

# BASh

## Developer Documentation v1.5.1

©2000 Deep Sky Technologies, Inc. All Rights Reserved.  
Published and Distributed Worldwide by Deep Sky Technologies, Inc.

Deep Sky Technologies, Inc.  
P.O. Box 6897  
Vero Beach, FL 32961-6897  
(561) 794-9494



<http://www.deepskytech.com/>

### Software Engineers

James T. Crate  
Robert T. McGoye  
Steven G. Willis

### Manual

Steven G. Willis



# Software License and Limited Warranty

Please read this license carefully before using the software. By using the software, you agree to become bound by the terms of this agreement, which includes the software license and warranty disclaimer (collectively referred to herein as the "agreement"). This agreement constitutes the complete agreement between you and Deep Sky Technologies, Inc. If you do not agree to the terms of this agreement, do not use the software and promptly destroy all copies in your possession, physical and electronically.

**1. Ownership of Software:** The enclosed manual and computer programs ("Software") were developed and are copyrighted by Deep Sky Technologies, Inc. ("DSTi") and are licensed, not sold, to you by DSTi for use under the following terms, and DSTi reserves any rights not expressly granted to you. DSTi retains ownership of all copies of the Software itself. Neither the manual nor the Software may be copied in whole or in part except as explicitly stated below.

**2. License:** DSTi, as Licensor, grants to you, the Licensee, a non-exclusive, non-transferable right to use this Software subject to the terms of the license as described below:

- a. You may make backup copies of the Software for your use provided they bear the DSTi copyright notice.
- b. You may use this Software in an unlimited number of custom or commercial databases or applications created by the original licensee. No additional product license or royalty is required.

**3. Restrictions:** You may not distribute copies of the Software to others (except as an integral part of a database or application within the terms of this License) or electronically transfer the Software from one computer to another over a network. You may distribute copies of the Software as an integral part of a development shell or non-compiled commercial database as long as the DSTi copyright notices and documentation remain intact with the distribution. The Software contains trade secrets and to protect them you may not decompile, reverse engineer, disassemble, or otherwise reduce the Software to a human perceivable form. You

may not modify, adapt, translate, rent, lease, loan or resell for profit the software or any part thereof.

**4. Termination:** This license is effective until terminated. This license will terminate immediately without notice from DSTi if you fail to comply with any of its provisions. Upon termination you must destroy the Software and all copies thereof, and you may terminate this license at any time by doing so.

**5. Update Policy:** DSTi may create, from time to time, updated versions of the Software. At its option, DSTi will make such updates available to the Licensee.

**6. Warranty Disclaimer:** The software is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. DSTi does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of the software or written materials in the terms of correctness, accuracy, reliability, currentness or otherwise. The entire risk as to the results and performance of the software is assumed by the Licensee. If the software or written materials are defective you, and not DSTi or its dealers, distributors, agents, or employees, assume the entire cost of all necessary servicing, repair or correction. No oral or written information or advice given by DSTi, its dealers, distributors, agents, or employees shall create a warranty or in any way increase the scope of this warranty, and you may not rely on such information or advice. This warranty gives you specific legal rights. You may have other rights, which vary from state to state.

**7. Governing Law:** This agreement shall be governed by the laws of the State of Florida.

# Copyrights and Trademarks

All trade names referenced in this document are the trademark or registered trademark of their respective holder.

BASh is copyright Deep Sky Technologies, Inc.

4th Dimension, ACI, ACI US, 4D Compiler, 4D, 4D Server, 4D Client, and 4D Insider are trademarks of 4D, Inc.

Windows is a trademark of Microsoft Corporation.

Macintosh and MacOS are trademarks of Apple Computer, Inc.

Marshmallow Peeps and Just Born are trademarks of Just Born, Inc.

# Table of Contents

Items in **Bold** were just added in the latest release of the BASH component.  
Items in *Italic* have not had their content filled in completely as of yet.

- Software License and Limited Warranty
- Copyrights and Trademarks
- Table of Contents
- Preface
- About this Manual*
- Acknowledgements**
- Features*
  - System Requirements
  - Support
- Components**
  - Installing and Updating BASH**
  - Managing Installation Conflicts**
  - Uninstalling BASH**
- Modules
  - ARR Module
    - ARR\_Add\_Elements\_to\_End
    - ARR\_Add\_Elements\_to\_Top
    - ARR\_Clear
    - ARR\_Convert\_Longint\_to\_Text
    - ARR\_Convert\_Text\_to\_Longint**
    - ARR\_Convert\_Type\_to\_Longint**
    - ARR\_Cycle\_Stack
    - ARR\_ERROR
    - ARR\_Order\_to\_Match
    - ARR\_Pack\_to\_Text**
    - ARR\_Populate\_Text\_Array
    - ARR\_Remove\_Elements\_from\_End
    - ARR\_Remove\_Elements\_from\_Top
    - ARR\_Set\_Size
  - CODEC Module
    - CLE Encoding Scheme
      - CODEC\_Decode\_Base64\_x
      - CODEC\_Decode\_Base64\_z
      - CODEC\_Decode\_CLE\_x
      - CODEC\_Decode\_URL\_x
      - CODEC\_Encode\_Base64\_x
      - CODEC\_Encode\_Base64\_z
      - CODEC\_Encode\_CLE\_x

- CODEC\_Encode\_URL\_x
- CONV Module
  - Hex2 and Hex8 Values
  - Type Values
  - Dotted IP Values
    - CONV\_ASCII\_to\_Hex2
    - CONV\_ASCII\_to\_PrintableText
    - CONV\_Coerce\_from\_Text
    - CONV\_Coerce\_to\_Text
    - CONV\_ERROR
    - CONV\_Hex8\_to\_Longint
    - CONV\_Hex8\_to\_Text
    - CONV\_IP\_to\_Longint
    - CONV\_Longint\_to\_Hex8
    - CONV\_Longint\_to\_IP
    - CONV\_Longint\_to\_Type
    - CONV\_Text\_to\_Hex8
    - CONV\_Text\_to\_Longint
    - CONV\_Text\_to\_Real
    - CONV\_Type\_to\_Longint
- DSS Module
  - Variable Classifications
    - DSS\_ERROR
    - DSS\_Get\_Variable\_by\_Type
    - DSS\_Get\_Array\_by\_Unary\_Type
    - DSS\_Get\_Unary\_by\_Array\_Type
    - DSS\_Return\_Variable
- DTS Module
  - DTS\_Add
  - DTS\_Get\_Current
  - DTS\_Get\_Date
  - DTS\_Get\_Date\_Time
  - DTS\_Get\_Day
  - DTS\_Get\_Maximum
  - DTS\_Get\_Month
  - DTS\_Get\_Range
  - DTS\_Get\_Time
  - DTS\_Get\_Year
  - DTS\_Make\_from\_DateTime
  - DTS\_Make\_from\_Values
- ENV Module
  - ENV\_Get\_4DApplication\_FullPath
  - ENV\_Get\_Application\_Name

- ENV\_Get\_Application\_Name\_Short
- ENV\_Get\_Application\_Type
- ENV\_Get\_DataFile\_FullPath
- ENV\_Get\_DirectorySymbol
- ENV\_Get\_Platform
- ENV\_Get\_Structure\_RF\_FileName
- ENV\_Get\_Structure\_RF\_FullPath
- ENV\_q\_Windows
- FILE Module
  - FILE\_Convert\_to\_ResFork\_Windows
  - FILE\_FullPath\_to\_FileName
- NULL Module
  - NULL\_Dereference\_by\_Type
  - NULL\_ERROR
  - NULL\_Get\_DTS
  - NULL\_Get\_Pointer
  - NULL\_Set\_Variables
- NVP Module
  - NVP\_ERROR
  - NVP\_Extract\_Values\_by\_Name\_s
  - NVP\_Extract\_Values\_by\_Name\_x
  - NVP\_Pack\_to\_Text
  - NVP\_Parse\_to\_Arrays
- RES Module
  - 'fMap' Resource*
  - 'LoCK' Resource*
  - 'MENV' Resource*
  - 'WAGr' Resource*
  - 'WPGr' Resource*
  - RES\_Close
  - RES\_Create\_File
  - RES\_Delete\_Resource
  - RES\_ERROR
  - RES\_Get\_Resource\_List
  - RES\_Get\_TEXT\_Resource
  - RES\_Load\_cicn
  - RES\_Load\_fMap*
  - RES\_Loadack\_LoCK*
  - RES\_Load\_MBAR
  - RES\_Load\_MENU
  - RES\_Load\_MENV*
  - RES\_Load\_PICT
  - RES\_Load\_TMPL



*RES\_Load\_WAGr*  
*RES\_Load\_WPGr*  
RES\_Make\_TMPL\_f\_Arrays  
RES\_Open  
RES\_Open\_4DApplication  
RES\_Open\_DateFile  
RES\_Open\_Structure  
RES\_Parse\_TMPL  
RES\_Set\_Resource\_Name  
RES\_Set\_Resource\_Properties  
RES\_Set\_TEXT\_Resource

SEM Module

SEM\_Clear\_One  
SEM\_Set\_One

STR Module

STR\_Clean\_EmailAddress  
STR\_Concatenate\_Text  
STR\_Count\_Occurrences\_of\_ASCII  
STR\_Count\_Occurrences\_of\_String  
STR\_Get\_CharPosition\_by\_ASCII  
STR\_Get\_Line\_First  
STR\_Pad\_String  
STR\_Parse\_to\_Array\_by\_ASCII  
STR\_Parse\_to\_Array\_by\_Str  
STR\_qi\_Match\_Filter\_NonCase  
STR\_Remove\_After\_Last\_by\_ASCII  
STR\_Remove\_Line\_First  
STR\_Remove\_NonAlphaNumeric  
STR\_Remove\_Spaces\_Post  
STR\_Remove\_Spaces\_Pre  
STR\_Remove\_Spaces\_PrePost  
STR\_Replace\_ASCII\_All  
STR\_Wrap\_in\_DoubleQuotes

TIME Module

TIME\_Add\_Normalize  
TIME\_Get\_Hours  
TIME\_Get\_Minutes  
TIME\_Get\_Seconds  
TIME\_Get\_Sum\_Offset

TYPE Module

**Unary and Array Classifications**

TYPE\_Compare\_Dereferenced\_Type  
TYPE\_Get\_Array\_by\_Unary

- TYPE\_Get\_Unary\_by\_Array
    - TYPE\_qi\_Array
    - TYPE\_qi\_BLOB
  - VAR Module
    - VAR\_Get\_Variable\_Name
    - VAR\_qi\_Null\_Pointer
- Version History
  - BASh v1.5.1
  - BASh v1.4.7
  - BASh v1.4.6
  - BASh v1.4.1
  - BASh v1.4.0
  - BASh pre-v1.4.0
- Using BASh*
- BASh Error Codes*
- Index*

# Preface

The BASH component is distributed solely as “Peepware”. Of course, this is as opposed to distribution as freeware or postcardware.

In case you are not familiar with peepware, it is considered very similar to postcardware. If you find the methods and functionality available within this software to be useful, we ask that you send along your favorite variety of marshmallow peeps to the programming staff at Deep Sky Technologies, Inc.

Just in case, if you have no idea what marshmallow peeps are, please visit <http://www.marshmallowpeeps.com/> and become a fan of the greatest programmer food to have ever existed.

## About this Manual

asd

# Acknowledgements

The creation of the BASH component is not directly attributable to any single person. Particular pieces of functionality within the BASH component may be from the direct knowledge and experience of certain developers, but the overall concept and construction of the BASH component has come from all of the developers at Deep Sky Technologies, Inc.

In particular the tireless efforts of Robert T. McGoye have contributed the most to the BASH component. His ability, and patience, to be able to tolerate the swings in the atmosphere at DSTi have proven to be invaluable in the development of the BASH component. Mr. McGoye fill a roll early in expansion of DSTi which proved invaluable to the eventual release of the BASH component.

Later additions and enhancements to the BASH component have resulted from the training of James T. Crate. Mr. Crate's experience in many different programming environments has provided refreshing insights into the overall structure and organization of the core routines at DSTi, the same core routines which are available in the BASH component.

For different particular pieces of functionality within the BASH component, many different individuals provided interesting feedback and techniques which have contributed to the code with the BASH component. These list of these people include, though is not limited to: David Adams, Charles Albrecht, Stewart Buskirk, Tim Crusher, Michael Erickson, Kieth Goebel, Bryan Green, Jerry Hale, Tim Klein, and Tom Lundeen.

Finally, I, Steven G. Willis, might have had something to do with the creation of the BASH component...



## Features

asd

# System Requirements

The BASH component is compatible with both Macintosh and Windows installations of 4th Dimension.

Since it is a component, it does require at least version 6.7 of 4th Dimension or above, including 4D Insider v6.7 or above for installation.

Other than the normal hardware and software requirements for your version of 4th Dimension, there are no other minimum requirements for proper use of this software.



# Support

As peepware, there obviously is no guaranteed support which is available for the BASH component.

But, on an “as available” basis, with no implied or expressed warranty for support whatsoever, Deep Sky Technologies, Inc., can be contacted for support for the software in the BASH component.

Contact information, including email address(es), phone number(s), and a Contact Us request form, for Deep Sky Technologies, Inc., can be found on the DSTi web site located at:

<http://www.deepskytech.com/>



# Components

A component groups various 4D objects (tables, project methods, forms, menu bars, variables, etc.) representing one or more additional functions. Developing a 4D component providing electronic mail functionality is one such example. A component is autonomous and must be able to be installed in any 4D structure.

Components are defined, generated, and installed with the help of 4D Insider. The component definition is based on the cross referencing analysis performed by 4D Insider (target objects and source objects).

Unlike libraries and groups, components embed the idea of security of objects that they compose. During the development phase of the component, each object is attributed an access type, "Public", "Protected" or "Private". This attribute determines whether each object will be visible or modifiable in 4th Dimension and in 4D Insider once the component is installed within a 4D database.

# Installing and Updating BASH

Installing BASH or updating an existing version of BASH within a 4D database is performed using 4D Insider. The activity primarily consists of installing the BASH component in a database structure opened with 4D Insider (installing the BASH component in a library is not supported at this time).

4D Insider will manage possible conflict issues within the installation and will inform as they are detected. Though, with the naming conventions used within the BASH component and the limited number of object names, conflicts should be very rare in the worst of cases.

To install or update the BASH component, follow these very simple steps:

**Open the database that you wish to install BASH using 4D Insider**

**Choose the "Install/Update..." command in the "Components" menu**

A standard open file dialog box will appear.

**Select the BASH component file and click on the "Open" button.**

4D Insider parses the BASH component and prepares to integrate it with your open database. 4D Insider will detect if the operation is an installation or an update of the BASH component.

In the event of a new installation, all BASH objects are installed.

In the event of an update, 4D Insider compares the version numbers of both the currently installing BASH component and the already installed BASH component. If the date of the "new" component is older than the already installed component, a dialog box will alert you, allowing you to then "Continue" or "Cancel" the update.

4D Insider replaces old objects with newer objects within the BASH component and adds new objects from the new BASH component. 4D Insider takes into account "public" objects having been modified by you (e.g. "\_ERROR" methods) and will prompt you to either save or replace them. If any other conflicts arise from the installation or update of the BASH component, 4D Insider will prompt you with an appropriate dialog box.

### **Save the database in 4D Insider.**

The BASH component is now installed/updated in your database and is listed on the "Components" page of the 4D Explorer.

## **Managing Installation Conflicts**

On very rare occasions, when the BASH component is installed or updated in your 4D database, several questions and conflicts may arise. In the event of an update, 4D Insider will detect that you have modified one of more "Public" objects in BASH after the initial installation. Or, one or more objects of the same type and of the same name may already exist in your database and in the BASH component.

4D Insider detects and solves these conflicts during installation:

### **Modified public objects (updates only)**

In this case, 4D Insider alerts you by a dialog box, allowing you to choose an update mode:

Replace the object

Replace all objects

Do not replace the object

Stop installation

## Name conflicts

In this case, 4D Insider stops the BASH's installation process, alerts you through a dialog box and saves the list of objects in conflict. This list is stored as a text file in the 4D database folder.

Naming conflicts between logical objects, such as variables, are managed by 4D Insider, in a manner that allows database compilation and avoids conflicts between BASH and other 4D components.

It may be necessary to rename certain objects in your database or in other components in order to be able to install the BASH.

If any naming conflicts do occur between BASH and other 4D components, please notify Deep Sky Technologies, Inc., immediately.

# Uninstalling BASH

4D Insider allows you to uninstall the BASH component from your 4D database.

To uninstall BASH from your 4D database:

Using 4D Insider, open your database containing the copy of BASH to be uninstalled.

In the "Main" listing window, select the BASH component.

Consider again how great the BASH component is and make certain that you will *really* no longer need it in your 4D database.

Select the "Uninstall..." command in the "Components" menu.

This command is only active when a component is installed in the database. A dialog box appears allowing you to confirm or cancel the operation. If you uncertain about the previous step then the cancel option is probably your best choice at this time.

Click "OK" to validate the operation.

All objects from the BASH component are deleted from your 4D database. Obviously, you are now very sad to no longer have the BASH component in your 4D database. Crying is allowed...





# Modules

All of the code within the BASH component is organized into modules. Each module is designated by a three (3) to five (5) character module prefix. All of the module prefixes are used within the name of every object within the module (methods names, variable names, semaphore names, etc.). This allows for the easy identification of any object within the BASH component.

Each module contains a set of methods which can be used throughout your database once the BASH component is installed. Method names all begin with the module prefix followed by an underscore ("\_") characters. The remainder of the method name then describes the function of the method.

# ARR Module

The ARR Module is for managing and manipulating arrays within 4th Dimension. The ARR module includes routines for sizing arrays, inserting elements, converting between types of arrays, parsing of values into arrays, element cycling, and sort matching, among many others to come.



## ARR\_Add\_Elements\_to\_End

**ARR\_Add\_Elements\_to\_End** ( *Referenced Array; Elements to Add* )

**ARR\_Add\_Elements\_to\_End**  
(  
    -> *Referenced Array* : **Pointer**  
    -> *Elements to Add* : **Longint**  
)

Parameter	Type	Description
<i>Referenced Array</i>	<b>Pointer</b>	Reference to a single dimensional array of any type which will have a specified number of elements added to the end of it
<i>Elements to Add</i>	<b>Longint</b>	Positive number of elements to add to the end of <i>Referenced Array</i>

The method **ARR\_Add\_Elements\_to\_End** will insert *Elements to Add* elements to the end of the array *Referenced Array*. Each element which is added to *Referenced Array* will be initialised to a **NULL** value.



## ARR\_Add\_Elements\_to\_Top

**ARR\_Add\_Elements\_to\_Top** ( *Referenced Array; Elements to Add* )

**ARR\_Add\_Elements\_to\_Top**  
(  
    -> *Referenced Array* : **Pointer**  
    -> *Elements to Add* : **Longint**  
)

)

Parameter	Type	Description
<i>Referenced Array</i>	Pointer	Reference to a single dimensional array of any type which will have a specified number of elements added to the beginning of it
<i>Elements to Add</i>	Longint	Positive number of elements to add to the beginning of <i>Referenced Array</i>

The method ***ARR\_Add\_Elements\_to\_Top*** will insert *Elements to Add* elements to the beginning of the array *Referenced Array*. Each element which is added to *Referenced Array* will be initialised to a **NULL** value.



## **ARR\_Clear**

***ARR\_Clear***( *Referenced Array*)

***ARR\_Clear***

(  
    -> *Referenced Array*: Pointer  
)

Parameter	Type	Description
<i>Referenced Array</i>	Pointer	Reference to a single dimensional array of any type to be cleared

The method ***ARR\_Clear*** will set the size of the array *Referenced Array* to zero (0). This will be done regardless of the type of array.

This command is equivalent to using native command **DELETE ELEMENT** to remove all of the elements of an array. ***ARR\_Clear*** will make certain though that *Referenced Array* is a valid array before clearing the array of all elements.



## **ARR\_Convert\_Longint\_to\_Text**

**ARR\_Convert\_Longint\_to\_Text**( *Referenced Longint Array*; *Referenced Text Array*)

**ARR\_Convert\_Longint\_to\_Text**

(  
    -> *Referenced Longint Array*: **Pointer**  
    -> *Referenced Text Array*: **Pointer**  
)

Parameter	Type	Description
<i>Referenced Longint Array</i>	<b>Pointer</b>	Reference to a longint array holding values to convert
<i>Referenced Text Array</i>	<b>Pointer</b>	Reference to text array which will hold textual equivalent of longint values from <i>Referenced Longint Array</i> .

The method **ARR\_Convert\_Longint\_to\_Text** will convert all of the values in the array *Referenced Longint Array* to their textual equivalents and store them in *Referenced Text Array*. The size of *Referenced Text Array* will be set to be the same as *Referenced Longint Array*. Values will be converted element by element. The native **String** command is used to make the conversion of each element.

Usage:

This method is ideal for usage within interfaces in which numeric values must be selected from a list but the values must be text. This is often true when build a drop down interface item within HTML



## **ARR\_Convert\_Text\_to\_Longint**

**ARR\_Convert\_Text\_to\_Longint** ( *Referenced Text Array*; *Referenced Longint Array*)

**ARR\_Convert\_Text\_to\_Longint**

(  
    -> *Referenced Text Array*: **Pointer**  
    -> *Referenced Longint Array*: **Pointer**  
)

Parameter	Type	Description
<i>Referenced Text Array</i>	Pointer	Reference to a text or string array containing values to convert
<i>Referenced Longint Array</i>	Pointer	Referenced to a longint array to contain converted values

The method ***ARR\_Convert\_Text\_to\_Longint*** will convert all of the elements of a referenced text or string array into 32 bit integers and place them into a referenced longint array. Elements are maintained in a well formed stack and the native **Int** and **Num 4D** commands are used to make the conversion of each element.

*Referenced Text Array* is a pointer to the text array containing the elements to convert.

*Referenced Longint Array* is the destination array which is to contain the converted elements from *Referenced Text Array*. *Referenced Longint Array* will be resized to match the size of *Referenced Text Array*.

**Note:** the method ***ARR\_Convert\_Text\_to\_Longint*** was added in BASH v1.5.1.



## **ARR\_Convert\_Type\_to\_Longint**

***ARR\_Convert\_Type\_to\_Longint*** (*Referenced Types Array*; *Referenced Longint Array*)

***ARR\_Convert\_Type\_to\_Longint***

```
(
    -> Referenced Types Array: Pointer
    -> Referenced Longint Array: Pointer
)
```

Parameter	Type	Description
<i>Referenced Types Array</i>	Pointer	Reference to a text or string array containing type values to convert
<i>Referenced Longint Array</i>	Pointer	Referenced to a longint array to contain converted type values

The method ***ARR\_Convert\_Type\_to\_Longint*** will convert all of the elements of a referenced types array into 32 bit integers and place them into a referenced longint array. Elements are maintained in a well formed stack.

*Referenced Types Array* is a pointer to the text or string array containing the elements to convert.

*Referenced Longint Array* is the destination array which is to contain the converted elements from *Referenced Types Array*. *Referenced Longint Array* will be resized to match the size of *Referenced Types Array*.

**Note:** a type array is any string or text array which is used to store standard Macintosh type codes. A type code can be any four (4) byte value used as a key of some sort on the Macintosh; this can include Macintosh creator codes, Macintosh file type codes, and resource type codes. See the section **Type Values**, below, for more information on Type values

**Note:** the method ***ARR\_Convert\_Type\_to\_Longint*** was added in BASH v1.5.1.



## **ARR\_Cycle\_Stack**

**ARR\_Cycle\_Stack** ( *Referenced Array; Beginning Element; Ending Element; Cycle Count* )

**ARR\_Cycle\_Stack**

(  
-> *Referenced Array: Pointer*  
-> *Beginning Element: Longint*  
-> *Ending Element: Longint*  
-> *Cycle Count: Longint*  
)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Array</i>	<b>Pointer</b>	asd
<i>Beginning Element</i>	<b>Longint</b>	asd

<i>Ending Element</i>	<b>Longint</b>	<b>asd</b>
<i>Cycle Count</i>	<b>Longint</b>	<b>asd</b>

The method **ARR\_Cycle\_Stack** will cycle a specified range of elements, as specified by *Beginning Element* and *Ending Element*, in the array *Referenced Array* a number of elements. The direction of the cycle and the number of elements cycled is determined by *Cycle Count*.

If *Cycle Count* is less than zero (0), elements in the array range are cycled towards the beginning of *Referenced Array*. If *Cycle Count* is greater than zero (0), elements in the array range are cycled towards the ending of *Referenced Array*.

#### Example:

Assume the array **axMyArray** has the following elements:

axMyArray{1}	Andrew
axMyArray{2}	Bobby
axMyArray{3}	Carolyn
axMyArray{4}	Debby
axMyArray{5}	Evelyn
axMyArray{6}	Frederick
axMyArray{7}	Guido

After making the call:

**ARR\_Cycle\_Stack** (->axMyArray; 2; 5 1)

the array **axMyArray** would now have the following elements:

axMyArray{1}	Andrew
axMyArray{2}	Carolyn
axMyArray{3}	Debby
axMyArray{4}	Evelyn
axMyArray{5}	Bobby
axMyArray{6}	Frederick
axMyArray{7}	Guido

Subsequently, if the call:

**ARR\_Cycle\_Stack** (->axMyArray; 4; 7 -2)

were to be made, the array **axMyArray** would then have the following elements:

axMyArray{1}	Andrew
axMyArray{2}	Carolyn
axMyArray{3}	Debby
axMyArray{4}	Frederick
axMyArray{5}	Guido
axMyArray{6}	Evelyn
axMyArray{7}	Bobby

**Usage:**

This method is exceedingly useful in interfaces in which elements in a scrollable array can be subject to drag and drop reordering of elements. Elements can be cycled within the array with a single method call.



## **ARR\_ERROR**

**ARR\_ERROR** (*BASH Error Number; Special Error Text; Calling Method Name*)

### **ARR\_ERROR**

(  
-> *BASH Error Number: Longint*  
-> *Special Error Text: Text*  
-> *Calling Method Name: Text*  
)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>BASH Error Number</i>	<b>Longint</b>	Internal BASH error number
<i>Special Error Text</i>	<b>Text</b>	Special text to describe the exact error instance
<i>Calling Method Name</i>	<b>Text</b>	Name of the method that the error condition occurred in



The method **ARR\_ERROR** acts as a callback method from within the ARR module for errors that may occur. Any time an error condition is detected within the ARR module, a call to the method **ARR\_ERROR** is made.

The internal *BASH Error Number* is passed to this method as the first parameter. The *Special Error Text* parameter will contain any relevant error text which is specific to the error which occurred. It is not uncommon for the *Special Error Text* value to be empty. The *Calling Method Name* will always contain the name of the ARR method which call the **ARR\_ERROR** method.

The **ARR\_ERROR** method has been implemented as a source for a consistent interface and/or error tracking mechanism to be available while using the BASH component. This method can be modified to suit the needs of the database in which the BASH module has been installed.



## **ARR\_Order\_to\_Match**

**ARR\_Order\_to\_Match** ( *Referenced Ordered Array; Referenced Match Array  
{; Parallel Array {; ... } }* )

### **ARR\_Order\_to\_Match**

```
(
    -> Referenced Ordered Array: Pointer
    -> Referenced Match Array: Pointer
    { -> Parallel Array: Pointer }
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Ordered Array</i>	<b>Pointer</b>	Referenced array holding longint values sorted as desired
<i>Referenced Match Array</i>	<b>Pointer</b>	Referenced array holding longint values which should be ordered to match the sorting within the <i>Referenced Order Array</i>
<i>Parallel Array</i>	<b>Pointer</b>	Up to five (5) more optional array references which will be kept in parallel with the reordering of the <i>Referenced Match Array</i>

The method ***ARR\_Order\_to\_Match*** will reorder a series of arrays to match the ordering within *Referenced Ordered Array*. The reordering will be done by matching the values in *Referenced Match Array* with values stored in *Referenced Ordered Array*, keeping all of the optional *Parallel Array* elements row-wise in line with the *Referenced Match Array*.

### Example:

Assume the following arrays:

aiOrderedLong {1}	2
aiOrderedLong {2}	5
aiOrderedLong {3}	7
aiOrderedLong {4}	9
aiMatchLong{1}	7
aiMatchLong{2}	9
aiMatchLong{3}	2
aiMatchLong{4}	5
axWhatever{1}	seven
axWhatever{2}	nine
axWhatever{3}	two
axWhatever{4}	five

After making the following call:

```
ARR_Order_to_Match (->aiOrderedLong;
                    ->aiMatchLong; ->axWhatever)
```

the same three arrays would now be:

aiOrderedLong {1}	2
aiOrderedLong {2}	5
aiOrderedLong {3}	7
aiOrderedLong {4}	9
aiMatchLong{1}	2
aiMatchLong{2}	5
aiMatchLong{3}	7

aiMatchLong{4}	9
axWhatever{1}	two
axWhatever{2}	five
axWhatever{3}	seven
axWhatever{4}	nine

#### Usage:

This method is very useful for keeping array synchronised when the elements may be subject to reordering. By keeping a single key array into the stack, all of the columns within the stack can be kept together row-wise with a single method call. This method is ideal for use with interfaces that involve scrollable arrays which can be reordered through drag and drop.



### **ARR\_Pack\_to\_Text**

**ARR\_Pack\_to\_Text**( *Referenced Values Array*; *Delimiter Value*)

**ARR\_Pack\_to\_Text**

```
(
    -> Referenced Values Array: Pointer
    -> Delimiter Value: Text
)
=> Packed Text: Text
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Values Array</i>	<b>Pointer</b>	Reference to a text, string, date, longint, integer, or real array containing values to pack
<i>Delimiter Value</i>	<b>Text</b>	Delimiter value to place between elements being packed
<i>Packed Text</i>	<b>Text</b>	Packed elements from <i>Referenced Values Array</i> which are separate by <i>Delimiter Value</i>

The method **ARR\_Pack\_to\_Text** will pack the elements from a referenced array into a text value separated by a specified delimiter value.

*Referenced Values Array* is a text, string, date, longint, integer, or real array containing values which are going to be packed. Before packing, each element is coerced to text.

*Delimiter Value* is a character or string which is placed between each value from *Referenced Values Array* when packed.

*Packed Text* is the resulting values all packed into a single text variable.

**Note:** the method *ARR\_Pack\_to\_Text* was added in BASH v1.5.1.



## **ARR\_Populate\_Text\_Array**

**ARR\_Populate\_Text\_Array** ( *Referenced Text Array* {; *Fill Text* {; ... } } )

**ARR\_Populate\_Text\_Array**

(  
    -> *Referenced Text Array*: **Pointer**  
    { -> *Fill Text*: **Text** }  
)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Text Array</i>	<b>Pointer</b>	Text array to be filled by specified text values
<i>Fill Text</i>	<b>Text</b>	Optional number of text values to fill into <i>Referenced Text Array</i>

The method *ARR\_Populate\_Text\_Array* provides a simple mechanism to popular many text values into a text array. *Referenced Text Array* is resized so the number of elements match the number of *Fill Text* values which are passed to the method. The only limitation on the number of *Fill Text* values which can be passed to *ARR\_Populat\_Text\_Array* is that which is imposed by the development environment.



## **ARR\_Remove\_Elements\_from\_End**

**ARR\_Remove\_Elements\_from\_End** ( *Referenced Array*; *Elements to Remove* )

**ARR\_Remove\_Elements\_from\_End**  
(  
    -> *Referenced Array*: **Pointer**  
    -> *Elements to Remove*: **Longint**  
)

Parameter	Type	Description
<i>Referenced Array</i>	<b>Pointer</b>	Reference to a single dimensional array of any type which will have a specified number of elements removed from the end of it
<i>Elements to Remove</i>	<b>Longint</b>	Positive number of elements to remove from the end of <i>Referenced Array</i>

The method **ARR\_Remove\_Elements\_from\_End** will remove *Elements to Remove* elements from the end of the array *Referenced Array*.



## **ARR\_Remove\_Elements\_from\_Top**

**ARR\_Remove\_Elements\_from\_Top** ( *Referenced Array*; *Elements to Remove* )

**ARR\_Remove\_Elements\_from\_Top**  
(  
    -> *Referenced Array*: **Pointer**  
    -> *Elements to Remove*: **Longint**  
)

Parameter	Type	Description
<i>Referenced Array</i>	<b>Pointer</b>	Reference to a single dimensional array of any type which will have a specified number of elements removed from the beginning of it
<i>Elements to Remove</i>	<b>Longint</b>	Positive number of elements to remove from the beginning of <i>Referenced Array</i>

The method ***ARR\_Remove\_Elements\_from\_Top*** will remove *Elements to Remove* elements from the beginning of the array *Referenced Array*.



## **ARR\_Set\_Size**

***ARR\_Set\_Size*** ( *Referenced Array*; *New Maximum Element Number* )

***ARR\_Set\_Size***

(  
    -> *Referenced Array*: **Pointer**  
    -> *New Maximum Element Number*: **Longint**  
)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Array</i>	<b>Pointer</b>	Reference to a single dimensional array of any type which will be resized
<i>New Maximum Element Number</i>	<b>Longint</b>	New maximum element number which <i>Referenced Array</i> should be resized to

The method ***ARR\_Set\_Size*** will resized the array *Referenced Array* to have a maximum element number of *New Maximum Element Number*. If new elements are added to *Referenced Array*, they will be added to the end of the array and will have all elements initialized to NULL.

# CODEC Module

The CODEC module provides basic encoding and decoding routines for many popular coding formats. Where applicable, encoding and decoding routines for both text and BLOB variable types have been included to handle all needs within 4th Dimension. With the exception of the variable types employed, encoding and decoding of text and BLOBs work exactly the same.

## CLE Encoding Scheme

The CLE (Carriage return, Linefeed, Equal) encoding scheme is merely a shortened version of the URL encoding scheme. It provides for for basic named value pair storage within a single text value. It is ideal for the storage of NVP values within a single text value in 4th Dimension. This makes it very easy to make human readable preferences files which are stored in the file system of the OS.

In essence, the CLE encoding scheme provides for the following translation of bytes:

ASCII Value	ASCII Title	Encoded
13	Carriage Return	&D
10	Linefeed	&A
61	Equal Sign	&3D

**Note:** the CODEC module was initially added in BASH v1.5.1.



## CODEC\_Decompile\_Base64\_x

**CODEC\_Decompile\_Base64\_x**( Base64 Encoded Text) => Decoded Text

**CODEC\_Decompile\_Base64\_x**

(  
    -> Base64 Encoded Text: Text  
)

=> *Decoded Text*: **Text**

Parameter	Type	Description
<i>Base64 Encoded Text</i>	<b>Text</b>	Base64 encoded text which is to be decoded
<i>Decoded Text</i>	<b>Text</b>	Decoding of <i>Base64 Encoded Text</i>

The method ***CODEC\_Decode\_Base64\_x*** decodes a base64 encoded text value. Specifications for the base64 encoding scheme are available in RFC 2045, available at:

[http://www.deepskytech.com/rfcs/rfc\\_2045.txt](http://www.deepskytech.com/rfcs/rfc_2045.txt)

*Base64 Encoded Text* is the base64 encoded text which is to be decoded.

*Decoded Text* is *Base64 Encoded Text* decoded.

**Note:** the method ***CODEC\_Decode\_Base64\_x*** was added in BASH v1.5.1.



## **CODEC\_Decode\_Base64\_z**

***CODEC\_Decode\_Base64\_z***( *Referenced BLOB*)

***CODEC\_Decode\_Base64\_z***

(  
    -> *Referenced BLOB: Pointer*  
)

Parameter	Type	Description
<i>Referenced BLOB</i>	<b>Pointer</b>	Reference to base64 encoded BLOB

The method ***CODEC\_Decode\_Base64\_z*** decodes a base64 encoded BLOB value. Specifications for the base64 encoding scheme are available in RFC 2045, available at:

[http://www.deepskytech.com/rfcs/rfc\\_2045.txt](http://www.deepskytech.com/rfcs/rfc_2045.txt)



*Referenced BLOB* is the referenced base64 encoded BLOB which is to be decoded. The decoded BLOB is returned directly in this referenced value.

**Note:** the method *CODEC\_Decode\_Base64\_z* was added in BASH v1.5.1.



## **CODEC\_Decode\_CLE\_x**

**CODEC\_Decode\_CLE\_x**( *CLE Encoded Text*) => *Decoded Text*

**CODEC\_Decode\_CLE\_x**  
(  
    -> *CLE Encoded Text*: **Text**  
)  
=> *Decoded Text*: **Text**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>CLE Encoded Text</i>	<b>Text</b>	CLE encoded text value to be decoded
<i>Decoded Text</i>	<b>Text</b>	Decoding of <i>CLE Encoded Text</i>

The method *CODEC\_Decode\_CLE\_x* returns the decoding of a single CLE encoded value. Details on CLE encoding are available in the section entitled **CLE Encoding Scheme**, above.

*CLE Encoded Text* is a CLE encoded text value which is to be decoded.

*Decoded Text* is *CLE Encoded Text* decoded.

**Note:** the method *CODEC\_Decode\_CLE\_x* was added in BASH v1.5.1.



## **CODEC\_Decode\_URL\_x**

**CODEC\_Decode\_URL\_x**( *URL Encoded Text*; *Translation Option*) =>  
*Decoded Text*

**CODEC\_Decode\_URL\_x**  
(

```

-> URL Encoded Text: Text
-> Translation Option: Longint
)
=> Decoded Text: Text

```

Parameter	Type	Description
URL Encoded Text	Text	URL encoded text value to be decoded
Translation Option	Longint	Translation option is qi for decoding "+" values into spaces
Decoded Text	Text	Decoding of URL Encoded Text

The method *CODEC\_Decode\_URL\_x* returns the decoding of a single URL encoded value. Details on URL encoding are available in RFC 1738, available at:

[http://www.deepskytech.com/rfcs/rfc\\_1738.txt](http://www.deepskytech.com/rfcs/rfc_1738.txt)

*URL Encoded Text* is an URL encoded text value which is to be decoded.

*Translation Option* is option flag for how to decode plus ("+", ASCII value 43) values within *URL Encoded Text*. If *Translation Option* equals zero (0) then pluses are translated into spaces (ASCII value 32). Otherwise, plusses will not be translated into spaces.

*Decoded Text* is *URL Encoded Text* decoded.

**Note:** the method *CODEC\_Decode\_URL\_x* was added in BASH v1.5.1.



## **CODEC\_Encode\_Base64\_x**

*CODEC\_Encode\_Base64\_x*(Text Value) => Base64 Encoded Text

```

CODEC_Encode_Base64_x
(
    -> Text Value: Text
)
=> Base64 Encoded Text: Text

```

Parameter	Type	Description
-----------	------	-------------

<i>Text Value</i>	<b>Text</b>	<b>Text value to be base64 encoded</b>
<i>Base64 Encoded Text</i>	<b>Text</b>	<b>Base64 encoding of <i>Text Value</i></b>

The method ***CODEC\_Encode\_Base64\_x*** base64 encodes a supplied text value. Specifications for the base64 encoding scheme are available in RFC 2045, available at:

[http://www.deepskytech.com/rfcs/rfc\\_2045.txt](http://www.deepskytech.com/rfcs/rfc_2045.txt)

*Text Value* is the text which is to be base64 encoded.

*Base64 Encoded Text* is *Text Value* base64 encoded.

**Note:** the method ***CODEC\_Encode\_Base64\_x*** was added in BASH v1.5.1.



## **CODEC\_Encode\_Base64\_z**

***CODEC\_Encode\_Base64\_z***( *Referenced BLOB*)

***CODEC\_Encode\_Base64\_z***

(  
     -> *Referenced BLOB: Pointer*  
 )

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced BLOB</i>	<b>Pointer</b>	<b>Reference to BLOB to be base64 encoded</b>

The method ***CODEC\_Encode\_Base64\_z*** base64 encodes a referenced BLOB value. Specifications for the base64 encoding scheme are available in RFC 2045, available at:

[http://www.deepskytech.com/rfcs/rfc\\_2045.txt](http://www.deepskytech.com/rfcs/rfc_2045.txt)

*Referenced BLOB* is the referenced BLOB which is to be base64 encoded. The encoded BLOB is returned directly in this referenced value.

**Note:** the method ***CODEC\_Encode\_Base64\_z*** was added in BASH v1.5.1.



## CODEC\_Encode\_CLE\_x

**CODEC\_Encode\_CLE\_x**( *Text Value*) => *CLE Encoded Text*

**CODEC\_Encode\_CLE\_x**

```
(  
    -> Text Value: Text  
)  
=> CLE Encoded Text: Text
```

Parameter	Type	Description
<i>Text Value</i>	<b>Text</b>	Text value to be CLE encoded
<i>CLE Encoded Text</i>	<b>Text</b>	CLE encoding of <i>Text Value</i>

The method **CODEC\_Encode\_CLE\_x** returns a supplied text value CLE encoded. Details on CLE encoding are available in the section entitled **CLE Encoding Scheme**, above.

*Text Value* is the text value to be

*CLE Encoded Text* is *Text Value* CLE encoded.

**Note:** the method **CODEC\_Encode\_CLE\_x** was added in BASH v1.5.1.



## CODEC\_Encode\_URL\_x

**CODEC\_Encode\_URL\_x**( *Text Value*) => *URL Encoded Text*

**CODEC\_Encode\_URL\_x**

```
(  
    -> Text Value: Text  
)  
=> URL Encoded Text: Text
```

Parameter	Type	Description
<i>Text Value</i>	<b>Text</b>	Text value to be URL encoded
<i>URL Encoded Text</i>	<b>Text</b>	URL encoding of <i>Text Value</i>

The method *CODEC\_Encode\_URL\_x* returns a single text value URL encoded. Details on URL encoding are available in RFC 1738, available at:

[http://www.deepskytech.com/rfcs/rfc\\_1738.txt](http://www.deepskytech.com/rfcs/rfc_1738.txt)

The exact listing of ASCII byte values which are allowable can be modified. This listing is contained within the 'STR#' resource, ID 29821, entitled "Allowable ASCII Values for URLs"; note that this resource ID might change with the installation of the BASH component in the event that another 'STR#' resource with the same ID already exists within the current 4D structure (the BASH component handles this remapping internally and automatically). The values in this resource can be modified, as desired. Any ASCII values which are not contained in this resource will be URL encoded by this method.

*Text Value* is the text to be URL encoded.

*URL Encoded Text* is *Text Value* URL encoded.

**Note:** the method *CODEC\_Encode\_URL\_x* was added in BASH v1.5.1.

## CONV Module

The CONV module handles conversions between different data types available within 4th Dimension. The CONV module also handle conversions between particularly common auxiliary formats which are commonly employed in 4th Dimension, including dotted IP addresses, single byte hexadecimal values (Hex2), four byte hexadecimal values (Hex8), ASCII values, etc.

### Hex2 and Hex8 Values

Hex2 and Hex8 values are representations of ASCII values, integers, longints, and text within hexadecimal notation. The Hex2 and Hex8 values are text values which contain the hexadecimal equivalent of converted bytes.

Hex2 values are text of length two bytes which represent a single byte value in hexadecimal. For instance, the following conversion table lists example ASCII value and their equivalent Hex2 values:

<u>ASCII Value</u>	<u>Hex2 Value</u>
10	0A
13	0D
27	1B
32	20
61	3D
63	3F
64	40

**Note:** Hex2 values are *always* two bytes in length. This is true even for Hex2 values which when translated to decimal would be less than 16. Also, Hex2 values never have the commonly used ampersand ("&") at the beginning to indicate that the value is representing hexadecimal digits.

Hex8 values are text of length eight bytes which represent four byte values in hexadecimal. For instance,

the following conversion table lists example ASCII values and their equivalent Hex8 values:

<u>ASCII Value</u>	<u>Hex8 Value</u>
65, 115, 115, 13	4073730D
13, 27, 13, 27	0D1B0D1B
13, 10, 13, 10	0D0A0D0A

**Note:** Hex8 values are *always* eight bytes in length. Also, Hex8 values never have the commonly used ampersand ("&") at the beginning to indicate that the value is representing hexadecimal digits.

## Type Values

A Type value is text of length four (4) which represents a four (4) byte coercion of a Macintosh Type code. A Macintosh Type code is any four (4) byte data structure used as a key or attribute indicator. Commonly, Type values are known, depending on their usage, as Macintosh creator codes, Macintosh file type codes, resource types, and much more.

Technically, these values as they exist in 4th Dimension should be handled as longints. But, in many instances, 4th Dimension provides access only to the Type values as coerced textual equivalents. The CONV methods which deal with Type values allow for the easy conversion between standard 4D longint values and their equivalent values as coerced to and from text for use in 4th Dimension.

It is highly recommended that such type values, when stored within 4th Dimension, be handled exclusively as longints. Conversion to and from text values should be done immediately surrounding native calls to 4th Dimension commands. The importance of this matter is exemplified by the fact that resources of type 'TEXT' and 'text' are very different. But, when comparing their textual type values in 4th Dimension, they will be

considered equivalent. This can lead to no end of hassles and confusion when working in 4th Dimension.

**Note:** all BASH methods, in all modules of contained within the BASH component, deal with type values as longints, not coerced text. This includes all resource handling, file typing, etc., throughout the component. The advantages of using this format are obvious.

### Dotted IP Values

Dotted IP values are an alternate representation of a 32 bit (4 byte) values. A Dotted IP address will always have the format:

a.b.c.d

where a, b, c, and d are all values between 0 and 255, inclusive.

A common four (4) byte value in 4th Dimension is the longint variable type. So, it is a simple matter to store Dotted IP addresses within longint variables in 4th Dimension. By doing this, the storage is much more compact, manipulation is much easier, and operations perform much more quickly.

**Note:** the storage of dotted IP addresses within 4th Dimension as longints is not as straightforward as it may initially seem. For instance, a common operation to calculate for dotted IP addresses is to determine if a particular dotted IP address falls within a specified range of IP addresses (or matches a particular mask). Using integer comparison on the Dotted IP values as stored in longints would not yield consistent results for such an operation. Instead, bit arithmetic must be used on the Dotted IP values stored in longints. This is not a complexity or limitation of the storing Dotted IP values in longints. Rather, the code for such comparisons is actually much more efficient and not as prone to silly textual and typing errors.



**Note:** the CONV module was initially added in BASH v1.5.1.



## **CONV\_ASCII\_to\_Hex2**

**CONV\_ASCII\_to\_Hex2** ( *ASCII Value* ) => *Hex2 Value*

**CONV\_ASCII\_to\_Hex2**

(  
    -> *ASCII Value*: Longint  
)  
    => *Hex2 Value*: String2

Parameter	Type	Description
<i>ASCII Value</i>	Longint	ASCII value of a single byte
<i>Hex2 Value</i>	String2	Hex2 equivalent of <i>ASCII Value</i>

The method **CONV\_ASCII\_to\_Hex2** will convert a single, integral byte value into its Hex2 equivalent.

*ASCII Value* is a single ASCII byte value, a number between 0 and 255 (inclusive). If *ASCII Value* is outside the accepted range of valid values, only the lower eight (8) bits) will be used to calculate the converted return value.

*Hex2 Value* is *ASCII Value* represented in hexadecimal. *Hex2 Value* will always be two bytes in length.

**Note:** the method **CONV\_ASCII\_to\_Hex2** was added in BASH v1.5.1.



## **CONV\_ASCII\_to\_PrintableText**

**CONV\_ASCII\_to\_PrintableText**( *ASCII Value* ) => *Printable Character*

**CONV\_ASCII\_to\_PrintableText**

(

-> *ASCII Value*: **Longint**  
 )  
 => *Printable Character*: **Text**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>ASCII Value</i>	<b>Longint</b>	ASCII value of a single byte
<i>Printable Character</i>	<b>Text</b>	Printable version of <i>ASCII Value</i>

The method *CONV\_ASCII\_to\_PrintableText* will convert a single ASCII value into the textual equivalent which is printable. ASCII values which are non-printable are converted to a period (".") for display and printing.

The range of directly printable ASCII values varies depending on the platform. On Macintosh, the range of printable ASCII values is from 32 to 255, inclusive. On Windows, the range is reduced to 32 to 126, inclusive. All ASCII values which are considered non-printable on the current platform will be converted to periods (".") for display and printing.

*ASCII Value* is the integral ASCII value to be converting to text for display and/or printing. All values of *ASCII Value* which are outside the range of acceptable ASCII values (0 to 255, inclusive) will be translated to periods (".").

*Printable Character* is the textual equivalent of *ASCII Value* which is directly printable.

**Note:** the method *CONV\_ASCII\_to\_PrintableText* was added in BASH v1.5.1.



## **CONV\_Coerce\_from\_Text**

**CONV\_Coerce\_from\_Text**( *Text Value*; *Reference to Coerced Text*)

**CONV\_Coerce\_from\_Text**

(  
 -> *Text Value*: **Text**  
 -> *Reference to Coerced Text*: **Pointer**

)

Parameter	Type	Description
<i>Text Value</i>	Text	Text value to coerce
<i>Reference to Coerced Text</i>	Pointer	Reference to variable to contain coerced text value

The method *CONV\_Coerce\_from\_Text* will coerce a text value into a referenced variable of most any type. Coercion is done blindly on the text value, as it is merely forced into the referenced value (the only exception is that for referenced data variables, in which case the data value is normalized).

*Text Value* is the text value which is going to be coerced.

*Reference to Coerced Text* is a pointer to a variable which will contain the coerced *Text Value*. This referenced variable can be a field or variable of type boolean, date, integer, longint, real, string, text, or time.

**Note:** the method *CONV\_Coerce\_from\_Text* was added in BASH v1.5.1.



## CONV\_Coerce\_to\_Text

*CONV\_Coerce\_to\_Text*(*Referenced Value*) => *Coerced Text*

*CONV\_Coerce\_to\_Text*  
(  
    -> *Referenced Value*: **Pointer**  
)  
=> *Coerced Text*: **Text**

Parameter	Type	Description
<i>Referenced Value</i>	Pointer	Pointer to value to coerce into text
<i>Coerced Text</i>	Text	Textually coerced value referenced by <i>Referenced Value</i>

The method *CONV\_Coerce\_to\_Text* will coerce a referenced value of most any type to text. Coercion is

done blindly to the textual equivalent, as it is merely forced from the referenced value.

*Reference Value* is a pointer to a variable which contains the value to coerce to text. This referenced variable can be a field or variable of type boolean, date, integer, longint, real, string, text, or time.

*Coerced Text* contains the coerced value referenced by *Reference Value*.

**Note:** the method *CONV\_Coerce\_to\_Text* was added in BASH v1.5.1.



## CONV\_ERROR

**CONV\_ERROR** ( *BASH Error Number*; *Special Error Text*; *Calling Method Name* )

### CONV\_ERROR

```
(  
  -> BASH Error Number: Longint  
  -> Special Error Text: Text  
  -> Calling Method Name: Text  
)
```

Parameter	Type	Description
<i>BASH Error Number</i>	Longint	Internal BASH error number
<i>Special Error Text</i>	Text	Special text to describe the exact error instance
<i>Calling Method Name</i>	Text	Name of the method that the error condition occurred in

The method *CONV\_ERROR* acts as a callback method from within the CONV module for errors that may occur. Any time an error condition is detected within the CONV module, a call to the method *CONV\_ERROR* is made.

The internal *BASH Error Number* is passed to this method as the first parameter. The *Special Error Text* parameter will contain any relevant error text which is specific to the error which occurred. It is not uncommon for the

*Special Error Text* value to be empty. The *Calling Method Name* will always contain the name of the DSS method which call the **CONV\_ERROR** method.

The **CONV\_ERROR** method has been implemented as a source for a consistent interface and/or error tracking mechanism to be available while using the BASH component. This method can be modified to suit the needs of the database in which the BASH module has been installed.

**Note:** the method **CONV\_ERROR** was added in BASH v1.5.1.



## **CONV\_Hex8\_to\_Longint**

**CONV\_Hex8\_to\_Longint**(*Hex8 Value*) => *Longint Value*

**CONV\_Hex8\_to\_Longint**

(  
    -> *Hex8 Value*: Text  
)  
    => *Longint Value*: Longint

Parameter	Type	Description
<i>Hex8 Value</i>	Text	Hex8 value to convert to longint
<i>Longint Value</i>	Longint	Longint conversion of <i>Hex8 Value</i>

The method **CONV\_Hex8\_to\_Longint** converts a Hex8 value to longint.

*Hex8 Value* is the Hex8 value to convert to longint. If the length of *Hex8 Value* is greater than eight (8) bytes, then only the first eight bytes are used to create the longint conversion.

*Longint Value* is the conversion of the first eight bytes of *Hex8 Value*.

**Note:** the method **CONV\_Hex8\_to\_Longint** was added in BASH v1.5.1.



## CONV\_Hex8\_to\_Text

**CONV\_Hex8\_to\_Longint**( *Hex8 Value*) => *Text Value*

**CONV\_Hex8\_to\_Longint**

```
(  
    -> Hex8 Value: Text  
)  
=> Text Value: Text
```

Parameter	Type	Description
<i>Hex8 Value</i>	Text	Hex8 value to convert to text
<i>Text Value</i>	Text	Textual conversion of <i>Hex8 Value</i>

The method **CONV\_Hex8\_to\_Text** converts a Hex8 value to text.

*Hex8 Value* is the Hex8 value to convert to text. If the length of *Hex8 Value* is greater than eight (8) bytes, then only the first eight bytes are used to create the text conversion.

*Text Value* is the conversion of the first eight bytes of *Hex8 Value*. *Text Value* will always be four (4) bytes in length. The conversion to text is done blindly, without respecting printable ASCII values or even invalid ASCII values within a 4th Dimension text variable (i.e. ASCII value 0 is not a good value to set into a text variable in 4D).

**Note:** the method **CONV\_Hex8\_to\_Text** was added in BASH v1.5.1.



## CONV\_IP\_to\_Longint

**CONV\_IP\_to\_Longint**( *Dotted IP Address*) => *Longint Value*

**CONV\_IP\_to\_Longint**

```
(  
    -> Dotted IP Address: Text  
)
```

=> *Longint Value: Longint*

Parameter	Type	Description
<i>Dotted IP Address</i>	Text	Dotted IP address value to convert to longint
<i>Longint Value</i>	Longint	Longint conversion of <i>Dotted IP Address</i>

The method *CONV\_IP\_to\_Longint* converts a Dotted IP value to longint.

*Dotted IP Address* is the text value of the dotted IP address to convert to longint. It must be a complete and well formed dotted IP address value in text. The conversion is done blindly once the complete and well formed nature of *Dotted IP Address* is determined.

*Longint Value* is the conversion of *Dotted IP Address* to a longint value.

**Note:** the method *CONV\_IP\_to\_Longint* was added in BASH v1.5.1.



## CONV\_Longint\_to\_Hex8

*CONV\_Longint\_to\_Hex8*(*Longint Value*) => *Hex8 Value*

*CONV\_Longint\_to\_Hex8*

(  
    -> *Longint Value: Longint*  
)  
=> *Hex8 Value: Text*

Parameter	Type	Description
<i>Longint Value</i>	Longint	Longint value to convert to Hex8
<i>Hex8 Value</i>	Text	Hex8 conversion of <i>Longint Value</i>

The method *CONV\_Longint\_to\_Hex8* converts a longint value to Hex8.

*Longint Value* is the longint value to convert to Hex8.

*Hex8 Value* is the conversion of *Longint Value* to Hex8. *Hex8 Value* will always be eight (8) bytes in length.

**Note:** the method *CONV\_Longint\_to\_Hex8* was added in BASH v1.5.1.



## **CONV\_Longint\_to\_IP**

**CONV\_Longint\_to\_IP**( *Longint Value* ) => *Dotted IP Address*

**CONV\_Longint\_to\_IP**

(  
    -> *Longint Value*: Longint  
)  
    => *Dotted IP Address*: Text

Parameter	Type	Description
<i>Longint Value</i>	Longint	Longint value to covert to a Dotted IP value
<i>Dotted IP Address</i>	asdText	Dotted IP conversion of <i>Longint Value</i>

The method *CONV\_Longint\_to\_IP* converts a longint to a complete and well form Dotted IP value.

*Longint Value* is the longint value to convert to a well formed and complete Dotted IP value.

*Dotted IP Address* is the text value of the dotted IP address converted from *Longint Value*. It will always be a complete and well formed dotted IP address value in text.

**Note:** the method *CONV\_Longint\_to\_IP* was added in BASH v1.5.1.



## **CONV\_Longint\_to\_Type**

**CONV\_Longint\_to\_Type**( *Longint Value* ) => *Type Value*

**CONV\_Longint\_to\_Type**



```
(
    -> Longint Value: Longint
)
=> Type Value: String4
```

Parameter	Type	Description
<i>Longint Value</i>	Longint	Longint value to covert to a Type value
<i>Type Value</i>	String4	Type conversion of <i>Longint Value</i>

The method ***CONV\_Longint\_to\_Type*** converts a longint to a valid Type value in text.

*Longint Value* is the longint value to convert to a textual Type value.

*Type Value* is the text value of the Type converted from *Longint Value*. *Type Value* will always be four (4) bytes in length. The conversion to text is done blindly, without respecting printable ASCII values or even invalid ASCII values within a 4th Dimension text variable (i.e. ASCII value 0 is not a good value to set into a text variable in 4D).

**Note:** the method ***CONV\_Longint\_to\_Type*** was added in BASH v1.5.1.



## CONV\_Text\_to\_Hex8

***CONV\_Text\_to\_Hex8***(*Text Value*) => *Hex8 Value*

***CONV\_Text\_to\_Hex8***

```
(
    -> Text Value: Text
)
=> Hex8 Value: Text
```

Parameter	Type	Description
<i>Text Value</i>	Text	Text value to convert to Hex8
<i>Hex8 Value</i>	Text	Hex8 conversion of <i>Text Value</i>

The method ***CONV\_Text\_to\_Hex8*** converts a text value to Hex8.

*Text Value* is the text to convert to a Hex8 value. If *Text Value* is longer than four (4) bytes, then only the first four (4) bytes are used to create the resulting Hex8 value.

*Hex8 Value* is converted first four (4) bytes of *Text Value*. *Hex8 Value* will always be eight (8) bytes in length.

**Note:** the method *CONV\_Text\_to\_Hex8* was added in BASH v1.5.1.



## **CONV\_Text\_to\_Longint**

**CONV\_Text\_to\_Longint**( *Text Value* ) => *Longint Value*

**CONV\_Text\_to\_Longint**

(  
    -> *Text Value*: Text  
)  
    => *Longint Value*: Longint

Parameter	Type	Description
<i>Text Value</i>	Text	Text value to convert to longint
<i>Longint Value</i>	Longint	Longint conversion of <i>Text Value</i>

The method *CONV\_Text\_to\_Longint* converts a text value to longint. It is functionally equivalent to using the native **Int** and **Num** commands in 4th Dimension.

*Text Value* is the text value to be converted to longint.

*Longint Value* is the converted *Text Value*.

**Note:** the method *CONV\_Text\_to\_Longint* was added in BASH v1.5.1.



## **CONV\_Text\_to\_Real**

**CONV\_Text\_to\_Real**( *Text Value* ) => *Real Value*

## **CONV\_Text\_to\_Real**

```
(  
    -> Text Value: Text  
)  
=> Real Value: Real
```

Parameter	Type	Description
Text Value	Text	Text value to convert to real
Real Value	Real	Real conversion of Text Value

The method *CONV\_Text\_to\_Real* converts a text value to real. It is functionally equivalent to using the native **Num** command in 4th Dimension.

*Text Value* is the text value to be converted to real.

*Real Value* is the converted *Text Value*.

**Note:** the method *CONV\_Text\_to\_Real* was added in BASH v1.5.1.



## **CONV\_Type\_to\_Longint**

**CONV\_Type\_to\_Longint**(Type Value) => Longint Value

```
CONV_Type_to_Longint  
(  
    -> Type Value: String4  
)  
=> Longint Value: Longint
```

Parameter	Type	Description
Type Value	String4	Type value to convert to longint
Longint Value	Longint	Longint conversion of Type Value

The method *CONV\_Type\_to\_Longint* converts a valid Type value in text to longint.

*Type Value* is a valid Type value to convert to longint.

*Longint Value* is the longint value of the Type converted from *Type Value*.

**Note:** the method *CONV\_Type\_to\_Longint* was added in BASH v1.5.1.

## DSS Module

The DSS module is probably the single most used module of code which we have ever developed in 4th Dimension. It provides functionality which is exceedingly useful for any 4D programmer, regardless of the particular coding style and conventions used.

The simplicity of the DSS module is what helps make it so universally applicable. There are really only two methods to be called in the DSS module. And, the functionality within these two methods is very compact. But, the practical functionality gained is immeasurable.

Basically, the DSS module provides a means for a central pool of variables of every type to be shared across your 4th Dimension application. Whenever a variable of a particular type is needed, it can be retrieved from the DSS module for use. Once the variable is done being used, it can be returned to the DSS module for future use by other areas of your code. The variable management system within the DSS module provides a locking mechanism for each variable which is currently in use; the DSS module will not return to be used a variable which is still being used within another area of your code.

Consistent use of the DSS module can easily lead you to no longer require as many process and interprocess variables within your 4D based applications. And, it can even alleviate the commonly held shortcoming of 4D's lack of references to local variables.

As of version 1.4.6 of BASH, one hundred (100) variables of each type can be used concurrently.

### Variable Classifications

It is worth knowing, also, that variable types in 4th Dimension have two different classifications: unary and array. Unary variables are capable of storing a single

data value. Array variables are capable of storing zero, one, or more data values.

A complete listing of the unary variable types and their equivalent array data types follows, including the actual DSS variable types which used (separate unary or array data types are used within DSS):

<u>Unary</u>	<u>Array</u>	<u>DSS</u>
BLOB	n/a	<i>BLOB (unary only)</i>
Boolean	Boolean	Boolean
Date	Date	Date
Graph	n/a	<i>Graph (unary only)</i>
Integer	Integer	Integer
Longint	Longint	Longint
Picture	Picture	Picture
Pointer	Pointer	Pointer
Real	Real	Real
String	String	<i>Text</i>
Text	Text	Text
Time	Longint	<i>Time (unary only)</i> <i>Longint (array only)</i>



## **DSS\_ERROR**

**DSS\_ERROR** ( *BASh Error Number; Special Error Text; Calling Method Name* )

**DSS\_ERROR**

(  
     -> *BASh Error Number: Longint*  
     -> *Special Error Text: Text*  
     -> *Calling Method Name: Text*  
 )

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>BASh Error Number</i>	Longint	Internal BASh error number
<i>Special Error Text</i>	Text	Special text to describe the exact error instance
<i>Calling Method Name</i>	Text	Name of the method that the error condition occurred in

The method **DSS\_ERROR** acts as a callback method from within the DSS module for errors that may occur. Any

time an error condition is detected within the DSS module, a call to the method **DSS\_ERROR** is made.

The internal *BASH Error Number* is passed to this method as the first parameter. The *Special Error Text* parameter will contain any relevant error text which is specific to the error which occurred. It is not uncommon for the *Special Error Text* value to be empty. The *Calling Method Name* will always contain the name of the DSS method which call the **DSS\_ERROR** method.

The **DSS\_ERROR** method has been implemented as a source for a consistent interface and/or error tracking mechanism to be available while using the BASH component. This method can be modified to suit the needs of the database in which the BASH module has been installed.



## **DSS\_Get\_Array\_by\_Unary\_Type**

**DSS\_Get\_Array\_by\_Unary\_Type** (Unary Type;) => Referenced DSS Array Variable

**DSS\_Get\_Array\_by\_Unary\_Type**  
(  
    -> Unary Type: Longint  
)  
=> Referenced DSS Array Variable: Pointer

Parameter	Type	Description
Unary Type	Longint	Type of array variable wanted (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )
Referenced DSS Array Variable	Pointer	Reference to DSS array variable to be used matching type of Unary Type

The method **DSS\_Get\_Array\_by\_Unary\_Type** will return a reference to an available DSS array variable matching the type supplied. Unary variable types will have the array equivalent DSS variable type returned for use. This variable is locked within the DSS module from

being returned as a variable which can be used by subsequent calls to any of the *DSS\_Get\_* methods.

*Unary Type* is the unary data type desired. Array variable types can also be passed to this routine, for convenience.

*Referenced DSS Array Variable* is a pointer to a DSS array variable matching the type requested. If all DSS variables of a given type are locked and a subsequent call to any of the *DSS\_Get\_* methods is made, a NULL pointer is returned.

It is a mandatory that once a DSS variable is done being utilized that it be returned to the pool of available variables by calling *DSS\_Return\_Variable*. Failure to do so may result in an error condition.

See the method *DSS\_Get\_Variable\_by\_Type* for a more detailed explanation of the variable usage within the DSS module.

**Note:** the method *DSS\_Get\_Array\_by\_Unary\_Type* was added in BASH v1.5.1.



## **DSS\_Get\_Unary\_by\_Array\_Type**

***DSS\_Get\_Unary\_by\_Array\_Type*** ( Array Type;) => *Referenced DSS Unary Variable*

***DSS\_Get\_Unary\_by\_Array\_Type***  
(  
    -> Array Type: Longint  
)  
=> *Referenced DSS Unary Variable: Pointer*

Parameter	Type	Description
Array Type	Longint	Type of unary variable wanted (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )



<i>Referenced DSS Unary Variable</i>	<b>Pointer</b>	<b>Reference to DSS unary variable to be used matching type of Array Type</b>
--	----------------	---

The method *DSS\_Get\_Unary\_by\_Array\_Type* will return a reference to an available DSS unary variable matching the type supplied. Array variable types will have the unary equivalent DSS variable type returned for use. This variable is locked within the DSS module from being returned as a variable which can be used by subsequent calls to any of the *DSS\_Get\_* methods.

*Array Type* is the array data type desired. Unary variable types can also be passed to this routine, for convenience.

*Referenced DSS Unary Variable* is a pointer to a DSS unary variable matching the type requested. If all DSS variables of a given type are locked and a subsequent call to any of the *DSS\_Get\_* methods is made, a NULL pointer is returned.

It is a mandatory that once a DSS variable is done being utilized that it be returned to the pool of available variables by calling *DSS\_Return\_Variable*. Failure to do so may result in an error condition.

See the method *DSS\_Get\_Variable\_by\_Type* for a more detailed explanation of the variable usage within the DSS module.

**Note:** the method *DSS\_Get\_Unary\_by\_Array\_Type* was added in BASH v1.5.1.

---

## **DSS\_Get\_Variable\_by\_Type**

***DSS\_Get\_Variable\_by\_Type*** ( *Variable Type* ) => *Referenced DSS Variable*

```
DSS_Get_Variable_by_Type
(
    -> Variable Type: Longint
)
=> Referenced DSS Variable: Pointer
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Variable Type</i>	Longint	Type of variable wanted (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )
<i>Referenced DSS Variable</i>	Pointer	Reference to DSS variable to be used matching type of <i>Variable Type</i>

The method ***DSS\_Get\_Variable\_by\_Type*** returns a reference to a variable matching the type of *Variable Type*. This variable is locked within the DSS module from being returned as a variable which can be used by subsequent calls to ***DSS\_Get\_Variable\_by\_Type***.

The DSS module supports every variable type which is supported by 4th Dimension, with the sole exception of 2D arrays.

For each variable type which can be requested, the DSS module has a pool of one hundred (100) variables to select from. If all DSS variables of a given type are locked and a subsequent call to ***DSS\_Get\_Variable\_by\_Type*** is made, a NULL pointer is returned.

DSS variables which are put into use within code will be locked from being returned as available variables. Calling the DSS method ***DSS\_Return\_Variable*** will return a DSS variable to the pool of available variables, effectively unlocking it for subsequent use.

It is a mandatory that once a DSS variable is done being utilized that it be returned to the pool of available variables by calling ***DSS\_Return\_Variable***. Failure to do so may result in an error condition.

### Example:

The following snippet of code shows a short example of properly using the ***DSS\_Get\_Variable\_by\_Type*** method in place of references to local variables:

```
`FUNCTION: FILE_qi_Volume_Exists
`
```

```

` This will make sure that a volume exists
`
` $0 is qi for volume
`   = 0 volume does not exist
`   = 1 volume does exist
` $1 is the volume name to check
`

C_LONGINT ($0)
$0:=0 `default to doesn't exist
C_TEXT ($1;$xVolumeName)
$xVolumeName:=$1
`

C_POINTER ($pVolumes)
C_LONGINT ($iIndex)
`

` get a temp var from the dss
$pVolumes:=DSS_Get_Variable_by_Type (Text_array )
` get all of the current volumes
FILE_Get_All_VolumeNames ($pVolumes)
` check to see if the volume exists
$iIndex:=Find in array ($pVolumes->$xVolumeName;1)
If ($iIndex#-1)
    $0:=1
End if
` clean up
DSS_Return_Variable ($pVolumes)
` eof

```

The method *FILE\_Get\_All\_VolumeNames* requires a reference to a text array to get the list of all volumes. Instead of wasting a process or interprocess variable specifically for this method call, the DSS module provides a temporary text array as referenced by the variable \$pVolumes. This text array is used to get the list of volume names, then searched on to find a specific volume name. Once done, the text array is returned to the pool of available DSS variables for use elsewhere.



## DSS\_Return\_Variable

**DSS\_Return\_Variable** ( *Referenced DSS Variable* )

**DSS\_Return\_Variable**

```

(
    -> Referenced DSS Variable: Pointer
)

```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced DSS Variable</i>	Pointer	Reference to DSS variable to be returned to pool of available DSS variables

The method *DSS\_Return\_Variable* will return a variable to the pool of available variables within the DSS module. It should be called after using a DSS variable which was acquired from the method *DSS\_Get\_Variable\_by\_Type*.

## DTS Module

The DTS (Date-Time Stamps) module contains routines for creating, handling, and manipulating DTS values. The code within the DTS module is exceedingly simple to implement in 4th Dimension, providing basic DTS functionality needed in any system.

A DTS value is merely a fourteen (14) byte string which contains a date and time value. The format for a DTS value is as follows:

YYYYMMDDhhmmss

where

*YYYY* is four (4) byte year value (e.g. "1970")

*MM* is two (2) byte month value (e.g. "04")

*DD* is two (2) byte day value (e.g. "15")

*hh* is two (2) byte hour value (e.g. "18")

*mm* is two (2) byte minute value (e.g. "39")

*ss* is two (2) byte second value (e.g. "13")

So, for example, a DTS value of "19700415183913" would translate to April 15th, 1970, at 6:39:13 in the afternoon.

DTS value have a significant advantage over native date and time values in 4th Dimension. The advantages include:

- i) Sortability on a single, indexed field value for true chronological ordering;
- ii) Consistent storage format for date and time values, regardless of localization or machine settings;
- iii) Encoding format for DTS values is easily human readable;
- iv) Interfaces can continue to use standard date and time values, merely encoding into a DTS value for storage and usage internally within the database and code;

- v) Date and time math can now be accomplished much more easily and consistently;
- vi) The DTS system is often considered easier to understand and handle, especially when considering the large number of "gotchas" when dealing with native date and time values in 4th Dimension.

With repeated, consistent use of DTS values and the DTS module, every 4D programmer can realise significant gains in legibility and functionality when dealing with date and time values. As well, maintenance is significantly reduced as DTS values are easier to understand and deal with, thereby making it much simpler to produce error free code for handling date and time values. As well, since the DTS value type is completely contrived, there are no restrictions on how it can be utilized and implemented in 4th Dimension; you are free to utilize the concepts and routines within the DTS module to maximize the potential of each system placed into production.

**Note:** the DTS module was initially added in BASH v1.4.7.



## **DTS\_Add**

*DTS\_Add(Primary DTS Value; Secondary DTS Value) => Summed DTS Value*

```
DTS_Add
(-> Primary DTS Value: String[14]
-> Secondary DTS Value: String[14]
)=> Summed DTS Value: String[14]
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Primary DTS Value</i>	<b>String[14]</b>	First DTS addend
<i>Secondary DTS Value</i>	<b>String[14]</b>	Second DTS addend

*Summed DTS Value*      **String[14]**      **Sum of DTS addends**

The method ***DTS\_Add*** will add together two DTS values and return a normalized DTS value.

*Primary DTS Value* and *Secondary DTS Value* are both valid DTS values which are to be added.

*Summed DTS Value* is a well formed addition of *Primary DTS Value* and *Secondary DTS Value*.

#### Example:

The following table shows the values of two addends and the results which will be returned from the method ***DTS\_Add***:

<u>1st Addend</u>	<u>2nd Addend</u>	<u>Sum</u>
20001122040506	00000000010101	20001122050607
20001122040506	00000014000000	20001206040506
20001031040506	00000100000000	20001201040506
20000229040506	00010000000000	20010301040506
20001122040506	00000000200000	20001123000506

As can be seen by the above table, DTS values need not respect the rules of being well formed as date and time values often must in 4th Dimension. If a DTS value must be incremented by a certain amount, for instance by fourteen (14) days (i.e. two weeks), a single month, or merely twenty (20) hours, just it is as simple as creating a DTS value with only the amount to be incremented (use the method ***DTS\_Make\_from\_Values***, detailed below) and adding it to the original DTS value by using the ***DTS\_Add*** method.

**Note:** the method ***DTS\_Add*** was added in BASH v1.4.7.



## **DTS\_Get\_Current**

***DTS\_Get\_Current***=> *Current DTS Value*

***DTS\_Get\_Current***

=> *Current DTS Value*: String[14]

Parameter	Type	Description
<i>Current DTS Value</i>	String[14]	DTS value corresponding to the date and time on the current machine

The method *DTS\_Get\_Current* returns a DTS value containing the currently set date and time as it exists on the current machine.

*Current DTS Value* will always be a well formed date.

Take care to note though that the date and time is retrieved from the current machine even when this method is called in a client/server environment; so, the client machine will never retrieve the date and time from the server when calling *DTS\_Get\_Current*.

If the current DTS value for the server is needed, use the method *DTS\_Make\_from\_DateTime* and native 4D commands to create the DTS value.

**Note:** the method *DTS\_Get\_Current* was added in BASH v1.4.7.



## **DTS\_Get\_Date**

*DTS\_Get\_Date*(*DTS Value*) => *Date Value*

```
DTS_Get_Date
(
    -> DTS Value: String[14]
)
=> Date Value: Date
```

Parameter	Type	Description
<i>DTS Value</i>	String[14]	Valid DTS value
<i>Date Value</i>	Date	Valid 4D date value extracted from <i>DTS Value</i>



The method *DTS\_Get\_Date* will return a valid 4D date value in *Date Value* corresponding to the date stored in the supplied *DTS Value*.

*DTS Value* is any valid DTS value which the date is to be extracted from.

*Date Value* is a valid 4D date value. It is extracted from the date portion of *DTS Value* and returned as a result of *DTS\_Get\_Date*.

**Note:** the method *DTS\_Get\_Date* was added in BASH v1.4.7.



## **DTS\_Get\_Date\_Time**

*DTS\_Get\_Date\_Time*(*DTS Value*; *Referenced Date Variable*; *Referenced Time Variable*)

*DTS\_Get\_Date\_Time*

```
(  
    -> DTS Value: String[14]  
    -> Referenced Date Variable: Pointer  
    -> Referenced Time Variable: Pointer  
)
```

Parameter	Type	Description
<i>DTS Value</i>	String[14]	Valid DTS value
<i>Referenced Date Variable</i>	Pointer	Reference to a 4D date variable to hold the date contained within <i>DTS Value</i>
<i>Referenced Time Variable</i>	Pointer	Reference to a 4D time variable to hold the time contained within <i>DTS Value</i>

The method *DTS\_Get\_Date\_Time* will return in referenced date and time variables the date and time contained within a supplied DTS value.

*DTS Value* is any valid DTS value which the date and time is to be extracted from.

*Referenced Date Variable* and *Referenced Time Variable* are pointers to date and time variables, respectively. The date and time contained in *DTS Value* will be extracted and placed into *Referenced Date Variable* and *Referenced Time Variable*, respectively.

**Note:** the method *DTS\_Get\_Date\_Time* was added in BASH v1.4.7.



## **DTS\_Get\_Day**

*DTS\_Get\_Day*( *DTS Value*) => *Day Value*

*DTS\_Get\_Day*  
(  
    -> *DTS Value*: String[14]  
)  
    => *Day Value*: Longint

Parameter	Type	Description
<i>DTS Value</i>	String[14]	Valid DTS value
<i>Day Value</i>	Longint	Day value as stored in supplied <i>DTS Value</i>

The method *DTS\_Get\_Day* returns the day value for a supplied DTS value.

*DTS Value* is any valid DTS value which the day value is to be extracted from.

*Day Value* is the day value that is stored in the supplied *DTS Value*.

**Note:** the method *DTS\_Get\_Day* was added in BASH v1.4.7.



## **DTS\_Get\_Maximum**

*DTS\_Get\_Maximum* => *Maximum DTS Value*

*DTS\_Get\_Maximum*

=> *Maximum DTS Value*: String[14]

Parameter	Type	Description
<i>Maximum DTS Value</i>	String[14]	Maximum DTS value, "99991231235959"

The method *DTS\_Get\_Maximum* returns the maximum valid DTS value which is allowed.

*Maximum DTS Value* is equivalent to "99991231235959".

**Note:** remember that it is perfectly acceptable to have DTS values in which individual elements exceed the maximum individual elements that exist in *Maximum DTS Value*. For instance, the DTS value "00002000000000" is completely acceptable and can be used to easily add twenty (20) months to any other DTS value using the method *DTS\_Add*.

**Note:** the method *DTS\_Get\_Maximum* was added in BASH v1.4.7.



## **DTS\_Get\_Month**

*DTS\_Get\_Month*( *DTS Value* ) => *Month Value*

```
DTS_Get_Month
(
    -> DTS Value: String[14]
)
=> Month Value: Longint
```

Parameter	Type	Description
<i>DTS Value</i>	String[14]	Valid DTS value
<i>Month Value</i>	Longint	Month value as stored in supplied <i>DTS Value</i>

The method *DTS\_Get\_Month* returns the month value for a supplied DTS value.

*DTS Value* is any valid DTS value which the month value is to be extracted from.

*Month Value* is the month value that is stored in the supplied *DTS Value*.

**Note:** the method *DTS\_Get\_Month* was added in BASH v1.4.7.



## **DTS\_Get\_Range**

*DTS\_Get\_Range*( *Reference DTS*; *Range Type*; *Referenced Beginning DTS*;  
*Referenced Ending DTS*)

*DTS\_Get\_Range*

(  
    -> *Reference DTS*: **String[14]**  
    -> *Range Type*: **Longint**  
    -> *Referenced Beginning DTS*: **Pointer**  
    -> *Referenced Ending DTS*: **Pointer**  
)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Reference DTS</i>	<b>String[14]</b>	asd
<i>Range Type</i>	<b>Longint</b>	asd
<i>Referenced Beginning DTS</i>	<b>Pointer</b>	asd
<i>Referenced Ending DTS</i>	<b>Pointer</b>	asd

The method *DTS\_Get\_Range* returns valid DTS values matching a specified range type in respect to a specified reference DTS value.

*Reference DTS* is a valid DTS value which will be used as the reference DTS to build *Referenced Beginning DTS* and *Referenced Ending DTS* values from.

*Range Type* is a selector value which will determine what range values will be returned in *Referenced Beginning DTS* and *Referenced Ending DTS*. Valid values for *Range Type* are:

<u><i>Range Type</i></u>	<u>Range returned</u>
1	Last week

- 2 Last month
- 3 Last quarter

The following rule apply for range calculations:

- i) Weeks start on Sunday and end on Saturday;
- ii) Quarters work off of a standard calendar quarter;
- iii) Time values are always set to midnight for range types that involve values of a day or more.

More values for *Range Type* will be added in future versions of the BASH component.

**Note:** the method *DTS\_Get\_Range* was added in BASH v1.4.7.



## DTS\_Get\_Time

*DTS\_Get\_Time*( *DTS Value*) => *Time Value*

```
DTS_Get_Time
(
    -> DTS Value: String[14]
)
=> Time Value: Time
```

Parameter	Type	Description
<i>DTS Value</i>	String[14]	Valid DTS value
<i>Time Value</i>	Time	Valid 4D time value extracted from <i>DTS Value</i>

The method *DTS\_Get\_Time* will return a valid 4D time value in *Time Value* corresponding to the time stored in the supplied *DTS Value*.

*DTS Value* is any valid DTS value which the time is to be extracted from.

*Time Value* is a valid 4D time value. It is extracted from the time portion of *DTS Value* and returned as a result of *DTS\_Get\_Time* .

**Note:** the method *DTS\_Get\_Time* was added in BASH v1.4.7.



## **DTS\_Get\_Year**

*DTS\_Get\_Year*( *DTS Value*) => *Year Value*

```
DTS_Get_Year
(
    -> DTS Value: String[14]
)
=> Year Value: Longint
```

Parameter	Type	Description
<i>DTS Value</i>	String[14]	Valid DTS value
<i>Year Value</i>	Longint	Year value as stored in supplied <i>DTS Value</i>

The method *DTS\_Get\_Year* returns the year value for a supplied DTS value.

*DTS Value* is any valid DTS value which the year value is to be extracted from.

*Year Value* is the year value that is stored in the supplied *DTS Value*.

**Note:** the method *DTS\_Get\_Year* was added in BASH v1.4.7.



## **DTS\_Make\_from\_DateTime**

*DTS\_Make\_from\_DateTime*( *Date Value*; *Time Value*) => *DTS Value*

```
DTS_Make_from_DateTime
(
    -> Date Value: Date
```

```

    -> Time Value: Time
)
=> DTS Value: String[14]

```

Parameter	Type	Description
<i>Date Value</i>	Date	Valid 4th Dimension date value
<i>Time Value</i>	Time	Valid 4th Dimension time value
<i>DTS Value</i>	String[14]	Valid DTS value formed from <i>Date Value</i> and <i>Time Value</i>

The method *DTS\_Make\_from\_DateTime* will return a valid DTS value for supplied date and time values.

*Date Value* is a valid 4th Dimension date value.

*Time Value* is a valid 4th Dimension time value.

*DTS Value* is a valid DTS value formed from *Date Value* and *Time Value*.

**Note:** the method *DTS\_Make\_from\_DateTime* was added in BASH v1.4.7.



## **DTS\_Make\_from\_Values**

*DTS\_Make\_from\_Values*( *Years*; *Months*; *Days*; *Hours*; *Minutes*; *Seconds*) => *DTS Value*

```

DTS_Make_from_Values
(
    -> Years: Longint
    -> Months: Longint
    -> Days: Longint
    -> Hours: Longint
    -> Minutes: Longint
    -> Seconds: Longint
)
=> DTS Value: String[14]

```

Parameter	Type	Description
<i>Years</i>	Longint	Years value
<i>Months</i>	Longint	Months value
<i>Days</i>	Longint	Days value

<i>Hours</i>	Longint	Hours value
<i>Minutes</i>	Longint	Minutes value
<i>Seconds</i>	Longint	Seconds value
<i>DTS Value</i>	String[14]	DTS value formed directly from supplied parameters

The method *DTS\_Make\_from\_Values* creates directly a DTS value for the supplied incremental values.

*Years* is the numeric value to use for years. Values for *Years* are formatted to fit into four (4) bytes.

*Months* is the numeric value to use for months. Values for *Months* are formatted to fit into two (2) bytes.

*Days* is the numeric value to use for days. Values for *Days* are formatted to fit into two (2) bytes.

*Hours* is the numeric value to use for hours. Values for *Hours* are formatted to fit into two (2) bytes.

*Minutes* is the numeric value to use for minutes. Values for *Minutes* are formatted to fit into two (2) bytes.

*Seconds* is the numeric value to use for seconds. Values for *Seconds* are formatted to fit into two (2) bytes.

*DTS Value* is the DTS value created directly from the incrementally values.

The method *DTS\_Make\_from\_Values* does not perform any normalization upon the DTS value it creates. So, it is create a DTS value using this method which is used as a unique addend value for the method *DTS\_Add*.

### Example:

All of the following conditionals will result to True:

```
DTS_Make_from_Values (1999;12;30;4;5;6)="19991230040506"
```

```
DTS_Make_from_Values (0;0;20;0;0;0)="00000020000000"  
0" `20 months
```



*DTS\_Make\_from\_Values* (0;0;90;0;0;0)="0000009000000  
0" `90 days

*DTS\_Make\_from\_Values* (0;0;0;48;0;0)="0000000048000  
0" `48 hours

*DTS\_Make\_from\_Values* (0;0;0;0;0;90)="0000000000009  
0" `90 seconds

**Note:** the method *DTS\_Make\_from\_Values* was added  
in BASH v1.4.7.

# ENV Module

The ENV module provides information about the 4D and application environment as it currently exists. The methods provide file names and paths to many of the items used within 4D development, as well as other basic information which may be variable over different instances of running 4D based applications.

**Note:** the ENV module was initially added in BASH v1.5.1.



## **ENV\_Get\_4DApplication\_FullPath**

**ENV\_Get\_4DApplication\_FullPath** => *4D Application Path*

**ENV\_Get\_4DApplication\_FullPath**  
=> *4D Application Path: Text*

Parameter	Type	Description
<i>4D Application Path</i>	Text	Full path to the currently running 4D application

The method **ENV\_Get\_4DApplication\_FullPath** returns the full path to the current 4D application. This is functionally equivalent to using the native **Application file** command within the 4D language, except no time slicing is given to the 4D application.

*4D Application Path* is the full path to the current 4D application which is running.

**Note:** the method **ENV\_Get\_4DApplication\_FullPath** was added in BASH v1.5.1.



## **ENV\_Get\_Application\_Name**

**ENV\_Get\_Application\_Name** => *Application Name*

**ENV\_Get\_Application\_Name**

=> *Application Name*: Text

Parameter	Type	Description
<i>Application Name</i>	Text	Long name of the current application as stored in the resource fork of the current structure document

The method *ENV\_Get\_Application\_Name* returns the application name as stored in resource fork of the current structure document. The standard location to store the application name is in a resource of type 'STR#', ID 1, index 2, as defined by Apple developer documentation.

*Application Name* is the contents of the resource used for storing the application name within the current structure document.

**Note:** the method *ENV\_Get\_Application\_Name* was added in BASH v1.5.1.



## **ENV\_Get\_Application\_Name\_Short**

*ENV\_Get\_Application\_Name\_Short*=> *Application Short Name*

*ENV\_Get\_Application\_Name\_Short*  
=> *Application Short Name*: Text

Parameter	Type	Description
<i>Application Short Name</i>	Text	Short name of the current application as stored in the resource fork of the current structure document

The method *ENV\_Get\_Application\_Name\_Short* returns the application short name as stored in resource fork of the current structure document. The standard location to store the application short name is in a resource of type 'STR#', ID 1, index 1, as defined by Apple developer documentation.

*Application Short Name* is the contents of the resource used for storing the application short name within the current structure document.

Note: the method *ENV\_Get\_Application\_Name\_Short* was added in BASH v1.5.1.

---

## **ENV\_Get\_Application\_Type**

**ENV\_Get\_Application\_Type**=> *Application Indicator*

**ENV\_Get\_Application\_Type**  
=> *Application Indicator: Longint*

Parameter	Type	Description
<i>Application Indicator</i>	Longint	Application type of the currently running 4D application (application types are designated by the 4D constants <i>4D Environemnt</i> )

The method *ENV\_Get\_Application\_Type* returns the application type of the current 4D application. This is functionally equivalent to using the native **Application type** command within the 4D language, except no time slicing is given to the 4D application.

*Application Indicator* is the application type of the current 4D application which is running. Application type values are equivalent to the 4D constants within the *4D Environment* grouping.

Note: the method *ENV\_Get\_Application\_Type* was added in BASH v1.5.1.

---

## **ENV\_Get\_DataFile\_FullPath**

**ENV\_Get\_DataFile\_FullPath**=> *Data File Path*

**ENV\_Get\_DataFile\_FullPath**  
=> *Data File Path: Text*

Parameter	Type	Description
<i>Data File Path</i>	Text	Full path to the current date file

The method *ENV\_Get\_DataFile\_FullPath* returns the full path to the current data file. This is functionally equivalent to using the native **Data file** command within the 4D language, except no time slicing is given to the 4D application.

*Data File Path* is the full path to the current data file which is in use.

**Note:** the method *ENV\_Get\_DataFile\_FullPath* was added in BASH v1.5.1.



## ENV\_Get\_DirectorySymbol

*ENV\_Get\_DirectorySymbol*=> *Directory Symbol*

*ENV\_Get\_DirectorySymbol*  
=> *Directory Symbol*: String1

Parameter	Type	Description
<i>Directory Symbol</i>	String1	Directory delimiter used on current platform

The method *ENV\_Get\_DirectorySymbol* returns the character used as the delimiter between directory items on the current OS.

*Directory Symbol* is the characters used as the delimiter between directory items on the current OS. On Macintosh, this is the colon (":") and on Windows this is the back slash ("/").

**Note:** the method *ENV\_Get\_DirectorySymbol* was added in BASH v1.5.1.



## ENV\_Get\_Platform

**ENV\_Get\_Platform**=> *Platform Indicator*

**ENV\_Get\_Platform**  
=> *Platform Indicator: Longint*

Parameter	Type	Description
<i>Platform Indicator</i>	Longint	Current platform indicator (platform indicators are designated by the 4D constants <i>Platform Properties</i> )

The method **ENV\_Get\_Platform** returns the platform indicator for the current platform being used. This is functionally equivalent to using the native **PLATFORM PROPERTIES** command within the 4D language, except no time slicing is given to the 4D application.

*Platform Indicator* is the value indicating the current platform which 4D is running under. *Platform Indicator* values are equivalent to the 4D constants within the *Platform Properties* grouping.

**Note:** the method **ENV\_Get\_Platform** was added in BASH v1.5.1.



## **ENV\_Get\_Structure\_RF\_FileName**

**ENV\_Get\_Structure\_RF\_FileName**=> *Structure Resource Fork File Name*

**ENV\_Get\_Structure\_RF\_FileName**  
=> *Structure Resource Fork File Name: Text*

Parameter	Type	Description
<i>Structure Resource Fork File Name</i>	Text	File name of the resource fork of the current structure file as it exists on the current machine

The method **ENV\_Get\_Structure\_RF\_FileName** returns the file name of the resource fork of the structure file as it exists on the current machine.

When this method is called on Windows under single user 4D systems, it will return the correct file name, including the file extension, to the structure file as it

currently exists. This will work even for merged 4D applications.

When this method is called on Macintosh under single user 4D systems, it will return the file name of the structure file as it currently exists. This file name will be the same as the file name of the data fork of the current structure file due to the nature of dual fork files under MacOS.

When this method is called under 4D Client, the file name will be correct file name of the "RES" file as stored in the 4D folder. The "RES" file is just a copy of the resource fork of the current structure file, maintained automatically by 4D Client when running in client/server applications. Keep in mind, though, that resources modified in the "RES" file on 4D Client are *not* updated to the structure file on the server. In these cases, the resource fork on the server should be edited directly and flagged to be updated by the clients the next time they connect.

*Structure Resource Fork File Name* is the file name of the resource fork of the current structure on the current machine.

**Note:** the method *ENV\_Get\_Structure\_RF\_FileName* was added in BASH v1.5.1.



## **ENV\_Get\_Structure\_RF\_FullPath**

**ENV\_Get\_Structure\_RF\_FullPath** => *Structure Resource Fork Full Path*

**ENV\_Get\_Structure\_RF\_FullPath**

=> *Structure Resource Fork Full Path: Text*

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Structure Resource Fork Full Path</i>	Text	Full path of the resource fork of the current structure file as it exists on the current machine

The method *ENV\_Get\_Structure\_RF\_FullPath* returns the full path of the resource fork of the structure file as it exists on the current machine.

When this method is called on Windows under single user 4D systems, it will return the correct full path, including the file extension, to the structure file as it currently exists. This will work even for merged 4D applications.

When this method is called on Macintosh under single user 4D systems, it will return the full path to the structure file as it currently exists. This path will be the same as the full path to the data fork of the current structure file due to the nature of dual fork files under MacOS.

When this method is called under 4D Client, the full path will be correct path of the "RES" file as stored in the 4D folder. The "RES" file is just a copy of the resource fork of the current structure file, maintained automatically by 4D Client when running in client/server applications. Keep in mind, though, that resources modified in the "RES" file on 4D Client are *not* updated to the structure file on the server. In these cases, the resource fork on the server should be edited directly and flagged to be updated by the clients the next time they connect.

*Structure Resource Fork Full Path* is the full path of the resource fork of the current structure on the current machine.

**Note:** the method *ENV\_Get\_Structure\_RF\_FullPath* was added in BASH v1.5.1.



## **ENV\_q\_Windows**

**ENV\_q\_Windows** => *Windows Flag*

**ENV\_q\_Windows**  
=> *Windows Flag: Boolean*



<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Windows Flag</i>	Boolean	Flag for current machine running Windows

The method *ENV\_q\_Windows* returns a flag for whether the current machine is running under type of Windows OS, including Windows 95/98/2000 and Windows NT.

*Windows Flag* is the boolean value for whether the current machine is running under Windows.

**Note:** the method *ENV\_q\_Windows* was added in BASH v1.5.1.

# FILE Module

The FILE modules deals with path and file names directly within the operating system (OS). The current functionality available within the FILE module is fairly limited. But, with future versions of the BASH component, enhancements to the FILE module will be made available.

**Note:** the FILE module was initially added in BASH v1.5.1.



## **FILE\_Convert\_to\_ResFork\_Windows**

**FILE\_Convert\_to\_ResForm\_Windows**( *Data Fork Full Path*) => *Resource Fork Full Path*

**FILE\_Convert\_to\_ResForm\_Windows**  
(  
    -> *Data Fork Full Path: Text*  
)  
    => *Resource Fork Full Path: Text*

Parameter	Type	Description
<i>Data Fork Full Path</i>	Text	asd
<i>Resource Fork Full Path</i>	Text	asd

The method **FILE\_Convert\_to\_ResFork\_Windows** will convert a full path, relative path, or file name for the data fork of a file on Windows to the instead be to the resource fork on Windows. This will work for both plugins, applications, structure, and data documents used commonly in the 4D environment.

This method will check the *Data Fork Full Path* parameter to see if the file pointed to a 4D data file. If so, then the file extension is changed to "4DR". If the

*Data Fork Full Path* does not name a 4D data file, then the extension is changed to "RSR". No confirmation is done as to the well form nature of the *Data Fork Full Path* parameter.

*Data Fork Full Path* is a valid full path, relative path, or file name for Windows.

*Resource Fork Full Path* is the *Data Fork Full Path* value modified to be a valid resource file name on Windows. *Resource Fork Full Path* is merely a string substitution done on the last four bytes of *Data Fork Full Path*.

**Note:** the method *FILE\_Convert\_to\_ResFork\_Windows* was added in BASH v1.5.1.



## **FILE\_FullPath\_to\_FileName**

**FILE\_FullPath\_to\_FileName**( *Full Path* ) => *File Name*

**FILE\_FullPath\_to\_FileName**  
(  
    -> *Full Path*: Text  
)  
    => *File Name*: Text

Parameter	Type	Description
<i>Full Path</i>	Text	asd
<i>File Name</i>	Text	asd

The method *FILE\_FullPath\_to\_FileName* will take a full or relative path and return the filename portion. The currently directory delimiter value is respected so that full cross-platform compatibility is available within this method.

*Full Path* is the full or relative path to a file on the current platform.

*File Name* is the file name portion of *Full Path* as it would exist on the current platform.

**Note:** the method *FILE\_FullPath\_to\_FileName* was added in BASH v1.5.1.

## NULL Module

The NULL module is just a feeble attempt to make up for 4th Dimension lacking support for actual NULL/Nil values. At its core, the NULL module contains variables of each type that contain empty values (zero, empty string, zero-byte picture and BLOB, Nil pointer, etc.). Using these values, the methods within the NULL module make it easier to clear the values in a set of variables, test whether a value is NULL, and more. The functionality available in the NULL module is very simplistic; its power comes from consistent use of it throughout your code.

For the sake of reference, the following is a listing of the actual NULL variables, their types, and their values:

Variable	Type	Value
zNULL	BLOB	Blob Size is 0
qNULL	Boolean	False
dNULL	Date	!00/00/00!
isNULL	Integer	0
iNULL	LongInt	0
yNULL	Picture	(*0, empty Picture)
pNULL	Pointer	Nil
rNULL	Real	0
xNULL	Text	""
tNULL	Time	†00:00:00†
dt\$NULL	String, 14	"0000000000000000"

Under **no** circumstances should you **ever** set values into these variables. They are listed here solely for your convenience and reference purposes. Their values can be assigned to other variables, but no value should ever be set into these NULL variables.

The initial assignment of the empty values to these variables is handled internally within the NULL module. Assigning different values to any of these variables will result in erroneous operation of the code with the BASH component.



## NULL\_Dereference\_by\_Type

**NULL\_Dereferenced\_by\_Type** ( Variable Type) => Referenced NULL Variable

**NULL\_Dereference\_by\_Type**

```
(
    -> Variable Type: Longint
)
=> Referenced DSS Variable: Pointer
```

Parameter	Type	Description
Variable Type	Longint	Type of NULL variable reference wanted (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )
Referenced NULL Variable	Pointer	Reference to NULL variable matching type of Variable Type

The method **NULL\_Reference\_by\_Type** returns a reference to one of the NULL variables matching the variable type specified.

### Example:

It is common to use such a reference as an empty value within a method. For instance, consider the following code fragment:

```
C_TEXT($xMyText;$1)
$xMyText:=$1
C_LONGINT($iErrorCode)
$iErrorCode:=MyMethod ($xMyText;"";"";69)
```

It is not exactly clear what is being done or even how many parameters are being passed to *MyMethod* (well, for us old people that have poor eyesight). The following code fragment is much easier to read and understand:

```
C_TEXT($xMyText;$1)
$xMyText:=$1
C_LONGINT($iErrorCode)
C_TEXT($xNoValue)
$xNoValue:=NULL_Dereference_by_Type (Is Text) ->
$iErrorCode:=MyMethod ($xMyText;$xNoValue;$xNoValue;69)
```



## NULL\_ERROR

**NULL\_ERROR** (*BASH Error Number*; *Special Error Text*; *Calling Method Name*)

### NULL\_ERROR

(  
    -> *BASH Error Number*: **Longint**  
    -> *Special Error Text*: **Text**  
    -> *Calling Method Name*: **Text**  
)

Parameter	Type	Description
<i>BASH Error Number</i>	<b>Longint</b>	Internal NULL error number
<i>Special Error Text</i>	<b>Text</b>	Special text to describe the exact error instance
<i>Calling Method Name</i>	<b>Text</b>	Name of the method that the error condition occurred in

The method **NULL\_ERROR** acts as a callback method from within the DSS module for errors that may occur. Any time an error condition is detected within the NULL module, a call to the method **NULL\_ERROR** is made.

The internal *BASH Error Number* is passed to this method as the first parameter. The *Special Error Text* parameter will contain any relevant error text which is specific to the error which occurred. It is not uncommon for the *Special Error Text* value to be empty. The *Calling Method Name* will always contain the name of the DSS method which call the **NULL\_ERROR** method.

The **NULL\_ERROR** method has been implemented as a source for a consistent interface and/or error tracking mechanism to be available while using the BASH component. This method can be modified to suit the needs of the database in which the BASH module has been installed



## NULL\_Get\_DTS

**NULL\_Get\_DTS** => NULL DTS Value

**NULL\_Get\_DTS**

=> NULL DTS Value: String [14]

Parameter	Type	Description
NULL DTS Value	String[14]	Returns NULL DTS value, which is "00000000000000"

The method **NULL\_Get\_DTS** returns a NULL DTS value. That is, the value returned from **NULL\_Get\_DTS** is equal to "00000000000000", or 14 zeros ("0"\*14).

See the DTS library for more information on DTS values.

**Note:** the method **NULL\_Get\_DTS** was added in BASH v1.4.7.



## NULL\_Get\_Pointer

**NULL\_Get\_Pointer** => nil pointer

**NULL\_Get\_Pointer**

=> nil pointer: Pointer

Parameter	Type	Description
nil pointer	Pointer	Returns a Nil pointer

The method **NULL\_Get\_Pointer** returns a Nil pointer. That is, the pointer returned from the method **NULL\_Get\_Pointer** will evaluate in the Nil command in 4th Dimension as TRUE.

### Example:

Due to an *anomaly* in compiled and merged 4th Dimension for Windows applications, assigning a dereferenced reference to a Nil reference will crash the merged application. In other words, the following code fragment



will fail horribly as a compiled and merged Windows application:

```
C_POINTER($pClearThisVar)
$pClearThisVar:=NULL_Dereference_by_Type (Is Pointer)
->
```

To get around this slight aberration in 4D, the following code fragment should be used instead:

```
C_POINTER($pClearThisVar)
$pClearThisVar:=NULL_Get_Pointer
```

Hopefully, a future version of 4th Dimension will make this unnecessary, though for now it is needed to consistently utilize the NULL module of code.



## NULL\_Set\_Variables

**NULL\_Set\_Variables** ( *Referenced Variable {; ... }* )

**NULL\_Set\_Variables**

```
(
    -> Referenced Variable: Pointer
    { -> Referenced Variable: Pointer }
)
```

Parameter	Type	Description
<i>Referenced Variable</i>	Pointer	Reference to variable(s) to clear (set to NULL values)

The method **NULL\_Set\_Variables** will clear between one (1) and sixteen (16) referenced variables, setting their values to NULL.

This makes the initialization and clearing of variables extremely simple, saving many lines of code. For instance, consider the following code fragment:

```
SET BLOB SIZE (zMyBlob;0)
xMyText:=""
yMyPicture:=yMyPicture*0
```

This can quickly become cumbersome. Instead, a single line of code can be written:

```
NULL_Set_Variables (->zMyBlob; ->xMyText; ->yMyPicture)
```

This single line of code is much more compact and easier to read. As well, it provides a good reminder to always be tidy in the management of your memory as used by variables in your 4th Dimension applications.

# NVP Module

The NVP module is available for handling named value pairs. The methods available in the NVP module provide easy access to packing, unpacking, and extracting values from NVP stacks and text variables.

An NVP stack is a pair of text arrays which manage named data values. One array holds the names (keys) of the values. The other array contains the values associated with each named element. A NVP stack must always be well form, meaning that both arrays must always contain the same number of elements.

The routines available in the NVP modules provide a simple means for NVP stacks to be formatted into a single text variable. And, there is even a method to extract a named value from a formatted NVP stack stored in a text variable.

**Note:** the NVP module was initially added in BASH v1.5.1.



## NVP\_ERROR

**NVP\_ERROR** (*BASH Error Number; Special Error Text; Calling Method Name*)

### NVP\_ERROR

(  
-> *BASH Error Number: Longint*  
-> *Special Error Text: Text*  
-> *Calling Method Name: Text*  
)

Parameter	Type	Description
<i>BASH Error Number</i>	Longint	Internal NULL error number
<i>Special Error Text</i>	Text	Special text to describe the exact error instance

Calling Method Name	Text	Name of the method that the error condition occurred in
---------------------	------	---

The method ***NVP\_ERROR*** acts as a callback method from within the NVP module for errors that may occur. Any time an error condition is detected within the NVP module, a call to the method ***NVP\_ERROR*** is made.

The internal *BASH Error Number* is passed to this method as the first parameter. The *Special Error Text* parameter will contain any relevant error text which is specific to the error which occurred. It is not uncommon for the *Special Error Text* value to be empty. The *Calling Method Name* will always contain the name of the DSS method which call the ***NVP\_ERROR*** method.

The ***NVP\_ERROR*** method has been implemented as a source for a consistent interface and/or error tracking mechanism to be available while using the BASH component. This method can be modified to suit the needs of the database in which the BASH module has been installed

**Note:** the method ***NVP\_ERROR*** was added in BASH v1.5.1.



## **NVP\_Extract\_Values\_by\_Name\_s**

***NVP\_Extract\_Values\_by\_Name\_s*** (*NVP Name*; *Referenced Names Array*; *Referenced Values Array* ) => *NVP Value*

***NVP\_Extract\_Values\_by\_Name\_s***

```
(
    -> NVP Name: Text
    -> Referenced Names Array: Pointer
    -> Referenced Values Array: Pointer
)
=> NVP Value: Text
```

Parameter	Type	Description
<i>NVP Name</i>	<b>Text</b>	Named value to extract

<i>Referenced Names Array</i>	<b>Pointer</b>	<b>Pointer to names array</b>
<i>Referenced Values Array</i>	<b>Pointer</b>	<b>Pointer to values array</b>
<i>NVP Value</i>	<b>Text</b>	<b>Value matching <i>NVP Name</i></b>

The method ***NVP\_Extract\_Values\_by\_Names\_s*** returns the value associated with a particular name from an NVP stack. The value returned will be the first occurrence of the specified name which is found in the names array of the NVP stack.

*NVP Name* is the name to find in the names array in the NVP stack, *Referenced Names Array*.

*Referenced Names Array* is a pointer to the names array for the NVP stack to be searching.

*Referenced Values Array* is a pointer to the values array for the NVP stack to be searching.

*NVP Value* is the value found matching *NVP Name* within the referenced NVP stack. If no match is found in the NVP stack for *NVP Name*, *NVP Value* will be set to empty.

**Note:** the method ***NVP\_Extract\_Values\_by\_Names\_s*** was added in BASH v1.5.1.



## **NVP\_Extract\_Values\_by\_Name\_x**

***NVP\_Extract\_Values\_by\_Name\_x***( *Packed NVP Text*; *NVP Name*; *Column Delimiter ASCII Value*; *Row Delimiter ASCII Value*) => *NVP Value*

***NVP\_Extract\_Values\_by\_Name\_x***  
 (  
     -> *Packed NVP Text*: **Text**  
     -> *NVP Name*: **Text**  
     -> *Column Delimiter ASCII Value*: **Longint**  
     -> *Row Delimiter ASCII Value*: **Longint**  
 )  
 => *NVP Value*: **Text**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Packed NVP Text</i>	Text	NVP stack packed in text
<i>NVP Name</i>	Text	Named value to extract
<i>Column Delimiter ASCII Value</i>	Longint	ASCII Value of the column delimiter used to pack the NVP stack into text
<i>Row Delimiter ASCII Value</i>	Longint	ASCII Value of the row delimiter used to pack the NVP stack into text
<i>NVP Value</i>	Text	Value matching <i>NVP Name</i>

The method ***NVP\_Extract\_Values\_by\_Names\_x*** returns the values associated with a particular name from a packed NVP stack. The value returned will be the first occurrence of the specified name which is found in the packed NVP stack.

*Packed NVP Text* is the NVP stack packed into a text variable (see the method ***NVP\_Pack\_to\_Text*** for packing and NVP stack into a text variable).

*NVP Name* is the name to find in the names array in the NVP stack, *Referenced Names Array*.

*Column Delimiter ASCII Value* is the ASCII value of the column delimiter used to pack the NVP stack.

*Row Delimiter ASCII Value* is the ASCII value of the row delimiter used to pack the NVP stack.

*NVP Value* is the value found matching *NVP Name* within the referenced NVP stack. If no match is found in the NVP stack for *NVP Name*, *NVP Value* will be set to empty.

**Note:** the method ***NVP\_Extract\_Values\_by\_Names\_x*** was added in BASH v1.5.1.



## **NVP\_Pack\_to\_Text**

***NVP\_Pack\_to\_Text***( *Referenced Names Array*; *Referenced Values Array*;  
*Column Delimiter ASCII Value*; *Row Delimiter ASCII Value*;  
*Trailing Row Delimiter Flag*) => *Packed NVP Text*

***NVP\_Pack\_to\_Text***

```
(
    -> Referenced Names Array: Pointer
    -> Referenced Values Array: Pointer
    -> Column Delimiter ASCII Value: Longint
    -> Row Delimiter ASCII Value: Longint
    -> Trailing Row Delimiter Flag: Boolean
)
=> Packed NVP Text: Text
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Names Array</i>	<b>Pointer</b>	Pointer to names array of NVP stack
<i>Referenced Values Array</i>	<b>Pointer</b>	Pointer to values array of NVP stack
<i>Column Delimiter ASCII Value</i>	<b>Longint</b>	ASCII Value of the column delimiter to use to pack the NVP stack into text
<i>Row Delimiter ASCII Value</i>	<b>Longint</b>	ASCII Value of the row delimiter to use to pack the NVP stack into text
<i>Trailing Row Delimiter Flag</i>	<b>Boolean</b>	Flag to include the trailing row delimiter value in the packed NVP stack
<i>Packed NVP Text</i>	<b>Text</b>	NVP stack packed into text with the options specified

The method ***NVP\_Pack\_to\_Text*** packs a well formed NVP stack into a text variable using the options specified. The NVP stack must have at least one (1) pairing of names and values and it must be a well formed NVP stack.

*Referenced Names Array* is a pointer to the names array for the NVP stack to be packed.

*Referenced Values Array* is a pointer to the values array for the NVP stack to be packed.

*Column Delimiter ASCII Value* is the ASCII value of the column delimiter to use to pack the NVP stack.

*Row Delimiter ASCII Value* is the ASCII value of the row delimiter to use to pack the NVP stack.

*Trailing Row Delimiter Flag* is a boolean value to indicate whether the a row delimiter is to be placed after the last pairing within the packed NVP stack.

*Packed NVP Text* is the NVP stack packed into a text variable using all of options specified in calling this method.

**Note:** the method *NVP\_Pack\_to\_Text* was added in BASH v1.5.1.



## **NVP\_Parse\_to\_Arrays**

**NVP\_Parse\_to\_Arrays**( *Packed NVP Text*; *Referenced Names Array*; *Referenced Values Array*; *Column Delimiter ASCII Value*; *Row Delimiter ASCII Value*; *Comment ASCII Value*) => *NVP Count*

### **NVP\_Parse\_to\_Arrays**

```
(  
    -> Packed NVP Text: Text  
    -> Referenced Names Array: Pointer  
    -> Referenced Values Array: Pointer  
    -> Column Delimiter ASCII Value: Longint  
    -> Row Delimiter ASCII Value: Longint  
    -> Comment ASCII Value: Longint  
)  
=> NVP Count: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Packed NVP Text</i>	<b>Text</b>	NVP stack packed into text
<i>Referenced Names Array</i>	<b>Pointer</b>	Pointer to array to use as names array for unpacked NVP pairings
<i>Referenced Values Array</i>	<b>Pointer</b>	Pointer to array to use as values array for unpacked NVP pairings
<i>Column Delimiter ASCII Value</i>	<b>Longint</b>	ASCII Value of the column delimiter used to pack the NVP stack into text
<i>Row Delimiter ASCII Value</i>	<b>Longint</b>	ASCII Value of the row delimiter used to pack the NVP stack into text



<i>Comment ASCII Value</i>	Longint	ASCII Value of the comment character used in the packed NVP stack
<i>NVP Count</i>	Longint	Number of pairings successfully extracted from the packed NVP stack

The method ***NVP\_Parse\_to\_Arrays*** unpacks an NVP stack from a text variable using the options specified. The packed NVP stack must be well formed and formatted for the parsing routine to be successful.

*Packed NVP Text* is the NVP stack packed into a text variable which is to be parsed.

*Referenced Names Array* is a pointer to the array to hold the names of the pairings unpacked from *Packed NVP Text*..

*Referenced Values Array* is a pointer to the array to hold the values of the pairings unpacked from *Packed NVP Text*..

*Column Delimiter ASCII Value* is the ASCII value of the column delimiter used to pack the NVP stack.

*Row Delimiter ASCII Value* is the ASCII value of the row delimiter used to pack the NVP stack.

*Comment ASCII Value* is the ASCII value of the comment character to skip within the packed NVP stack. A comment byte used at the beginning of a row in a packed NVP stack will force the row to be skipped when parsed.

*NVP Count* is the number of pairings which this method parses from *Packed NVP Text* using the options specified.

**Note:** the method ***NVP\_Parse\_to\_Arrays*** was added in BASH v1.5.1.

## RES Module

The RES module contains numerous resource management methods for use within 4th Dimension. This includes methods to manage opening and closing resource forks and accessing specific resource types. Methods are also available for setting and getting different resource properties and attributes. With future versions of the BASH component, the RES module will expand significantly.

### **'fMAP' Resource**

Documentation for the 'fMap' resource will be provided in a future release of the BASH component.

### **'LoCK' Resource**

Documentation for the 'LoCK' resource will be provided in a future release of the BASH component.

### **'MENV' Resource**

Documentation for the 'MENV' resource will be provided in a future release of the BASH component.

### **'WAGr' Resource**

Documentation for the 'fMap' resource will be provided in a future release of the BASH component.

### **'WPGr' Resource**

Documentation for the 'fMap' resource will be provided in a future release of the BASH component.

**Note:** the RES module was initially added in BASH v1.5.1.



## **RES\_Close**

**RES\_Close**( *Resource Fork File Reference* )

**RES\_Close**

```
(  
    -> Resource Fork File Reference: Time  
)
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to close

The method **RES\_Close** will close a resource fork which was previously opened within 4th Dimension. This method functions exactly the same as calling the native 4D command **CLOSE RESOURCE FORK**.

*Resource Fork File Reference* is the reference value to the resource fork previously opened to close within this method.

**Note:** the method **RES\_Close** was added in BASH v1.5.1.



## **RES\_Create\_File**

**RES\_Create\_File**( *Full Path*; *File Type* ) => *asd*

**RES\_Create\_File**

```
(  
    -> Full Path: Text  
    -> File Type: Longint  
)
```

=> *Resource Fork File Reference: Time*

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Full Path</i>	<b>Text</b>	Full path to new resource document to create
<i>File Type</i>	<b>Longint</b>	File type of new resource document
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference for new resource document

The method ***RES\_Create\_File*** will create a new resource document, open it within 4th Dimension, and return the resource fork file reference for it. This method functions exactly the same as calling the native 4D command **Create resource file**.

*Full Path* is the full path to the resource document to create.

*File Type* is the file type of the new resource document being created. If *File Type* is zero (0), then the default file type will be assigned to the newly created resource document (i.e. 'res ' or ".res", as applicable by platform).

*Resource Fork File Reference* is the reference value to the resource fork of the newly created resource document.

**Note:** the method ***RES\_Create\_File*** was added in BASH v1.5.1.



## **RES\_Delete\_Resource**

***RES\_Delete\_Resource***( *Resource Fork File Reference*; *Resource Type*;  
*Resource ID*) => *Success Indicator*

### ***RES\_Delete\_Resource***

(  
-> *Resource Fork File Reference: Time*  
-> *Resource Type: Longint*  
-> *Resource ID: Longint*  
)  
=> *Success Indicator: Longint*

Parameter	Type	Description
<i>Resource Fork File Reference</i>	Time	Resource fork file reference to delete from
<i>Resource Type</i>	Longint	Resource type to delete
<i>Resource ID</i>	Longint	Resource ID to delete
<i>Success Indicator</i>	Longint	qi for successfully deleting resource

The method *RES\_Delete\_Resource* will delete a particular resource. The resource must be identified by the resource file, resource type, and resource ID. This method is functionally equivalent to using the netive 4D command **DELETE RESOURCE**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to delete from.

*Resource Type* is the resource type to delete.

*Resource ID* is the resource ID to delete.

*Success Indicator* is an indicator value for whether the deletion was successful. If *Success Indicator* is zero (0) then the resource was not deleted; if *Success Indicator* is one (1) then the resource was successully deleted.

**Note:** the method *RES\_Delete\_Resource* was added in BASH v1.5.1.



## **RES\_ERROR**

**RES\_ERROR** ( *BASH Error Number*; *Special Error Text*; *Calling Method Name* )

**RES\_ERROR**  
 (  
     -> *BASH Error Number*: Longint  
     -> *Special Error Text*: Text  
     -> *Calling Method Name*: Text  
 )

Parameter	Type	Description
-----------	------	-------------

<i>BASH Error Number</i>	<b>Longint</b>	<b>Internal NULL error number</b>
<i>Special Error Text</i>	<b>Text</b>	<b>Special text to describe the exact error instance</b>
<i>Calling Method Name</i>	<b>Text</b>	<b>Name of the method that the error condition occurred in</b>

The method ***RES\_ERROR*** acts as a callback method from within the RES module for errors that may occur. Any time an error condition is detected within the RES module, a call to the method ***RES\_ERROR*** is made.

The internal *BASH Error Number* is passed to this method as the first parameter. The *Special Error Text* parameter will contain any relevant error text which is specific to the error which occurred. It is not uncommon for the *Special Error Text* value to be empty. The *Calling Method Name* will always contain the name of the DSS method which call the ***RES\_ERROR*** method.

The ***RES\_ERROR*** method has been implemented as a source for a consistent interface and/or error tracking mechanism to be available while using the BASH component. This method can be modified to suit the needs of the database in which the BASH module has been installed

**Note:** the method ***RES\_ERROR*** was added in BASH v1.5.1.



## **RES\_Get\_Resource\_List**

***RES\_Get\_Resource\_List***( *Resource Fork File Reference*; *Resource Type*;  
*Referenced Resource IDs*; *Referenced Resource Names*)

***RES\_Get\_Resource\_List***

```
(
    -> Resource Fork File Reference: Time
    -> Resource Type: Longint
    -> Referenced Resource IDs: Pointer
    -> Referenced Resource Names: Pointer
)
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource Type Referenced</i>	<b>Longint Pointer</b>	Resource type to work with Pointer to array to hold resource IDs
<i>Resource IDs Referenced</i>	<b>Pointer</b>	Pointer to array to hold resource names
<i>Resource Names</i>		

The method ***RES\_Get\_Resource\_List*** retrieves a list of all of the resources of a specified resource type from a specified resource document. This method is functionally equivalent to the native 4D command **RESOURCE LIST**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource Type* is the resource type to work with.

*Referenced Resource IDs* is a pointer to a longint array which will contain the resource IDs matching *Resource Type* in the resource document *Resource Fork File Reference*.

*Referenced Resource Names* is a pointer to a text array which will contain the resource names matching *Resource Type* in the resource document *Resource Fork File Reference*.

**Note:** the values within *Referenced Resource IDs* and *Referenced Resource Names* are a well formed stack.

**Note:** the method ***RES\_Get\_Resource\_List*** was added in BASH v1.5.1.



## **RES\_Get\_TEXT\_Resource**

***RES\_Get\_TEXT\_Resource***( *Resource Fork File Reference*; *Resource ID*) =>  
*Resource Text Value*

## ***RES\_Get\_TEXT\_Resource***

```
(  
    -> Resource Fork File Reference: Time  
    -> Resource ID: Longint  
)  
=> Resource Text Value: Text
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Resource Text Value</i>	<b>Text</b>	Text value stored in specified resource

The method *RES\_Get\_TEXT\_Resource* retrieves the contents of a 'TEXT' resource from a specified resource document. This method is functionally equivalent to the native 4D command **Get text resource**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

*Resource Text Value* is the contents of the specified 'TEXT' resource.

**Note:** the method *RES\_Get\_TEXT\_Resource* was added in BASH v1.5.1.



## ***RES\_Load\_cicn***

***RES\_Load\_cicn***( *Resource Fork File Reference*; *Resource ID*; *Referenced Picture*) => *Success Indicator*

```
RES_Load_cicn  
(  
    -> Resource Fork File Reference: Time  
    -> Resource ID: Longint  
    -> Referenced Picture: Pointer  
)  
=> Success Indicator: Longint
```



<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Picture</i>	<b>Pointer</b>	Pointer to picture variable to load resource into
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method ***RES\_Load\_cicn*** retrieves a the contents of an icon resource from a specified resource document. This method is functionally equivalent to the native 4D command **GET ICON RESOURCE**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

*Referenced Picture* is a pointer to a picture variable to load the resource contents into.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method ***RES\_Load\_cicn*** was added in BASH v1.5.1.



## **RES\_Load\_fMap**

***RES\_Load\_fMAP***( *Resource Fork File Reference*; *Resource ID*; *Referenced Title*; *Referenced Macintosh Types*; *Referenced Windows Types*; *Referenced Macintosh Creators*; *Referenced MIME Types*; *Referenced Parm Bits*) => *Success Indicator*

### ***RES\_Load\_fMAP***

(

- > *Resource Fork File Reference*: **Time**
- > *Resource ID*: **Longint**
- > *Referenced Title*: **Pointer**
- > *Referenced Macintosh Types*: **Pointer**
- > *Referenced Windows Types*: **Pointer**

-> *Referenced Macintosh Creators*: **Pointer**  
 -> *Referenced MIME Types*: **Pointer**  
 -> *Referenced Parm Bits*: **Pointer**  
 )  
 => *Success Indicator*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Title</i>	<b>Pointer</b>	not available at this time
<i>Referenced Macintosh Types</i>	<b>Pointer</b>	not available at this time
<i>Referenced Windows Types</i>	<b>Pointer</b>	not available at this time
<i>Referenced Macintosh Creators</i>	<b>Pointer</b>	not available at this time
<i>Referenced MIME Types</i>	<b>Pointer</b>	not available at this time
<i>Referenced Parm Bits</i>	<b>Pointer</b>	not available at this time
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method *RES\_Load\_fMap* retrieves the contents of an 'fMap' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

The remainder of the documentation for this method, including the exact structure of the 'fMap' resource is being left for future versions of the BASH component.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method *RES\_Load\_fMap* was added in BASH v1.5.1.



## RES\_Load.LoCK

**RES\_Load.LoCK**( *Resource Fork File Reference*; *Resource ID*; *Referenced Lock Pin Names*; *Referenced Low Flags*; *Referenced High Flags*)  
=> *Success Indicator*

### **RES\_Load.LoCK**

(  
-> *Resource Fork File Reference*: **Time**  
-> *Resource ID*: **Longint**  
-> *Referenced Lock Pin Names*: **Pointer**  
-> *Referenced Low Flags*: **Pointer**  
-> *Referenced High Flags*: **Pointer**  
)  
=> *Success Indicator*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Lock Pin Names</i>	<b>Pointer</b>	not available at this time
<i>Referenced Low Flags</i>	<b>Pointer</b>	not available at this time
<i>Referenced High Flags</i>	<b>Pointer</b>	not available at this time
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method **RES\_Load.LoCK** retrieves the contents of a 'LoCK' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

The remainder of the documentation for this method, including the exact structure of the 'LoCK' resource is being left for future versions of the BASH component.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0)

then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method *RES\_Load\_LoCK* was added in BASH v1.5.1.



## **RES\_Load\_MBAR**

**RES\_Load\_MBAR**( *Resource Fork File Reference*; *Resource ID*; *Referenced MENU Resource IDs*) => *Success Indicator*

### **RES\_Load\_MBAR**

(  
-> *Resource Fork File Reference*: **Time**  
-> *Resource ID*: **Longint**  
-> *Referenced MENU Resource IDs*: **Pointer**  
)  
=> *Success Indicator*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced MENU Resource IDs</i>	<b>Pointer</b>	Pointer to array to hold 'MENU' resource IDs within specified 'MBAR' resource
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method *RES\_Load\_MBAR* retrieves the contents of an 'MBAR' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

*Referenced MENU Resource IDs* is a pointer to a longint array which will hold the 'MENU' resource IDs referenced in the specified 'MBAR' resource.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0)

then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method *RES\_Load\_MBAR* was added in BASH v1.5.1.



## **RES\_Load\_MENU**

**RES\_Load\_MENU**( *Resource Fork File Reference*; *Resource ID*; *Referenced Menu Item Titles*; *Referenced Menu Item States*; *Referenced Menu Item Styles*; *Referenced Menu Item Keys*; *Referenced Menu Item Marks*) => *Success Indicator*

### **RES\_Load\_MENU**

```
(  
    -> Resource Fork File Reference: Time  
    -> Resource ID: Longint  
    -> Referenced Menu Item Titles: Pointer  
    -> Referenced Menu Item States: Pointer  
    -> Referenced Menu Item Styles: Pointer  
    -> Referenced Menu Item Keys: Pointer  
    -> Referenced Menu Item Marks: Pointer  
)  
=> Success Indicator: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Menu Item Titles</i>	<b>Pointer</b>	Pointer to array to hold menu item titles
<i>Referenced Menu Item States</i>	<b>Pointer</b>	Pointer to array to hold menu item states
<i>Referenced Menu Item Styles</i>	<b>Pointer</b>	Pointer to array to hold menu item styles
<i>Referenced Menu Item Keys</i>	<b>Pointer</b>	Pointer to array to hold menu items keys
<i>Referenced Menu Item Marks</i>	<b>Pointer</b>	Pointer to array to hold menu items marks
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method *RES\_Load\_MENU* retrieves the contents of a 'MENU' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

*Referenced Menu Item Titles* is a pointer to a text array which will hold the menu item names for the specified 'MENU' resource.

*Referenced Menu Item States* is a pointer to a longint array which will hold the menu item states for the specified 'MENU' resource.

*Referenced Menu Item Styles* is a pointer to a longint array which will hold the menu item styles for the specified 'MENU' resource.

*Referenced Menu Item Keys* is a pointer to a longint array which will hold the menu item keys for the specified 'MENU' resource.

*Referenced Menu Item Marks* is a pointer to a longint array which will hold the menu item marks for the specified 'MENU' resource.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the values within *Referenced Template Referenced Menu Item Titles*, *Referenced Menu Item States*, *Referenced Menu Item Styles*, *Referenced Menu Item Keys*, and *Referenced Menu Item Marks* are a well formed stack.

**Note:** the method *RES\_Load\_MENU* was added in BASH v1.5.1.



## **RES\_Load\_MENU**

**RES\_Load\_MENV**( *Resource Fork File Reference*; *Resource ID*; *Referenced Menu Item Names*; *Referenced Method Names*) => *Success Indicator*

**RES\_Load\_MENV**

(  
 -> *Resource Fork File Reference*: **Time**  
 -> *Resource ID*: **Longint**  
 -> *Referenced Menu Item Names*: **Pointer**  
 -> *Referenced Method Names*: **Pointer**  
 )  
 => *Success Indicator*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Menu Item Names</i>	<b>Pointer</b>	not available at this time
<i>Referenced Method Names</i>	<b>Pointer</b>	not available at this time
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method **RES\_Load\_MENV** retrieves the contents of a 'MENV' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

The remainder of the documentation for this method, including the exact structure of the 'MENV' resource is being left for future versions of the BASH component.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method **RES\_Load\_MENV** was added in BASH v1.5.1.



## RES\_Load\_PICT

**RES\_Load\_PICT**( *Resource Fork File Reference*; *Resource ID*; *Referenced Picture*) => *Success Indicator*

### RES\_Load\_PICT

(  
    -> *Resource Fork File Reference*: **Time**  
    -> *Resource ID*: **Longint**  
    -> *Referenced Picture*: **Pointer**  
)  
    => *Success Indicator*: **Longint**

Parameter	Type	Description
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Picture</i>	<b>Pointer</b>	Pointer to picture variable to load resource into
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method **RES\_Load\_PICT** retrieves the contents of a 'PICT' resource from a specified resource document. This method is functionally equivalent to the native 4th Dimension command **GET PICTURE RESOURCE**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

*Referenced Picture* is a pointer to a picture variable to load the resource contents into.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method **RES\_Load\_PICT** was added in BASH v1.5.1.





## RES\_Load\_TMPL

**RES\_Load\_TMPL**( *Resource Fork File Reference*; *Resource ID*; *Referenced Template Item Titles*; *Referenced Template Field Types*) => *Success Indicator*

### RES\_Load\_TMPL

```
(
    -> Resource Fork File Reference: Time
    -> Resource ID: Longint
    -> Referenced Template Item Titles: Pointer
    -> Referenced Template Field Types: Pointer
)
=> Success Indicator: Longint
```

Parameter	Type	Description
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Template Item Titles</i>	<b>Pointer</b>	Pointer to text array to hold template item titles which are loaded
<i>Referenced Template Field Types</i>	<b>Pointer</b>	Pointer to longint array to hold template field types which are loaded
<i>Success Indicator</i>	<b>Longint</b>	qi for loaded successfully

The method **RES\_Load\_TMPL** retrieves the contents of a 'PICT' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

*Referenced Template Item Titles* is a pointer to a text array which will hold the template item titles which are loaded from the specified resource.

*Referenced Template Field Types* is a pointer to a longint array which will hold the template field types which are loaded from the specified resource.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the values within *Referenced Template Item Titles* and *Referenced Template Item Titles* are a well formed stack.

**Note:** the method *RES\_Load\_TMPL* was added in BASH v1.5.1.



## **RES\_Load\_WAGr**

**RES\_Load\_WAGr**( *Resource Fork File Reference*; *Resource ID*; *Referenced Lock Code*; *Referenced Table Name*; *Referenced Action Titles*; *Referenced Action Values*; *Referenced Method Names*; *Referenced Link Titles*; *Referenced Resource IDs*; *Referenced Pin Indices*; *Referenced Page Flags*) => *Success Indicator*

### **RES\_Load\_WAGr**

```
(  
-> Resource Fork File Reference: Time  
-> Resource ID: Longint  
-> Referenced Lock Code: Pointer  
-> Referenced Table Name: Pointer  
-> Referenced Action Titles: Pointer  
-> Referenced Action Values: Pointer  
-> Referenced Method Names: Pointer  
-> Referenced Link Titles: Pointer  
-> Referenced Resource IDs: Pointer  
-> Referenced Pin Indices: Pointer  
-> Referenced Page Flags: Pointer  
)  
=> Success Indicator: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Referenced Lock Code</i>	<b>Pointer</b>	not available at this time

<i>Referenced Table Name</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Referenced Action Titles</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Referenced Action Values</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Referenced Method Names</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Referenced Link Titles</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Referenced Resource IDs</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Referenced Pin Indices</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Referenced Page Flags</i>	<b>Pointer</b>	<b>not available at this time</b>
<i>Success Indicator</i>	<b>Longint</b>	<b>qi for loaded successfully</b>

The method *RES\_Load\_WAGr* retrieves the contents of a 'WAGr' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

The remainder of the documentation for this method, including the exact structure of the 'WAGr' resource is being left for future versions of the BASH component.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method *RES\_Load\_WAGr* was added in BASH v1.5.1.



## **RES\_Load\_WPGr**

**RES\_Load\_WPGr**( *Resource Fork File Reference*; *Resource ID*; *Referenced Global Lock Code*; *Referenced Global Pin Index*; *Referenced File Names*; *Referenced Resource IDs*; *Referenced Pin Indices*;

Referenced Link Titles; Referenced Method Names; Referenced Low Product Flags; Referenced High Product Flags) => Success Indicator

## **RES\_Load\_WPGr**

```
(
    -> Resource Fork File Reference: Time
    -> Resource ID: Longint
    -> Referenced Global Lock Code: Pointer
    -> Referenced Global Pin Index: Pointer
    -> Referenced File Names: Pointer
    -> Referenced Resource IDs: Pointer
    -> Referenced Pin Indices: Pointer
    -> Referenced Link Titles: Pointer
    -> Referenced Method Names: Pointer
    -> Referenced Low Product Flags: Pointer
    -> Referenced High Product Flags: Pointer
)
=> Success Indicator: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
Resource Fork File Reference	<b>Time</b>	Resource fork file reference to work with
Resource ID	<b>Longint</b>	Resource ID to work with
Referenced Global Lock Code	<b>Pointer</b>	not available at this time
Referenced Global Pin Index	<b>Pointer</b>	not available at this time
Referenced File Names	<b>Pointer</b>	not available at this time
Referenced Resource IDs	<b>Pointer</b>	not available at this time
Referenced Pin Indices	<b>Pointer</b>	not available at this time
Referenced Link Titles	<b>Pointer</b>	not available at this time
Referenced Method Names	<b>Pointer</b>	not available at this time
Referenced Low Product Flags	<b>Pointer</b>	not available at this time
Referenced High Product Flags	<b>Pointer</b>	not available at this time
Success Indicator	<b>Longint</b>	qi for loaded successfully

The method **RES\_Load\_WPGr** retrieves the contents of a 'WPGr' resource from a specified resource document.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

The remainder of the documentation for this method, including the exact structure of the 'WPGr' resource is being left for future versions of the BASH component.

*Success Indicator* is an indicator value for whether the loading was successful. If *Success Indicator* is zero (0) then the resource was not loaded; if *Success Indicator* is one (1) then the resource was successfully loaded.

**Note:** the method *RES\_Load\_WPGr* was added in BASH v1.5.1.

---

## **RES\_Make\_TMPL\_f\_Arrays**

**RES\_Make\_TMPL\_f\_Arrays** (*Referenced BLOB; Referenced Template Item Titles; Referenced Template Field Types*) => *Success Indicator*

**RES\_Make\_TMPL\_f\_Arrays**  
(  
    -> *Referenced BLOB: Pointer*  
    -> *Referenced Template Item Titles: Pointer*  
    -> *Referenced Template Field Types: Pointer*  
)  
=> *Success Indicator: Longint*

Parameter	Type	Description
<i>Referenced BLOB</i>	Pointer	Pointer to BLOB to contain 'TMPL' resource created
<i>Referenced Template Item Titles</i>	Pointer	Pointer to text array containing template item titles
<i>Referenced Template Field Types</i>	Pointer	Pointer to longint array containing template field types
<i>Success Indicator</i>	Longint	qi for created successfully

The method *RES\_Make\_TMPL\_f\_Arrays* will create a 'TMPL' resource from the values specified. The 'TMPL' resource will be placed into a referenced BLOB value for further use and manipulation.

*Referenced BLOB* is a pointer to a BLOB to place the 'TMPL' resource created into.

*Referenced Template Item Titles* is a pointer to a text array containing the template item titles to be created in the 'TMPL' resource.

*Referenced Template Field Types* is a pointer to a longint array containing the template field types to be created in the 'TMPL' resource.

*Success Indicator* is an indicator value for whether the creation was successful. If *Success Indicator* is zero (0) then the resource was not created; if *Success Indicator* is one (1) then the resource was successfully created.

**Note:** the method *RES\_Make\_TMPL\_f\_Arrays* was added in BASH v1.5.1.



## RES\_Open

**RES\_Open**( Full Path) => Resource Fork File Reference

### RES\_Open

```
(  
    -> Full Path: Text  
)  
=> Resource Fork File Reference: Time
```

Parameter	Type	Description
Full Path	Text	Full path to resource document to open
Resource Fork File Reference	Time	Resource fork file reference opened

The method **RES\_Open** will open a resource fork within 4th Dimension and return a resource fork file reference for accessing the contents. This method functions exactly the same as calling the native 4D command **Open resource fork**.

*Full Path* is the full path to the resource document to open within this method.

*Resource Fork File Reference* is the reference value to the resource fork opened within this method. If *Resource Fork File Reference* is NULL then the resource document specified failed to be opened.

**Note:** there is no harm or error condition which occurs if a resource document that is already opened within 4D is opened again. The same resource fork file reference will again be returned for use and there is no need to subsequently close the resource document multiple times.

**Note:** the method **RES\_Open** was added in BASH v1.5.1.



## **RES\_Open\_4DApplication**

**RES\_Open\_4DApplication**=> *Resource Fork File Reference*

**RES\_Open\_4DApplication**

=> *Resource Fork File Reference: Time*

Parameter	Type	Description
<i>Resource Fork File Reference</i>	Time	Resource fork file reference opened

The method **RES\_Open\_4DApplication** will open the resource fork of the current 4D application and return a resource fork file reference for accessing the contents.

*Resource Fork File Reference* is the reference value to the resource fork opened within this method. If *Resource Fork File Reference* is NULL then the resource fork failed to be opened.

**Note:** there is no harm or error condition which occurs if this method is called multiple times within the same 4D session. The same resource fork file reference will be returned for each subsequent call to this method. Under no circumstances though should you ever close the resource fork of the current 4D application.

**Note:** the method *RES\_Open\_4DApplication* was added in BASH v1.5.1.



## **RES\_Open\_DataFile**

**RES\_Open\_DataFile**=> *Resource Fork File Reference*

**RES\_Open\_DataFile**

=> *Resource Fork File Reference: Time*

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference opened

The method *RES\_Open\_DataFile* will open the resource fork of the current 4D data file and return a resource fork file reference for accessing the contents.

*Resource Fork File Reference* is the reference value to the resource fork opened within this method. If *Resource Fork File Reference* is NULL then the resource fork failed to be opened.

**Note:** there is no harm or error condition which occurs if this method is called multiple times within the same 4D session. The same resource fork file reference will be returned for each subsequent call to this method.

**Note:** the method *RES\_Open\_DataFile* was added in BASH v1.5.1.



## **RES\_Open\_Structure**



**RES\_Open\_Structure**=> *Resource Fork File Reference*

**RES\_Open\_Structure**

=> *Resource Fork File Reference: Time*

Parameter	Type	Description
<i>Resource Fork File Reference</i>	Time	Resource fork file reference opened

The method **RES\_Open\_Structure** will open the resource fork of the current 4D structure file and return a resource fork file reference for accessing the contents.

*Resource Fork File Reference* is the reference value to the resource fork opened within this method. If *Resource Fork File Reference* is NULL then the resource fork failed to be opened.

**Note:** there is no harm or error condition which occurs if this method is called multiple times within the same 4D session. The same resource fork file reference will be returned for each subsequent call to this method. Under no circumstances though should you ever close the resource fork of the current 4D structure file.

**Note:** the method **RES\_Open\_Structure** was added in BASH v1.5.1.



**RES\_Parse\_TMPL**

**RES\_Parse\_TMPL**( *Referenced BLOB; Referenced Template Item Titles; Referenced Template Field Types* ) => *Success Indicator*

**RES\_Parse\_TMPL**

(  
-> *Referenced BLOB: Pointer*  
-> *Referenced Template Item Titles: Pointer*  
-> *Referenced Template Field Types: Pointer*  
)  
=> *Success Indicator: Longint*

Parameter	Type	Description
-----------	------	-------------

<i>Referenced BLOB</i>	<b>Pointer</b>	Pointer to BLOB containing 'TMPL' resource to parse
<i>Referenced Template Item Titles</i>	<b>Pointer</b>	Pointer to text array to contain template item titles
<i>Referenced Template Field Types</i>	<b>Pointer</b>	Pointer to longint array to contain template field types
<i>Success Indicator</i>	<b>Longint</b>	qi for parsed successfully

The method ***RES\_Parse\_TMPL*** will parse a 'TMPL' resource individual pairings of template item titles and template field types.

*Referenced BLOB* is a pointer to a BLOB containing the 'TMPL' resource to parse.

*Referenced Template Item Titles* is a pointer to a text array which will contain the template item titles parsed from the 'TMPL' resource.

*Referenced Template Field Types* is a pointer to a longint array which will contain the template field types parsed from the 'TMPL' resource.

*Success Indicator* is an indicator value for whether the parsing was successful. If *Success Indicator* is zero (0) then the resource was not parsed; if *Success Indicator* is one (1) then the resource was successfully parsed.

**Note:** the method ***RES\_Parse\_TMPL*** was added in BASH v1.5.1.



## **RES\_Set\_Resource\_Name**

***RES\_Set\_Resource\_Name***( *Resource Fork File Reference*; *Resource Type*; *Resource ID*; *Resource Name*) => *Success Indicator*

***RES\_Set\_Resource\_Name***

(

- > *Resource Fork File Reference*: **Time**
- > *Resource Type*: **Longint**
- > *Resource ID*: **Longint**

-> *Resource Name*: **String255**  
 )  
 => *Success Indicator*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork</i>	<b>Time</b>	Resource fork file reference to work with
<i>File Reference</i>		
<i>Resource Type</i>	<b>Longint</b>	Resource type to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Resource Name</i>	<b>String255</b>	Value to set the resource name to
<i>Success Indicator</i>	<b>Longint</b>	qi for set successfully

The method ***RES\_Set\_Resource\_Name*** set the name of a specified resource in a specified resource document. This method is functionally equivalent to the native 4D command **SET RESOURCE NAME**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource Type* is the resource type to work with.

*Resource ID* is the resource ID to work with.

*Resource Name* is the value to set the specified resource name to.

*Success Indicator* is an indicator value for whether the setting was successful. If *Success Indicator* is zero (0) then the resource was not set; if *Success Indicator* is one (1) then the resource was successfully set.

**Note:** the method ***RES\_Set\_Resource\_Name*** was added in BASH v1.5.1.



## **RES\_Set\_Resource\_Properties**

***RES\_Set\_Resource\_Properties***( *Resource Fork File Reference*; *Resource Type*; *Resource ID*; *Resource Properties*) => *Success Indicator*

***RES\_Set\_Resource\_Properties***

(  
 -> *Resource Fork File Reference*: **Time**

```

-> Resource Type: Longint
-> Resource ID: Longint
-> Resource Properties: Longint
)
=> Success Indicator: Longint

```

Parameter	Type	Description
Resource Fork File Reference	Time	Resource fork file reference to work with
Resource Type	Longint	Resource type to work with
Resource ID	Longint	Resource ID to work with
Resource Properties	Longint	Value to set the resource properties to
Success Indicator	Longint	qi for set successfully

The method *RES\_Set\_Resource\_Properties* set the properties of a specified resource in a specified resource document. This method is functionally equivalent to the native 4D command **SET RESOURCE PROPERTIES**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource Type* is the resource type to work with.

*Resource ID* is the resource ID to work with.

*Resource Properties* is the value to set the specified resource properties to.

*Success Indicator* is an indicator value for whether the setting was successful. If *Success Indicator* is zero (0) then the resource was not set; if *Success Indicator* is one (1) then the resource was successfully set.

**Note:** the method *RES\_Set\_Resource\_Properties* was added in BASH v1.5.1.



## **RES\_Set\_TEXT\_Resource**

**RES\_Set\_TEXT\_Resource**( Resource Fork File Reference; Resource ID; Resource Text Value) => Success Indicator

## ***RES\_Set\_TEXT\_Resource***

```
(  
    -> Resource Fork File Reference: Time  
    -> Resource ID: Longint  
    -> Resource Text Value: Text  
)  
=> Success Indicator: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Resource Fork File Reference</i>	<b>Time</b>	Resource fork file reference to work with
<i>Resource ID</i>	<b>Longint</b>	Resource ID to work with
<i>Resource Text Value</i>	<b>Text</b>	Value to set the resource contents to
<i>Success Indicator</i>	<b>Longint</b>	qi for set successfully

The method ***RES\_Set\_TEXT\_Resource*** sets the contents of a 'TEXT' resource within a specified resource document. This method is functionally equivalent to the native 4D command **SET TEXT RESOURCE**.

*Resource Fork File Reference* is the reference value to the resource fork of the resource document to work with.

*Resource ID* is the resource ID to work with.

*Resource Text Value* is the contents to set for the specified 'TEXT' resource.

*Success Indicator* is an indicator value for whether the setting was successful. If *Success Indicator* is zero (0) then the resource was not set; if *Success Indicator* is one (1) then the resource was successfully set.

**Note:** the method ***RES\_Set\_TEXT\_Resource*** was added in BASH v1.5.1.

# SEM Module

The SEM module is the beginning of wrapper functionality for semaphores in 4th Dimension. The current functionality available within the SEM module is fairly limited and may at times seem redundant for what is already available natively within 4th Dimension. But, with future versions of the BASH component, enhancements to the SEM module will be made available.



## SEM\_Clear\_One

**SEM\_Clear\_One** (*Semaphore Name*)

**SEM\_Clear\_One**  
(  
    -> *Semaphore Name*: String [32]  
)

Parameter	Type	Description
<i>Semaphore Name</i>	String [32]	Name of semaphore to clear

The method **SEM\_Clear\_One** will clear a named semaphore.

The *Semaphore Name* parameter is the name of the semaphore which should be cleared.

This method is equivalent to calling the built in 4th Dimension command **CLEAR SEMAPHORE**. It is recommended that the method **SEM\_Clear\_One** be used in place of **CLEAR SEMAPHORE** though as future enhancements to the SEM module will provide more functionality within this method.



## SEM\_Set\_One

**SEM\_Set\_One** (*Semaphore Name*; *Ticks Between Checks*; *Total Ticks to Wait*) => *Semaphore Status*

## ***SEM\_Set\_One***

```
(  
    -> Semaphore Name: String [32]  
    -> Ticks Between Checks: Longint  
    -> Total Ticks to Wait: Longint  
)  
=> Semaphore Status: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Semaphore Name</i>	<b>String [32]</b>	Name of the semaphore to set
<i>Ticks Between Checks</i>	<b>Longint</b>	Number of ticks to wait between checking for the availability of the semaphore to the current process
<i>Total Ticks to Wait</i>	<b>Longint</b>	Total number of ticks to wait for the semaphore in the current process until returning from the method
<i>Semaphore Status</i>	<b>Longint</b>	Status of attempt to set named semaphore

The method ***SEM\_Set\_One*** is used to set a semaphore from within the current process. It handles checking whether the semaphore is already set, and will wait for it to be cleared for a specified amount of time before setting it within this call or returning without having set it.

The *Semaphore Name* parameter is the name of the semaphore which should be set.

The *Ticks Between Checks* parameter is the number of ticks the method will wait between calls to check whether the named semaphore is available to be set; if the named semaphore is already set, then this method will loop with a delay of *Ticks Between Checks* between each loop waiting for the semaphore to become cleared before setting it within this method. Setting this parameter to zero (0) will product no delay in the loop.

The *Total Ticks to Wait* parameter is the total number of ticks this method will wait for the named semaphore to become available before it stops trying. If the named semaphore is already set and this method has already

waited for *Total Ticks to Wait* for the semaphore to be cleared, the method will return a status code indicating such (see below). If the *Total Ticks to Wait* parameter is set to zero (0), then the named semaphore will be checked for availability only once and set if available; otherwise, no waiting for the named semaphore will be done within this method. If the *Total Ticks to Wait* parameter is set to **MAXLONG** (the native 4th Dimension constant), then this method will wait indefinitely for the named semaphore to become available for setting herein.

The returned parameter *Semaphore Status* indicates the success or failure of this method to set the named semaphore. If *Semaphore Status* is zero (0) then the named semaphore was not set at all. If *Semaphore Status* is one (01) then the named semaphore was successfully set. If *Semaphore Status* is two (2) then the named semaphore was not set because it was already set from another location and did not become available within the time allowed to this method.

It is recommended that the method **SEM\_Set\_One** be used in place of the 4th Dimension **SEMAPHORE** command as future enhancements to the SEM module will provide more functionality within this method. In any case, the **SEM\_Set\_One** method provides more functionality than is available by the native **SEMAPHORE** command.



# STR Module

The STR module provides numerous methods for manipulating text and string values. The methods in the STR module comprise many basic and common operations which are performed on text values. As with many of the modules in the BASH component, this module will be continuously growing with future releases.

**Note:** the STR module was initially added in BASH v1.5.1.



## **STR\_Clean\_EmailAddress**

**STR\_Clean\_EmailAddress**( *Text Value* ) => *Cleansed Text Value*

**STR\_Clean\_EmailAddress**

(  
    -> *Text Value: Text*  
)  
    => *Cleansed Text Value: Text*

Parameter	Type	Description
<i>Text Value</i>	Text	Text value to clean
<i>Cleansed Text Value</i>	Text	Cleansed text value

The method **STR\_Clean\_EmailAddress** will removed all invalid byte values for a properly comprised email address.

Valid values for email addresses include: uppercase letters, lowercase letters, numbers, hyphen, period, underscore, and the at sign. All other byte values are consider invalid and will be removed with this method.

*Text Value* is the supplied text value to be cleansed.

*Cleansed Text Value* is *Text Value* with all invalid email address byte removed.

**Note:** the method *STR\_Clean\_EmailAddress* was added in BASH v1.5.1.



## **STR\_Concatenate\_Text**

**STR\_Concatenate\_Text**( *Referenced Concatenated Text*; *Overflow Option*{*Referenced Text Value*{ ...}}) => *Byte Count*

### **STR\_Concatenate\_Text**

(  
    -> *Referenced Concatenated Text*: **Pointer**  
    -> *Overflow Option*: **Longint**  
    {-> *Referenced Text Value*: **Pointer**}  
)  
    => *Byte Count*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Concatenated Text</i>	<b>Pointer</b>	Pointer to text variable to hold concatenated text
<i>Overflow Option</i>	<b>Longint</b>	Overflow option setting
<i>Referenced Text Value</i>	<b>Pointer</b>	One or more pointer to text values to concatenate
<i>Byte Count</i>	<b>Longint</b>	Number of bytes in resulting <i>Referenced Concatenated Text</i>

The method *STR\_Concatenate\_Text* concatenates multiple referenced text values while properly respecting overflow limitations. An overflow option setting determines whether concatenation is performed up to the overflow state in the event that concatenation would normally produce an overflow condition.

This method provides a fullproof means to concatenate text values and avoid possible overflow conditions. This is essential for many unattended systems which must run continuously in an error and dialog free state.

*Referenced Concatenated Text* is a pointer to a text variable which will contain the concatenated text values. Depending on the value of *Overflow Option* and potential

overflow conditions, *Referenced Concatenated Text* may actually be empty.

*Overflow Option* is an indicator setting for what this method will do in the even that the concatenation of the *Referenced Text Value* parameters' values would product an overflow condition. If *Overflow Option* is zero (0) then *Referenced Concatenated Text* will be empty in the event of a possible overflow. If *Overflow Option* is one (1) then *Referenced Concatenated Text* will be the first MAXTEXTLEN bytes of the concatenation of the *Referenced Text Value* parameters' values; the remaining overflow text will be truncated.

*Referenced Text Value* is between one (1) and fourteen (14) pointers to text variables which will be concatenated within this method. The referenced values are concatenated in the order of the parameter references.

*Byte Count* contains the total number of bytes within *Referenced Concatenated Text* upon returning from this method.

**Note:** the method *STR\_Concatenate\_Text* was added in BASH v1.5.1.



## **STR\_Count\_Occurrences\_of\_ASCII**

**STR\_Count\_Occurrences\_of\_ASCII**( *Text Value*; *ASCII Value*) =>  
*Occurrence Count*

**STR\_Count\_Occurrences\_of\_ASCII**  
(  
    -> *Text Value*: **Text**  
    -> *ASCII Value*: **Longint**  
)  
=> *Occurrence Count*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Text Value</i>	<b>Text</b>	Text value to scan
<i>ASCII Value</i>	<b>Longint</b>	ASCII value of bytes to count

*Occurrence Count*   **Longint**      **Count of occurrences of ASCII Value within Text Value**

The method ***STR\_Count\_Occurrences\_of\_ASCII*** will count the number of occurrences of a specified ASCII value within a block of text.

For single byte scanning, this method is much faster than the method ***STR\_Count\_Occurrences\_of\_String***.

*Text Value* is the text block to scan.

*ASCII Value* is the ASCII value to actually scan for within *Text Value*.

*Occurrence Count* is the total number of occurrences of the *ASCII Value* within the text block *Text Value*.

**Note:** the method ***STR\_Count\_Occurrences\_of\_ASCII*** was added in BASH v1.5.1.



## **STR\_Count\_Occurrences\_of\_String**

***STR\_Count\_Occurrences\_of\_String***( *Text Value*; *Match Text*) => *Occurrence Count*

```
STR_Count_Occurrences_of_String
(
    -> Text Value: Text
    -> Match Text: Text
)
=> Occurrence Count: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Text Value</i>	Text	Text value to scan
<i>Match Text</i>	Text	Text value to match for counting
<i>Occurrence Count</i>	Longint	Count of occurrences of <i>Match Text</i> within <i>Text Value</i>

The method ***STR\_Count\_Occurrences\_of\_String*** will count the number of occurrences of a specified match

text value within a block of text. The string matching is done using native 4D string comparison operators and comparators and is inclusive for multiple match values within and overlapping match strings.

For single byte scanning, the method ***STR\_Count\_Occurrences\_of\_ASCII*** is much faster for counting occurrences.

*Text Value* is the text block to scan.

*Match Text* is the actual match string to scan for within *Text Value*.

*Occurrence Count* is the total number of occurrences of the *Match Text* within the text block *Text Value*.

**Note:** the method ***STR\_Count\_Occurrences\_of\_String*** was added in BASH v1.5.1.



## **STR\_Get\_CharPosition\_by\_ASCII**

**STR\_Get\_CharPosition\_by\_ASCII**( *Text Value*; *ASCII Value*; *Start Byte*; *Direction Option*) => *Found Byte*

**STR\_Get\_CharPosition\_by\_ASCII**  
(  
    -> *Text Value*: **Text**  
    -> *ASCII Value*: **Longint**  
    -> *Start Byte*: **Longint**  
    -> *Direction Option*: **Longint**  
)  
=> *Found Byte*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Text Value</i>	<b>Text</b>	Text value to scan
<i>ASCII Value</i>	<b>Longint</b>	ASCII value of byte to scan for
<i>Start Byte</i>	<b>Longint</b>	Starting byte position within <i>Text Value</i>
<i>Direction Option</i>	<b>Longint</b>	Direction option for whether to scan forward or backwards within <i>Text Value</i>

*Found Byte*                      *Longint*                      *Byte position of first byte within Text Value matching scanning options*

The method *STR\_Get\_CharPosition\_by\_ASCII* returns the byte position of the first byte found within a specified text block matching a particular value. This byte can be scanned for either forward or backwards from any starting byte position within the specified text block.

*Text Value* is the text block to scan.

*ASCII Value* is the ASCII value to actually scan for within *Text Value*.

*Start Byte* is the starting byte position to begin the scanning from.

*Direction Option* is the looping parameter for which direction to scan for the matching byte value within *Text Value*. If *Direction Option* is one (1) then the scanning is done forward through *Text Value* starting at *Start Byte*. If *Direction Option* is negative one (-1) then the scanning is done backwards through *Text Value* starting at *Start Byte*.

*Found Byte* is the position of the first occurrence of *ASCII Value* found within *Text Value* when using the specified scanning options within this method. The following table summarizes result values for *Found Byte* after calling this method:

<u>Found Byte value</u>	<u>Indicates</u>
> 0	position of next match
0	no match found
- 1	text block to scan is empty
- 2	ASCII value invalid
- 3	start byte is too high (out of range)

**Note:** since *Direction Option* is actually the looping parameter within this method, scanning can be done on everything *nth* byte if the need arises. There are no limitations within this method to prevent different integral values, both positive and negative, from being

passed to this method. Positive values will scan forward and negative values will scan backwards.

**Note:** the method *STR\_Get\_CharPosition\_by\_ASCII* was added in BASH v1.5.1.



## **STR\_Get\_Line\_First**

**STR\_Get\_Line\_First**( *Text Value*) => *First Line Text*

**STR\_Get\_Line\_First**

```
(  
    -> Text Value: Text  
)  
=> First Line Text: Text
```

Parameter	Type	Description
<i>Text Value</i>	Text	Text block to scan
<i>First Line Text</i>	Text	Text of first line within <i>Text Value</i>

The method *STR\_Get\_Line\_First* will return the first line within a specified text block. Lines of text are considered to be delimited by carriage returns (ASCII value 13) within this method.

*Text Value* is the text block to scan.

*First Line Text* is the text of the first line within *Text Value*. If no carriage returns are found within *Text Value* then *First Line Text* is set to the complete *Text Value*.

**Note:** the method *STR\_Get\_Line\_First* was added in BASH v1.5.1.



## **STR\_Pad\_String**

**STR\_Pad\_String**( *Referenced Text Value*; *Preferred Length*; *ASCII Value*; *Padding Flag*)

**STR\_Pad\_String**

```
(
```

-> *Referenced Text Value*: **Pointer**  
 -> *Preferred Length*: **Longint**  
 -> *ASCII Value*: **Longint**  
 -> *Padding Flag*: **Boolean**  
 )

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Text Value</i>	<b>Pointer</b>	Pointer to text value to pad
<i>Preferred Length</i>	<b>Longint</b>	Preferred minimum length of <i>Referenced Text Value</i>
<i>ASCII Value</i>	<b>Longint</b>	ASCII value of bytes used for padding
<i>Padding Flag</i>	<b>Boolean</b>	Flag for whether padding goes after string or before

The method ***STR\_Pad\_String*** will pad out a referenced text value to always be a minimum length. This method is ideal for providing formatting columnar data.

*Referenced Text Value* is a pointer to the text value to pad to a minimum length.

*Preferred Length* is the minimum length required for *Referenced Text Value*. If the length of *Referenced Text Value* is less than *Preferred Length* then it will be padded to this minimum length. If the length of *Referenced Text Value* is greater than *Preferred Length* then no change will be made to *Referenced Text Value*.

*ASCII Value* is the ASCII value to use for any padding bytes which are added to *Referenced Text Value*.

*Padding Flag* is a boolean value for whether padding bytes added should be done after *Referenced Text Value*. If *Padding Flag* is not set then padding bytes will be added before *Referenced Text Value*, if length requirements deem it necessary..

**Note:** the method ***STR\_Pad\_String*** was added in BASH v1.5.1.



## **STR\_Parse\_to\_Array\_by\_ASCII**



***STR\_Parse\_to\_Array\_by\_ASCII***( *Referenced Text Array*; *Text Value*;  
*Delimiter ASCII Value*) => *Element Count*

***STR\_Parse\_to\_Array\_by\_ASCII***  
(  
    -> *Referenced Text Array*: **Pointer**  
    -> *Text Value*: **Text**  
    -> *Delimiter ASCII Value*: **Longint**  
)  
    => *Element Count*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Text Array</i>	<b>Pointer</b>	Pointer to text array to hold parsed values
<i>Text Value</i>	<b>Text</b>	Text block to parse
<i>Delimiter ASCII Value</i>	<b>Longint</b>	ASCII value of value delimiter within <i>Text Value</i>
<i>Element Count</i>	<b>Longint</b>	Number of elements parsed from <i>Text Value</i>

The method ***STR\_Parse\_to\_Array\_by\_ASCII*** will parse a text block into one or more elements. The elements can be separate within the text block by any ASCII value.

For single byte delimiters, this method is much faster than the method ***STR\_Parse\_to\_Array\_by\_Str***.

*Referenced Text Array* is a pointer to a text array that will hold the parsed values.

*Text Value* is the text block to parse.

*Delimiter ASCII Value* is the ASCII value of the value delimiter within *Text Value*.

*Element Count* is the number of elements parsed from *Text Value*. The following table summarizes possible values for the *Element Count* return value:

<u>Found Byte value</u>	<u>Indicates</u>
> 0	element count
- 1	referencing not to a text array
- 2	no reference parameter

Note: the method *STR\_Parse\_to\_Array\_by\_ASCII* was added in BASH v1.5.1.



## **STR\_Parse\_to\_Array\_by\_Str**

*STR\_Parse\_to\_Array\_by\_Str*( *Referenced Text Array*; *Text Value*; *Delimiter Text*) => *Element Count*

***STR\_Parse\_to\_Array\_by\_Str***

(  
    -> *Referenced Text Array*: **Pointer**  
    -> *Text Value*: **Text**  
    -> *Delimiter Text*: **Text**  
)  
    => *Element Count*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Text Array</i>	<b>Pointer</b>	Pointer to text array to hold parsed values
<i>Text Value</i>	<b>Text</b>	Text block to parse
<i>Delimiter Text</i>	<b>Text</b>	Text value of delimiter within <i>Text Value</i>
<i>Element Count</i>	<b>Longint</b>	Number of elements parsed from <i>Text Value</i>

The method *STR\_Parse\_to\_Array\_by\_Str* will parse a text block into one or more elements. The elements can be separate within the text block by any text value. Delimiter matches are determined by using native 4D string operators and comparators.

For single byte delimiters, the method *STR\_Parse\_to\_Array\_by\_ASCII* is much faster and efficient.

*Referenced Text Array* is a pointer to a text array that will hold the parsed values.

*Text Value* is the text block to parse.

*Delimiter Text* is the text value of the value delimiter within *Text Value*.

*Element Count* is the number of elements parsed from *Text Value*. The following table summarizes possible values for the *Element Count* return value:

<u>Found Byte value</u>	<u>Indicates</u>
> 0	element count
- 1	referencing not to a text array
- 2	no reference parameter
- 3	delimiter longer than text block

**Note:** the method *STR\_Parse\_to\_Array\_by\_Str* was added in BASH v1.5.1.



## **STR\_qi\_Match\_Filter\_NonCase**

**STR\_qi\_Match\_Filter\_NonCase**( *Source Text Value*; *Filter*; *Multiple Wilcard ASCII Value*; *Single Wilcard ASCII Value*) => *Match Successful*

**STR\_qi\_Match\_Filter\_NonCase**

```
(
-> Source Text Value: Text
-> Filter: Text
-> Multiple Wilcard ASCII Value: Longint
-> Single Wilcard ASCII Value: Longint
)
=> Match Successful: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Source Text Value</i>	Text	Text to compare with a particular filter and filter criteria
<i>Filter</i>	Text	Filter to compare against <i>Soruce Text Value</i>
<i>Multiple Wilcard ASCII Value</i>	Longint	ASCII value of multiple wildcard byte within <i>Filter</i>
<i>Single Wilcard ASCII Value</i>	Longint	ASCII value of single wildcard byte within <i>Filter</i>
<i>Match Successful</i>	Longint	qi for whether <i>Source Text Value</i> matches specified filter and filter criteria

The method *STR\_qi\_Match\_Filter\_NonCase* will return an indicator for whether a specified text block matches a specified filter string. The single and

multiple wildcard values are specified as parameters into this method.

Comparisons are done using standard 4D operators and comparators, though wildcard byte matches must be exact.

*Source Text Value* is the text value to compare for matching a specified filter and filter criteria.

*Filter* is the filter string to use for comparison against *Source Text Value*. *Filter* can contain zero, one, or more than one of both multiple and single wildcard values.

*Multiple Wildcard ASCII Value* is the ASCII value of the multiple wildcard value used within *Filter*.

*Single Wildcard ASCII Value* is the ASCII value of the single wildcard value used within *Filter*.

*Match Successful* is an indicator value for whether *Source Text Value* matches the specified filter and filter criteria.

**Note:** the method *STR\_qi\_Match\_Filter\_NonCase* was added in BASH v1.5.1.



## **STR\_Remove\_After\_Last\_by\_ASCII**

**STR\_Remove\_After\_Last\_by\_ASCII**( *Text Value*; *Delimiter ASCII Value*)  
=> *Truncated Text Value*

**STR\_Remove\_After\_Last\_by\_ASCII**  
(  
-> *Text Value*: **Text**  
-> *Delimiter ASCII Value*: **Longint**  
)  
=> *Truncated Text Value*: **Text**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Text Value</i>	<b>Text</b>	Text block to truncate

<i>Delimiter ASCII Value</i>	<b>Longint</b>	ASCII value to truncate after
<i>Truncated Text Value</i>	<b>Text</b>	Truncated text block

The method ***STR\_Remove\_After\_Last\_by\_ASCII*** will truncate a specified text block after the *last* occurrence of a specified byte value.

*Text Value* is the text block to be scanned and truncated.

*Delimiter ASCII Value* is the ASCII value of the byte which will be truncated after. The last occurrence of *Delimiter ASCII Value* within *Text Value* will be the last byte returned in *Truncated Text Value*.

*Truncated Text Value* is the resulting truncated text. If the *Delimiter ASCII Value* is invalid then the complete *Text Value* will be returned. If the *Delimiter ASCII Value* is not found within *Text Value* then the value returned in *Truncated Text Value* will be empty.

**Note:** the method ***STR\_Remove\_After\_Last\_by\_ASCII*** was added in BASH v1.5.1.



## **STR\_Remove\_Line\_First**

***STR\_Remove\_Line\_First***( *Text Value*) => *Truncated Text Value*

```
STR_Remove_Line_First
(
    -> Text Value: Text
)
=> Truncated Text Value: Text
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Text Value</i>	<b>Text</b>	Text block to scan
<i>Truncated Text Value</i>	<b>Text</b>	Text block with first line removed

The method ***STR\_Remove\_Line\_First*** will return a specified text block with the first line removed. Lines of text are considered to be delimited by carriage returns (ASCII value 13) within this method.

*Text Value* is the text block to scan.

*Truncated Text Value* is the text of *Text Value* with the first line removed. If there are no carriage returns with *Text Value* then *Truncated Text Value* will be empty.

**Note:** the method ***STR\_Remove\_Line\_First*** was added in BASH v1.5.1.



## **STR\_Remove\_NonAlphaNumeric**

***STR\_Remove\_NonAlphaNumeric***(*Text Value*) => *Filtered Text Value*

***STR\_Remove\_NonAlphaNumeric***

(  
    -> *Text Value*: **Text**  
)  
    => *Filtered Text Value*: **Text**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Text Value</i>	<b>Text</b>	Text value to filter
<i>Filtered Text Value</i>	<b>Text</b>	Filtered text value

The method ***STR\_Remove\_NonAlphaNumeric*** will removed all nonalphanumeric bytes from a text block.

Valid alphanumeric values include: lowercase letters, uppercase letters, and numbers.

*Text Value* is the supplied text value to be filtered.

*Filtered Text Value* is *Text Value* with all invalid bytes removed.

**Note:** the method ***STR\_Remove\_NonAlphaNumeric*** was added in BASH v1.5.1.



## **STR\_Remove\_Spaces\_Post**

**STR\_Remove\_Spaces\_Post**( *Text Value*) => *Trimmed Text Value*

**STR\_Remove\_Spaces\_Post**

```
(  
    -> Text Value: Text  
)  
=> Trimmed Text Value: Text
```

Parameter	Type	Description
<i>Text Value</i>	Text	Text value to trim
<i>Trimmed Text Value</i>	Text	Trimmed text value

The method **STR\_Remove\_Spaces\_Post** will removed all trailing spaces (ASCII 32) from a text block.

*Text Value* is the supplied text value to be trimmed.

*Trimmed Text Value* is *Text Value* with all trailing spaces removed.

**Note:** the method **STR\_Remove\_Spaces\_Post** was added in BASH v1.5.1.



## **STR\_Remove\_Spaces\_Pre**

**STR\_Remove\_Spaces\_Pro**( *Text Value*) => *Trimmed Text Value*

**STR\_Remove\_Spaces\_Pro**

```
(  
    -> Text Value: Text  
)  
=> Trimmed Text Value: Text
```

Parameter	Type	Description
<i>Text Value</i>	Text	Text value to trim
<i>Trimmed Text Value</i>	Text	Trimmed text value

The method *STR\_Remove\_Spaces\_Pre* will removed all leading spaces (ASCII 32) from a text block.

*Text Value* is the supplied text value to be trimmed.

*Trimmed Text Value* is *Text Value* with all leading spaces removed.

**Note:** the method *STR\_Remove\_Spaces\_Pre* was added in BASH v1.5.1.



## **STR\_Remove\_Spaces\_PrePost**

**STR\_Remove\_Spaces\_PrePost**( *Text Value*) => *Trimmed Text Value*

**STR\_Remove\_Spaces\_PrePost**

```
(  
    -> Text Value: Text  
)  
=> Trimmed Text Value: Text
```

Parameter	Type	Description
<i>Text Value</i>	Text	Text value to trim
<i>Trimmed Text Value</i>	Text	Trimmed text value

The method *STR\_Remove\_Spaces\_PrePost* will removed all leading and trailing spaces (ASCII 32) from a text block.

*Text Value* is the supplied text value to be trimmed.

*Trimmed Text Value* is *Text Value* with all leading and trailing spaces removed.

**Note:** the method *STR\_Remove\_Spaces\_PrePost* was added in BASH v1.5.1.



## **STR\_Replace\_ASCII\_All**



**STR\_Replace\_ASCII\_All**( *Text Value*; *Old ASCII Value*; *New ASCII Value*)  
=> *New Text Value*

**STR\_Replace\_ASCII\_All**  
(  
-> *Text Value*: **Text**  
-> *Old ASCII Value*: **Longint**  
-> *New ASCII Value*: **Longint**  
)  
=> *New Text Value*: **Text**

Parameter	Type	Description
<i>Text Value</i>	<b>Text</b>	Text value to scan
<i>Old ASCII Value</i>	<b>Longint</b>	ASCII value to be replaced
<i>New ASCII Value</i>	<b>Longint</b>	ASCII value to replace with
<i>New Text Value</i>	<b>Text</b>	Text value with replacement completed

The method **STR\_Replace\_ASCII\_All** will replace all occurrences of a specified ASCII value within a specified text block.

*Text Value* is the supplied text value to be scanned.

*Old ASCII Value* is the ASCII value within *Text Value* which is to be replaced.

*New ASCII Value* is the ASCII value within *Text Value* which will replace all occurrences of *Old ASCII Value*.

*New Text Value* is *Text Value* with all occurrences of *Old ASCII Value* replaced with *New ASCII Value*.

**Note:** the method **STR\_Replace\_ASCII\_All** was added in BASH v1.5.1.

---

## **STR\_Wrap\_in\_DoubleQuotes**

**STR\_Wrap\_in\_DoubelQuotes**( *Text Value*) => *Wrapped Text Value*

**STR\_Wrap\_in\_DoubelQuotes**  
(  
-> *Text Value*: **Text**

)  
=> *Wrapped Text Value*: Text

Parameter	Type	Description
<i>Text Value</i>	Text	Text block to wrap
<i>Wrapped Text Value</i>	Text	<i>Text Value</i> wrapped in double quotes

The method *STR\_Wrap\_in\_DoubleQuotes* will wrap any text value with double quotes.

*Text Value* is the text block to wrap in double quotes.

*Wrapped Text Value* is *Text Value* wrapped in double quotes.

**Note:** the method *STR\_Wrap\_in\_DoubleQuotes* was added in BASH v1.5.1.

# TIME Module

The TIME module comprises basic routines for dealing with time variables in 4th Dimension. The routines are fairly simplistic in this release of the BASH component. But, similar to the SEM module, additional functionality will be added to the TIME module in future releases and it is strongly suggested that these routines be used in place of traditional native implementations.

**Note:** the TIME module was initially added in BASH v1.4.7.



## TIME\_Add\_Normalize

*TIME\_Add\_Normalize(First Time Addend; Second Time Addend) =>  
Normalized Time Sum*

*TIME\_Add\_Normalize  
(  
-> First Time Addend: Time  
-> Second Time Addend: Time  
)  
=> Normalized Time Sum: Time*

Parameter	Type	Description
<i>First Time Addend</i>	Time	Time value to add
<i>Second Time Addend</i>	Time	Time value to add
<i>Normalized Time Sum</i>	Time	Sum of two supplied time values, normalized to be a valid time value within a 24 hour day

The method *TIME\_Add\_Normalize* returns the sum of two time supplied time values. The returned sum is in the format of a time value which has been normalized to be a valid time value.

*First Time Addend* and *Second Time Addend* are both supplied time values which are to be added together.

*Normalized Time Sum* is the sum of *First Time Addend* and *Second Time Addend*. *Normalized Time Sum* is normalized into a time value such; if the sum of *First Time Addend* and *Second Time Addend* would create a time value which is above the 24th hour, then 24 hours is removed from the sum to create a valid, normalized time value in *Normalized Time Sum*.

To determine whether *Normalized Time Sum* was normalized in the method *TIME\_Add\_Normalize*, call the method *TIME\_Get\_Sum\_Offset* afterwards.

### Example:

The following fragment of code exemplifies the use of the *TIME\_Add\_Normalize* and *TIME\_Get\_Sum\_Offset* methods. This fragment of code will add two time values together properly and determine whether a corresponding date value should be incremented to indicate that the sum of the two time values rolled over past midnight.

```
tTimeSum:= TIME_Add_Normalize (tTime1; tTime2)
If ( TIME_Get_Sum_Offset (tTime1; tTime2; tTimeSum)=1)
    `sum rolled over past midnight
Else
    `sum did not roll over past midnight
End if
```

**Note:** the method *TIME\_Add\_Normalize* was added in BASH v1.4.7.



## TIME\_Get\_Hours

*TIME\_Get\_Hours*(Time Value) => Number of Hours

```
TIME_Get_Hours
(
    -> Time Value: Time
)
=> Number of Hours: Longint
```

Parameter	Type	Description
<i>Time Value</i>	Time	Supplied time value
<i>Number of Hours</i>	Longint	Number of hours in supplied time value <i>Time Value</i>

The method ***TIME\_Get\_Hours*** returns the number of whole hours in a supplied time value.

*Time Value* is any given time value to extract the number of hours from.

*Number of Hours* is the number of whole hours within the supplied time value *Time Value*. No concessions are made to make *Number of Hours* correspond to a standard twelve (12) hour clock. Rather, the *Number of Hours* is merely a mathematical calculation performed on *Time Value*.

**Note:** the method ***TIME\_Get\_Hours*** was added in BASH v1.4.7.



## TIME\_Get\_Minutes

***TIME\_Get\_Minutes*** (*Time Value*) => *Number of Minutes*

```
TIME_Get_Minutes
(
    -> Time Value: Time
)
=> Number of Minutes: Longint
```

Parameter	Type	Description
<i>Time Value</i>	Time	Supplied time value
<i>Number of Minutes</i>	Longint	Number of minutes past hour in supplied time value <i>Time Value</i>

The method ***TIME\_Get\_Minutes*** returns the number of whole minutes past the last whole hour in a supplied time value.

*Time Value* is any given time value to extract the number of minutes past the last whole hour from.

*Number of Minutes* is the number of whole minutes past the last whole hour within the supplied time value *Time Value*. The *Number of Minutes* is merely retrieved from a mathematical calculation on the supplied time value *Time Value*.

**Note:** the method *TIME\_Get\_Minutes* was added in BASH v1.4.7.



## TIME\_Get\_Seconds

*TIME\_Get\_Seconds* ( *Time Value* ) => *Number of Seconds*

```
TIME_Get_Seconds
(
    -> Time Value: Time
)
=> Number of Seconds: Longint
```

Parameter	Type	Description
<i>Time Value</i>	<b>Time</b>	Supplied time value
<i>Number of Seconds</i>	<b>Longint</b>	Number of seconds past minute in supplied time value <i>Time Value</i>

The method *TIME\_Get\_Seconds* returns the number of whole seconds past the last whole minute in a supplied time value.

*Time Value* is any given time value to extract the number of seconds past the last whole minute from.

*Number of Seconds* is the number of whole seconds past the last whole minute within the supplied time value *Time Value*. The *Number of Seconds* is merely retrieved from a mathematical calculation on the supplied time value *Time Value*.

**Note:** the method *TIME\_Get\_Seconds* was added in BASH v1.4.7.



## TIME\_Get\_Sum\_Offset

*TIME\_Get\_Sum\_Offset*(Primary Time Value; Secondary Time Value;  
Tertiary Time Value) => Sum Cycle Status

*TIME\_Get\_Sum\_Offset*  
(  
    -> Primary Time Value: Time  
    -> Secondary Time Value: Time  
    -> Tertiary Time Value: Time  
)  
=> Sum Cycle Status: Longint

Parameter	Type	Description
Primary Time Value	Time	First time value to check against
Secondary Time Value	Time	Second time value to check against
Tertiary Time Value	Time	Third time value to check against
Sum Cycle Status	Longint	Indicator for whether Tertiary Time Value is less than either Primary Time Value or Secondary Time Value

The method *TIME\_Get\_Sum\_Offset* returns an indicator in for whether a particular time value is less than two other time values.

The *Primary Time Value* and *Secondary Time Value* must both be greater than *Tertiary Time Value* for *Sum Cycle Status* to be set to one (1). Otherwise, *Sum Cycle Status* will be set to zero (0).

The method *TIME\_Get\_Sum\_Offset* is meant to be used in conjunction with the method *TIME\_Add\_Normalize* to properly add two time values together and determine whether a day rollover occurred as a result of the summing.

**Example:**

The following fragment of code exemplifies the use of the *TIME\_Add\_Normalize* and *TIME\_Get\_Sum\_Offset* methods. This fragment of code will add two time values together properly and determine whether a corresponding date value should be incremented to indicate that the sum of the two time values rolled over past midnight.

```
tTimeSum:= TIME_Add_Normalize (tTime1; tTime2)
If ( TIME_Get_Sum_Offset (tTime1; tTime2; tTimeSum)=1)
    `sum rolled over past midnight
Else
    `sum did not roll over past midnight
End if
```

**Note:** the method *TIME\_Get\_Sum\_Offset* was added in BASH v1.4.7.



# TYPE Module

The TYPE module comprises basic routines for checking and confirming data types of different objects. The routines are fairly simplistic in this release of the BASH component. But, similar to the SEM module, additional functionality will be added to the TYPE module in future releases and it is strongly suggested that these routines be used in place of traditional native implementations.

## Unary and Array Classifications

It is worth knowing, also, that variable types in 4th Dimension have two different classifications: unary and array. Unary variables are capable of storing a single data value. Array variables are capable of storing zero, one, or more data values.

A complete listing of the unary variable types and their equivalent array data types follows:

<u>Unary</u>	<u>Array</u>
BLOB	n/a
Boolean	Boolean
Date	Date
Graph	n/a
Integer	Integer
Longint	Longint
Picture	Picture
Pointer	Pointer
Real	Real
String	String
Text	Text
Time	Longint



## **TYPE\_Compare\_Dereferenced\_Type**

**TYPE\_Compare\_Dereferenced\_Type** ( *Referenced Variable*; *Variable Type*) =>  
*Match State*

**TYPE\_Compare\_Dereferenced\_Type**

```
(
    -> Referenced Variable: Pointer
    -> Variable Type: Longint
)
=> Match State: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Variable</i>	<b>Pointer</b>	Reference to a variable which will have its type checked
<i>Variable Type</i>	<b>Longint</b>	Variable type to compare to (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )
<i>Match State</i>	<b>Longint</b>	State of type match check

The method *TYPE\_Compare\_Dereferenced\_Type* will compare a referenced variable type with a supplied variable type. It will return an indicator for whether the types match or an error condition.

The *Referenced Variable* parameter is a pointer to any variable whose type is going to be checked.

The *Variable Type* parameter is a variable type value to check against the *Referenced Variable* parameter. The list of valid type values are available as native constants in 4th Dimension under the *Field and Variable Types* grouping.

The *Match State* return parameter is the indicator for the match validity of the *Referenced Variable* parameter and the *Variable Type* parameter. Valid values for the *Match State* return parameter are as follows:

<u>Value</u>	<u>Meaning</u>
1	Types match
0	Types do not match
-1	<i>Referenced Variable</i> is Nil
-2	<i>Referenced Variable</i> is not a pointer



## **TYPE\_Get\_Array\_by\_Unary**

**TYPE\_Get\_Array\_by\_Unary** ( *Unary Type* ) => *Array Type*

## **TYPE\_Get\_Array\_by\_Unary**

```
(  
    -> Unary Type: Longint  
)  
=> Array Type: Longint
```

Parameter	Type	Description
Unary Type	Longint	Unary data type (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )
Array Type	Longint	Equivalent array data type (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )

The method **TYPE\_Get\_Array\_by\_Unary** will return the array equivalent data type for any specified unary data type. Variable data type values are designated by the 4D constants *Field* and *Variable Types*.

*Unary Type* is any unary variable type. Array variable types can be specified within *Unary Type* for convenience.

*Array Type* is the equivalent array data type for a value passed in *Unary Type*. If *Unary Type* is invalid, *Array Type* will be set to -1. If *Unary Type* has no equivalent array variable type, *Array Type* will be set to -2.

**Note:** the method **TYPE\_Get\_Array\_by\_Unary** was added in BASH v1.5.1.



## **TYPE\_Get\_Unary\_by\_Array**

**TYPE\_Get\_Unary\_by\_Array** (Array Type) => Unary Type

```
TYPE_Get_Unary_by_Array  
(  
    -> Array Type: Longint  
)  
=> Unary Type: Longint
```

Parameter	Type	Description
-----------	------	-------------

Array Type	Longint	Array data type (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )
Unary Type	Longint	Equivalent array data type (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )

The method ***TYPE\_Get\_Unary\_by\_Array*** will return the unary equivalent data type for any specified array data type. Variable data type values are designated by the 4D constants *Field* and *Variable Types*.

*Array Type* is any array variable type. Unary variable types can be specified within *Array Type* for convenience.

*Unary Type* is the equivalent unary data type for a value passed in *Array Type*. If *Array Type* is invalid, *Unary Type* will be set to -1.

**Note:** the method ***TYPE\_Get\_Unary\_by\_Array*** was added in BASH v1.5.1.



## **TYPE\_qi\_Array**

***TYPE\_qi\_Array*** ( Variable Type) => Array Match State

***TYPE\_qi\_Array***  
 (  
     -> Variable Type: Longint  
 )  
     => Array Match State: Longint

Parameter	Type	Description
Variable Type	Longint	Variable type to check (variable types are designated by the 4D constants <i>Field</i> and <i>Variable Types</i> )
Array Match State	Longint	State of type match check

The method ***TYPE\_qi\_Array*** will check whether a supplied variable type is any form of array variable.

The *Variable Type* parameter is any variable type value which is consistent with the native 4th Dimension variable type values found in the *Field and Variable Types* constants grouping. This variable type will be checked for whether it is some form of array, regardless of the exact type of array.

The returned *Array Match State* parameter is an indicator of whether the *Variable Type* value is an array type. If *Array Match State* is zero (0) then the supplied *Variable Type* is not an array. If *Array Match State* is one (1) then the supplied *Variable Type* is an array of some type.



## **TYPE\_qi\_BLOB**

**TYPE\_qi\_BLOB** ( *Variable Type*) => *BLOB Match State*

**TYPE\_qi\_BLOB**

```
(
    -> Variable Type: Longint
)
=> BLOB Match State: Longint
```

Parameter	Type	Description
<i>Variable Type</i>	Longint	asd
<i>BLOB Match State</i>	Longint	asd

The method **TYPE\_qi\_BLOB** will check whether a supplied variable type is a BLOB.

The *Variable Type* parameter is any variable type value which is consistent with the native 4th Dimension variable type values found in the *Field and Variable Types* constants grouping. This variable type will be checked for whether it is of type BLOB.

The returned *BLOB Match State* parameter is an indicator of whether the *Variable Type* value is BLOB type. If *BLOB Match State* is zero (0) then the supplied *Variable Type* is not a BLOB. If *BLOB Match State* is one (1) then the supplied *Variable Type* is a BLOB.



# VAR Module

The VAR module comprises basic routines for handling and manipulating variables directly. The routines are fairly simplistic in this release of the BASH component. But, similar to the SEM and TYPE modules, additional functionality will be added to the VAR module in future releases and it is strongly suggested that these routines be used in place of traditional native implementations.



## VAR\_Get\_Variable\_Name

**VAR\_Get\_Variable\_Name** (*Referenced Variable*) => *Variable Name*

**VAR\_Get\_Variable\_Name**  
(  
    -> *Referenced Variable*: **Pointer**  
)  
    => *Variable Name*: **Text**

Parameter	Type	Description
<i>Referenced Variable</i>	<b>Pointer</b>	Referenced variable to get the name of
<i>Variable Name</i>	<b>Text</b>	Actual name of <i>Referenced Variable</i>

The method **VAR\_Get\_Variable\_Name** will return the actual variable name for any referenced variable supplied in the *Referenced Variable* parameter. This method is practically equivalent to the native 4th Dimension command **RESOLVE POINTER**.



## VAR\_qi\_Null\_Pointer

**VAR\_qi\_Null\_Pointer** (*Reference Value*) => *nil Match State*

**VAR\_qi\_Null\_Pointer**  
(  
    -> *Reference Value*: **Pointer**  
)  
    => *nil Match State*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Reference Value</i>	<b>Pointer</b>	Any form of a referenced (pointer)
<i>nil Match State</i>	<b>Longint</b>	Indicator for whether <i>Reference Value</i> is a NULL pointer

The method *VAR\_qi\_Null\_Pointer* will check the *Reference Value* parameter and return an indicator for whether it is a NULL pointer. This method is practically equivalent to the native 4th Dimension command **Nil**. If the *Reference Value* parameter is a NULL pointer, the returned value *nil Match State* will be set to one (1); if the *Reference Value* parameter is not a NULL pointer, the returned value *nil Match State* will be set to zero (0).





# Version History

The following is a brief version history of the BASH component. It details release notes, bug fixes, and changes for each version publicly available.

It is worth noting that the version jumps of the BASH component are intentional. The code within the BASH module is actually stored in a master 4th Dimension application within Deep Sky Technologies, Inc. As core code within DSTi changes, it is changed in this master application and the version is incremented at specific intervals. The BASH component is kept synchronous with the master structure for ease of maintenance of version numbers.

In general, version numbers should not be an issue within your usage of the BASH component. And, in any case, the full history of different versions of the BASH component are fully detailed in this section.

# BASh v1.5.1

*released 20001127*

## Changes:

Added documentation for CLE encoding within the CODEC module. This new section is entitled **CLE Encoding Scheme**.

Added documentation for CONV value types which are not standard 4th Dimension data types. This includes Hex2, Hex8, Type, and Dotted IP values. These new section are entitled **Hex2 and Hex8 Values, Type Values, Dotted IP Values**.

Updated documentation within DSS module to explain the difference between unary and array variable classifications within 4th Dimension. Also, included explanation of mapping table for 4D data type and DSS variable types. This section was entitled **Variable Classifications**.

Added documentation sections within the RES module for the custom resource types. These documentation sections were left for future completion once the functionality for these custom resource types has been integrated into BASh and other DSTi components. These documentation sections are entitled **'fMap' Resource, 'LoCK' Resource, 'MENV' Resource, 'WAGr' Resource, 'WPGGr' Resource**.

Updated documentation within TYPE module to explain the difference between unary and array variable classifications within 4th Dimension. This section was entitled **Unary and Array Classifications**.

Added methods to the ARR module:

```
ARR_Convert_Text_to_Longint  
ARR_Convert_Type_to_Longint  
ARR_Pack_to_Text
```

Added the CODEC module to the BASh component. This included the following methods:

```
CODEC_Decode_Base64_x
```

```
CODEC_Decode_Base64_z  
CODEC_Decode_CLE_x  
CODEC_Decode_URL_x  
CODEC_Encode_Base64_x  
CODEC_Encode_Base64_z  
CODEC_Encode_CLE_x  
CODEC_Encode_URL_x
```

Added the CONV module to the BASH component. This included the following methods:

```
CONV_ASCII_to_Hex2  
CONV_ASCII_to_PrintableText  
CONV_Coerce_from_Text  
CONV_Coerce_to_Text  
CONV_ERROR  
CONV_Hex8_to_Longint  
CONV_Hex8_to_Text  
CONV_IP_to_Longint  
CONV_Longint_to_Hex8  
CONV_Longint_to_IP  
CONV_Longint_to_Type  
CONV_Text_to_Hex8  
CONV_Text_to_Longint  
CONV_Text_to_Real  
CONV_Type_to_Longint
```

Added methods to the DSS module:

```
DSS_Get_Array_by_Unary_Type  
DSS_Get_Unary_by_Array_Type
```

Added the ENV module to the BASH component. This included the following methods:

```
ENV_Get_4DApplication_FullPath  
ENV_Get_Application_Name  
ENV_Get_Application_Name_Short  
ENV_Get_Application_Type  
ENV_Get_DataFile_FullPath  
ENV_Get_DirectorySymbol  
ENV_Get_Platform  
ENV_Get_Structure_RF_FileName
```

ENV\_Get\_Structure\_RF\_FullPath  
ENV\_q\_Windows

Added the FILE module to the BASH component. This included the following methods:

FILE\_Convert\_to\_ResFork\_Windows  
FILE\_FullPath\_to\_FileName

Added the NVP module to the BASH component. This included the following methods:

NVP\_ERROR  
NVP\_Extract\_Values\_by\_Name\_s  
NVP\_Extract\_Values\_by\_Name\_x  
NVP\_Pack\_to\_Text  
NVP\_Parse\_to\_Arrays

Added the RES module to the BASH component. This included the following methods:

RES\_Close  
RES\_Create\_File  
RES\_Delete\_Resource  
RES\_ERROR  
RES\_Get\_Resource\_List  
RES\_Get\_TEXT\_Resource  
RES\_Load\_cicn  
RES\_Load\_fMap  
RES\_Load\_Loack\_LoCK  
RES\_Load\_MBAR  
RES\_Load\_MENU  
RES\_Load\_MENV  
RES\_Load\_PICT  
RES\_Load\_TMPL  
RES\_Load\_WAGr  
RES\_Load\_WPGr  
RES\_Make\_TMPL\_f\_Arrays  
RES\_Open  
RES\_Open\_4DApplication  
RES\_Open\_DateFile  
RES\_Open\_Structure  
RES\_Parse\_TMPL

RES\_Set\_Resource\_Name  
RES\_Set\_Resource\_Properties  
RES\_Set\_TEXT\_Resource

Added the STR module to the BASH component. This included the following methods:

STR\_Clean\_EmailAddress  
STR\_Concatenate\_Text  
STR\_Count\_Occurrences\_of\_ASCII  
STR\_Count\_Occurrences\_of\_String  
STR\_Get\_CharPosition\_by\_ASCII  
STR\_Get\_Line\_First  
STR\_Pad\_String  
STR\_Parse\_to\_Array\_by\_ASCII  
STR\_Parse\_to\_Array\_by\_Str  
STR\_qi\_Match\_Filter\_NonCase  
STR\_Remove\_After\_Last\_by\_ASCII  
STR\_Remove\_Line\_First  
STR\_Remove\_NonAlphaNumeric  
STR\_Remove\_Spaces\_Post  
STR\_Remove\_Spaces\_Pre  
STR\_Remove\_Spaces\_PrePost  
STR\_Replace\_ASCII\_All  
STR\_Wrap\_in\_DoubleQuotes

Added methods to the TYPE module:

TYPE\_Get\_Array\_by\_Unary  
TYPE\_Get\_Unary\_by\_Array

# BASh v1.4.7

*released 20001029*

## Changes:

Added the TIME module to the BASh component. This included the following methods:

- TIME\_Add\_Normalize
- TIME\_Get\_Hours
- TIME\_Get\_Minutes
- TIME\_Get\_Seconds
- TIME\_Get\_Sum\_Offset

Added the DTS modules to the BASh component. This included the following methods:

- DTS\_Add
- DTS\_Get\_Current
- DTS\_Get\_Date
- DTS\_Get\_Date\_Time
- DTS\_Get\_Day
- DTS\_Get\_Maximum
- DTS\_Get\_Month
- DTS\_Get\_Range
- DTS\_Get\_Time
- DTS\_Get\_Year
- DTS\_Make\_from\_DateTime
- DTS\_Make\_from\_Values

Added the method NULL\_Get\_DTS to the NULL module.

Fixed a bug in the DSS module wherein the getting and returning of variables was not indexed correctly. This was introduced with v1.4.6 when the indexing went from 2 digits to 3 digits to support the increased number of variables of each type.

# BASh v1.4.6

*released 20001025*

## Changes:

Expanded the number of supported variables of each type within the DSS module from 50 to 100.

Provided a mapping mechanism within the DSS module to handle string and alpha variable types as text and string array variable types as text arrays.

Completed the documentation of all of the protected and public methods which are available within the BASh component.



# BASh v1.4.1

*released 20001001*

## Changes:

Included limited PDF documentation for the BASh component.

Added some functionality to the SEM module, particularly for greater control in the ***SEM\_Set\_One*** method.

Fixed a bug in the initialization of the different modules. The inclusion of auto-initialization within v1.4.0 caused a small bug which could lead to duplicate initialization calls being made concurrently. This is now fixed in v1.4.1.

## BASh v1.4.0

*released 20000905*

BASh v1.4.0 was released mainly for inclusion on the 4D Summit 2000 CD. It accompanied the notes and materials for two (2) different classes: *If Memory Serves Me Right...* and *Styles Vary*. Notes for both of these classes are available on the Deep Sky Technologies, Inc., web site (<http://www.deepskytech.com/>) and the 4D Zine web site (<http://www.4dzine.com/>).

## BASh pre-v1.4.0

*never released publicly*

Prior to version 1.4.0 of the BASh component, the code which comprised the BASh component was used internally at Deep Sky Technologies, Inc., in all of our projects. It forms the basis for much of the code work and application development which is done at DSTi in 4th Dimension. The code which is in the BASh component has been in use in one form or another at DSTi for at least four (4) years, since the beginning of 1997.

Over the years, the DSS module, contained in the BASh component, has become the single most used module of code within DSTi. No other module of code has been as useful or widely used in all of the projects done at DSTi. The functionality it provides makes variable management and variable reuse simpler than anything else currently available for use when developing in 4th Dimension.



## Using BASH

asd



# BASh Error Codes

asd





## Index

asd