

$GETSTATUS（ブロック終了状態取得関数）

処理の終了状態を取得します。取得できる終了状態は、NORMAL（正常）、NODATA（データなし）、ERROR（異常）、MULTI（別ユーザでの更新あり）の４つです。

形式

```
返却値=$GETSTATUS()
```

指定する値

- 返却値

  取得した終了状態を設定する変数名を指定します。

記述例

資産番号「R11111」の情報を検索し、その資産の状態を「在庫」に更新する場合の記述例を次に示します。この例では、検索後、ほかの制御で「R11111」の情報がすでに更新されている場合はエラーとなります。

```
[TRANSACTION]
  [CLASS_FIND]
    AssetInfo
  [FIND_DATA]
    (AssetInfo.AssetNo  = 'R11111')
  [GET_VALUE]
    ASSETID     = AssetInfo.AssetID
    ASSETSTATUS = AssetInfo.AssetStatus
    ASSETNO     = AssetInfo.AssetNo
    UPDCK       = AssetInfo.UpdateTime

  [SET_VALUE]
    STATUS = $GETSTATUS()
  [IF]
    STATUS = NORMAL
    [THEN]
      [UPDATE]
        AssetInfo
      [DATA]
        AssetInfo.AssetID     = ASSETID
        AssetInfo.AssetStatus = '301'
        AssetInfo.UpdateTime  = UPDCK

      [SET_VALUE]
        STATUS = $GETSTATUS()
      [IF]
        (STATUS = NORMAL)
        [THEN]
          [SET_VALUE]
            MSG = 'ASSETNO('+ASSETNO+') の状態を更新しました'
            $ECHO(MSG)
      [IF_END]
```

```
      [IF]
        (STATUS = MULTI)
        [THEN]
          [SET_VALUE]
            MSG = 'ASSETNO(' +ASSETNO+ ') はすでに他制御で更新されています'
            $ECHO(MSG)
      [IF_END]
      [IF]
        (STATUS != NORMAL) and (STATUS != MULTI)
        [THEN]
          [SET_VALUE]
            MSG = 'ASSETNO(' +ASSETNO+ ') の状態を更新できません'
            $ECHO(MSG)
      [IF_END]

    [ELSE]
      [SET_VALUE]
        MSG = 'ASSETNO(' +ASSETNO+ ') は登録されていません'
        $ECHO(MSG)
  [IF_END]
[TRANSACTION_END]
```

この例のように同時更新を抑止する場合は、「クラス名.UpdateTime」を検索時に取得して、取得した値をそのまま[UPDATE]タグで指定します。同時更新の抑止は、更新の場合だけに有効です。追加・削除では同時更新を抑止できません。

$GETTEMPNAME（一時ファイル名指定関数）

セッション単位に管理する一時ファイル名を指定します。

組み込み関数$GETTEMPNAME で指定したファイル名のファイルは、ログアウト、強制ログアウト、およびタイムアウトのセッション終了時に削除されます。CSV ファイルなどをダウンロードする場合は、組み込み関数$GETTEMPNAME で指定した名称を使用すると、作成したファイルを自動的に削除できます。jamscript コマンドで使用するときは、コマンドの終了時に削除されます。

また、ダウンロードするファイルをブラウザのヘルパーアプリケーションで開きたい場合は、ファイルに関連づけられたヘルパーアプリケーションの拡張子を指定します。

形式

```
返却値=$GETTEMPNAME(ユニーク文字列)
```

指定する値

- **返却値**
  取得した一時ファイル名を設定する変数名を指定します。
- **ユニーク文字列**
  セッション内でユニークとなる文字列を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

組み込み関数$GETTEMPNAME を使用して、一時ファイル名を取得する場合の記述例を次に示します。一時ファイル名は、実行結果に示すような Asset Console が自動的に設定したファイル名になります。

```
[SET_VALUE]
  FILENAME = $GETTEMPNAME('Sample.csv')
```

```
MSG = 'FILENAME = ' +FILENAME
$ECHO(MSG)
```

実行結果

FILENAME = csv/$$3FAE0F01000004EC0001$4$Sample.csv

$GOSUB（サブルーチン実行関数）

[SUB]タグで定義したサブルーチンを実行します。実行するサブルーチンは、この関数より前に定義しておく必要があります。

形式

```
$GOSUB(サブルーチン名)
```

指定する値

- サブルーチン名

  サブルーチン名には、半角英数字と「_（アンダーバー）」が使用できます。ただし、サブルーチン名の先頭に、半角数字は使用できません。また、サブルーチン名は、英文字の大小文字を区別して使用してください。

記述例

セッション情報「MSG」の値に従って、メッセージを出力する処理をサブルーチン化した場合の記述例を次に示します。

```
[SUB]
  ECHO
  [IF]
    MSG = '1'
  [THEN]
    [SET_VALUE]
      $ECHO(ECHOMSG)
  [IF_END]
[SUB_END]

[BEGIN]
  [SET_VALUE]
    MSG = $GETSESSION('MSG')

  ECHOMSG = 'Hello world'
  $GOSUB(ECHO)
[END]
```

$ISNULL（NULL 判定関数）

[CSV_COLUMN_NAME]タグで取得したデータが、NULL か 0 バイト文字かどうかを判定します。

形式

```
返却値=$ISNULL(カラム名)
```

指定する値

- 返却値

  判定した結果を設定する変数名を指定します。カラム名で指定したカラムの値が NULL の場合 1 を、0 バイト文字の場合は 0 を返します。

- カラム名

  [CSV_COLUMN_NAME]タグで定義したカラムの変数名を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

ファイル「input.csv」からデータを取得し、「COLUMN1」が NULL のときは「COLUMN1 IS NULL」を、0 バイト文字のときは「COLUMN1 LENGTH IS 0」を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  FILENAME = 'input.csv'
  CNT = 1
[CSV_FILE_LOOP]
  FILENAME
  [CSV_COLUMN_NAME]
    COLUMN1 = 1
  [BEGIN]
    [SET_VALUE]
      LEN = $LENGTH(COLUMN1)
    [IF]
      LEN = 0
      [THEN]
```

```
        [SET_VALUE]
          VAL=$ISNULL(COLUMN1)
        [IF]
          VAL = 1
          [THEN]
            [SET_VALUE]
              MSG = 'LINE('+CNT+') COLUMN1 IS NULL'
          [ELSE]
            [SET_VALUE]
              MSG = 'LINE('+CNT+') COLUMN1 LENGTH IS 0'
        [IF_END]

      [ELSE]
        [SET_VALUE]
          MSG = 'LINE('+CNT+') COLUMN1 LENGTH IS '+LEN
    [IF_END]
    [SET_VALUE]
      $ECHO(MSG)
     CNT = $ADD(CNT,1)
  [END]

  [SET_VALUE]
    $SETSTATUS(NORMAL)
[CSV_FILE_LOOP_END]
```

「input.csv」の内容

```
,bbb,ccc
"",bbb,ccc
aaa,bbb,ccc
"aaa",bbb,ccc
```

実行結果

LINE(1) COLUMN1 IS NULL

LINE(2) COLUMN1 LENGTH IS 0

LINE(3) COLUMN1 LENGTH IS 3

LINE(4) COLUMN1 LENGTH IS 3

$LDAPACS（ディレクトリ情報へのアクセス関数）

ディレクトリサービスへの接続認証、検索、エントリの取得、属性の取得など、ディレクトリ情報にアクセスできます。ただし、この組み込み関数を使用してディレクトリ情報を操作するには、ディレクトリ情報へのアクセス方法と関数を習得している必要があります。

形式

```
$LDAPACS(関数名,引数1（,引数2（,…））)
```

指定する値

- 関数名

  関数名を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- 引数

  関数への引数を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

注意事項

情報の取得に失敗した場合は、0バイトの文字列が返されます。

ディレクトリ情報へのアクセス関数で使用できる関数の詳細

組み込み関数$LDAPACSで使用できる関数と機能を次の表に示します。

表5-3　組み込み関数$LDAPACSで使用できる関数一覧

関数名	機能
CONNECT	ディレクトリサービスへの接続認証
CONVERT	ディレクトリ情報検索用文字列への変換
DISCONNECT	ディレクトリサービスとの接続解除
FIRSTENTRY	最初に検索されたエントリの取得
FREEENTRY	エントリの解放
FREERESULT	検索結果の解放
GETDN	エントリのDNの取得
NEXTENTRY	2番目以降に検索されたエントリの取得
SEARCH	ディレクトリサービスの検索
SELECTVALUE	属性値の取得

各関数の機能、記述形式、引数、および終了状態について、これ以降で説明します。終了状態は、組み込み関数$GETSTATUS で取得した終了状態によって異なります。

■CONNECT

ディレクトリサービスに接続認証をして、ディレクトリ情報オブジェクトを返します。

形式

```
$LDAPACS('CONNECT',LDAPOBJ,HOST,PORT,USERDN,PASSWD)
```

引数

引数	指定型	説明
LDAPOBJ	ディレクトリ情報オブジェクト	ディレクトリ情報オブジェクトを設定する変数名を指定します。
HOST	変数、定数	ディレクトリサーバのホスト名または IP アドレスを指定します。
PORT	変数、定数	ディレクトリサーバのポート番号を指定します。
USERDN	変数、定数	接続認証のためのユーザ DN を指定します。
PASSWD	変数、定数	接続認証のためのパスワードを指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	引数の誤り
スクリプト中断	次のどれかを示す • ディレクトリの接続の際に、エラーが発生した • 認証エラー • そのほかのエラー

（凡例）

－：該当しない

■CONVERT

ディレクトリサービスで検索するための文字列に変換します。

形式

```
返却値= $LDAPACS('CONVERT',SOURCE)
```

- **返却値**

  変換した文字列を設定する変数名を指定します。

引数

引数	指定型	説明
SOURCE	変数、定数	変換する文字列を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	次のどちらかを示す • 変換失敗 • 引数の誤り
スクリプト中断	次のどちらかを示す • 変数が定義されていない • そのほかのエラー

（凡例）

－：該当しない

■DISCONNECT

ディレクトリサービスの接続解除およびその下位のオブジェクトをすべて解放します。

形式

```
$LDAPACS('DISCONNECT',LDAPOBJ)
```

引数

引数	指定型	説明
LDAPOBJ	ディレクトリ情報オブジェクト	CONNECT で求めたディレクトリ情報オブジェクトを指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了

終了状態	内容
NODATA	－
ERROR	CONNECT で求めたディレクトリ情報オブジェクトではない
スクリプト中断	次のどちらかを示す • 変数が定義されていない • そのほかのエラー

（凡例）

－：該当しない

■FIRSTENTRY

検索結果オブジェクトから最初に検索されたエントリオブジェクトを取得します。取得したオブジェクトを解放するためには、FREEENTRY を呼び出す必要があります。

形式

```
$LDAPACS('FIRSTENTRY',LDAPENT,LDAPRST)
```

引数

引数	指定型	説明
LDAPENT	エントリオブジェクト	エントリオブジェクトを設定する変数名を指定します。
LDAPRST	検索結果オブジェクト	検索結果オブジェクトを指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	エントリがない
ERROR	次のどちらかを示す • SEARCH で求めた検索オブジェクトではない • 引数の誤り
スクリプト中断	そのほかのエラー

■FREEENTRY

指定したエントリオブジェクトおよびその下位のオブジェクトをすべて解放します。

形式

```
$LDAPACS('FREEENTRY',LDAPENT)
```

引数

引数	指定型	説明
LDAPENT	エントリオブジェクト	エントリオブジェクトを指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	次のどちらかを示す • FIRSTENTRY または NEXTENTRY で求めたエントリオブジェクトではない • 引数の誤り
スクリプト中断	そのほかのエラー

（凡例）

－：該当しない

■FREERESULT

指定した検索結果オブジェクトおよびその下位のオブジェクトをすべて解放します。

形式

```
$LDAPACS('FREERESULT',LDAPRST)
```

引数

引数	指定型	説明
LDAPRST	検索結果オブジェクト	検索結果オブジェクトを指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	次のどちらかを示す • SEARCH で求めた検索オブジェクトではない • 引数の誤り
スクリプト中断	そのほかのエラー

（凡例）

－：該当しない

■GETDN

エントリオブジェクトから、識別名（文字列）を取得します。取得した文字列は解放できません。FREEENTRY を使用して、その上位のオブジェクトを解放する必要があります。

形式

```
$LDAPACS('GETDN',LDAPDN,LDAPENT)
```

引数

引数	指定型	説明
LDAPDN	DN 識別名	DN 識別名を設定する変数名を指定します。
LDAPENT	エントリオブジェクト	エントリオブジェクトを指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	次のどちらかを示す • FIRSTENTRY または NEXTENTRY で求めたエントリオブジェクトではない • 引数の誤り
スクリプト中断	そのほかのエラー

（凡例）

－：該当しない

■NEXTENTRY

検索結果オブジェクトから 2 番目以降に検索されたエントリオブジェクト取得します。FIRSTENTRY を呼び出さないで、この関数を呼び出すことはできません。取得したオブジェクトを解放するためには、FREEENTRY を呼び出す必要があります。

形式

```
$LDAPACS('NEXTENTRY',LDAPENT,LDAPRST)
```

引数

引数	指定型	説明
LDAPENT	エントリオブジェクト	エントリオブジェクトを設定する変数名を指定します。
LDAPRST	検索結果オブジェクト	検索結果オブジェクトを指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	次のどちらかを示す • SEARCH で求めた検索結果オブジェクトではない • 引数の誤り
スクリプト中断	そのほかのエラー

（凡例）

－：該当しない

■SEARCH

LDAP サーバに対して、同期検索を行います。

検索結果オブジェクトを解放するためには、FREERESULT を呼ぶか、その上位のオブジェクトを解放する必要があります。

形式

```
$LDAPACS('SEARCH',LDAPRST,LDAPOBJ,BASE,FILTER,SCOPE)
```

引数

引数	指定型	説明
LDAPRST	検索結果オブジェクト	検索結果オブジェクトを設定する変数名を指定します。
LDAPOBJ	ディレクトリ情報オブジェクト	CONNECT で取得したディレクトリ情報オブジェクトを指定します。
BASE	変数、定数	検索を開始するベースオブジェクトを指定します。
FILTER	変数、定数	検索フィルタを指定します。
SCOPE	変数、定数	ベースオブジェクトを基点として検索するディレクトリ情報の階層を次の中から指定します。 • LDAP_SCOPE_SUBTREE（ベース下のすべてのオブジェクトに対して検索） • LDAP_SCOPE_ONELEVEL（ベース直下のオブジェクトに対して検索） • LDAP_SCOPE_BASE（ベースそのものを検索）

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	該当するデータがない
ERROR	引数の誤り
スクリプト中断	そのほかのエラー

■SELECTVALUE

エントリオブジェクトから、属性名称を指定して、1 つ目の属性値（文字列）を取得します。

取得した文字列は、解放できません。FREEENTRY によってその上位のエントリオブジェクトを解放する必要があります。

形式

```
$LDAPACS('SELECTVALUE',LDAPSEL,LDAPENT,KEYNAME)
```

引数

引数	指定型	説明
LDAPSEL	属性値	属性値（文字列）を設定する変数名を指定します。
LDAPENT	エントリオブジェクト	エントリオブジェクトを指定します。
KEYNAME	変数、定数	取得したい属性の属性名称を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	指定した属性の値がない
ERROR	次のどちらかを示す • FIRSTENTRY または NEXTENTRY で求めたエントリオブジェクトではない • 引数の誤り
スクリプト中断	そのほかのエラー

記述例

ディレクトリの「ou=people,o=xxxxxx.co.jp」に登録されているユーザで、属性「title;lang-ja」が「主任」のユーザの、DN と名称を出力する場合の記述例を次に示します。

```
[VAR]
STATUS
MSG
HOST
PORT
FILTER
BASE
SCOPE
FIRST
LDOBJ
LDRST
LDENT
DN
NAME

[SET_VALUE]
  HOST = 'localhost'
  PORT = '389'
  BASE = 'ou=people,o=xxxxxxx.co.jp'
  SCOPE= 'LDAP_SCOPE_ONELEVEL'

[SET_VALUE]
  $LDAPACS('CONNECT',LDOBJ,HOST,PORT,'','')              # CONNECT
  STATUS = $GETSTATUS()

  [SET_VALUE]
    FILTER = '(&(objectclass=*)(title;lang-ja='
    FILTER = FILTER+$LDAPACS('CONVERT','主任')           # CONVERT
    FILTER = FILTER+'))'
    # FILETER=(&(objectclass=*)(title;lang-ja=¥E4¥B8¥BB¥E4¥BB¥BB))

  $LDAPACS('SEARCH',LDRST,LDOBJ,BASE,FILTER,SCOPE)    # SEARCH
  FIRST = 1

    [DO]
      [IF]
        FIRST = 1
        [THEN]
          [SET_VALUE]
            $LDAPACS('FIRSTENTRY',LDENT,LDRST)          # GET FIRST ENTRY
            STATUS = $GETSTATUS()
            FIRST = 0
        [ELSE]
          [SET_VALUE]
            $LDAPACS('NEXTENTRY',LDENT,LDRST)           # GET NEXT ENTRY
            STATUS = $GETSTATUS()
      [IF_END]

      [IF]
        STATUS = NORMAL
        [THEN]
          [SET_VALUE]
            $LDAPACS('GETDN',DN,LDENT)                  # GET DN
            $LDAPACS('SELECTVALUE',NAME,LDENT,'cn')     # GET VALUE OF CN
            MSG='DN ['+DN+'] is '+NAME
            $ECHO(MSG)
            $LDAPACS('FREEENTRY',LDENT)                 # FREE ENTRY OBJECT
```

```
        [ELSE]
          [SET_VALUE]
            $BREAK()
        [IF_END]
    [DO_END]

  [SET_VALUE]
    $LDAPACS('FREERESULT',LDRST)                          # FREE SEARCH OBJECT

[SET_VALUE]
  $LDAPACS('DISCONNECT',LDOBJ)                            # FREE LDAP OBJECT
```

実行結果

DN [uid=user1, ou=people, o=xxxxxxx.co.jp] is Taro Asset

DN [uid=user3, ou=people, o=xxxxxxx.co.jp] is Hanako Asset

$LENGTH（文字列長取得関数）

指定した文字列の長さ（バイト数）を取得します。

形式

```
返却値=$LENGTH(文字列)
```

指定する値

- **返却値**

  取得した文字列の長さを設定する変数名を指定します。

- **文字列**

  長さを取得する文字列を定数、変数または配列変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  配列変数を指定すると、その配列に設定されている要素の数を返します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

文字列「string length」の長さを取得して出力する場合の記述例を次に示します。

```
[SET_VALUE]
  DATA = 'string length'
  VAL = $LENGTH(DATA)
  MSG = 'LENGTH = ' +VAL
  $ECHO(MSG)
```

実行結果

LENGTH = 13

$LOGMSG（ログ出力関数）

指定したメッセージをメッセージログファイル「ASTMESn.LOG」に出力します。

形式

```
$LOGMSG(種別,メッセージ)
```

指定する値

- 種別

  出力するメッセージの種別を指定します。指定できる種別について説明します。

  - E（ERROR）

    プログラムを終了させなければならない、重度のトラブルが発生したことを通知するメッセージです。

  - EW（WARNING）

    プログラムを終了させる必要はありませんが、一部機能が使えないなどのトラブルが発生したことを通知するメッセージです。

    メッセージの種別には、「W」が出力されます。

  - EI（INFORMATION）

    情報を通知するメッセージです。

    メッセージの種別には、「I」が出力されます。

- メッセージ

  メッセージを定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

注意事項

メッセージ種別に「E」、「EW」および「EI」以外の値を指定した場合、スクリプトが中断されます。

記述例

メッセージ種別を「ERROR」として、指定したメッセージをログファイルに出力する場合の記述例を次に示します。メッセージの先頭には、実行結果に示すような Asset Console で自動的に設定された数字が出力されます。実行結果のメッセージの見方については、マニュアル「JP1/IT Desktop Management 2 - Asset Console 構築・運用ガイド」を参照してください。

```
[SET_VALUE]
  MSG = 'メールの送信先アドレスが設定されていません。'
  $LOGMSG ('E', MSG)
```

実行結果

20040519212834.673 00000594(00000664) KDAM2G14-E メールの送信先アドレスが設定されていません。

$LOWER（文字列変換関数）

引数文字列中の半角英文字を小文字に変換します。

形式

```
返却値=$LOWER(文字列)
```

指定する値

- 返却値

  小文字に変換した文字列を設定する変数名を指定します。

- 文字列

  変換する文字列を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

システムインベントリ情報から取得したホスト名を、英小文字に変換する場合の記述例を次に示します。

```
[SET_VALUE]
  NAME = 'HOSTNAME'
  VAL = $LOWER(NAME)
  MSG = 'LOWER = ' +VAL
  $ECHO(MSG)
```

実行結果

LOWER = hostname

$MATCH（文字列評価関数）

文字列に使用されている文字を評価し適合する部分までの文字数を返します。

形式

```
返却値＝$MATCH（文字列,評価書式）
```

指定する値

- 返却値

  評価書式に適合する部分までの文字数を設定する変数名を指定します。評価に適合する文字がない場合、0 バイトの文字列を返します。

- 文字列

  評価対象の文字列を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- 評価書式

  文字列を評価するための書式を定数で指定します。許可する文字を範囲で指定する場合は、[a-b]の形式で指定します。

  評価書式には単純文字列のほかに、次に示す正規表現が使用できます。ただし、日本語文字コードの正規表現は指定できません。

  - .（ピリオド）

    任意の 1 文字に適合します。

  - [ ]（角括弧）

    括弧に囲まれた文字の任意の 1 文字、または「-（ハイフン）」で区切られた文字範囲のうち任意の 1 文字に適合します。例えば、「R[OAI]M」は「ROM」、「RAM」、「RIM」に適合します。

    また、「S[AE]+D」は「SAD」、「SED」、「SEED」、「SAAD」には適合しますが、「SAED」や「SEAD」には適合しません。

    「C[0-9]」は「C0」、「C1」、「C2」などに適合します。

    角括弧内の最初の文字として「^（アクサンシルコンフレックス）」を指定すると、意味が反対になり、「^」に続く文字以外のすべての文字に適合します。

  - [^]

    「^」に続く文字以外、または「-」で区切られた範囲内の文字以外の、任意の 1 文字に適合します。

    例えば、「x[^0-9]」は「xa」、「xb」、「xc」などには適合しますが、「x0」、「x1」、「x2」などには適合しません。

  - ^

    文字列の先頭に適合します。

  - $（ドル記号）

文字列の末尾に適合します。

- *（アスタリスク）

  直前にある文字または正規表現の 0 回以上の繰り返しに適合します。

  例えば、「ba*c」は「bc」、「bac」、「baac」、「baaac」などに適合します。

- +（加算記号）

  直前にある文字または正規表現の 1 回以上の繰り返しに適合します。

  例えば、「ba+c」は「bac」、「baac」、「baaac」などには適合しますが、「bc」には適合しません。

- ¥（円記号）

  エスケープ文字指定で、後続の 1 文字に適合します。正規表現文字列の中で特殊な意味を持つ文字（*や$など）の意味を無効にします。ただし、t が続く場合には、特殊文字のタブ文字に適合します。

- ¥t

  タブ文字に適合します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	評価に適合する文字がない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

例 1

半角英数字以外をチェックする場合の記述例を次に示します。

```
[SET_VALUE]
  DATA = 'user$1'
  VAL  = $MATCH(DATA,'[^a-zA-Z0-9]')
[IF]
  VAL = ''
  [THEN]
    [SET_VALUE]
      MSG  = 'MATCH OK'
      $ECHO(MSG)
  [ELSE]
    [SET_VALUE]
      MSG  = 'MATCH NG (' +VAL+ ')'
```

```
      $ECHO(MSG)
[IF_END]
```

実行結果

MATCH NG (4)

例 2

日付形式をチェックする場合の記述例を次に示します。

```
[SET_VALUE]
  DATA = '2015/04/01'
  VAL  = $MATCH(DATA,'^[1-2][0-9][0-9][0-9]/[0-1][0-9]/[0-3][0-9]$')
[IF]
  VAL != ''
  [THEN]
    [SET_VALUE]
      MSG  = 'MATCH OK'
      $ECHO(MSG)
  [ELSE]
    [SET_VALUE]
      MSG  = 'MATCH NG'
      $ECHO(MSG)
[IF_END]
```

実行結果

MATCH OK

$MOD（除算結果の余りの取得関数）

文字列を数値と見なして除算を実行し、その演算結果の余りを返します。

形式

```
返却値=$MOD(文字列,数字)
```

指定する値

- **返却値**

  演算結果を設定する変数名を指定します。結果として取得できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。結果が範囲外となった場合は 0 を返します。

- **文字列**

  被除数を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

- **数字**

  除数を定数または変数で指定します。小数点を含む数値を定数で指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。0 を指定した場合は、返却値に 0 が返されます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 文字列または数値に指定した値に誤りがある • 演算結果が表現できる値の範囲外
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

注意事項

「文字列」、「数字」に指定できない値が指定されている場合、および演算結果が表現できる範囲外となった場合は、返却値には 0 が返されます。

記述例

「10÷3」を演算し、余りを出力する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL1 = 10
  VAL2 = $MOD(VAL1, 3)

MSG = 'MOD: ' +VAL1+ ' MOD 3 = ' +VAL2
$ECHO(MSG)
```

実行結果

MOD: 10 MOD 3 = 1

$MUL（乗算結果の取得関数）

文字列を数値と見なして乗算を実行し、その演算結果を返します。

形式

```
返却値=$MUL(文字列,数字)
```

指定する値

- **返却値**

  演算結果を設定する変数名を指定します。結果として取得できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

- **文字列**

  被乗数となる定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

- **数字**

  乗数となる定数または変数で指定します。小数点を含む数値を定数で指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 文字列または数値に指定した値に誤りがある • 演算結果が表現できる値の範囲外
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

注意事項

「文字列」、「数字」に指定できない値が記述されている場合、および演算結果が表現できる値の範囲外となった場合は、返却値には 0 が返されます。

記述例

「10×5」を演算し、結果を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL1 = 10
  VAL2 = $MUL(VAL1, 5)

MSG = 'MUL:' +VAL1+ ' * 5 = ' +VAL2
$ECHO(MSG)
```

実行結果

MUL: 10 * 5 = 50

$NUMBER（データベースを利用した通番採番関数）

オブジェクトクラス「FunctionInfo」を使用して、資産管理システムのデータベースでユニークな番号を採番します。指定した機能 ID とサブ機能 ID 単位で、1～4,294,967,295 の 10 けたの整数を採番します。

4,294,967,295 を超えた場合、1 に戻ります。また、10 けたに満たない場合は、前に 0 を挿入します。

形式

```
返却値=$NUMBER(機能ID,サブ機能ID)
```

指定する値

- 返却値

  取得した通番を設定する変数名を指定します。

- 機能 ID

  オブジェクトクラス「FunctionInfo」で規定した用途の値を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- サブ機能 ID

  オブジェクトクラス「FunctionInfo」で規定された用途の中でカテゴリ分けをする場合、任意の文字列を 1～251 バイトで指定します。サブ機能 ID は、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	次のどちらかを示す • データベースアクセスエラー • 引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

注意事項

オブジェクトクラス「FunctionInfo」には、「機能 ID」に対応した排他制御用のロックレコードも登録する必要があります。

機能 ID「USER」、サブ機能 ID「Number」の場合に、「FunctionInfo」にインポートするデータファイルの例を次に示します。

```
OP,CreationClassName,FunctionID,ExtendID,UpdateDate,SequenceNo
a,FunctionInfo,USER,Number,2003/1/1,0
a,FunctionInfo,USER,NumberLock,2003/1/1,0
```

記述例

機能 ID「USER」、サブ機能 ID「Number」を使用して番号（0000000001）を取得する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL = $NUMBER('USER' , 'Number')
  MSG = 'NUMBER = ' +VAL
  $ECHO(MSG)
```

実行結果

NUMBER = 0000000001

$SETARRAY（配列へのデータ設定関数）

配列に情報を追加します。

形式

```
$SETARRAY(配列変数名,文字列)
```

指定する値

- 配列変数名
  値を追加する配列変数名を指定します。
- 文字列
  配列変数に追加する値を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

配列「ARY」を初期化し、ARY[1]「aaa」、ARY[2]「bbb」、ARY[3]「ccc」の配列データを作成する場合の記述例を次に示します。この例では、さらに、作成した配列変数の配列番号が 2 番目の変数の値を「ddd」に変更し、その内容を取得して出力しています。

```
[SET_VALUE]
  $CLEARARRAY(ARY)

  $SETARRAY(ARY,'aaa')
  $SETARRAY(ARY,'bbb')
  $SETARRAY(ARY,'ccc')

  $UPDARRAY(ARY,2,'ddd')

VAL = $GETARRAY(ARY, 2)
```

```
MSG = 'ARY = ' +VAL
$ECHO(MSG)
```

実行結果

ARY = ddd

$SETARRAYBYKEY（配列へのキー付きデータ設定関数）

キー付きで配列に情報を追加します。

形式

```
$SETARRAYBYKEY(配列変数名,キー値,文字列)
```

指定する値

- 配列変数名
  値を追加する配列変数の変数名を指定します。
- キー値
  配列変数に追加する値のキーを指定します。
- 文字列
  配列変数に追加する値を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

キーが「MEM」でキー内の配列番号が 1 番目のキーの値を、「128」から「1024」に変更して、変更した内容をキー指定で取得する場合の記述例を次に示します。

```
[SET_VALUE]
  $CLEARARRAY(ARY)

  $SETARRAYBYKEY(ARY,'CPU','100')    # ARY[1] CPU[1]
  $SETARRAYBYKEY(ARY,'CPU','200')    # ARY[2] CPU[2]
  $SETARRAYBYKEY(ARY,'HD' ,'40')     # ARY[3] HD[1]
  $SETARRAYBYKEY(ARY,'HD' ,'20')     # ARY[4] HD[2]
  $SETARRAYBYKEY(ARY,'MEM','128')    # ARY[5] MEM[1]
  $SETARRAYBYKEY(ARY,'MEM','256')    # ARY[6] MEM[2]
```

```
  $UPDARRAYBYKEY(ARY,'MEM',1,'1024')

VAL = $GETARRAYBYKEY(ARY,'MEM',1)
MSG = 'ARY MEM[1] = ' +VAL
$ECHO(MSG)
```

実行結果

ARY MEM[1] = 1024

$SETOPTION（実行オプション設定関数）

スクリプトでエラーが発生しても、スクリプトの処理を中断しないように設定します。

形式

```
$SETOPTION(’オプション名’,パラメーター)
```

指定する値

- オプション名
  指定するオプションを次に示します。1 つの関数には、1 つのオプションを指定します。
  - ErrorFlush
    スクリプトでエラーが発生した場合にスクリプトの処理を中断するかどうかを指定します。
- パラメーター
  オプション「ErrorFlush」で指定できるパラメーターを次に示します。
  - 0
    エラーが発生したらスクリプトの処理を中断します。
  - 1
    エラーが発生してもスクリプトの処理を続行します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
ERROR	パラメーターに規定値以外の値を指定した
FLUSH	エラーが発生したが処理を続行した（パラメーターに「1」を指定した場合）
スクリプト中断	次のどちらかを示す • 無効なオプションを指定した • 引数の誤り、またはそのほかのエラー

記述例

資産番号「1000000001」がすでに使用されていても、スクリプトの処理は中断しないで「MSG」の内容をログに出力する場合の記述例を次に示します。

```
[SET_VALUE]
  $SETOPTION(’ErrorFlush’, 1)
[APPEND]
 AssetInfo
```

```
[DATA]
 AssetInfo.AssetID = '1000000001'
 AssetInfo.AssetNo = '1000000001'
[SET_VALUE]
  $LOGMSG('E', MSG)
```

$SETSESSION（セッション情報設定関数）

文字列をセッション情報として設定します。

形式

```
$SETSESSION(セッション変数名,文字列)
```

指定する値

- セッション変数名
  定数または変数を指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。ただし、&（アンパサンド）で始まるセッション変数名は、登録および変更できません。
- 文字列
  定数または変数を指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

セッション変数「SID」に、「123456789」の値を設定する場合の記述例を次に示します。

```
[SET_VALUE]
  $SETSESSION('SID','123456789')

  VAL = $GETSESSION('SID')
  MSG = 'SID = ' +VAL
  $ECHO(MSG)
```

実行結果

SID = 123456789

$SETSTATUS（ブロック終了状態設定関数）

処理の終了状態を変更します。

形式

```
$SETSTATUS(ステータス定数)
```

指定する値

- ステータス定数

  NORMAL（正常）、NODATA（データなし）、ERROR（異常）、MULTI（別ユーザでの更新あり）、変数、または予約文字列を指定します。定数（NORMAL、ERROR、NODATA、MULTI）を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

ステータス定数に指定した状態になります。規定以外のステータス定数を指定した場合は、スクリプトの処理が中断されます。

記述例

機器状態が在庫（301）の資産情報をすべて検索し、機器状態を運用（002）に更新して、処理を終了する場合の記述例を次に示します。

```
[ASSET_ITEM_LOOP]
  [CLASS_FIND]
    AssetInfo
  [FIND_DATA]
    (AssetInfo.AssetStatus = '301')AND
    (AssetInfo.AssetKind = '001')
  [GET_VALUE]
    ASSETID = AssetInfo.AssetID
  [UPDATE]
    AssetInfo
  [DATA]
    AssetInfo.AssetID = ASSETID
    AssetInfo.AssetStatus = '002'
  [SET_VALUE]
    $SETSTATUS('NORMAL')
[ASSET_ITEM_LOOP_END]
```

$STRCMP（文字列比較関数）

文字列の比較をします。

形式

```
返却値=$STRCMP(文字列1,文字列2)
```

指定する値

- 返却値

  比較した結果を設定する変数名を指定します。

  - 文字列 1 が文字列 2 より小さい場合、戻り値に 0 が返されます。
  - 文字列 1 と文字列 2 が等しい場合、戻り値に 1 が返されます。
  - 文字列 1 が文字列 2 より大きい場合、戻り値に 2 が返されます。

- 文字列 1

  被比較文字列を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- 文字列 2

  比較する文字列を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

「DATA1」と「DATA2」の文字列を比較して、同じ場合「STRCMP IDENTICAL」を、異なる場合「STRCMP DIFFERENT ($STRCMP 返却値)」を出力する記述例を次に示します。

```
[SET_VALUE]
  DATA1 = 'Asset Console1'
```

```
  DATA2 = 'Asset Console2'
  VAL = $STRCMP(DATA1,DATA2)
[IF]
  VAL = 1
  [THEN]
    [SET_VALUE]
      MSG  = 'STRCMP IDENTICAL'
      $ECHO(MSG)
  [ELSE]
    [SET_VALUE]
      MSG  = 'STRCMP DIFFERENT (' +VAL+ ')'
      $ECHO(MSG)
[IF_END]
```

実行結果

```
STRCMP DIFFERENT (0)
```

$SUB（減算結果の取得関数）

文字列を数値と見なして減算を実行し、その演算結果を返します。

形式

```
返却値=$SUB(文字列,数字)
```

指定する値

- **返却値**
  演算結果を設定する変数名を指定します。結果として取得できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。
- **文字列**
  被減数となる定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。
- **数字**
  減数となる定数または変数で指定します。小数点を含む数値を定数で指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は、0.0001～999,999,999,999,999（15 けた）です。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 文字列、数値に指定した値に誤りがある • 演算結果が表現できる値の範囲外
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

注意事項

「文字列」、「数字」に指定できない値が記述されている場合、および演算結果が表現できる範囲外となった場合は、返却値に 0 が返されます。

記述例

「10－5」を演算し、結果を出力する場合の記述例を次に示します。

```
[SET_VALUE]
  VAL1 = 10
  VAL2 = $SUB(VAL1, 5)

MSG = ’SUB:’ +VAL1+ ’ - 5 = ’ +VAL2
$ECHO(MSG)
```

実行結果

SUB: 10 - 5 = 5

$SUBSTR（部分文字列抽出関数）

指定した文字列、抽出開始位置および文字列の長さの定義に従って、部分的に文字列を抽出します。

形式

```
返却値=$SUBSTR(処理対象文字列,抽出開始位置,抽出する文字列の長さ)
```

指定する値

- **返却値**

  部分抽出した文字列を設定する変数名を指定します。

- **処理対象文字列**

  抽出の対象となる文字列を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- **抽出開始位置**

  処理対象文字列の先頭文字を 0 として、抽出を開始する位置を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

  抽出開始位置には、0 以上の値を指定してください。存在しない抽出開始位置を指定した場合、0 バイトの文字列を返します。

- **抽出する文字列の長さ**

  抽出する部分の文字列の長さ（バイト数）を定数または変数で指定します。処理対象文字列の長さを超えて指定した場合、処理対象文字列の長さまでが有効となります。なお、抽出開始位置から最後の文字列をすべて指定する場合は、負数を指定します。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	指定した抽出開始位置が文字列に存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

「NAME」に文字列「ABCDEFG」が格納されている場合、結果として「VALUE」に「ABC」が抽出される記述例を次に示します。

```
[SET_VALUE]
  NAME =  ‘ABCDEFG’
  VALUE=$SUBSTR(NAME,0,3)
  $ECHO(VALUE)
```

実行結果

ABC

$TOKEN（トークン抽出関数）

指定した文字列、抽出するトークンの位置およびセパレータ文字の定義に従って、トークンを抽出します。

形式

```
返却値=$TOKEN(処理対象文字列,抽出トークン位置,セパレータ文字)
```

指定する値

- **返却値**

  抽出したトークンを設定する変数名を指定します。

- **処理対象文字列**

  トークンを抽出する対象となる文字列を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- **抽出トークン位置**

  抽出するトークンの位置を、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。セパレータ文字で切り出したトークンの先頭を 0 として、抽出するトークンの通番を指定します。例えば、aaa、bbb、ccc、ddd という文字列の場合、セパレータ文字を区切りとして抽出する文字（aaa）がトークンとなります。この例の場合は、次に示す 0～3 がトークン位置に当たります。

  0：aaa

  1：bbb

  2：ccc

  3：ddd

  また、トークン位置を誤って、切り出し範囲外を指定した場合、0 バイトの文字列を返します。

- **セパレータ文字**

  トークンを切り出す文字を定数または変数で指定します。使用できる文字は、半角の英数字および記号です。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	処理対象文字列に指定した値が抽出トークン位置にない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

「NAME」に文字列「ABC/DEF/GHI」が格納されている場合、結果として「VALUE」に「DEF」が抽出される記述例を次に示します。

```
[SET_VALUE]
  DATA = 'ABC/DEF/GHI'
  VAL = $TOKEN(DATA,1,'/')
  MSG = 'TOKEN = ' +VAL
  $ECHO(MSG)
```

実行結果

TOKEN = DEF

$UPDARRAY（配列データの更新関数）

配列変数の配列要素の値を更新します。配列要素は配列番号で指定します。

形式

```
$UPDARRAY(配列変数名,配列番号,文字列)
```

指定する値

- 配列変数名
  対象となる配列変数名を指定します。
- 配列番号
  変更する配列要素の配列番号を定数または、変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。指定できる値の範囲は 1～2,147,483,647 です。
- 文字列
  変更する値を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	指定した配列番号の情報が存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）
－：該当しない

記述例

$SETARRAY（配列へのデータ設定関数）の記述例を参照してください。

$UPDARRAYBYKEY（キー付き配列データの更新関数）

配列変数の配列要素の値を更新します。配列要素は、キーおよびキー内の配列番号で指定します。

形式

```
$UPDARRAYBYKEY（配列変数名,キー値（,キー内配列番号）,文字列）
```

指定する値

- **配列変数名**

  対象となる配列変数名を指定します。

- **キー値**

  更新する配列要素のキー値を指定します。定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

- **キー内配列番号**

  更新するキーに対応するデータが複数ある場合、キー内の配列番号を数字、定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。また、指定できる値の範囲は、1～2,147,483,647 です。

  この値は省略することができます。省略した場合のデフォルト値は 1 です。

- **文字列**

  変更する値を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	次のどちらかを示す • 指定したキーが存在しない • 指定したキー内の配列番号の情報が存在しない
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

$SETARRAYBYKEY（配列へのキー付きデータ設定関数）の記述例を参照してください。

$UPPER（文字列変換関数）

引数文字列中の半角英文字を大文字に変換します。

形式

```
返却値=$UPPER(文字列)
```

指定する値

- 返却値

  大文字に変換した文字列を設定する変数名を指定します。

- 文字列

  変換する文字列を定数または変数で指定します。定数を指定する場合は、「'（シングルクォーテーション）」で囲みます。

終了状態

処理の終了状態とその内容を次に示します。

終了状態	内容
NORMAL	正常終了
NODATA	－
ERROR	－
スクリプト中断	引数の誤り、またはそのほかのエラー

（凡例）

－：該当しない

記述例

変数「NAME」の内容を、大文字に変換して出力する場合の記述例を次に示します。

```
[SET_VALUE]
  NAME = 'computer name'
  VAL = $UPPER(NAME)
  MSG = 'UPPER = ' +VAL
  $ECHO(MSG)
```

実行結果

UPPER = COMPUTER NAME

付録

付録 A　各バージョンの変更内容

付録 A.1　11-50 の変更内容

- なし

付録 A.2　11-10 の変更内容

- Windows Server 2016 を次の製品の適用 OS に追加した。
  - JP1/IT Desktop Management 2 - Manager
  - JP1/IT Desktop Management 2 - Agent
  - JP1/IT Desktop Management 2 - Network Monitor
  - JP1/IT Desktop Management 2 - Asset Console
  - Remote Install Manager

付録 A.3　11-01 の変更内容

- Windows 10 を JP1/IT Desktop Management 2 - Network Monitor の適用 OS に追加した。

付録 A.4　11-00 の変更内容

- Windows 10 を次の製品の適用 OS に追加した。
  - JP1/IT Desktop Management 2 - Agent
  - JP1/IT Desktop Management 2 - RC Manager
  - Remote Install Manager
- Windows Server 2003 および Windows Server 2008（Windows Server 2008 R2 を除く）を次の製品の適用 OS 外とした。
  - JP1/IT Desktop Management 2 - Manager
  - JP1/IT Desktop Management 2 - Agent
  - JP1/IT Desktop Management 2 - Network Monitor
  - JP1/IT Desktop Management 2 - RC Manager

付録 B　このマニュアルの参考情報

付録 B.1　関連マニュアル

このマニュアルの関連マニュアルを次に示します。必要に応じてお読みください。

- JP1 Version 11 資産・配布管理 基本ガイド（3021-3-B51）
- JP1 Version 11 JP1/IT Desktop Management 2 導入・設計ガイド（3021-3-B52）
- JP1 Version 11 JP1/IT Desktop Management 2 構築ガイド（3021-3-B53）
- JP1 Version 11 JP1/IT Desktop Management 2 運用ガイド（3021-3-B54）
- JP1 Version 11 JP1/IT Desktop Management 2 配布機能 運用ガイド（3021-3-B55）
- JP1 Version 11 JP1/IT Desktop Management 2 - Asset Console 構築・運用ガイド（3021-3-B56）
- JP1 Version 11 JP1/IT Desktop Management 2 メッセージ（3021-3-B58）

付録 B.2　このマニュアルでの表記

このマニュアルでは、次の製品を略称で表記しています。

このマニュアルでの表記		正式名称
Asset Console		JP1/IT Desktop Management 2 - Asset Console
Windows 7※1	Windows 7 Enterprise	Microsoft(R) Windows(R) 7 Enterprise
	Windows 7 Home Basic	Microsoft(R) Windows(R) 7 Home Basic
	Windows 7 Home Premium	Microsoft(R) Windows(R) 7 Home Premium
	Windows 7 Professional	Microsoft(R) Windows(R) 7 Professional
	Windows 7 Starter	Microsoft(R) Windows(R) 7 Starter
	Windows 7 Ultimate	Microsoft(R) Windows(R) 7 Ultimate
Windows 8※1	Windows 8	Windows(R) 8
	Windows 8 Enterprise	Windows(R) 8 Enterprise
	Windows 8 Pro	Windows(R) 8 Pro
Windows 8.1※1	Windows 8.1	Windows(R) 8.1
	Windows 8.1 Enterprise	Windows(R) 8.1 Enterprise
	Windows 8.1 Pro	Windows(R) 8.1 Pro
Windows 10※1	Windows 10 Enterprise	Windows(R) 10 Enterprise

このマニュアルでの表記			正式名称
Windows 10※1	Windows 10 Pro		Windows(R) 10 Pro
Windows Server 2008 R2※1※2			Microsoft(R) Windows Server(R) 2008 R2 Datacenter
			Microsoft(R) Windows Server(R) 2008 R2 Enterprise
			Microsoft(R) Windows Server(R) 2008 R2 Standard
Windows Server 2012※1※3	Windows Server 2012※3	Windows Server 2012 Datacenter	Microsoft(R) Windows Server(R) 2012 Datacenter
		Windows Server 2012 Standard	Microsoft(R) Windows Server(R) 2012 Standard
	Windows Server 2012 R2		Microsoft(R) Windows Server(R) 2012 R2 Datacenter
			Microsoft(R) Windows Server(R) 2012 R2 Standard
Windows Server 2016※1	Windows Server 2016 Datacenter		Microsoft(R) Windows Server(R) 2016 Datacenter
	Windows Server 2016 Standard		Microsoft(R) Windows Server(R) 2016 Standard

注※1

OS による機能差がない場合、Windows 7、Windows 8、Windows 8.1、Windows 10、Windows Server 2008 R2、Windows Server 2012、および Windows Server 2016 を総称して Windows と表記します。

注※2

Server Core インストールは対象外です。

注※3

Windows Server 2012 R2 を併記している場合は、Windows Server 2012 に Windows Server 2012 R2 は含みません。

付録 B.3　このマニュアルで使用する英略語

このマニュアルで使用する主な英略語を次に示します。

英略語	正式名称
CPU	Central Processing Unit
CSV	Comma Separated Value
DLL	Dynamic Linking Library
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
ID	IDentifier
IP	Internet Protocol
IT	Information Technology

英略語	正式名称
JIS	Japanese Industrial Standards
LDAP	Lightweight Directory Access Protocol
OS	Operating System
PC	Personal Computer
RDB	Relational Database
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	Uniform Resource Locator
WS	Work Station
WWW	World Wide Web

付録 B.4　このマニュアルで使用している書式について

説明文で使用する書式

このマニュアルで画面の構成要素を説明するときに使用している記号を、次のように定義します。

記号	意味と例
[　]	ウィンドウ名、ダイアログ名、ボタン名およびキーボードのキーを示しています。 (例) ［コンソール］ウィンドウ ［登録］ダイアログ ［OK］ボタン ［Ctrl］
「　」	画面中に表示されている項目を示しています。 (例) 「資産番号」テキストボックス

コマンドまたはスクリプトの文法に使用する記号

このマニュアルで文法を説明するときに使用する文法記述記号を次に示します。

記号	意味と例
(　)	この記号で囲まれている項目は省略してもよいことを示します。複数の項目が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、どれか 1 つを選択します。 (例) (A) は「何も指定しない」か「A を指定する」ことを示します。
| (ストローク)	複数の項目がある場合に項目間の区切りを示し、「または」の意味を示します。 (例) A|B|C は、「A、B または C」を示します。

記号	意味と例
{ } （波括弧）	この記号で囲まれている複数の項目の中から、必ず 1 組の項目を選択します。項目の区切りは|で示します。 （例） {A|B|C}は「A、B または C のどれかを指定する」ことを示します。
… （点線）	この記号の直前に示す項目を、繰り返し複数個指定できることを示します。 （例） A…　は「A のあとに A を必要個数指定できる」ことを示します。
__ （下線）	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示します。標準値がない場合は、指定した項目だけが有効です。 （例） （A|B）はこの項目を指定しなかった場合に、A を選択したと見なすことを示します。
△ （白三角）	空白を示します。

このマニュアルで使用する構文要素

このマニュアルで使用している構文要素（ユーザの指定値の範囲）とその意味を次に示します。

構文要素	意味
英字	A～Z　a～z
英小文字	a～z
英大文字	A～Z
数字	0～9
英数字	英字または数字
記号	! " # $ % & ' ( ) + , - . / : ; < = > @ [ ] ^ _ { } ~ ? スペース

付録 B.5　オンラインヘルプについて

JP1/IT Desktop Management 2 では、次に示すオンラインヘルプを提供しています。

画面説明のヘルプ

表示中の操作画面について説明するヘルプです。操作画面に表示される［ヘルプ］ボタンから起動できます。

付録 B.6　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。

索引

記号

A

B

C

D

E

F

G

I

J

N

O

S

T

へ

ま

む

め

も

ゆ

れ

ろ

株式会社 日立製作所

〒 100-8280　東京都千代田区丸の内一丁目 6 番 6 号