BASh

Developer Documentation

v1.8.2

©1998-2003 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
772.794.9494

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 it's 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, it's 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, TCP Deux, SMTP Deux, POP3 Client Deux, FTP Client Deux, HTTP Client Deux, eTrans, TCP Server Deux, and HTTP Server Deux are 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.

4D Pack plugin provided courtesy, and with permission, of 4D, Inc.

Macintosh and MacOS are trademarks of Apple Computer, Inc.

Windows is a trademark of Microsoft Corporation.

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

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.

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 significantly 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 filled 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 A. 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, Ken Ishimoto, Tim Klein, and Tom Lundeen.

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

Features

The BASh component consists of many different modules of methods which provide extensive utility services in 4th Dimension applications. The list of modules currently available in the BASh component is:

ARR - Array sizing, management, and manipulation methods;
BLOB - BLOB management and manipulation methods;
CODEC - encoding and decoding support for common formats;
CONV - conversion methods to handle data and type conversions;
CRYPT - encryption and decryption routines;
DATE - provides basic date manipulation routines;
DSS - Dynamic Stack Space; reusable variable management system;
DTS - Date-Time Stamps; generation and manipulation methods;
ENV - environmental information (program, OS, CPU, etc.);
FILE - path and file name utilities;
FMAP - file mapping for document types, create and type codes, and MIME types;
IB - indexed BLOB management routines;
INIT - module and component initialization;
INT - interruption manager;
NULL - Variable clearing and initialization methods;
NVP - named value pair utilities and management;
PROS - process information accessors;
PTEXT - parameter text replacement routines;
QUIT - controlled clean up of BASh data structures;
RES - resource management and utility methods;
RW - read/write module for handling record access;
SEM - Semaphore management methods;
SEQ – Sequence number generation and management methods;
SERNO - serial number generation and confirmation methods;
STR - string manipulation, formatting, and filtering methods;
TIME - Time manipulation methods;
TYPE - Variable type comparison methods;
URL - URL creation and extraction routines;
VAR - Variable utility methods;
WORD - manipulators for non-native four-byte values;
XML - Extensible Markup Language parsing and retrieval code.

Items in **Bold** were just added in the latest release of the BASh component.
Items in Underline have had important updates in BASh v1.8.2, the latest release.

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 6.7.0 of 4th Dimension or above, including 4D Insider v6.7.0 or above for installation.

BASh is compatible with all versions of 4D, including the 4D v6.7.x, v6.8.x, and v7.0.x (2003) product lines.

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 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 you 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.

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

Open the uncompiled structure that you wish to install BASh into 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.

Place a copy of the Affix BASH document in the 4DX folder.

The Affix BASh document contains essential data and resources for many of the methods within the BASh component. For many of the methods within the BASh component to function properly, the Affix BASh document must be in the 4DX folder for the current structure.

On Macintosh, the Affix BASh document is entitled **Affix_BASh.4DX** (under 4D v6.8.x and 4D v7.0.x [4D 2003], there is now a carbonized version entitled **Affix_BASh.4CX**) and is located in the Mac4DX directory in the BASh component's archive. The document should be copied into the Mac4DX folder of your current structure. If the BASh component is going to be used in all of your 4D projects, the Affix BASh document can instead be placed within the Mac4DX folder within the 4D folder of your system.

On Windows, the Affix BASh document is actually two documents: **Affix_BASh.4DX** and **Affix_BASh.RSR**. These two document correspond to the data fork and the resource fork of the Affix BASh document used on Macintosh. These documents are located in the Win4DX directory in the BASh component's archive. These documents should be copied into the Win4DX folder of your current structure. If the BASh component is going to be used in all of your 4D projects, the Affix BASh document can instead be placed within the Win4DX folder within the 4D folder of your system.

For client/server installations in cross-platform environments, both the Macintosh and Windows versions of the Affix BASh document should be installed.

Place a copy of the 4D Pack plugin in the 4DX folder.

The 4D Pack plugin contains essential commands and functionality used within the BASh component. For many of the methods within the BASh component to function properly, the 4D Pack plugin must be in the 4DX folder for the current structure.

On Macintosh, the 4D Pack plugin included with the BASh component download is entitled **4D_Pack_v671.4DX** (for 4D v6.8.x, the plugin is entitled **4D_Pack_v682.4CX**; for 4D v7.0.x [4D 2003], the plugin is entitled **4D_Pack_v700mc01.4CX**) and is located in the Mac4DX directory in the BASh component's archive. The plugin should be copied into the Mac4DX folder of your current structure. If the BASh component is going to be used in all of your 4D projects, the plugin can instead be placed within the Mac4DX folder within the 4D folder of your system.

On Windows, the 4D Pack plugin included with the BASh component is actually two documents: **4D_Pack_v671.4DX** and **4D_Pack_v671.RSR** (for 4D v6.8.x, the plugin documents are entitled **4D_Pack_v681.4DX** and **4D_Pack_v681.RSR**, respectively). These two document correspond to the data fork and the resource fork of the 4D Pack plugin used on Macintosh. These documents are located in the Win4DX directory in the BASh component's archive. These documents should be copied into the Win4DX folder of your current structure. If the BASh component is going to be used in all of your 4D projects, the 4D Pack plugin documents can instead be placed

within the Win4DX folder within the 4D folder of your system.

For client/server installations in cross-platform environments, both the Macintosh and Windows versions of the 4D Pack plugin should be installed.

Call the method *INIT_BASh* early in the On Startup and On Server Startup database methods.

To initialize the BASh component in your code, place a call to the method ***INIT_BASh*** early in your **On Startup** database method.

A call to ***INIT_BASh*** must also be placed early in the **On Server Startup** database method.

Details about the INIT module and the ***INIT_BASh*** method can be found in the module and method documentation, below.

Call the method *QUIT_BASh* late in the On Exit and On Server Shutdown database methods.

To properly clean up after the BASh component in your code, place a call to the method ***QUIT_BASh*** late in your **On Exit** and **On Server Shutdown** database methods.

Details about the **QUIT** module and the ***QUIT_BASh*** method can be found in the module and method documentation, below.

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.

Affix BASh Document

The Affix BASh document contains essential data and resources for many of the methods within the BASh component. For many of the methods within the BASh component to function properly, the Affix BASh document must be in the 4DX folder for the current structure. For distributed and installed versions of your 4D projects, the Affix BASh document must be available in the 4DX folder for the BASh methods to continue to function properly.

Note: it is best to consider the Affix BASh document another plug-in within your 4D project. Though there no actual plug-in calls available within the Affix BASh document, it does contain data and resources essential to the operation of the BASh methods. The BASh component has been designed to find the Affix BASh document correctly in all environments (any platform, any 4DX folder, single user or clients/server, etc.). Since the Affix BASh document is configured similarly to plug-ins, 4D and the BASh component will automatically manage the document for you in all of the possible installations of a 4D project.

4D Pack Plugin

The 4D Pack plugin contains essential calls for many of the methods within the BASh component. For many of the methods within the BASh component to function properly, the 4D Pack plugin must be in the 4DX folder for the current structure. For distributed and installed versions of your 4D projects, the 4D Pack plugin must be available in the 4DX folder for the BASh methods to continue to function properly.

The 4D Pack plugin is provided unmodiied directly from 4D, Inc., and 4D SA. Copies of the plugin are included with the BASh component merely for convenience.

4D v6.8.x

With the availability of 4D v6.8.x, the complete 4th Dimension environment is now fully carbonized. 4D as a carbonized application allows for a single set of tools to function on either MacOS X or MacOS v9.x using CarbonLib.

With the release of 4D v6.8.x, there is now a new plugin architecture available for third party developers. As well, there are some changes which have been made in the actually plugin hierarchy and naming conventions. There have also been changes made to the component architecture within the 4D product line. Reading the release notes for 4D v6.8.x is a great source of information regarding these changes.

BASh is currently available with compatibility for 4D v6.8.x. This comes in the form of new affix and plugin documents for use with 4D v6.8.x. The BASh archive contains both 4D v6.7.x and 4D v6.8.x compatible versions of the component, affix documents, and plugins.

When updating an existing database from 4D v6.7.x to 4D v6.8.x, we have found that installed components will no longer update properly. The first time that a component is updated after upgrading a 4D structure, the component must first be removed from the structure before installed the new version of the component. We have not seen any problems with doing this other than the extra step required to remove the component using 4D Insider.

It is also important that you install the correct affix and plugin documents for your environment. Whether you are running under MacOS X or MacOS 9 is not a factor. Rather, whether you are using 4D v6.7.x or 4D v6.8.x is the determining factor. Copy the appropriate documents for your current version of 4D from the component archive for use in your 4D structure. The differences between the 4D v6.7.x and 4D v6.8.x affix and plugin documents is simple to determine; the names of the documents compatible with 4D v6.7.x end in ".4DX" and the names of the documents compatible with 4D v6.8.x end in ".4CX".

4D v7.0.x (4D v2003)

BASh is currently available with compatibility for 4D v7.0.x. This comes in the form of new affix and plugin documents for use with 4D v7.0.x. The BASh archive contains 4D v6.7.x, 4D v6.8.x, and 4D v7.0.x compatible versions of the component, affix documents, and plugins.

It is also important that you install the correct affix and plugin documents for your environment. Whether you are running under MacOS X or MacOS 9 is not a factor. Rather, whether you are using 4D v6.7.x, 4D v6.8.x, or 4D v7.0.x is the determining factor. Copy the appropriate documents for your current version of 4D from the component archive for use in your 4D structure.

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.

Remove the Affix_BASh document and 4D Pack plugin from your 4DX folder.

Remove the call to the method *INIT_BASh* from your On Startup and On Server Startup database methods.

Remove the call to the method *QUIT_BASh* from your On Exit and On Server Shutdown database methods.

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...

Updating to BASh v1.8.2

Updating to the latest release of the BASh component is a simple procedure. Follow the instructions contained in the section **“Installing BASh” on page 10** to update the code within the structure.

In particular, for BASh v1.8.2, make certain that the latest release of the Affix BASh document is placed in the 4DX folder.

All previous copies of the 4D Pack plugin should be removed from the 4DX folders. Included in the archive with BASh v1.8.2 are copies of 4D Pack for 4D v6.7.x, 4D v6.8.x and 4D v7.0.x (4D 2003) for both Macintosh and Windows. 4D Pack should be placed in the platform appropriate 4DX folder next to your structure.

Also, make certain that calls to ***INIT_BASh*** are placed in both the **On Startup** and **On Server Startup** database methods. It is now important that these calls are placed early in both of these database methods.

The **QUIT** module cleans up after BASh when your application/structure is closing. Properly cleaning up after the BASh component is now very important to the continued error free operation of the functionality managed by BASh. To support this, calls to ***QUIT_BASh*** must be made late in the **On Exit** and **On Server Shutdown** database methods.

Constants

There are a minimal number of custom constants included with the BASh component package. These constants are grouped into a few convenient constant groups for easier referencing and organization.

Where appropriate, it is highly recommended that the custom constants included with the BASh component be utitlized within your code; this will considerably simplify future feature enhancements to the core code within BASh.

TYPE, Array 2D, Variable Types

The **TYPE, Array 2D, Variable Types** constants group contains one constant for each of the supported 2 dimensional variable types available within 4D. Each of the 2D array types has a separate constant within this constants group. With the support of 2D array types throughout many areas of BASh v1.8.0 and above, the use of the constants in this constants group complements the single dimensional variable type constants already available within the native 4D environment.

The following is a listing of all of the constants in this constants group:

Constant	Value
TYPE_A2D_Real	114
TYPE_A2D_Integer	115
TYPE_A2D_Longint	116
TYPE_A2D_Date	117
TYPE_A2D_Text	118
TYPE_A2D_Picture	119
TYPE_A2D_Pointer	120
TYPE_A2D_String	121
TYPE_A2D_Boolean	122

It is worth pointing out that each of the constants in this constants group has a value of 100. This makes it very easy to distinguish type values for 2 dimensional variable from type values for all other variable

types. As well, notice the value of all constants in this constants group has a value of exactly 100 more than the equivalent one dimensional array type (e.g. Text Array, a native 4D constant, has a type value of 18; TYPE_A2D_Text, a constant in this constant group, has a type value of 118).

Note: the **TYPE, Array 2D, Variable Types** constant group was added to BASh in v1.8.0.

URL Schemes

The URL Schemes constants group contains one constant for each of the supported URL schemes supported within the URL modules of the BASh component package. The following is a listing of the constants, and their values, within the URL Schemes constant group:

Constant	Value
URL_Scheme_FTP	1
URL_Scheme_HTTP	2
URL_Scheme_Gopher	3
URL_Scheme_MailTo	4
URL_Scheme_News	5
URL_Scheme_NNTP	6
URL_Scheme_Telnet	7
URL_Scheme_File	8
URL_Scheme_HTTPS	9

These constants are used only when initializing the URL module of the BASh component. See the section of this manual dealing with the URL module for more details on using the constants within the URL Schemes constants group.

Code 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
	Referenced Array	Pointer	Reference to a single dimensional array of any type
	Elements to Add	Longint	Positive number of elements to add to the end of *Referenced Array*

The method ***ARR_Add_Elements_to_End*** will add the specified number of elements to the end of the referenced array. Each element which is added will be initialised to a **NULL** value.

Referenced Array is a pointer to a single dimensional array of any type.

Elements to Add is a positive number of elements to add to the end of *Referenced Array*.

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
	Referenced Array	Pointer	Reference to a single dimensional array of any type
	Elements to Add	Longint	Positive number of elements to add to the top of *Referenced Array*

The method ***ARR_Add_Elements_to_Top*** will add the specified number of elements to the top of the referenced array. Each element which is added will be initialised to a **NULL** value.

Referenced Array is a pointer to a single dimensional array of any type.

Elements to Add is a positive number of elements to add to the top of *Referenced Array*.

ARR_Clear

ARR_Clear *(Referenced Array)*

ARR_Clear

(

-> *Referenced Array :* Pointer

)

	Parameter	Type	Description
	Referenced Array	Pointer	Reference to a single dimensional array of any type

The method ***ARR_Clear*** will set the size of the array to zero (0). This will be done regardless of the type of array.

Referenced Array is a pointer to a single dimensional array of any type.

ARR_Coerce_from_Text

ARR_Coerce_from_Text *(Referenced Source Text Array ; Referenced Target Array)*

ARR_Coerce_from_Text
(
-> *Referenced Source Text Array :* Pointer
-> *Referenced Target Array :* Pointer
)

	Parameter	Type	Description
	Referenced Source Text Array	Pointer	Referenced text array to coerce data from
	Referenced Target Array	Pointer	Referenced array to coerce data into

The method ***ARR_Coerce_from_Text*** will coerce a series of values stored in a refererenced text array into a separate referenced array of most any type. The target array is resized to contain the same number of elements as the source array, all elements including the zeroth (0th) element are coerced into the target array, and the array index is copied from the source to the target array.

Referenced Source Text Array is a pointer to the source text array to coerce values from.

Referenced Target Array is a pointer to the destination array which is to be resized and have its values coerced from text. Array types supported include string, text, boolean, date, integer, longint, real, and time.

Note: the method ***ARR_Coerce_from_Text*** was added in BASh v1.5.5.

ARR_Coerce_to_Text

ARR_Coerce_to_Text *(Referenced Source Array ; Referenced Target Text Array)*

ARR_Coerce_to_Text

(

-> *Referenced Source Array :* Pointer
-> *Referenced Target Text Array :* Pointer

)

	Parameter	Type	Description
	Referenced Source Array	Pointer	Referenced array to coerce data from
	Referenced Target Text Array	Pointer	Referenced text array to coerce data into

The method ***ARR_Coerce_to_Text*** will coerce a series of values stored in a refererenced array of most any type into a separate referenced text array. The target text array is resized to contain the same number of elements as the source array, all elements including the zeroth (0th) element are coerced into the target array, and the array index is copied from the source to the target text array.

Referenced Source Array is a pointer to the source array to coerce values from. Array types supported include string, text, boolean, date, integer, longint, real, and time.

Referenced Target Array is a pointer to the destination text array which is to be resized and have its values coerced to text.

Note: the method ***ARR_Coerce_to_Text*** was added in BASh v1.5.5.

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
	Referenced Longint Array	Pointer	Reference to a longint array holding values to convert
	Referenced Text Array	Pointer	Reference to text array which will hold textual equivalent of longint values from *Referenced Longint Array*.

The method ***ARR_Convert_Longint_to_Text*** will convert all of the values in a longint array to their textual equivalents and store them in a text array. The size of the text array will be set to be the same as the 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.

Referenced Longint Array is a reference to a longint array holding values to be converted.

Referenced Text Array is a reference to a text array to hold the converted values from *Referenced Longint Array*.

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
	Referenced Text Array	Pointer	Reference to a text or string array containing values to convert

	Parameter	Type	Description
	Referenced Longint Array	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
	Referenced Types Array	Pointer	Reference to a text or string array containing type values to convert
	Referenced Longint Array	Pointer	Referenced to a longint array to contain converted 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; Begining Element; Ending Element; Cycle Count)*

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

	Parameter	Type	Description
	Referenced Array	Pointer	Referenced array to cycle elements within
	Beginning Element	Longint	Element number to begin cycle at
	Ending Element	Longint	Element number to end cycle at
	Cycle Count	Longint	Number of elements to cycle

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
)

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***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_Find_First_Binary

ARR_Find_First_Binary *(Referenced Array ; Referenced Value) => Found Index*

ARR_Find_First_Binary
(
-> *Referenced Array :* Pointer
-> *Referenced Value :* Pointer
)
=> *Found Index :* Longint

	Parameter	Type	Description
	Referenced Array	Pointer	Pointer to an array which is going to be searched using a binary search algorithm
	Referenced Value	Pointer	Value to search for
	Found Index	Longint	Index of first element found matching specified value

The method ***ARR_Find_First_Binary*** runs a binary search on a referenced array for a referenced value and returns the index of the first element matching. Of course, since the search performed is a binary search, the values within the specified referenced array must be sorted; this method assumes that the values in the specified referenced array are already sorted.

Referenced Array is a pointer to an array which is already sorted and is to be searched.

Referenced Value is a pointer to a value which is to be searched for within *Referenced Array*.

Found Index is the index of the first element within *Referenced Array* that matches *Referenced Value*.

Note: the method ***ARR_Find_First_Binary*** was added in BASh v1.5.9.

ARR_Find_in_Range

ARR_Find_in_Range *(Referenced Array ; Referenced Search Value ; Start Index ; End Index) => Found Index*

ARR_Find_in_Range
(
-> *Referenced Array :* Pointer
-> *Referenced Search Value :* Pointer
-> *Start Index :* Longint
-> *End Index :* Longint
)
=> *Found Index :* Longint

	Parameter	Type	Description
	Referenced Array	Pointer	Pointer to an array which is going to be sequentially scanned
	Referenced Search Value	Pointer	Pointer to value to be searched for in array
	Start Index	Longint	Index to begin scan at
	End Index	Longint	Index to end scan at
	Found Index	Longint	Index of first element within specified range matching specified value

The method ***ARR_Find_in_Range*** will scan a range of indices within a referenced array of values for the first matching value. The index of the first matching value will be returned.

Referenced Array is a pointer to an array to be scanned through.

Referenced Search Value is a pointer to a value of the same type as the elements in *Referenced Array* . The value of *Referenced Search Value* will be used to find the first element within the specified range of *Referenced Array* which contains a matching value.

Start Index is the first index within *Referenced Array* which is to be searched. To begin at the start of *Referenced Array*, set *Start Index* to one (1). To also search the zeroth (e.g. myarray {0}) element of *Referenced Array*, pass a value of zero (0) for *Start Index*.

End Index is the last index within *Referenced Array* which is to be scanned for matching values. To search through the last element in *Referenced Array*, regardless of the the number of elements in the array, set *End Index* to MAXLONG.

Found Index is the first element within the specified range of *Start Index* and *End Index* within *Referenced Array* matching the value of *Referenced Search Value*. If no matching elements are found, *Found Index* will be set to negative one (-1).

Note: the method ***ARR_Find_in_Range*** was added in BASh v1.6.0

ARR_Find_Last_Same

ARR_Find_Last_Same *(Referenced Array ; Start Index) => Last Match Index*

ARR_Find_Last_Same
(
-> *Referenced Array :* Pointer
-> *Start Index :* Longint
)
=> *Last Match Index :* Longint

	Parameter	Type	Description
	Referenced Array	Pointer	Pointer to an array which is going to be sequentially scanned
	Start Index	Pointer	Index to begin scan at

	Parameter	Type	Description
	Last Match Index	Longint	Index of last element sequentially found matching *Start Index* value

The method ***ARR_Find_Last_Same*** will sequentially scan a specified array starting at a specified starting index and return the first following element which has a value different from the value of the specified starting index.

Referenced Array is a pointer to an array which is to be sequentially scanned.

Start Index is the index which is to be the beginning of the sequential element scan of array values.

Last Match Index is the index of the first element following *Start Index* within *Referenced Array* which has the same value of that within *Start Index*.

Note: the method ***ARR_Find_Last_Same*** was added in BASh v1.5.9.

ARR_Insert_Elements_f_Array

ARR_Insert_Elements_f_Array *(Referenced Target Array ; Referenced Source Array ; Target Insert Index ; Source Copy First Index ; Source Copy Last Index) => ErrorCode*

ARR_Insert_Elements_f_Array

(

-> Referenced Target Array : Pointer
-> Referenced Source Array : Pointer
-> Target Insert Index : Longint
-> Source Copy First Index : Longint
-> Source Copy Last Index : Longint

)

=> ErrorCode : Longint

	Parameter	Type	Description
	Referenced Target Array	Pointer	Pointer to array to insert values into
	Referenced Source Array	Pointer	Pointer to array to copy values from

	Parameter	Type	Description
	Target Insert Index	Longint	Index into *Referenced Target Array* to insert new values after
	Source Copy First Index	Longint	Index into *Referenced Source Array* to begin copying values from (inclusive)
	Source Copy Last Index	Longint	Index into *Referenced Source Array* to stop copying values from (inclusive)
	Error Code	Longint	Error code from running of method

The method ***ARR_Insert_Elements_f_Array*** will copy a specified range of elements for a specified referenced source array into a specified referenced target array, placing the elements in their existing order after a specified element in the target array.

Referenced Target Array is a pointer to an array to have values copied into. New elements will be added to this array to accommodate the new elements being copied from the source array. The type of this array must match the type of the source array.

Referenced Source Array is a pointer to an array to copy elements from. The range of elements specified will be copied in order from this array and insert into the target array. The type of this array must match the type of the target array.

Target Insert Index is a longint containing the index into the target array to insert new elements after. Setting this parameter to zero (0) will always prepend the elements in the target array. Setting this parameter to MAXLONG will always append the new elements to the target array.

Source Copy First Index is a longint containing the index in the source array to begin copying elements from. Setting this index value to anything less than one (1) will be assumed in the routine to indicate the first element in the source array.

Source Copy Last Index is a longint containing the index in the source array to end copying elements from (inclusive). This value should always be greater than the first index parameter value. Setting this parameter to MAXLONG will copy the elements up to the end of the source array.

Error Code is a longint error code from calling this method. This return value will be set to zero (0) if no error occurred or will be set to a negative value in the event of an error.

Note: the method ***ARR_Insert_Elements_f_Array*** was added in BASh v1.8.2.

ARR_Insert_Element_Binary

ARR_Insert_Element_Binary *(Referenced Array ; Referenced Value) => New Element Index*

ARR_Insert_Element_Binary
(
-> *Referenced Array :* Pointer
-> *Referenced Value :* Pointer
)
=> *New Element Index :* Longint

	Parameter	Type	Description
	Referenced Array	Pointer	Pointer to sorted array to insert new value into
	Referenced Value	Pointer	Pointer to new value to insert into sorted array
	New Element Index	Longint	Index of new element containing value inserted into array

The method ***ARR_Insert_Element_Binary*** will use a binary search algorithm to quickly insert a specified referenced value into a specified referenced, ascending sorted array. The index of the new element inserted will be returned. After the insertion of the new value into the array, the values in the array are still all in sorted order (hence the use of the binary search for insertion).

Referenced Array is a pointer to an array to insert the new value into. This array must be sorted in ascending order as the algorithm to insert the new value will use a binary search to preserve the sorted order of the array. The array type must match the type of the value to be inserted.

Referenced Value is a pointer to a value to insert into *Referenced Array*. The type of this value must match the type of the array.

New Element Index is a longint containing the index of the new element containing the value insert into the array.

Note: the method ***ARR_Insert_Element_Binary*** was added in BASh v1.8.2.

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 *}*

)

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

he 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)* => *Packed Text*

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

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

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 }
)

	Parameter	Type	Description
	Referenced Text Array	Pointer	Text array to be filled by specified text values
	Fill Text	Text	Optional number of text values to fill into *Referenced Text Array*

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_Duplicates

ARR_Remove_Duplicates *(Referenced Sorted Array)* => *Removed Elements Count*

ARR_Remove_Duplicates
(
-> *Referenced Sorted Array :* Pointer
)
=> *Removed Elements Count :* Longint

	Parameter	Type	Description
	Referenced Sorted Array	Pointer	Pointer to array of sorted values to have duplicates removed from

	Parameter	Type	Description
	Removed Elements Count	Longint	Number of elements removed from referenced array

The method ***ARR_Remove_Duplicates*** will remove all duplicate values from a referenced array of sorted values. The number of elements removed will be returned from calling this function.

Referenced Sorted Array is a pointer to an array of sorted values. All duplicate values in the sorted array will be removed. The checking and removal algorithm relies on the elements in the array being sorted and unexpected results may occur in the event that the elements are not sorted.

Removed Elements Count is the number of elements removed from *Referenced Sorted Array* by calling this routine.

Note: the method ***ARR_Remove_Duplicates*** was added in BASh v1.7.0.

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
	Referenced Array	Pointer	Reference to a single dimensional array of any type
	Elements to Remove	Longint	Positive number of elements to remove from the end of *Referenced Array*

The method ***ARR_Remove_Elements_from_End*** will remove the specified number of elements from the end of the referenced array.

Referenced Array is a pointer to a single dimensional array of any type.

Elements to Remove is a positive number of elements to remove from the end of *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
	Referenced Array	Pointer	Reference to a single dimensional array of any type
	Elements to Remove	Longint	Positive number of elements to remove from the top of *Referenced Array*

The method ***ARR_Remove_Elements_from_Top*** will remove the specified number of elements from the beginning of the referenced array.

Referenced Array is a pointer to a single dimensional array of any type.

Elements to Remove is a positive number of elements to remove from the beginning of *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
)

	Parameter	Type	Description
	Referenced Array	Pointer	Reference to a single dimensional array of any type which will be resized
	New Maximum Element Number	Longint	New maximum element number which *Referenced Array* 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.

ARR_Set_Sizes

ARR_Set_Sizes *(New Maximum Element Number ; Referenced Array { ; Referenced Array { ; ... } })*

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

	Parameter	Type	Description
	New Maximum Element Number	Longint	New maximum element number which *Referenced Arrays* should be resized to
	Referenced Array	Pointer	References to one or more single dimensional arrays of any type which will be resized

The method ***ARR_Set_Sizes*** will resized one or more arrays *Referenced Array* to have a maximum element number of *New Maximum Element Number*. If new elements are added to the *Referenced Array* arrays, they will be added to the end of the arrays and will have all elements initialized to NULL.

This method is functionally equivalent to ***ARR_Set_Size*** except that it provides for the ability to resize multiple arrays, all to the same size, with a single method call.

Note: the method ***ARR_Set_Sizes*** was added in BASh v1.5.4.

ARR_Set_Values

ARR_Set_Values *(Referenced Array ; Starting Element ; Ending Element ; Referenced Value)*

ARR_Set_Values
(
-> *Referenced Array :* Pointer
-> *Starting Element :* Longint
-> *Ending Element :* Longint
-> *Referenced Value :* Pointer
)

	Parameter	Type	Description
	Referenced Array	Pointer	Pointer to array to set values into
	Starting Element	Longint	Index of first element in array to set value for
	Ending Element	Longint	Index of last element in array to set value for
	Referenced Value	Pointer	Pointer to value to set into *Referenced Array*

The method ***ARR_Set_Values*** will set a range of element in a referenced array to a specified referenced value. The elements must already exist in the referenced array and any previous values in the range of elements specified will be replaced with the specified referenced value.

Referenced Array is a pointer to an array which will have the value of elements set.

Starting Element is the index of the first element in *Referenced Array* to have values replaced. Any value less than zero (0) will generate an error.

Ending Element is the index of the first element in *Referenced Array* to have values replaced. Setting this parameter to MAXLONG will set the value of all elements to the end of *Referenced Array* to the specified value. A value for this parameter less than *Starting Element* will generate an error.

Note: the method ***ARR_Set_Values*** was added in BASh v1.8.0.

BLOB Module

The BLOB module provides basic operations on BLOBs. With the current release of the BASh component, the BLOB module basically just provides ancillary functionality which is used in other modules. In future versions of the BASh component, the BLOB module will be expanded considerably.

BLOB_Append_BLOB

BLOB_Append_BLOB *(Referenced Destination BLOB ; Referenced Source BLOB)*

BLOB_Append_BLOB
(
-> *Referenced Destination BLOB :* Pointer
-> *Referenced Source BLOB :* Pointer
)

	Parameter	Type	Description
	Referenced Destination BLOB	Pointer	Pointer to BLOB to be appended to
	Referenced Source BLOB	Pointer	Pointer to BLOB to append to *Referenced Destination BLOB*

The method ***BLOB_Append_BLOB*** will append the value of a referenced BLOB to another referenced BLOB. The complete source BLOB is appended to the end of the destination BLOB.

Referenced Destination BLOB is a pointer to the destination BLOB to append to.

Referenced Source BLOB is a pointer to a source BLOB to append the contents of to *Referenced Destination BLOB*.

Note: the method ***BLOB_Append_BLOB*** was added in BASh v1.8.0.

BLOB_Append_Text

BLOB_Append_Text *(Referenced BLOB ; Text to Append)*

BLOB_Append_Text

(

-> *Referenced BLOB :* Pointer
-> *Text to Append :* Text

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to append to
	Text to Append	Text	Text value to append to *Referenced BLOB*

The method ***BLOB_Append_Text*** will append a specified text value to a referenced BLOB. The text is appended directly to the end of the BLOB without adding any variable indicator or length values.

Referenced BLOB is a pointer to a BLOB to append text to.

Text to Append is the text value to append to *Referenced BLOB*.

Note: the method ***BLOB_Append_Textwas*** added in BASh v1.5.3.

BLOB_BLOB_to_Document

BLOB_BLOB_to_Document *(Referenced BLOB ; Full Document Path) => Error Status*

BLOB_BLOB_to_Document

(

-> *Referenced BLOB :* Pointer
-> *Full Document Path :* Text

)

=> *Error Status :* Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Reference to BLOB containing data fork of document

	Parameter	Type	Description
	Full Document Path	Text	Full path to document to contain *Referenced BLOB* as the data fork
	Error Status	Longint	qi for document data fork written successfully

The method ***BLOB_BLOB_to_Document*** will write referenced data fork contents to a document specified by a full path. If the document specified, and any of the enclosing directories in the path to the document do not exist, this method will create them.

Referenced BLOB is a pointer to a BLOB containing the data fork to be written into the document.

Full Document Path is the full path to the document to be written.

Error Status is the indicator for whether this routine successfully created the full path the document, if needed, created the document, if needed, and wrote the contents of *Referenced BLOB* into the data fork of the document specified by *Full Document Path*. *Error Status* will be zero (0) if the operation was successful; *Error Status* will be one (1) if the operation failed at any point.

Note: the method ***BLOB_BLOB_to_Document*** was added in BASh v1.5.4.

BLOB_Change_Case

BLOB_Change_Case *(Referenced BLOB ; Starting Offset ; Bytes to Scan ; Case Flag)*

BLOB_Change_Case
(
-> *Referenced BLOB :* Pointer
-> *Starting Offset :* Longint
-> *Bytes to Scan :* Longint
-> *Case Flag :* Longint
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to change the case of contents
	Starting Offset	Longint	Offset of first byte to possibly change case of
	Bytes to Scan	Longint	Number of bytes following *Starting Offset* to possibly change case of
	Case Flag	Longint	Flag to indicate whether to first bytes to uppercase or lowercase

The method ***BLOB_Change_Case*** will scan a specified byte range within a specified BLOB and force bytes to either uppercase or lowercase.

Referenced BLOB is a pointer to the BLOB to use to scan the contents of for changing case.

Starting Offset is the starting offset within *Referenced BLOB* to begin scanning bytes to check if the case should be changed. This value should be set to zero (0) to begin scanning *Referenced BLOB* from the beginning.

Bytes to Scan is the number of bytes within *Referenced BLOB* to scan. This value should be set to MAXLONG to scan through to the end of *Referenced BLOB* .

Case Flag is a flag to indicate whether case of bytes should be changed to uppercase or lowercase. Set this value to one (1) to change all bytes to uppercase; set this value to two (2) to change all bytes to lowercase.

Note: the method ***BLOB_Change_Case*** was added in BASh v1.5.9.

BLOB_Clear

BLOB_Clear *(Referenced BLOB)*

BLOB_Clear
(

-> *Referenced BLOB :* Pointer
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to clear

The method ***BLOB_Clear*** will clear completely the contents of a referenced BLOB. This method is functionally equivalent to calling the native 4D command **SET BLOB SIZE** with a size value of zero (0).

Referenced BLOB is a pointer to the BLOB to clear.

Note: the method ***BLOB_Clear*** was added in BASh v1.5.3.

BLOB_Count_Occurrences_of_ASCII

BLOB_Count_Occurrences_of_ASCII *(Referenced BLOB ; Starting Offset ; Ending Offset ; ASCII Value ; Maximum Occurrences) => Occurrences Found*

BLOB_Count_Occurrences_of_ASCII
(
-> *Referenced BLOB :* Pointer
-> *Starting Offset :* Longint
-> *Ending Offset :* Longint
-> *ASCII Value :* Longint
-> *Maximum Occurrences :* Longint
)
=> *Occurrences Found :* Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to count
	Starting Offset	Longint	Beginning offset within *Referenced BLOB* to start counting
	Ending Offset	Longint	Ending offset within *Referenced BLOB* to end counting
	ASCII Value	Longint	ASCII value of byte to count
	Maximum Occurrences	Longint	Maximum number of occurrences to count within range of *Referenced BLOB*

	Parameter	Type	Description
	Occurrences Found	Longint	Count of number of occurrences of *ASCII Value* found within specified range of *Referenced BLOB*

The method ***BLOB_Count_Occurrences_of_ASCII*** will count the number of occurrences of a specified byte value within a specified range of a referenced BLOB.

The range is inclusive, meaning that the specified low and high values will be searched for as well as all values between them.

Referenced BLOB is a pointer to a BLOB to search through.

Starting Offset is the offset within *Referenced BLOB* to begin searching for a byte with the specified *ASCII value*. Pass a value of zero (0) to begin searching at the first byte within *Referenced BLOB*.

Ending Offset is the offset within *Referenced BLOB* to end searching for a byte with the specified *ASCII value*. Pass a value of MAXLONG to search to the end of *Referenced BLOB*.

ASCII Value is the value of the byte to count within *Referenced BLOB*.

Maximum Occurrences is the maximum number of occurrences to count within *Referenced BLOB*. When the maximum number of occurrences of *ASCII Value* have been counted, this method will no longer count any further within *Referenced BLOB*. Pass a value of MAXLONG to count all of the occurrences of *ASCII Value* within the specified range of the *Referenced BLOB*.

Occurrences Found is the total number of occurrences of the specified byte value found within the specified range of *Referenced BLOB*.

Note: the method ***BLOB_Count_Occurrences_of_ASCII*** was added in BASh v1.6.1.

BLOB_Document_to_BLOB

BLOB_Document_to_BLOB *(Referenced BLOB ; Full Document Path) => Error Status*

BLOB_Document_to_BLOB
(
-> *Referenced BLOB :* Pointer
-> *Full Document Path :* Text
)
=> *Error Status :* Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Reference to BLOB to contain data fork of document specified
	Full Document Path	Text	Full path to document to read data fork into *Referenced BLOB*
	Error Status	Longint	qi for document data fork read successfully

The method ***BLOB_Document_to_BLOB*** will read data fork contents of a document specified by a full path into a referenced BLOB variable.

Referenced BLOB is a pointer to a BLOB to contain the data fork of a document read.

Full Document Path is the full path to the document to be read.

Error Status is the indicator for whether this routine successfully read the data fork of the document specified by *Full Document Path* into *Referenced BLOB*. *Error Status* will be zero (0) if the operation was successful; *Error Status* will be one (1) if the operation failed at any point.

Note: the method ***BLOB_Document_to_BLOB*** was added in BASh v1.5.4.

BLOB_Find_Between_Folding_x

BLOB_Find_Between_Folding_x *(Referenced BLOB ; Starting Offset ; Referenced Ending Offset ; Begin String ; End String ; Folding Characters) => Found Text*

BLOB_Find_Between_Folding_x

(

-> Referenced BLOB : Pointer
-> Starting Offset : Longint
-> Referenced Ending Offset : Pointer
-> Begin String : Text
-> End String String : Text
-> Folding Characters : Text

)

=> Found Text : Text

	Parameter	Type	Description
	Referenced BLOB	Pointer	Reference to a BLOB contents to search
	Starting Offset	Longint	Starting byte offset within *Referenced BLOB* to begin search
	Referenced Ending Offset	Pointer	Referenced longint to contain the byte offset immediately following the found value
	Begin String	Text	Text to find to mark the beginning of the found value
	End String	Text	Text to find to mark the ending of the found value
	Folding Characters	Text	Text to delineate a folded content
	Found Text	Longint	Return text found in *Referenced BLOB* after *Starting Offset* and between *Begin String* and *End String* , inclusively, with any folded values matching delineated by *Folding Characters*

The method ***BLOB_Find_Between_Folding_x*** will return the text found after a specified byte offset which falls after a specified beginning string, up to a specified end string, within a specified BLOB value, inclusively. Content which is valid folding content will be included in the resulting found text.

The search within *Referenced BLOB* will begin with the *Starting Offset* byte value. *Referenced BLOB* will then be

scanned until *Begin String* is found. Once *Begin String* is found following *Starting Offset* in *Referenced BLOB*, the contents after *Begin String* will be returned, up to but not including *Ending String* (or the end of *Referenced BLOB* if *Ending String* is not found).

If immediately following the found *End String* instance is a match for *Folding Characters*, thereby indicating a folded line, then *Referenced BLOB* is searched further for the next occurrence of *End String* to mark the end of the found content. The found content will be returned up to, but not including, the final instance of *End String* which does not proceed a match for *Folding Characters*, or the end of *Referenced BLOB*.

The search is done sequentially and is not case sensitive.

The byte offset immediately following the first non-folding *Ending String* for the found value will be returned in the referenced parameter *Referenced Ending Offset*.

Referenced BLOB is a reference to a BLOB value to be searched through.

Starting Offset is the first byte offset within *Referenced BLOB* which is to be searched sequentially for *Begin String*.

Referenced Ending Offset is a reference to a longint to hold the byte offset value immediately following *End String* following *Found Text* extracted from *Referenced BLOB*.

Begin String is the string of characters to search for within *Referenced BLOB* beginning at *Starting Offset* to delineate the beginning of the found value.

End String is the string of characters to be used as the ending delimeter for the found value. If *End String* is not found following *Begin String* or all found *End String* values are followed immediately by *Folding Characters* , the resulting found text will include the whole contents up to the end of *Referenced BLOB* (barring overflow conditions).

Folding Characters is the value to check for immediately following each instance of *End String* to indicate a folding line of content. Folded lines are considered valid contents of

Found Text and are included as such with all returned values for this method.

Found Text is the found text value matching all of the search criteria provided and detailed, above.

Note: the method ***BLOB_Find_Between_Folding_x*** was added in BASh v1.5.4.

BLOB_Find_Between_x

BLOB_Find_Between_x *(Referenced BLOB ; Starting Offset ; Referenced Ending Offset ; Begin String ; End String { ; Bytes to Search }) => Found Text*

BLOB_Find_Between_x
(
-> *Referenced BLOB :* Pointer
-> *Starting Offset :* Longint
-> *Referenced Ending Offset :* Pointer
-> *Begin String :* Text
-> *End String :* Text
{ -> *Bytes to Search :* Longint }
)
=> *Found Text :* Text

	Parameter	Type	Description
	Referenced BLOB	Pointer	Reference to a BLOB contents to search
	Starting Offset	Longint	Starting byte offset within *Referenced BLOB* to begin search
	Referenced Ending Offset	Pointer	Referenced longint to contain the byte offset immediately following the found value
	Begin String	Text	Text to find to mark the beginning of the found value
	End String	Text	Text to find to mark the ending of the found value

	Parameter	Type	Description
	Found Text	Longint	Return text found in *Referenced BLOB* after *Starting Offset* and between *Begin String* and *End String* , inclusively
	Bytes to Search	Longint	Number of bytes from *Starting Offset* to search

The method ***BLOB_Find_Between_x*** will return the text found after a specified byte offset which falls after a specified beginning string, up to a specified end string, within a specified BLOB value.

The search within *Referenced BLOB* will begin with the *Starting Offset* byte value. *Referenced BLOB* will then be scanned until *Begin String* is found. Once *Begin String* is found following *Starting Offset* in *Referenced BLOB*, the contents after *Begin String* will be returned, up to but not including *Ending String* that falls before the end optional *Bytes to Search* value (or the end of *Referenced BLOB*, or up to the *Bytes to Search* value if specified, if *Ending String* is not found). For searches in which the *Bytes to Search* value is specified and the search is a backwards search (*Bytes to Search* is negative), the search for *Begin String* will be run backwards through *Referenced BLOB* until found and then *Ending String* will be searched for forwards through *Referenced BLOB* up to *Starting Offset*.

The search is done sequentially and is not case sensitive.

The byte offset immediately following *Ending String* for the found value will be returned in the referenced parameter *Referenced Ending Offset*.

Referenced BLOB is a reference to a BLOB value to be searched through.

Starting Offset is the first byte offset within *Referenced BLOB* which is to be searched sequentially for *Begin String*. For searches which are being run backwards (see the optional *Bytes to Search* parameter, below), setting *Starting Offset* to MAXLONG will begin the search at the end of *Referenced BLOB*.

Referenced Ending Offset is a reference to a longint to hold the byte offset value immediately following the *Ending String* following *Found Text* extracted from *Referenced BLOB*.

Begin String is the string of characters to search for within *Referenced BLOB* beginning at *Starting Offset* to delineate the beginning of the found value.

End String is the string of characters to be used as the ending delimeter for the found value. If *End String* is not found following *Begin String* , the resulting found text will include the whole contents up to the end of *Referenced BLOB* (barring overflow conditions).

Found Text is the found text value matching all of the search criteria provided, above.

Bytes to Search is an optional longint parameter indicating the number of bytes to perform search within *Referenced BLOB*. Setting *Bytes to Search* to MAXLONG will perform the search to the end of *Referenced BLOB*. A negative value for *Bytes to Search* will do a backwards search (right to left, bottom to top, et al) through *Referenced BLOB* for *Begin String*. Setting *Bytes to Search* to negative MAXLONG will search to the beginning of *Referenced BLOB*.

Note: the method ***BLOB_Find_Between_x*** was added in BASh v1.5.4.

Note: the method ***BLOB_Find_Between_x*** was updated in BASh v1.7.0 to accept the optional *Bytes to Search* parameter.

BLOB_Find_Text_Case

BLOB_Find_Text_Case *(Search Text ; Referenced BLOB ; Beginning Offset ; Search Range Byte Count) => Found Offset*

BLOB_Find_Text_Case
(

-> *Search Text :* Text
-> *Referenced BLOB :* Pointer
-> *Beginning Offset :* Longint

-> *Search Range Byte Count :* Longint

)

=> *Found Offset :* Longint

	Parameter	Type	Description
	Search Text	Text	Text value to search for
	Referenced BLOB	Pointer	Pointer to BLOB to search for *Search Text* within
	Beginning Offset	Longint	Offset to byte within *Referenced BLOB* to begin search for *Search Text*
	Search Range Byte Count	Longint	Total number of bytes within *Referenced BLOB* in which the search should be performed
	Found Offset	Longint	Byte offset within *Referenced BLOB* after *Beginning Offset* in which *Search Text* is first found

The method ***BLOB_Find_Text_Case*** will perform a sequential, case sensitive search for a specifed text value within a referenced BLOB value, within a range of bytes, and return the offset of the first occurrence found.

The search is done sequentially from the specified beginning byte offset. It will continue sequential up to, but not past, the number of bytes specified.

Search Text is the text value to search for within *Referenced BLOB*.

Referenced BLOB is a pointer to a BLOB value which will be searched.

Beginning Offset is the byte offset within *Referenced BLOB* to begin the search for *Search Text*. If the search is to done from the beginning of *Referenced BLOB*, pass zero (0) as the value for *Beginning Offset*.

Search Range Byte Count is the number of bytes past *Beginning Offset* within *Referenced BLOB* which the value *Search Text* should be searched for. The search will terminate after *Search Range Byte Count* if no match is found. To search until the end of *Referenced BLOB*, pass MaxLong as the value for *Search Range Byte Count*.

Found Offset is the byte offset within *Referenced BLOB* which *Search Text* is first found after *Beginning Offset* and before *Search Range Byte Offset* bytes. If no match is found within *Referenced BLOB* within the specified range, *Found Offset* will be -1.

Note: the method ***BLOB_Find_Text_Case*** was added in BASh v1.5.7.

BLOB_Find_Byte

BLOB_Find_Byte *(ASCII Value to Find ; Referenced BLOB ; Starting Offset ; Byte Count to Search) => Found Offset*

BLOB_Find_Byte
(
-> *ASCII Value to Find :* Longint
-> *Referenced BLOB :* Pointer
-> *Starting Offset :* Longint
-> *Byte Count to Search :* Longint
)
=> *Found Offset :* Longint

	Parameter	Type	Description
	ASCII Value to Find	Longint	ASCII value to search for
	Referenced BLOB	Pointer	Pointer to BLOB to scan
	Starting Offset	Longint	Beginning offset within *Referenced BLOB* to start scanning
	Byte Count to Search	Longint	Number of bytes after *Starting Offset* within *Referenced BLOB* to scan
	Found Offset	Longint	Offset of first byte within range matching *ASCII Value* to Find

The method ***BLOB_Find_Byte*** will find a particular byte value within a specified referenced BLOB within a specified byte range within the BLOB and will return the offset of the found byte.

ASCII Value to Find is the ASCII value of the byte to scan for.

Referenced BLOB is a pointer to a BLOB to search through.

Starting Offset is the offset within *Referenced BLOB* to begin scanning for *ASCII Value to Find*. Pass a value of zero (0) to begin search at the first byte within *Referenced BLOB*.

Byte Count to Search is the number of bytes following *Starting Offset* within *Referenced BLOB* which will be scanned for a matching *ASCII Value to Find*. If the complete length of *Referenced BLOB* is to be searched then pass a value of MAXLONG for this parameter.

Found Offset is the offset within *Referenced BLOB* of the first byte within the offset range specified that match *ASCII Value to Find*. If no matching byte was found within the specified offset range, *Found Offset* will be set to negative one (-1).

Note: the method ***BLOB_Find_Byte*** was added in BASh v1.5.8.

BLOB_Find_Byte_by_ASCII_Range

BLOB_Find_Byte_by_ASCII_Range *(Referenced BLOB ; Starting Offset ; Byte Count to Search ; ASCII Low Value ; ASCII High Value) => Found Offset*

BLOB_Find_Byte_by_ASCII_Range
(
-> Referenced BLOB : Pointer
-> Starting Offset : Longint
-> Byte Count to Search : Longint
-> ASCII Low Value : Longint
-> ASCII High Value : Longint
)
=> Found Offset : Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to scan
	Starting Offset	Longint	Beginning offset within *Referenced BLOB* to start scanning

	Parameter	Type	Description
	Byte Count to Search	Longint	Number of bytes after *Starting Offset* within *Referenced BLOB* to scan
	ASCII Low Value	Longint	Low end of ASCII range to search for
	ASCII Low Value	Longint	High end of ASCII range to search for
	Found Offset	Longint	Offset of first byte within the specified ASCII value range

The method ***BLOB_Find_Byte_by_ASCII_Range*** will find the first byte within the specified range of ASCII values in the specified byte range of the referenced BLOB and will return the offset of the first found byte.

The range is inclusive, meaning that the specified low and high values will be searched for as well as all values between them.

Referenced BLOB is a pointer to a BLOB to search through.

Starting Offset is the offset within *Referenced BLOB* to begin searching for a byte within the specified ASCII value range. Pass a value of zero (0) to begin searching at the first byte within *Referenced BLOB*, or a value of MAXLONG to begin searching at the end of the BLOB.

Byte Count to Search is the number of bytes following *Starting Offset* within *Referenced BLOB* which will be searched. Pass a positive value to search forward, or a negative value to search backward. Pass MAXLONG to search to the end of the blob, or negative MAXLONG (-MAXLONG) to search to the beginning of the BLOB.

ASCII Low Value is the low end of the ASCII range to search for.

ASCII High Value is the high end of the ASCII range to search for.

Found Offset is the offset within *Referenced BLOB* of the first byte within the offset range specified within the specified ASCII value range. If no matching byte was found within the specified offset range, *Found Offset* will be set to negative one (-1).

Note: the method ***BLOB_Find_Byte_by_ASCII_Range*** was added in BASh v1.6.1.

BLOB_Find_Text_NonCase

BLOB_Find_Text_NonCase *(Search Text ; Referenced BLOB ; Beginning Offset ; Search Range Byte Count) => Found Offset*

BLOB_Find_Text_NonCase
(
-> *Search Text :* Text
-> *Referenced BLOB :* Pointer
-> *Beginning Offset :* Longint
-> *Search Range Byte Count :* Longint
)
=> *Found Offset :* Longint

	Parameter	Type	Description
	Search Text	Text	Text value to search for
	Referenced BLOB	Pointer	Pointer to BLOB to search for *Search Text* within
	Beginning Offset	Longint	Offset to byte within *Referenced BLOB* to begin search for *Search Text*
	Search Range Byte Count	Longint	Total number of bytes within *Referenced BLOB* in which the search should be performed
	Found Offset	Longint	Byte offset within *Referenced BLOB* after *Beginning Offset* in which *Search Text* is first found

The method ***BLOB_Find_Text_NonCase*** will perform a sequential, non-case sensitive search for a specifed text value within a referenced BLOB value, within a range of bytes, and return the offset of the first occurrence found.

The search is done sequentially from the specified beginning byte offset. It will continue sequential up to, but not past, the number of bytes specified.

Search Text is the text value to search for within *Referenced BLOB*.

Referenced BLOB is a pointer to a BLOB value which will be searched.

Beginning Offset is the byte offset within *Referenced BLOB* to begin the search for *Search Text*. If the search is to done from the beginning of *Referenced BLOB*, pass zero (0) as the value for *Beginning Offset*.

Search Range Byte Count is the number of bytes past *Beginning Offset* within *Referenced BLOB* which the value *Search Text* should be searched for. The search will terminate after *Search Range Byte Count* if no match is found. To search until the end of *Referenced BLOB*, pass MaxLong as the value for *Search Range Byte Count*.

Found Offset is the byte offset within *Referenced BLOB* which *Search Text* is first found after *Beginning Offset* and before *Search Range Byte Offset* bytes. If no match is found within *Referenced BLOB* within the specified range, *Found Offset* will be -1.

Note: the method ***BLOB_Find_Text_NonCase*** was added in BASh v1.5.7.

BLOB_Finds_Between_Folding_x

BLOB_Finds_Between_Folding_x *(Referenced BLOB ; Begin String ; End String ; Folding Characters) => Found Text*

BLOB_Finds_Between_Folding_x

(

-> Referenced BLOB : Pointer
-> Begin String : Text
-> End String String : Text
-> Folding Characters : Text

)

=> Found Text : Text

	Parameter	Type	Description
	Referenced BLOB	Pointer	Reference to a BLOB contents to search

	Parameter	Type	Description
	Begin String	Text	Text to find to mark the beginning of the found value
	End String	Text	Text to find to mark the ending of the found value
	Folding Characters	Text	Text to delineate a folded content
	Found Text	Text	Return text found in Referenced BLOB between *Begin String* and *End String*, inclusively, with any folded values matching delineated by *Folding Characters*

The method ***BLOB_Finds_Between_Folding_x*** will return the concatenated all text found which falls between a specified beginning string and ending string, within a specified BLOB value, inclusively. Content which is valid folding content will be included in the resulting found text.

This method operates exactly the same as the method ***BLOB_Find_Between_Folding_x***, detailed above, except that it is done across the complete contents of *Referenced BLOB* and all content, including folded lines, which is between *Begin String* and *End String* is returned. This includes multiple occurrences of content which may match the delimeter and folding conditions.

The search is done sequentially and is not case sensitive.

Referenced BLOB is a reference to a BLOB value to be searched through.

Begin String is the string of characters to mark the beginning of content within *Referenced BLOB*.

End String is the string of characters to be used as the ending delimeter for content within *Referenced BLOB*.

Folding Characters is the value to check for immediately following each instance of *End String* to indicate a folding line of content. Folded lines are considered valid contents of *Found Text* and are included as such with all returned values for this method.

Found Text is the found text value matching all of the search criteria provided and detailed, above.

Note: the method ***BLOB_Finds_Between_Folding_x*** was added in BASh v1.5.4.

BLOB_Parse_to_Arrays_x

BLOB_Parse_to_Arrays_x *(Referenced BLOB ; Column Delimiter ; Row Delimiter { ; Referenced Destination Array { ; ... } }) => Row Count*

BLOB_Parse_to_Arrays_x
(
-> Referenced BLOB : Pointer
-> Column Delimiter : Text
-> Row Delimiter : Text
-> Referenced Destination Array : Pointer
{ -> ... : Pointer *}*
)
=> Row Count : Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing values to parse
	Column Delimiter	Text	Case sensitive text of column delimiter
	Row Delimiter	Text	Case sensitive text of row delimiter
	Referenced Destination Array	Pointer	Pointer to array to hold parsed values of column
	Row Count	Longint	Count of number of rows parsed from *Referenced BLOB*

The method ***BLOB_Parse_to_Arrays_x*** will parse a referenced BLOB in rows and columns of individual values based on specified, case sensitive, textual delimiters and return those values in referenced arrays. The count of the number of rows of values returned will be returned as a result of this function.

Each row parsed will be stored in the destination array in the same element index. For each column parsed, the value extracted will be placed sequentially into the destination arrays passed to this method. Extra column values parsed without an adequate number of destination arrays will be

disgarded. When columns are missing, the cooresponding elements in the destination arrays will be left as NULL values.

Referenced BLOB is a pointer to a BLOB containing the values to be parsed.

Column Delimiter is the case sensitive text of the column delimiter to use when parsing *Referenced BLOB*.

Row Delimiter is the case sensitive text of the row delimiter to use when parsing *Referenced BLOB*.

Referenced Destination Array is one or more pointers to arrays to contain the values parsed from *Referenced BLOB*. Values extracted are coerced into the cooresponding array index.

Note: the method ***BLOB_Parse_to_Arrays_x*** was added in BASh v1.8.0.

BLOB_Remove_Bytes_by_ASCII

BLOB_Remove_Bytes_by_ASCII *(Referenced BLOB ; ASCII Value ; Position Code)*

BLOB_Remove_Bytes_by_ASCII
(
-> Referenced BLOB : Pointer
-> ASCII Value : Longint
-> Position Code : Longint
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to scan for bytes to remove
	ASCII Value	Longint	Single ASCII value to scan for removing
	Position Code	Longint	Code for positioning within BLOB to remove bytes

The method ***BLOB_Remove_Bytes_by_ASCII*** will remove bytes matching a specified value from a referenced BLOB. A positioning parameter can be used to remove only leading bytes, only trailing bytes, or all bytes which match the specified value.

Referenced BLOB is a pointer to a BLOB which will be scanned for bytes to be removed.

ASCII Value is the byte value which is the scanned for and removed within *Referenced BLOB*.

Position Code is a longint value to indicate where the matching byte values will be scanned for and removed within *Referenced BLOB*. If *Position Code* is set to one (1), all leading bytes matching *ASCII Value* will be removed from *Referenced BLOB*. If *Position Code* is set to two (2), all trailing bytes matching *ASCII Value* will be removed from *Referenced BLOB*. If *Position Code* is set to three (3), all bytes within *Referenced BLOB* matching *ASCII Value* will be removed.

Note: the method ***BLOB_Remove_Bytes_by_ASCII*** was added in BASh v1.6.0.

BLOB_Replace_Byte

BLOB_Replace_Byte *(Referenced BLOB ; Replace Byte Value ; Replacement Byte Value)*

BLOB_Replace_Byte
(
-> *Referenced BLOB :* Pointer
-> *Replace Byte Value :* Longint
-> *Replacement Byte Value :* Longint
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Referenced BLOB to perform byte replacement upon
	Replace Byte Value	Longint	ASCII value of bytes to replace
	Replacement Byte Value	Longint	ASCII value of replacement bytes

The method ***BLOB_Replace_Byte*** will perform a single byte replacement through the contents of a referenced BLOB.

Referenced BLOB is a pointer to a BLOB containing bytes to be scanned for replacement. The replacement is done directly within *Referenced BLOB*.

Replace Byte Value is the ASCII value of the bytes within *Referenced BLOB* which will be replaced.

Replacement Byte Value is the ASCII value of the bytes to be the replacement for all *Replace Byte Value* bytes in *Referenced BLOB*.

Note: the method ***BLOB_Replace_Byte*** was added in BASh v1.5.5.

BLOB_Replace_by_Length_z

BLOB_Replace_by_Length_z *(Referenced Source BLOB ; Referenced Destination BLOB ; Source Starting Offset ; Source Bytes to Copy ; Destination Starting Offset ; Destination Replace Bytes)*

BLOB_Replace_by_Length_z

(

-> *Referenced Source BLOB :* Pointer
-> *Referenced Destination BLOB :* Pointer
-> *Source Starting Offset :* Longint
-> *Source Bytes to Copy :* Longint
-> *Destination Starting Offset :* Longint
-> *Destination Replace Bytes :* Longint

)

	Parameter	Type	Description
	Referenced Source BLOB	Pointer	Pointer to BLOB to copy bytes from
	Referenced Destination BLOB	Pointer	Pointer to BLOB to copy bytes into
	Source Starting Offset	Longint	Offset of first byte in source BLOB to copy from
	Source Bytes to Copy	Longint	Number of bytes in source BLOB to copy

	Parameter	Type	Description
	Destination Starting Offset	Longint	Offset of starting location in destination BLOB to copy into
	Destination Replace Bytes	Longint	Number of bytes in destination BLOB to copy over

The method ***BLOB_Replace_by_Length_z*** will copy a specified portion of a specified source BLOB over a specified series of bytes in a specified destionation BLOB. The source BLOB will not be modified in any way. The destination BLOB will be resized, as needed, for the specified destination byte range to be replaced by the source byte range.

Referenced Source BLOB is a pointer to a BLOB to copy from. This BLOB will not be modified in any way by this method.

Referenced Destination BLOB is a pointer to a BLOB to copy into. Depending on the number of bytes copied from the source BLOB and the number of bytes to be replaced herein, the size of this BLOB will be resized to accommodate the new range of bytes.

Source Starting Offset is the offset within *Referenced Source BLOB* to begin the copy of values at. Set this value to zero (0) to copy from the beginning of the source BLOB.

Source Bytes to Copy is the number of bytes to copy from *Referenced Source BLOB*. Set this values to MAXLONG to copy all bytes to the end of the source BLOB.

Destination Starting Offset is the offset within *Referenced Destination BLOB* to begin copying into. Setting this value past the end of the destination BLOB will merely append the copied bytes to the destination BLOB.

Destination Replace Bytes is the number of bytes from *Destination Starting Offset* to replace with the bytes to be copied into *Referenced Destination BLOB*.

Note: the method ***BLOB_Replace_by_Length_z*** was added in BASh v1.8.0.

BLOB_Replace_Text_by_Offset

BLOB_Replace_Text_by_Offset *(Referenced BLOB ; New Text ; Replace Offset ; Replace Length)*

BLOB_Replace_Text_by_Offset
(
-> *Referenced BLOB :* Pointer
-> *New Text :* Text
-> *Replace Offset :* Longint
-> *Replace Length :* Longint
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Referenced BLOB to perform replacement within
	New Text	Text	Text value to place into BLOB
	Replace Offset	Longint	Beginning offset of content to remove from BLOB
	Replace Length	Longint	Number of bytes of content of BLOB to remove

The method ***BLOB_Replace_Text_by_Offset*** will place a specified text value into a specified byte range within a specified BLOB. The byte range indicated will be removed and the text value will be inserted, without any length indicator, into the BLOB.

Referenced BLOB is a pointer to a BLOB to be operated upon.

New Text is the text value to place into *Referenced BLOB*.

Replace Offset is the offset within *Referenced BLOB* which the replacement of bytes is to begin at.

Replace Length is the number of bytes within *Referenced BLOB* which the replacement of bytes is the take place upon.

Note: the method ***BLOB_Replace_Text_by_Offset*** was added in BASh v1.6.0.

BLOB_Replace_Text_Case

BLOB_Replace_Text_Case *(Referenced BLOB ; Replace Text ; Replacement Text ; Beginning Offset ; Search Bytes) => Replacement Count*

BLOB_Replace_Text_Case

(

-> *Referenced BLOB :* Pointer
-> *Replace Text :* Text
-> *Replacement Text :* Text
-> *Beginning Offset :* Longint
-> *Search Bytes :* Longint

)

=> *Replacement Count :* Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Referenced BLOB to perform byte replacement upon
	Replace Text	Text	Text to be removed
	Replacement Text	Text	Text to be inserted
	Beginning Offset	Longint	Offset within specified BLOB to beginning scanning for specified text value
	Search Bytes	Longint	Number of bytes to scan for specified text values
	Replacement Count	Longint	Number of replacements done within *Referenced BLOB*

The method ***BLOB_Replace_Text_Case*** will perform a sequential, case sensitive replacement of a specified text value within a referenced BLOB.

The replacement is done sequentially from the specified beginning byte offset. It will continue sequential up to, but not past, the number of bytes specified.

Referenced BLOB is a pointer to a BLOB containing bytes to be scanned for replacement. The replacement is done directly within *Referenced BLOB*.

Replace Text is the text to search for within *Referenced BLOB*.

Replacement Text is the text value to replace the complete contents of every occurence of *Replace Text* found within the specified byte range within *Referenced BLOB*. For each replacement operation, the size of the *Referenced BLOB* will be resized, as needed, to accommodate length differences between *Replace Text* and *Replacement Text*.

Beginning Offset is the byte offset within *Referenced BLOB* to begin the search for *Search Text*. If the search is to done from the beginning of *Referenced BLOB*, pass zero (0) as the value for *Beginning Offset*.

Search Bytes is the number of bytes past *Beginning Offset* within *Referenced BLOB* which the value *Replace Text* should be searched for. The search will terminate after *Search Bytes* if no match is found. To search until the end of *Referenced BLOB*, pass MAXLONG as the value for *Search Bytes*.

Replacement Count is the number of occurrences where *Replace Text* has been replaced by *Replacement Text* within *Referenced BLOB*.

Note: the method ***BLOB_Replace_Text_Case*** was added in BASh v1.5.7.

BLOB_Replace_Text_NonCase

BLOB_Replace_Text_NonCase *(Referenced BLOB ; Replace Text ; Replacement Text ; Beginning Offset ; Search Bytes) => Replacement Count*

BLOB_Replace_Text_NonCase
(
-> *Referenced BLOB :* Pointer
-> *Replace Text :* Text
-> *Replacement Text :* Text
-> *Beginning Offset :* Longint
-> *Search Bytes :* Longint
)
=> *Replacement Count :* Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Referenced BLOB to perform byte replacement upon
	Replace Text	Text	Text to be removed
	Replacement Text	Text	Text to be inserted
	Beginning Offset	Longint	Offset within specified BLOB to beginning scanning for specified text value
	Search Bytes	Longint	Number of bytes to scan for specified text values
	Replacement Count	Longint	Number of replacements done within *Referenced BLOB*

The method ***BLOB_Replace_Text_NonCase*** will perform a sequential, non-case sensitive replacement of a specified text value within a referenced BLOB.

The replacement is done sequentially from the specified beginning byte offset. It will continue sequential up to, but not past, the number of bytes specified.

Referenced BLOB is a pointer to a BLOB containing bytes to be scanned for replacement. The replacement is done directly within *Referenced BLOB*.

Replace Text is the text to search for within *Referenced BLOB*.

Replacement Text is the text value to replace the complete contents of every occurence of *Replace Text* found within the specified byte range within *Referenced BLOB*. For each replacement operation, the size of the *Referenced BLOB* will be resized, as needed, to accommodate length differences between *Replace Text* and *Replacement Text*.

Beginning Offset is the byte offset within *Referenced BLOB* to begin the search for *Search Text*. If the search is to done from the beginning of *Referenced BLOB*, pass zero (0) as the value for *Beginning Offset*.

Search Bytes is the number of bytes past *Beginning Offset* within *Referenced BLOB* which the value *Replace Text* should be searched for. The search will terminate after

Search Bytes if no match is found. To search until the end of *Referenced BLOB*, pass MAXLONG as the value for *Search Bytes*.

Replacement Count is the number of occurrences where *Replace Text* has been replaced by *Replacement Text* within *Referenced BLOB*.

Note: the method ***BLOB_Replace_Text_NonCase*** was added in BASh v1.5.7.

BLOB_Set_Bytes

BLOB_Set_Bytes *(Referenced BLOB ; Starting Offset ; Byte Value { ; Byte Value })*

BLOB_Set_Bytes
(
-> *Referenced BLOB :* Pointer
-> *Starting Offset :* Longint
-> *Byte Value :* Longint
{ -> *Byte Value :* Longint }
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Referenced BLOB to set bytes in
	Starting Offset	Longint	Starting offset to set byte values
	Byte Values	Longint	Byte values to set in *Referenced BLOB*

The method ***BLOB_Set_Bytes*** will set byte values in a BLOB, starting at the specified offset. The BLOB will be increased in size if necessary.

Referenced BLOB is a pointer to a BLOB to set byte values in. *Referenced BLOB* will be directly modified.

Starting Offset is the starting offset within *Referenced BLOB* to start setting byte values.

*Byte Value*s are a variable number of parameters containing the byte values to be set in ***Referenced BLOB***.

Note: the method ***BLOB_Set_Bytes*** was added in BASh v1.7.0.

BLOB_Strip_Between_by_ASCII

BLOB_Strip_Between_by_ASCII *(Referenced BLOB ; Starting Delimiter ASCII Value ; Ending Delimiter ASCII Value ; Strip Delimiters Code)*

BLOB_Strip_Between_by_ASCII
(
-> Referenced BLOB : Pointer
-> Starting Delimiter ASCII Value : Longint
-> Ending Delimiter ASCII Value : Longint
-> Strip Delimiters Code : Longint
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to operate upon
	Starting Delimiter ASCII Value	Longint	ASCII value of begin delimiter byte for stripping within *Referenced BLOB*
	Ending Delimiter ASCII Value	Longint	ASCII value of end delimiter byte for stripping within *Referenced BLOB*
	Strip Delimiters Code	Longint	Code for stripping delimiter bytes as well as content between delimiters

The method ***BLOB_Strip_Between_by_ASCII*** will remove delimited contents from a specified BLOB. The delimiter byte values can be specified, as well as whether the delimiter bytes themselves will be removed.

When dealing with delimiters of a single byte, this routine will be much faster than ***BLOB_Strip_Between_x***, as the comparisons used within this routine are much more efficient.

Refereneced BLOB is a pointer to a BLOB to be scanned for delimited data.

Starting Delimiter ASCII Value is the byte value of the beginning delimiter to search for within *Referenced BLOB*.

Ending Delimiter ASCII Value is the byte value of the ending delimiter to search for within *Referenced BLOB*.

Strip Delimiters Code is a longint value code for indicating whether the delimiter bytes are to be removed with the content between any and all found delimiters. If *Strip Delimiters Code* is set to one (1) then the delimiter bytes will also be removed. If *Strip Delimiters Code* is set to zero (0) then the delimiter bytes will not be removed. All other values for *Strip Delimiters Code* will be consider the same as passing zero (0) for this parameter.

Note: the method ***BLOB_Strip_Between_by_ASCII*** was added in BASh v1.6.0.

BLOB_Strip_Between_x

BLOB_Strip_Between_x *(Referenced BLOB ; Starting Delimiter Text Value ; Ending Delimiter Text Value ; Strip Delimiters Code)*

BLOB_Strip_Between_x
(
-> Referenced BLOB : Pointer
-> Starting Delimiter Text Value : Text
-> Ending Delimiter Text Value : Text
-> Strip Delimiters Code : Longint
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB to operate upon
	Starting Delimiter Text Value	Text	Text value of begin delimiter byte for stripping within *Referenced BLOB*
	Ending Delimiter Text Value	Text	Text value of end delimiter byte for stripping within *Referenced BLOB*
	Strip Delimiters Code	Longint	Code for stripping delimiter bytes as well as content between delimiters

The method ***BLOB_Strip_Between_x*** will remove delimited contents from a specified BLOB. The delimiter values can be specified, as well as whether the delimiters themselves will be removed.

When dealing with delimiters of a single byte, the method ***BLOB_Strip_Between_by_ASCII*** is much faster than this method, as the comparisons used within this routine are much less efficient.

Refereneced BLOB is a pointer to a BLOB to be scanned for delimited data.

Starting Delimiter Text Value is the text value of the beginning delimiter to search for within *Referenced BLOB*.

Ending Delimiter Text Value is the text value of the ending delimiter to search for within *Referenced BLOB*.

Strip Delimiters Code is a longint value code for indicating whether the delimiters are to be removed with the content between any and all found delimiters. If *Strip Delimiters Code* is set to one (1) then the delimiters will also be removed. If *Strip Delimiters Code* is set to zero (0) then the delimiters will not be removed. All other values for *Strip Delimiters Code* will be consider the same as passing zero (0) for this parameter.

Note: the method ***BLOB_Strip_Between_x*** was added in BASh v1.6.0.

BLOB_XOR_Bytes

BLOB_XOR_Bytes *(Referenced Source BLOB ; Referenced Operand BLOB ; Referenced Result BLOB)*

BLOB_XOR_Bytes

(

-> Referenced Source BLOB : Pointer
-> Referenced Operand BLOB : Pointer
-> Referenced Result BLOB : Pointer

)

	Parameter	Type	Description
	Referenced Source BLOB	Pointer	Referenced BLOB used to create XOR result BLOB

	Parameter	Type	Description
	Referenced Operand BLOB	Pointer	Referenced BLOB used to create XOR result BLOB
	Referenced Result BLOB	Pointer	Referenced BLOB to contain resulting XORed BLOB

The method ***BLOB_XOR_Bytes*** will perform a sequential XOR operation on every byte within two specified BLOBs and return the resulting XORed BLOB.

If the two source BLOBs, *Referenced Source BLOB* and *Referenced Operand BLOB* are of different size, then the resulting BLOB *Referenced Result BLOB* will be the size of the larger with padding bytes of the shorter source BLOB assumed to be NULL bytes.

Referenced Source BLOB and *Referenced Operand BLOB* are pointers to BLOBs which will have their contents XORed byte for byte.

Referenced Result BLOB will contain the byte for byte XORing of the contents of the two referenced source BLOBs.

Note: the method ***BLOB_XOR_Bytes*** was added in BASh v1.5.9.

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_Convert_ShiftJIS_to_JIS_x

CODEC_Convert_ShiftJIS_to_JIS_x *(Shift JIS Encoded Text) => JIS Encoded Text*

CODEC_Convert_ShiftJIS_to_JIS_x
(
-> *Shift JIS Encoded Text* : Text
)

=> *JIS Encoded Text :* Text

	Parameter	Type	Description
	Shift JIS Encoded Text	Text	Shift JIS encoded text to be recoded
	JIS Encoded Text	Text	Text recoded to standard JIS encoding

The method ***CODEC_Convert_ShiftJIS_to_JIS_x*** recodes a Shift JIS encoded text value to JIS encoding.

Shift JIS Encoded Text is the Shift JIS encoded text that is to be recoded.

JIS Encoded Text is the recoded text value.

Note: the method ***CODEC_Convert_ShiftJIS_to_JIS_x*** was added in BASh v1.8.1.

CODEC_Convert_ShiftJIS_to_JIS_z

CODEC_Convert_ShiftJIS_to_JIS_z *(Referenced BLOB)* => *qi Success*

CODEC_Convert_ShiftJIS_to_JIS_z
(
-> *Referenced BLOB :* Pointer
)
=> *qi Success :* Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing Shift JIS encoded text to be recoded
	qi Success	Longint	Indicator for whether routine successfully performed recoding

The method ***CODEC_Convert_ShiftJIS_to_JIS_z*** recodes bytes stored in a referenced BLOB from Shift JIS encoding to JIS encoding.

Referenced BLOB is a pointer to a BLOB containing the bytes to recode from Shift JIS to JIS. If the BLOB contains text, it should be text stored in the BLOB without any length or

delimiter bytes, as this routine operates directly on the byte values.

qi Success is a longint indicator result code for whether the routine successfully performed the recoding on *Referenced BLOB*. A value of zero (0) for this indicates the routine failed. A value of one (1) indicates the recoding was successfully performed.

Note: the method ***CODEC_Convert_ShiftJIS_to_JIS_z*** was added in BASh v1.8.1.

CODEC_Decode_Base64_x

CODEC_Decode_Base64_x *(Base64 Encoded Text) => Decoded Text*

CODEC_Decode_Base64_x
(
-> *Base64 Encoded Text :* Text
)
=> *Decoded Text :* Text

	Parameter	Type	Description
	Base64 Encoded Text	Text	Base64 encoded text to be decoded
	Decoded Text	Text	Decoding of *Base64 Encoded Text*

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

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
	Referenced BLOB	Text	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

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

	Parameter	Type	Description
	CLE Encoded Text	Text	CLE encoded text to be decoded
	Decoded Text	Text	Decoding of *CLE Encoded Text*

The method ***CODEC_Decode_CLE_x*** returns the decoding of a single CLE encoded value. Details on CLE encoding are available in the **CLE Encoding Scheme** section.

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_Hex_x

CODEC_Decode_Hex_x *(Hex Encoded Text) => Decoded ASCII Text*

CODEC_Decode_Hex_x
(
-> *Hex Encoded Text :* Text
)
=> *Decoded ASCII Text :* Text

	Parameter	Type	Description
	Hex Encoded Text	Text	Hex encoded text value to be decoded
	Decoded ASCII Text	Text	Hex decoded text value

The method ***CODEC_Decode_Hex_x*** will hexadecimal decoded a specified text value and return the resulting decoded text value.

Hex Encoded Text is a text value containing a hexadecimal encoded text which is to be decoded.

Decoded ASCII Text is *Hex Encoded Text* hexadecimal decoded.

NOTE: remember that a hexadecimal decoded value occupies only half the number of bytes as its encoded value.

Note: the method ***CODEC_Decode_Hex_x*** was added in BASh v1.5.8.

CODEC_Decode_Hex_z

CODEC_Decode_Hex_z *(Referenced BLOB)*

CODEC_Decode_Hex_z

(

-> *Referenced BLOB :* Pointer

)

	Parameter	Type	Description
	Referenced BLOB	Text	Reference to BLOB containing hexadecimal encoded value to be decoded

The method ***CODEC_Decode_Hex_z*** will hexadecimal decode the contents of a referenced BLOB.

Referenced BLOB is a pointer to a BLOB containing a hexadecimal encoded value. The decoded value will be returned in the same referenced BLOB.

NOTE: remember that a hexadecimal decoded value occupies only half the number of bytes as its encoded value.

Note: the method ***CODEC_Decode_Hex_z*** was added in BASh v1.5.8.

CODEC_Decode_HTML_Entities_x

CODEC_Decode_HTML_Entities_x *(Encoded Text { ; Resource File Reference { ; Resource ID } }) => Decoded Text*

CODEC_Decode_HTML_Entities_x

(

-> *Encoded Text :* Text

-> *Resource File Reference :* Time

-> *Resource ID :* Longint

)

=> *Decoded Text :* Text

	Parameter	Type	Description
	Encoded Text	Text	Text to have HTML entities decoded
	Resource File Reference	Time	Resource file reference of open resource fork to use 'rSr#' resource for the HTML entities decoding
	Resource ID	Longint	Resource ID of 'rSr#' resource in specified resource file for the HTML entities decoding
	Decoded Text	Longint	Specified text with HTML entities decoded

The method ***CODEC_Decode_HTML_Entities_x*** will HTML entities decode a specified text value using the HTML entities provided by default in BASh or a listing of values optionally specified in a separate resource document. The decoded text value will be returned.

The HTML entities to decode are specified within a pair of resources listed within a 'rSr#' resource. The 'rSr#' resource must contain the referenced to a 'STR#' resource and a 'bYt#' resource. These two resources must contain the same number of list elements and correspond to the string values to be searched for and replaced with the byte values in the equivalent row of the resource lists. See the default 'rSr#' resource (resource ID 29011) stored in the Affix BASh document for a clear example of how this is done.

Encoded Text is a text value containing the text to decode.

Resource File Reference is a file reference to an open resource fork to use the 'rSr#' resource from. This parameter is optional and if not specified then the Affix BASh document is used.

Resource ID is a longint containing the resource ID of the 'rSr#' resource to use for HTML entities decoding. This value is optional and if not specified then the resource ID value is assumed to be 29011.

Decoded Text is a text value returned containing the specified text value decoded using the HTML entities specified through the 'rSr#' resource.

Note: the method ***CODEC_Decode_HTML_Entities_x*** was added in BASh v1.8.2.

CODEC_Decode_HTML_Entities_z

CODEC_Decode_HTML_Entities_z *(Referenced BLOB { ; Resource File Reference { ; Resource ID } })*

CODEC_Decode_HTML_Entities_z

(

-> Referenced BLOB : Pointer
-> Resource File Reference : Time
-> Resource ID : Longint

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing value to HTML entities decode
	Resource File Reference	Time	Resource file reference of open resource fork to use 'rSr#' resource for the HTML entities decoding
	Resource ID	Longint	Resource ID of 'rSr#' resource in specified resource file for the HTML entities decoding

The method ***CODEC_Decode_HTML_Entities_z*** will HTML entities decode a specified referenced BLOB value using the HTML entities provided by default in BASh or a listing of values optionally specified in a separate resource document.

The HTML entities to decode are specified within a pair of resources listed within a 'rSr#' resource. The 'rSr#' resource must contain the referenced to a 'STR#' resource and a 'bYt#' resource. These two resources must contain the same number of list elements and correspond to the string values to be searched for and replaced with the byte values in the equivalent row of the resource lists. See the default 'rSr#' resource (resource ID 29011) stored in the Affix BASh document for a clear example of how this is done.

Referenced BLOB is a pointer to a BLOB containing the value to scan and decode.

Resource File Reference is a file reference to an open resource fork to use the 'rSr#' resource from. This parameter is optional and if not specified then the Affix BASh document is used.

Resource ID is a longint containing the resource ID of the 'rSr#' resource to use for HTML entities decoding. This value is optional and if not specified then the resource ID value is assumed to be 29011.

Note: the method ***CODEC_Decode_HTML_Entities_z*** was added in BASh v1.8.2.

CODEC_Decode_ISO_x

CODEC_Decode_ISO_x(*Encoded Text*) => *Decoded Text*

CODEC_Decode_ISO_x

(

-> *Encoded Text :* Text

)

=> *Decoded Text :* Text

	Parameter	Type	Description
	Encoded Text	Text	ISO-8859-1 encoded text value
	Decoded Text	Text	ISO-8859-1 decoding to Mac standard of *Encoded Text*

The method ***CODEC_Decode_ISO_x*** will ISO-8859-1 decode a specified text value, returning the resulting Mac standard text value.

The lookup table used for the decoding and encoding of ISO-8859-1 is stored in the resource fork of the Affix BASh document. This is a 'bYt#' resource, resource ID 29001.

Encoded Text is a specifed text value containing the ISO-8859-1 text value to decode and convert to a Mac standard text value (as used in 4D).

Decoded Text is a text value returned containing the ISO-8859-1 decoded text.

Note: the method ***CODEC_Decode_ISO_x*** was added in BASh v1.8.2.

CODEC_Decode_ISO_z

CODEC_Decode_ISO_z *(Referenced BLOB)*

CODEC_Decode_ISO_z

(

-> *Referenced BLOB :* Pointer

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing value to ISO-8859-1 decode

The method ***CODEC_Decode_ISO_z*** will ISO-8859-1 decode a specified referenced value into standard Mac encoded format (the standard used within 4D).

The lookup table used for the decoding and encoding of ISO-8859-1 is stored in the resource fork of the Affix BASh document. This is a 'bYt#' resource, resource ID 29001.

Referenced BLOB is a pointer to a BLOB containing the value to decode.

Note: the method ***CODEC_Decode_ISO_z*** was added in BASh v1.8.2.

CODEC_Decode_MIME_Words_x

CODEC_Decode_MIME_Words_x *(MIME Encoded Text) => Decoded Text*

CODEC_Decode_MIME_Words_x

(

-> *MIME Encoded Text :* Text
)
=> *Decoded Text :* Text

	Parameter	Type	Description
	MIME Encoded Text	Text	MIME encoded text to be decoded
	Decoded Text	Text	MIME decoded text value

The method ***CODEC_Decode_MIME_Words_x*** will check to see if the text is MIME encoded, and decode it. If the text is not MIME encoded, it will be returned unchanged. Details on MIME encoding are available in RFC 2047, available at:

http://www.deepskytech.com/rfcs/rfc_2047.txt

MIME Encoded Text is a text value containing MIME encoded text which is to be decoded.

Decoded Text is *MIME Encoded Text* decoded.

Note: the method ***CODEC_Decode_MIME_Words_x*** was added in BASh v1.6.1.

CODEC_Decode_QuotedPrintable_x

CODEC_Decode_QuotedPrintable_x (*Encoded Text ; ISO 8859-1 Encoded*) =>
Decoded Text

CODEC_Decode_QuotedPrintable_x
(
-> *Encoded Text :* Text
-> *ISO 8859-1 Encoded :* Longint
)
=> *Decoded Text :* Text

	Parameter	Type	Description
	Encoded Text	Text	Quoted-Printable text to be decoded
	ISO 8859-1 Encoded	Longint	qi for whether ISO 8859-1 decoding should be applied after Quoted-Printable decoding

	Parameter	Type	Description
	Decoded Text	Text	Decoded text value

The method ***CODEC_Decode_QuotedPrintable_x*** will decode Quoted-Printable text. It can optionally apply ISO 8859-1 decoding as well. Details on Quoted-Printable encoding are available in RFC 1521, available at:

http://www.deepskytech.com/rfcs/rfc_1521.txt

Encoded Text is a text value containing Quoted-Printable encoded text which is to be decoded.

ISO 8859-1 Encoded is a qi for whether to apply ISO 8859-1 decoding to the Encoded Text after the Quoted-Printable decoding. Pass a zero (0) to not apply this decoding, or a one (1) to apply it.

Decoded Text is *Encoded Text* decoded.

Note: the method ***CODEC_Decode_QuotedPrintable_x*** was added in BASh v1.6.1.

CODEC_Decode_QuotedPrintable_z

CODEC_Decode_QuotedPrintable_z *(BLOB to Decode ; ISO 8859-1 Encoded)*

CODEC_Decode_QuotedPrintable_z

```
(
        -> BLOB to Decode : Pointer
        -> ISO 8859-1 Encoded : Longint
)
```

	Parameter	Type	Description
	BLOB to Decode	Pointer	Reference to BLOB containing Quoted-Printable text to be decoded
	ISO 8859-1 Encoded	Longint	qi for whether ISO 8859-1 decoding should be applied after Quoted-Printable decoding

The method ***CODEC_Decode_QuotedPrintable_z*** will decode Quoted-Printable text in the referenced BLOB. It can optionally apply ISO 8859-1 decoding as well. Details on Quoted-Printable encoding are available in RFC 1521, available at:

http://www.deepskytech.com/rfcs/rfc_1521.txt

BLOB to Decode is a referenced BLOB containing Quoted-Printable encoded text which is to be decoded.

ISO 8859-1 Encoded is a qi for whether to apply ISO 8859-1 decoding to the Encoded Text after the Quoted-Printable decoding. Pass a zero (0) to not apply this decoding, or a one (1) to apply it.

Note: the method ***CODEC_Decode_QuotedPrintable_z*** was added in BASh v1.6.1.

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	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

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, pluses 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_Decode_URL_z

CODEC_Decode_URL_z *(Referenced BLOB ; Translation Option)*

CODEC_Decode_URL_z

(

-> *Referenced BLOB :* Pointer
-> *Translation Option :* Text

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Referenced BLOB to URL decode
	Translation Option	Longint	qi for decoding "+" values into spaces

The method ***CODEC_Decode_URL_z*** 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

Referenced BLOB is a pointer to a BLOB containing URL encoded data which is to be decoded. The decoding is done directly within *Referenced BLOB*.

Translation Option is option flag for how to decode plus ("+", ASCII value 43) values within *Referenced BLOB*. If *Translation Option* equals zero (0) then pluses are translated into

spaces (ASCII value 32). Otherwise, pluses will not be translated into spaces.

Note: the method ***CODEC_Decode_URL_z*** was added in BASh v1.5.5.

CODEC_Decode_UUEncode_File

CODEC_Decode_UUEncode_File *(Referenced BLOB)* => *File Name*

CODEC_Decode_UUEncode_File
(
-> *Referenced BLOB :* Pointer
)
=> *File Name :* Text

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing UUEncoded file which is the be decoded
	File Name	Text	File name of the original document

The method ***CODEC_Decode_UUEncode_File*** will decode a UUEncoded document referenced in a BLOB and return the original document name. Details on UUEncoding are available in the following document:

http://www.deepskytech.com/rfcs/uuencode.txt

Referenced BLOB is a pointer to a BLOB containing the UUEncoded document to tbe decoded.

File Name is the name of the original document which was UUEncoded.

Note: the method ***CODEC_Decode_UUEncode_File*** was added in BASh v1.5.9.

CODEC_Decode_UUEncode_x

CODEC_Decode_UUEncode_x *(Encoded Text) => Decoded Text*

CODEC_Decode_UUEncode_x

(

-> *Encoded Text :* Text

)

=> *Decoded Text :* Text

	Parameter	Type	Description
	Encoded Text	Text	UUEncoded text which will be decoded
	Decoded Text	Text	UUDecoded text

The method ***CODEC_Decode_UUEncode_x*** will decode a UUEncoded text value and return the decoded text value. Details on UUEncoding are available in the following document:

http://www.deepskytech.com/rfcs/uuencode.txt

Encoded Text is a text value which has been UUEncoded. It wll be decoded in this method.

Decoded Text is the decoding of *Encoded Text* using the UUEncoding coding schema.

Note: the method ***CODEC_Decode_UUEncode_x*** was added in BASh v1.5.9.

CODEC_Decode_UUEncode_z

CODEC_Decode_UUEncode_z *(Referenced BLOB)*

CODEC_Decode_UUEncode_z

(

-> *Referenced BLOB :* Pointer

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to a BLOB containing UUEncoded data which is to be decoded

The method ***CODEC_Decode_UUEncode_z*** will decode previously UUEncoded data stored in a specified BLOB being referernced. Details on UUEncoding are available in the following document:

http://www.deepskytech.com/rfcs/uuencode.txt

Referenced BLOB is a pointer to a BLOB containing a UUEncoded encoded value. The decoded value will be returned in the same referenced BLOB.

Note: the method ***CODEC_Decode_UUEncode_z*** was added in BASh v1.5.9.

CODEC_Encode_Base64_x

CODEC_Encode_Base64_x *(Text Value { ; qi Include Linebreaks }) => Base64 Encoded Text*

CODEC_Encode_Base64_x

(

-> *Text Value :* Text
-> *qi Include Linebreaks :* Longint

)

=> *Base64 Encoded Text :* Text

	Parameter	Type	Description
	Text Value	Text	Text value to be base64 encoded
	Base64 Encoded Text	Text	Base64 encoding of *Text Value*
	qi Include Linebreaks	Longint	qi flag to include (default) or suppress linebreaks for standard Base64 encoding [optional]

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

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

Base64 Encoded Text is *Text Value* base64 encoded.

qi Include Linebreaks is an optional flag for whether the standard linebreaks inclusion is to be performed on the encoded value. To suppress the inclusion of linebreaks in the encoded value, pass a value of one (1) for this parameter. This parameter is not required and by default the linebreaks are included in the encoded value.

Note: the method ***CODEC_Encode_Base64_x*** was added in BASh v1.5.1.

Note: the optional parameter *qi Include Linebreaks* was added in BASh v1.8.0.

CODEC_Encode_Base64_z

CODEC_Encode_Base64_z *(Referenced BLOB { ; qi Include Linebreaks })*

CODEC_Encode_Base64_z

(

-> *Referenced BLOB :* Pointer
-> *qi Include Linebreaks :* Longint [optional]

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Reference to BLOB to be base64 encoded
	qi Include Linebreaks	Longint	qi flag to include (default) or suppress linebreaks for standard Base64 encoding [optional]

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

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

qi Include Linebreaks is an optional flag for whether the standard linebreaks inclusion is to be performed on the

encoded value. To suppress the inclusion of linebreaks in the encoded value, pass a value of one (1) for this parameter. This parameter is not required and by default the linebreaks are included in the encoded value.

Note: the method ***CODEC_Encode_Base64_z*** was added in BASh v1.5.1.

Note: the optional parameter *qi Include Linebreaks* was added in BASh v1.8.0.

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
	Text Value	Text	Text value to be CLE encoded
	CLE Encoded Text	Text	CLE encoding of *Text Value*

The method ***CODEC_Encode_CLE_x*** returns a supplied text value CLE encoded. Details on CLE encoding are available in the **CLE Encoding Scheme** section.

Text Value is the text value to be CLE encoded.

CLE Encoded Text is *Text Value* CLE encoded.

Note: the method ***CODEC_Encode_CLE_x*** was added in BASh v1.5.1.

CODEC_Encode_CRAMMD5_z

CODEC_Encode_CRAMMD5_z *(Username ; Key ; Referenced Source BLOB ; Referenced Result BLOB)*

CODEC_Encode_CRAMMD5_z

(

-> Username : Text
-> Key : Text
-> Referenced Source BLOB : Pointer
-> Referenced Result BLOB : Pointer

)

	Parameter	Type	Description
	Username	Text	Username to use in creating CRAM MD5 encoding
	Key	Text	Key to use in creating CRAM MD5 encoding
	Referenced Source BLOB	Pointer	Pointer to BLOB containing challenge to use in creating CRAM MD5 encoding
	Referenced Result BLOB	Pointer	Pointer to BLOB to contain CRAM MD5 encoding

The method ***CODEC_Encode_CRAMMD5_z*** will create a properly formatted CRAM MD5 digest from the parameter values supplied. Specifications for the CRAM MD5 encoding scheme are available in RFC 2195, available at:

http://www.deepskytech.com/rfcs/rfc_2195.txt

Username is the username to be used to create the CRAM MD5 encoding.

Key is the key to be used to create the CRAM MD5 encoding.

Referenced Source BLOB is a pointer to a BLOB container the challenge value which is to be used to create the CRAM MD5 encoding.

Referenced Result BLOB is a pointer to a BLOB which will contain the CRAM MD5 encoded value.

Note: the method ***CODEC_Encode_CRAMMD5_z*** was added in BASh v1.5.9.

CODEC_Encode_Hex_x

CODEC_Encode_Hex_x *(ASCII Text to Encode) => Hex Encoded Text*

CODEC_Encode_Hex_x
(
-> *ASCII Text to Encode :* Text
)
=> *Hex Encoded Text :* Text

	Parameter	Type	Description
	ASCII Text to Encode	Text	Text value to be hex encoded
	Hex Encoded Text	Text	Hex encoding of *Text Value*

The method ***CODEC_Encode_Hex_x*** will hexadecimal encoded a specified text value and return the resulting encoded text value.

ASCII Text to Encode is a text value to be encoded.

Hex Encoded Text is *ASCII Text to Encode* hexadecimal encoded.

NOTE: remember that a hexadecimal encoded value occupies twice the number of bytes as its decoded value.

Note: the method ***CODEC_Encode_Hex_x*** was added in BASh v1.5.8.

CODEC_Encode_Hex_z

CODEC_Encode_Hex_z *(Referenced BLOB)*

CODEC_Encode_Hex_z
(
-> *Referenced BLOB :* Pointer
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing value to be hexadecimal encoded

The method ***CODEC_Encode_Hex_z*** will hexadecimal encode the contents of a referenced BLOB.

Referenced BLOB is a pointer to a BLOB containing a value to be hexadecimal encoded. The encoded value will be returned in the same referenced BLOB.

NOTE: remember that a hexadecimal encoded value occupies twice the number of bytes as its decoded value.

Note: the method ***CODEC_Encode_Hex_z*** was added in BASh v1.5.8.

CODEC_Encode_HTML_Entities_x

CODEC_Encode_HTML_Entities_x (*Text to Encode* { ; *Resource File Reference* { ; *Resource ID* } }) => *Encoded Text*

CODEC_Encode_HTML_Entities_x

(

-> *Text to Encode :* Text
-> *Resource File Reference :* Time
-> *Resource ID :* Longint

)

=> *Encoded Text :* Text

	Parameter	Type	Description
	Text to Encode	Text	Text value to HTML entities encode
	Resource File Reference	Time	Resource file reference of open resource fork to use 'rSr#' resource for the HTML entities encoding
	Resource ID	Longint	Resource ID of 'rSr#' resource in specified resource file for the HTML entities encoding
	Encoded Text	Text	HTML entities encoded text value

The method ***CODEC_Encode_HTML_Entities_x*** will HTML entities encode a specified text value and return the encoded text.

The HTML entities to encode are specified within a pair of resources listed within a ‘rSr#’ resource. The ‘rSr#’ resource must contain the referenced to a ‘STR#’ resource and a ‘bYt#’ resource. These two resources must contain the same number of list elements and correspond to the byte values to be searched for and replaced with the string values in the equivalent row of the resource lists. See the default ‘rSr#’ resource (resource ID 2901) stored in the Affix BASh document for a clear example of how this is done.

Text to Encode is a text value containing the text to encode.

Resource File Reference is a file reference to an open resource fork to use the ‘rSr#’ resource from. This parameter is optional and if not specified then the Affix BASh document is used.

Resource ID is a longint containing the resource ID of the ‘rSr#’ resource to use for HTML entities encoding. This value is optional and if not specified then the resource ID value is assumed to be 29010.

Encoded Text is a text value returned containing the specified text value encoded using the HTML entities specified through the ‘rSr#’ resource.

Note: the method ***CODEC_Encode_HTML_Entities_x*** was added in BASh v1.8.2.

CODEC_Encode_HTML_Entities_z

CODEC_Encode_HTML_Entities_z *(Referenced BLOB { ; Resource File Reference { ; Resource ID } })*

ARR_
(

-> Referenced BLOB : Pointer
-> Resource File Reference : Time
-> Resource ID : Longint

)

	Parameter	Type	Description
	Referenced BLOB	Text	Pointer to BLOB containing value to HTML entities encode
	Resource File Reference	Time	Resource file reference of open resource fork to use 'rSr#' resource for the HTML entities encoding
	Resource ID	Longint	Resource ID of 'rSr#' resource in specified resource file for the HTML entities encoding

The method ***CODEC_Encode_HTML_Entities_z*** will HTML entities encode a specified referenced BLOB value.

The HTML entities to encode are specified within a pair of resources listed within a 'rSr#' resource. The 'rSr#' resource must contain the referenced to a 'STR#' resource and a 'bYt#' resource. These two resources must contain the same number of list elements and correspond to the byte values to be searched for and replaced with the string values in the equivalent row of the resource lists. See the default 'rSr#' resource (resource ID 2901) stored in the Affix BASh document for a clear example of how this is done.

Referenced BLOB is a pointer to a BLOB containing the value to scan and encode.

Resource File Reference is a file reference to an open resource fork to use the 'rSr#' resource from. This parameter is optional and if not specified then the Affix BASh document is used.

Resource ID is a longint containing the resource ID of the 'rSr#' resource to use for HTML entities encoding. This value is optional and if not specified then the resource ID value is assumed to be 29010.

Note: the method ***CODEC_Encode_HTML_Entities_z*** was added in BASh v1.8.2.

CODEC_Encode_ISO_x

CODEC_Encode_ISO_x *(Text to Encode)* => *Encoded Text*

CODEC_Encode_ISO_x
(
-> *Text to Encode :* Text
)
=> *Encoded Text :* Text

	Parameter	Type	Description
	Text to Encode	Text	Text value to ISO-8859-1 encode
	Encoded Text	Text	ISO-8859-1 encoding of *Encoded Text*

The method ***CODEC_Encode_ISO_x*** will ISO-8859-1 encode and return a specified text value.

The lookup table used for the decoding and encoding of ISO-8859-1 is stored in the resource fork of the Affix BASh document. This is a 'bYt#' resource, resource ID 29001.

Text to Encode is a specifed text value to ISO-8859-1 to encode.

Encoded Text is a text value returned containing the ISO-8859-1 encoded text.

Note: the method ***CODEC_Encode_ISO_x*** was added in BASh v1.8.2.

CODEC_Encode_ISO_z

CODEC_Encode_ISO_z *(Referenced BLOB)*

CODEC_Encode_ISO_z
(
-> *Referenced BLOB :* Pointer
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing value to ISO-8859-1 encode

The method ***CODEC_Encode_ISO_z*** will ISO-8859-1 encode and return a specified referenced BLOB value.

The lookup table used for the decoding and encoding of ISO-8859-1 is stored in the resource fork of the Affix BASh document. This is a 'bYt#' resource, resource ID 29001.

Referenced BLOB is a pointer to a BLOB containing the value to encode.

Note: the method ***CODEC_Encode_ISO_z*** was added in BASh v1.8.2.

CODEC_Encode_KeyedMD5_z

CODEC_Encode_KeyedMD5_z *(Key ; Referenced Source BLOB ; Referenced Result BLOB)*

CODEC_Encode_KeyedMD5_z

(

-> *Key :* Text
-> *Referenced Source BLOB :* Pointer
-> *Referenced Result BLOB :* Pointer

)

	Parameter	Type	Description
	Key	Text	Key to use in creating Keyed MD5 encoding
	Referenced Source BLOB	Pointer	Pointer to BLOB containing challenge to use in creating Keyed MD5 encoding
	Referenced Result BLOB	Pointer	Pointer to BLOB to contain Keyed MD5 encoding

The method ***CODEC_Encode_KeyedMD5_z*** will create a properly formatted Keyed MD5 digest (i.e. Keyed Hash) from the parameter values supplied. Specifications for the Keyed MD5 encoding scheme are available in RFC 2104, available at:

http://www.deepskytech.com/rfcs/rfc_2104.txt

Key is the key to be used to create the Keyed MD5 encoding.

Referenced Source BLOB is a pointer to a BLOB container the challenge value which is to be used to create the Keyed MD5 encoding.

Referenced Result BLOB is a pointer to a BLOB which will contain the Keyed MD5 encoded value.

Note: the method ***CODEC_Encode_KeyedMD5_z*** was added in BASh v1.5.9.

CODEC_Encode_MD4_x

CODEC_Encode_MD4_x *(Text Value) => MD4 Encoded Text Digest*

CODEC_Encode_MD4_x
(
-> *Text Value :* Text
)
=> *MD4 Encoded Text Digest :* Text

	Parameter	Type	Description
	Text Value	Text	Text value to create MD5 digest for
	MD4 Encoded Text Digest	Text	MD4 encoded digest of *Text Value*

The method ***CODEC_Encode_MD4_x*** creates a MD4 encoded digest of a supplied text value. Specifications for the MD4 encoded digest scheme are available in RFC 1320, available at:

http://www.deepskytech.com/rfcs/rfc_1320.txt

Text Value is the text which is to used to create the MD4 encoded digest.

MD4 Encoded Text Value is the MD4 encoded digest created using *Text Value*.

Note: the MD4 encoded digest is a one-way encoding schema. There is no way to retrieve the original value from an MD4 encoded digest.

Note: the method ***CODEC_Encode_MD4_x*** was added in BASh v1.7.0.

CODEC_Encode_MD4_z

CODEC_Encode_MD4_z *(Referenced BLOB)*

CODEC_Encode_MD4_z

(

-> Referenced BLOB : Pointer

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Reference to BLOB value to encode into MD4 digest

The method ***CODEC_Encode_MD4_z*** creates a MD4 encoded digest from a referenced BLOB value. Specifications for the MD4 encoded digest scheme are available in RFC 1320, available at:

http://www.deepskytech.com/rfcs/rfc_1320.txt

Referenced BLOB is the referenced BLOB to be used to create the MD4 encoded digest. The MD4 encoded digest is returned directly in this referenced value.

Note: the method ***CODEC_Encode_MD4_z*** was added in BASh v1.7.0.

CODEC_Encode_MD5_x

CODEC_Encode_MD5_x *(Text Value)* => *MD5 Encoded Text Digest*

CODEC_Encode_MD5_x
(
-> *Text Value :* Text
)
=> *MD5 Encoded Text Digest :* Text

	Parameter	Type	Description
	Text Value	Text	Text value to create MD5 digest for
	MD5 Encoded Text Digest	Text	MD5 encoded digest of *Text Value*

The method ***CODEC_Encode_MD5_x*** creates a MD5 encoded digest of a supplied text value. Specifications for the MD5 encoded digest scheme are available in RFC 1321, available at:

http://www.deepskytech.com/rfcs/rfc_1321.txt

Text Value is the text which is to used to create the MD5 encoded digest.

MD5 Encoded Text Value is the MD5 encoded digest created using *Text Value*.

Note: the MD5 encoded digest is a one-way encoding schema. There is no way to retrieve the original value from an MD5 encoded digest. Often, MD5 encoded digests are used for logging into POP3 email servers when using the optional APOP authentication option.

Note: the method ***CODEC_Encode_MD5_x*** was added in BASh v1.5.4.

CODEC_Encode_MD5_z

CODEC_Encode_MD5_z *(Referenced Unencoded BLOB ; Referenced Encoded BLOB)*

CODEC_Encode_MD5_z
(
-> *Referenced Unencoded BLOB :* Pointer
-> *Referenced Encoded BLOB :* Pointer

)

	Parameter	Type	Description
	Referenced Unencoded BLOB	Pointer	Reference to BLOB value to encode into MD5 digest
	Referenced Encoded BLOB	Pointer	Reference to BLOB value encoded into MD5 digest

The method ***CODEC_Encode_MD5_z*** creates a MD5 encoded digest from a referenced BLOB value. Specifications for the MD5 encoded digest scheme are available in RFC 1321, available at:

http://www.deepskytech.com/rfcs/rfc_1321.txt

Referenced Unencoded BLOB is a pointer to a BLOB to be used to create the MD5 encoded digest.

Referenced Encoded BLOB is a pointer to a BLOB to contain the MD5 encoded result. The MD5 encoded digest is returned directly in this referenced value.

Note: the method ***CODEC_Encode_MD5_z*** was added in BASh v1.5.4.

CODEC_Encode_MIME_Words_x

CODEC_Encode_MIME_Words_x *(Text Value ; Character Set ; Encoding Type)* => *Encoded Text*

CODEC_Encode_MIME_Words_x

(

-> *Text Value :* Text
-> *Character Set :* Text
-> *Encoding Type :* String[1]

)

-> *Encoded Text :* Text

	Parameter	Type	Description
	Text Value	Text	Text value to be MIME encoded

	Parameter	Type	Description
	Character Set	Text	Character set to use for MIME encoding
	Encoding Type	String[1]	Encoding type according to RFC 2047 Allowed Values: Q = Quoted-Printable B = Base64
	Encoded Text	Text	MIME encoding of Text Value

The method ***CODEC_Encode_MIME_Words_x*** encodes text with MIME encoding. Details on MIME encoding are available in RFC 2047, available at:

http://www.deepskytech.com/rfcs/rfc_2047.txt

Text Value is the text to be MIME encoded.

Character Set is the character set to use for the MIME encoding.

Encoding Type is the type of MIME encoding to be performed, according to RFC 2047. *Encoding Type* should be either a "Q", for Quoted-Printable encoding, or a "B" for Base64 encoding.

Encoded Text is *Text Value* MIME encoded.

Note: the method ***CODEC_Encode_MIME_Words_x*** was added in BASh v1.6.1.

CODEC_Encode_QuotedPrintable_x

CODEC_Encode_QuotedPrintable_x *(Text to Encode ; ISO 8859-1 Encoding)* => *Encoded Text*

CODEC_Encode_QuotedPrintable_x
(
-> *Text to Encode :* Text
-> *ISO 8859-1 Encoding :* Longint
)
=> *Encoded Text :* Text

	Parameter	Type	Description
	Text to Encode	Text	Text to be Quoted-Printable encoded
	ISO 8859-1 Encoding	Longint	qi for whether to apply ISO-8859-1 encoding before Quoted-Printable encoding
	Encoded Text	Text	Quoted-Printable encoding of *Text to Encode*

The method ***CODEC_Encode_QuotedPrintable_x*** encodes the text with Quoted-Printable encoding. Optionally, the text can also be ISO 8859-1 encoded too. Details on Quoted-Printable encoding are available in RFC 1521, available at:

http://www.deepskytech.com/rfcs/rfc_1521.txt

Text to Encode is the text to be Quoted-Printable encoded.

ISO 8859-1 Encoding is a qi for whether ISO 8859-1 encoding should be applied before the Quoted-Printable encoding. Pass a zero (0) to skip the ISO 8859-1 encoding, or a one (1) to apply this encoding.

Encoded Text is *Text Value* Quoted-Printable encoded.

Note: the method ***CODEC_Encode_QuotedPrintable_x*** was added in BASh v1.6.1.

CODEC_Encode_QuotedPrintable_z

CODEC_Encode_QuotedPrintable_z *(BLOB to Encode ; ISO 8859-1 Encoding)*

CODEC_Encode_QuotedPrintable_z
(
-> *BLOB to Encode :* Pointer
-> *ISO 8859-1 Encoding :* Longint
)

	Parameter	Type	Description
	BLOB to Encode	Pointer	Reference to BLOB containing text to be Quoted-Printable encoded

	Parameter	Type	Description
	ISO 8859-1 Encoding	Longint	qi for whether to apply ISO-8859-1 encoding before Quoted-Printable encoding

The method ***CODEC_Encode_QuotedPrintable_z*** encodes the referenced BLOB with Quoted-Printable encoding. Optionally, it can also be ISO 8859-1 encoded too. Details on Quoted-Printable encoding are available in RFC 1521, available at:

http://www.deepskytech.com/rfcs/rfc_1521.txt

BLOB to Encode is the a reference to a BLOB containing text to be Quoted-Printable encoded.

ISO 8859-1 Encoding is a qi for whether ISO 8859-1 encoding should be applied before the Quoted-Printable encoding. Pass a zero (0) to skip the ISO 8859-1 encoding, or a one (1) to apply this encoding.

Note: the method ***CODEC_Encode_QuotedPrintable_z*** was added in BASh v1.6.1.

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
	Text Value	Text	Text value to be URL encoded
	URL Encoded Text	Text	URL encoding of *Text Value*

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

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" within the Affix BASh document. 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.

Note: as of BASh v1.5.3, the listing of allowable ASCII byte values was moved to the Affix BASh document. Previous to this version, the allowable ASCII byte values list was stored in the resource fork of the structure file.

CODEC_Encode_URL_z

CODEC_Encode_URL_z *(BLOB to Encode)*

CODEC_Encode_URL_z

(

-> *BLOB to Encode :* Pointer

)

	Parameter	Type	Description
	BLOB to Encode	Pointer	Referened BLOB to URL encode

The method ***CODEC_Encode_URL_z*** returns a single referenced value URL encoded. Details on URL encoding are available in RFC 1738, available at:

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" within the Affix BASh document. 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.

Referenced BLOB is a pointer to a BLOB containing the data to be URL encoded. The encoding is done directly within *Referenced BLOB*.

Note: the method ***CODEC_Encode_URL_z*** was added in BASh v1.5.5.

CODEC_Encode_UUEncode_File

CODEC_Encode_UUEncode_File *(Referenced BLOB ; File Name)*

CODEC_Encode_UUEncode_File

(

-> *Referenced BLOB :* Text

-> *File Name :* Text

)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to BLOB containing file which is the be UUEncoded
	File Name	Text	File name of the document being UUEncoded

The method ***CODEC_Encode_UUEncode_File*** will encode a document referenced in a BLOB using the UUEncoded coding schema. Details on UUEncoding are available in the following document:

http://www.deepskytech.com/rfcs/uuencode.txt

Referenced BLOB is a pointer to a BLOB containing the document to be UUEncoded.

File Name is the name of the document which is to be UUEncoded.

Note: the method ***CODEC_Encode_UUEncode_File*** was added in BASh v1.5.9.

CODEC_Encode_UUEncode_x

CODEC_Encode_UUEncode_x *(Text to Encode)* => *Encoded Text*

CODEC_Encode_UUEncode_x
(
-> *Text to Encode :* Text
)
=> *Encoded Text :* Text

	Parameter	Type	Description
	Text to Encode	Text	Text value to be UUEncoded
	Encoded Text	Text	UUEncoded text value

The method ***CODEC_Encode_UUEncode_x*** will encode a text value using the UUEncode coding schema and return the encoded text value. Details on UUEncoding are available in the following document:

http://www.deepskytech.com/rfcs/uuencode.txt

Text to Encode is a text value to be encoded using the UUEncode coding schema in this method.

Encoded Text is the encoding of *Text to Encode* using the UUEncoding coding schema.

Note: the method ***CODEC_Encode_UUEncode_x*** was added in BASh v1.5.9.

CODEC_Encode_UUEncode_z

CODEC_Encode_UUEncode_z *(Referenced BLOB)*

CODEC_Encode_UUEncode_z
(

-> *Referenced BLOB :* Pointer
)

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to a BLOB which will be UUEncoded

The method ***CODEC_Encode_UUEncode_x*** will encode a text value using the UUEncode coding schema and return the encoded text value. Details on UUEncoding are available in the following document:

http://www.deepskytech.com/rfcs/uuencode.txt

Text to Encode is a text value to be encoded using the UUEncode coding schema in this method.

Encoded Text is the encoding of *Text to Encode* using the UUEncoding coding schema.

Note: the method ***CODEC_Encode_UUEncode_x*** was added in BASh v1.5.9.

CODEC_ERROR

CODEC_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***CODEC_ERROR*** acts as a callback method from within the CODEC module for errors that may occur. Any time an error condition is detected within the CODEC module, a call to the method ***CODEC_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 CODEC method which called the ***CODEC_ERROR*** method.

The ***CODEC_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 ***CODEC_ERROR*** was added in BASh v1.5.4.

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.

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

Hex2 and Hex8 Values

Hex2 and Hex8 values are represenations 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:

ASCII Value	Hex2 Value
10	0A
13	0D
27	1B
32	20
61	3D
63	3F
64	3F

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:

ASCII Value	Hex8 Value
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.

Hex6RGB Values

Hex6RGB values are six byte textual values used to represent an RGB color point. HEX6RGB values always follow the format of 'RRGGBB', where each two byte combination is the hexadecimal representation of the single, one byte color point. So, each two byte combination within a Hex6RGB value will be within the range of '00' (0x00) and 'FF' (0xFF).

Hex6RGB values are commonly used in HTML to represent color points for elements and objects. Hex6RGB values are commonly used for other implementations requiring a compact and human-readable color value.

The following table provides examples of many Hex6RGB values:

Integer RGB (R,G,B)	Hex RGB	Hex6RGB
(0,0,0)	(0,0,0)	000000
(3,4,5)	(3,4,5)	030405
(30,40,50)	(1E,28,32)	1E2832
(0,0,255)	(0,0,FF)	0000FF
(255,0,255)	(FF,0,FF)	FF00FF
(127,127,127)	(7F,7F,7F)	7F7F7F

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, by handled exclusively as longints. Conversion to and from text values should be done immediately surrounding native calls to 4th Dimension commands. The important 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 comparision 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.

CONV_ArrLong_to_LSTC

CONV_ArrLong_to_LSTC *(Referenced Longint Array ; Referenced BLOB ; Insertion Offset)*

CONV_ArrLong_to_LSTC
(
-> *Referenced Longint Array :* Pointer
-> *Referenced BLOB :* Pointer
-> *Insertion Offset :* Longint
)

	Parameter	Type	Description
	Referenced Longint Array	Pointer	Pointer to a longint array to be converted to a counted list
	Referenced BLOB	Pointer	Pointer to a BLOB to contain the new counted list
	Insertion Offset	Longint	Byte offset within *Referenced BLOB* to insert new counted list

The method ***CONV_ArrLong_to_LSTC*** will create a counted list within a specified offset of a referenced BLOB from values provided within a longint array.

A counted list has a specific format for storage of data within it. Basically, a counted list begins with a four byte value indicating how many elements are within the coounted list. Elements follow immediately and are individually stored as Pascal strings. For reference, Pascal strings have a one byte header indicating the number of following bytes that comprises the data for each individual element. A counted list is often a structure which is used in resources; for instance, a resource of type 'STR#' is actually a counted list.

Referenced Longint Array is a pointer to a longint array which will be converted into a counted list. All nonzero index

elements of the array will be placed into the counter list to be created.

Referenced BLOB is a pointer to a BLOB to place the counted list created into.

Insertion Offset is the offset within *Referenced BLOB* to insert the counted list at. Setting *Insertion Offset* to zero or less will insert the counted list at the beginning of *Referenced BLOB*. Setting *Insertion Offset* to MAXLONG will insert the counted list at the end of *Referenced BLOB*. The size of *Referenced BLOB* will be increased to accommodate the new counted list.

Note: the method ***CONV_ArrLong_to_LSTC*** was added in BASh v1.6.2.

CONV_ASCII_to_Hex2

CONV_ASCII_to_Hex2 *(ASCII Value) => Hex2 Value*

CONV_ASCII_to_Hex2
(
-> *ASCII Value :* Longint
)
-> *Hex2 Value :* String[2]

	Parameter	Type	Description
	ASCII Value	Longint	ASCII value of a single byte
	Hex2 Value	String[2]	Hex2 equivalent of *ASCII Value*

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

	Parameter	Type	Description
	ASCII Value	Longint	ASCII value of a single byte
	Printable Character	Text	Printable version of *ASCII Value*

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
	Text Value	Text	Text value to coerce
	Reference to Coerced Text	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
	Referenced Value	Pointer	Pointer to value to coerce into text
	Coerced Text	Text	Textually coerced value referenced by *Referenced Value*

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
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***CONV_ERROR*** acts as a callback method from within the CODEC module for errors that may occur. Any time an error condition is detected within the CODEC 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 CONV method which called 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_Hex6RGB_to_Longint_x

CONV_Hex6RGB_to_Longint_x *(Hex6RGB Value) => Longint RGB Value*

CONV_Hex6RGB_to_Longint_x

(

-> *Hex6RGB Value :* String[6]

)

=> *Longint RGB Value :* Longint

	Parameter	Type	Description
	Hex6RGB Value	String[6]	Hex6RGB value to be converted to longint equivalent
	Longint RGB Value	Longint	Longint RGB value equivalent to specified Hex6RGB value

The method ***CONV_Hex6RGB_to_Longint_x*** will convert a Hex6RGB value to the equivalent longint value. This allows for the much easier storage and manipulation within 4D of Hex6RGB values.

Hex6RGB Value is the Hex6RGB value to be converted to a longint.

Longint RGB Value is the equivalent *Hex6RGB Value* converted to a longint. The conversion is done very similarly as with Hex8 values except the top byte is considered a NULL byte for the conversion.

Note: the method ***CONV_Hex6RGB_to_Longint_x*** was added in BASh v1.8.0.

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
	Hex8 Value	Text	Hex8 value to convert to longint
	Longint Value	Longint	Longint conversion of *Hex8 Value*

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_Text *(Hex8 Value)* => *Text Value*

CONV_Hex8_to_Text
(
-> *Hex8 Value :* Text
)
=> *Text Value :* Text

	Parameter	Type	Description
	Hex8 Value	Text	Hex8 value to convert to text
	Text Value	Text	Text conversion of *Hex8 Value*

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
	Dotted IP Address	Text	Dotted IP address value to convert to longint
	Longint Value	Longint	Longint conversion of *Dotted IP Address*

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_Hex6RGB_x

CONV_Longint_to_Hex6RGB_x *(Longint RGB Value) => Hex6RGB Value*

CONV_Longint_to_Hex6RGB_x
(
-> *Longint RGB Value :* Longint
)
=> *Hex6RGB Value :* String[6]

	Parameter	Type	Description
	Longint RGB Value	Longint	Longint value to convert to Hex6RGB value
	Hex6RGB Value	String[6]	Converted longint value

The method ***CONV_Longint_to_Hex6RGB_x*** will convert a given longint value to an equivalent Hex6RGB value.

Longint RGB Value is the longint value to convert to Hex6RGB. The highest byte of this value will be disgarded in the conversion.

Hex6RGB Value is the converted longint value stored in a Hex6RGB format.

Note: the method ***CONV_Longint_to_Hex6RGB_x*** was added in BASh v1.8.0.

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
	Longint Value	Longint	Longint value to convert to Hex8
	Hex8 Value	Text	Hex8 conversion of *Longint Value*

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
	Longint Value	Longint	Longint value to covert to a Dotted IP value
	Dotted IP Address	Text	Dotted IP conversion of *Longint Value*

The method ***CONV_Longint_to_IP*** converts a longint to a complete and well formed 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 :* String[4]

	Parameter	Type	Description
	Longint Value	Longint	Longint value to covert to a Type value
	Type Value	String[4]	Type conversion of *Longint Value*

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 by 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 _LSTC_to _ArrLong

CONV _LSTC_to _ArrLong *(Referenced BLOB ; Referenced Longint Array ; Starting Offset ; qi Remove Counted List) => Counted List Size*

CONV _LSTC_to _ArrLong

(

-> Referenced BLOB : Pointer
-> Referenced Longint Array : Pointer
-> Starting Offset : Longint
-> qi Remove Counted List : Longint

)

=> Counted List Size : Longint

	Parameter	Type	Description
	Referenced BLOB	Pointer	Pointer to a BLOB containing the counted list to parse
	Referenced Longint Array	Pointer	Pointer to a longint array to be contain parsed counted list
	Starting Offset	Longint	Starting offset within *Referenced BLOB* where the counted list object begins
	qi Remove Counted List	Longint	Longint value to indicate whether the counted list is to be removed from *Referenced BLOB*

	Parameter	Type	Description
	Counted List Size	Longint	Number of items in the counted list

The method ***CONV_LSTC_to_ArrLong*** will parse the values from a counted list contained within a referenced BLOB and return the elements of the counted list and the count of the number of elements in the counted list. Elements parsed from the counted list will be assumed to be 32 bit integer values in 4D and converted as such.

A counted list has a specific format for storage of data within it. Basically, a counted list begins with a four byte value indicating how many elements are within the coounted list. Elements follow immediately and are individually stored as Pascal strings. For reference, Pascal strings have a one byte header indicating the number of following bytes that comprises the data for each individual element. A counted list is often a structure which is used in resources; for instance, a resource of type 'STR#' is actually a counted list.

Referenced BLOB is a pointer to a BLOB containing the counted list to be parsed.

Referenced Longint Array is a pointer to a longint array which will be filled with the elements of the counted list typed as 32 bit integers.

Starting Offset is the offset within *Referenced BLOB* which the counted list begins at.

qi Remove Counted List is a longint value for whether the counted list is to removed from *Referenced BLOB* after the values have been parsed from it.

Counted List Size is the number of elements parsed from the counted list.

Note: the method ***CONV_LSTC_to_ArrLong*** was added in BASh v1.6.2.

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
	Text Value	Text	Text value to convert to Hex8
	Hex8 Value	Text	Hex8 conversion of *Text Value*

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
	Text Value	Text	Text value to convert to longint
	Longint Value	Longint	Longint conversion of *Text Value*

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 :* String[4]

)

=> *Longint Value :* Longint

	Parameter	Type	Description
	Type Value	String[4]	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.

CRYPT Module

The CRYPT module provides basic encryption and decryption routines directly in 4th Dimension. With the first release of the CRYPT module, implementation of TEA is the sole purpose. With future releases of the BASh component, other encryption algorithms may be added.

Note: the CRYPT module was added in BASh v1.5.5.

Tiny Encryption Algorithm (TEA)

The Tiny Encryption Algorithm is one of the fastest and most efficient cryptographic algorithms in existence. It was developed by David Wheeler and Roger Needham at the Computer Laboratory of Cambridge University. It is a Feistel cipher which uses operations from mixed (orthogonal) algebraic groups - XORs and additions in this case. It encrypts 64 data bits at a time using a 128-bit key. It seems highly resistant to differential cryptanalysis, and achieves complete diffusion (where a one bit difference in the plaintext will cause approximately 32 bit differences in the ciphertext) after only six rounds. Performance on a modern desktop computer or workstation is very impressive.

You can obtain a copy of Roger Needham and David Wheeler's original paper describing TEA, from the Security Group ftp site at the world-famous Cambridge Computer Laboratory at Cambridge University. There is also a paper on extended variants of TEA which addresses a couple of minor weaknesses (irrelevant in almost all real-world applications), and introduces a block variant of the algorithm which can be even faster in some circumstances.

TEA is very secure. There have been no known successful cryptanalyses of TEA. It is believed (by James Massey) to be as secure as the IDEA algorithm, designed by Massey and Xuejia Lai. It uses the same mixed algebraic groups technique as IDEA, but it is much simpler, hence faster. Also, it is public domain, whereas IDEA is patented by Ascom-Tech AG in Switzerland. IBM’s Don Coppersmith and Massey independently showed that mixing operations from orthogonal algebraic groups performs the diffusion and confusion functions that a traditional block cipher would implement with P- and S-boxes. As a simple drop-in encryption routine, it is great. The code is lightweight and portable enough to be used just about anywhere. It even makes a great random number generator for Monte Carlo simulations and the like. The minor weaknesses identified by David Wagner at Berkeley are unlikely to have any impact in the real world, and you can always implement the new variant TEA which addresses them. If you want a low-overhead end-to-end cipher (for real-time data, for example), then TEA fits the bill.

The 128 bit key used in the TEA is handled in 4th Dimension through four (4) longint values. The spread use of the four longint values provides a full 128 bit key that maintains all of the advantages afforded by use of the TEA. A single routine for generating a full 128 bit key from a block of source text is provided merely for the convenience of the developer.

When encrypting and decrypting variables in 4D, it is worth noting that BLOB and text variables are not interchangeable. If a TEA encryption is run with a BLOB, the decryption must also be done with a BLOB. The same holds for text variables. The nature of the TEA algorithm and the limitations of 4D text variables makes this a necessary limitation of the TEA routines. As of BASh v1.6.0, the method ***CRYPT_Convert_TEA_x_to_z*** has been included to move encrypted 4D text variables into 4D BLOB variables.

You can learn more about the TEA algorithm online. There are plenty of sites which provide details and source for the Tiny Encryption Algorithm. One site which we found to be most help is:

http://vader.brad.ac.uk/tea/tea.shtml

Data Encryption Standard (DES)

DES was introduced in the mid 1970s by the National Bureau of Standards (NBS). It now has widespread use in many different applications, from banking to personal data encryption.

DES uses a 64-bit secret key to encrypt data in 64-bit data blocks. Only 56 bits of the key are used; every 8th bit is used as a parity bit for that byte. The algorithm uses a combination of permutation and substitution, as well as permutation and rotations of the secret key. The secret key is required to both encrypt and decrypt data.

While the 56-bit encryption was considered pretty strong in the 1970s, tremendous advances in computing power have rendered DES susceptible to brute-force attacks. As a result, variations such as Triple-DES have appeared to offer increased security.

DES also has several modes of operation. Electronic Code Book (DES) is the basic original DES algorithm, which operates on each 64-bit block of data individually. Other modes, such as Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB) add additional processing making the final result a more complex encryption of the entire data.

There is plenty of information about DES online. The first link is the official DES specification. The second link is to a description which we found to be most helpful.

http://www.itl.nist.gov/fipspubs/fip46-2.htm

http://www.orlingrabbe.com/des.htm

CRYPT_Convert_TEA_x_to_z

CRYPT_Convert_TEA_x_to_z *(TEA Encrypted Text Value ; Referenced BLOB ; Source Length ; First TEA Key ; Second TEA Key ; Third TEA Key ; Fourth TEA Key)*

CRYPT_Convert_TEA_x_to_z
(
-> *TEA Encrypted Text Value :* Text
-> *Referenced BLOB :* Pointer
-> *Source Length :* Longint
-> *First TEA Key :* Longint
-> *Second TEA Key :* Longint
-> *Third TEA Key :* Longint
-> *Fourth TEA Key :* Longint
)

	Parameter	Type	Description
	TEA Encrypted Text Value	Text	Text value previously encrypted using CRYPT module TEA routines
	Referenced BLOB	Pointer	Pointer to BLOB to hold TEA encrypted value
	Source Length	Longint	Number of bytes in original, unencrypted source text
	First TEA Key	Longint	First TEA key which was used to encrypt text
	Second TEA Key	Longint	Second TEA key which was used to encrypt text
	Third TEA Key	Longint	Third TEA key which was used to encrypt text
	Fourth TEA Key	Longint	Fourth TEA key which was used to encrypt text

The method ***CRYPT_Convert_TEA_x_to_z*** will convert a TEA encrypted 4D text variable into a TEA encrypted 4D BLOB variable.

TEA Encrypted Text Value is the previously TEA encrypted text value which is to be moved into a 4D BLOB variable.

Referenced BLOB is a pointer to a BLOB to hold the TEA encrypted value.

Source Length is the number of characters in the original source text before is was encrypted using the TEA routines.

First TEA Key is the first block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Second TEA Key is the second block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Third TEA Key is the third block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Fourth TEA Key is the fourth and final block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Note: the method ***CRYPT_Convert_TEA_x_to_z*** was added in BASh v1.6.0

CRYPT_Decrypt_DES_x

CRYPT_Decrypt_DES_x *(DES Encrypted Text ; Key First Half ; Key Second Half) => Decrypted Text*

CRYPT_Decrypt_DES_x
(
-> *DES Encrypted Text :* Text
-> *Key First Half :* Longint
-> *Key Second Half :* Longint
)

=> *Decrypted Text : Longint*

	Parameter	Type	Description
	DES Encrypted Text	Text	Encrypted text which is to be decrypted
	Key First Half	Longint	First 4 bytes of DES secret key
	Key Second Half	Longint	Second 4 bytes of DES secret key
	Decrypted Text	Text	Supplied source text decrypted using DES algorithm and supplied DES key

The method ***CRYPT_Decrypt_DES_x*** will decrypt a supplied text value using the supplied key for the decryption. The decryption is done using the DES encryption algorithm, documented above.

DES Encrypted Text is the text value which is to be decrypted using the DES encryption algorithm.

Key First Half is the first block of 32 bits (MSB to LSB) of the DES key used to encrypt the data using the DES encryption algorithm.

Key Second Half is the second block of 32 bits (MSB to LSB) of the DES key used to encrypt the data using the DES encryption algorithm.

Decrypted Text is the decrypted text of *DES Encrypted Text* using the supplied key for the DES encryption algorithm. No checks are done for whether the supplied key was actually used originally for the encryption; rather, the decryption processing is run blind using the supplied key and the result is then made available.

Note: the method ***CRYPT_Decrypt_DES_x*** was added in BASh v1.7.0.

CRYPT_Decrypt_DES_z

CRYPT_Decrypt_DES_z *(Referenced Encrypted BLOB ; Referenced Decrypted BLOB ; Key First Half ; Key Second Half)*

CRYPT_Decrypt_DES_z

(

-> *Referenced Encrypted BLOB :* Pointer
-> *Referenced Decrypted BLOB :* Pointer
-> *Key First Half :* Longint
-> *Key Second Half :* Longint

)

	Parameter	Type	Description
	Referenced Encrypted BLOB	Pointer	Referenced source BLOB which is to be decrypted
	Referenced Decrypted BLOB	Pointer	Referenced BLOB to hold source BLOB decrypted using DES algorithm and supplied DES key
	Key First Half	Longint	First 4 bytes of DES secret key
	Key Second Half	Longint	Second 4 bytes of DES secret key

The method ***CRYPT_Decrypt_DES_z*** will decrypt a referenced BLOB using the supplied key for the decryption. The decryption is done using the DES encryption algorithm, documented above.

Referenced Encrypted BLOB is a reference to a BLOB which is to be decrypted using the DES encryption algorithm.

Referenced Decrypted BLOB is a reference to a BLOB to contain the decrypted value. *Referenced Encrypted BLOB* will be decrypted using the supplied key for the DES encryption algorithm. No checks are done for whether the supplied key was actually used originally for the encryption; rather, the decryption processing is run blind using the supplied key and the result is then made available.

Key First Half is the first block of 32 bits (MSB to LSB) of the DES key used to encrypt the data using the DES encryption algorithm.

Key Second Half is the second block of 32 bits (MSB to LSB) of the DES key used to encrypt the data using the DES encryption algorithm.

Note: the method ***CRYPT_Decrypt_DES_z*** was added in BASh v1.7.0.

CRYPT_Decrypt_TEA_x

CRYPT_Decrypt_TEA_x *(TEAEncrypted Text ; First TEA Key ; Second TEA Key ; Third TEA Key ; Fourth TEA Key) => Decrypted Text*

CRYPT_Decrypt_TEA_x
(
-> *TEA Encrypted Text :* Text
-> *First TEA Key :* Longint
-> *Second TEA Key :* Longint
-> *Third TEA Key :* Longint
-> *Fourth TEA Key :* Longint
)
=> *Decrypted Text : Longint*

	Parameter	Type	Description
	TEA Encrypted Text	Text	Encrypted text which is to be decrypted
	First TEA Key	Longint	First TEA key which was used to encrypt text
	Second TEA Key	Longint	Second TEA key which was used to encrypt text
	Third TEA Key	Longint	Third TEA key which was used to encrypt text
	Fourth TEA Key	Longint	Fourth TEA key which was used to encrypt text
	Decrypted Text	Text	Supplied source text decrypted using TEA algorithm and supplied TEA keys

The method ***CRYPT_Decrypt_TEA_x*** will decrypt a supplied text value using the supplied keys for the decryption. The decryption is done using the tiny encryption algorithm, documented above.

Encrypted Text is the text value which is to be decrypted using the tiny encryption algorithm.

First TEA Key is the first block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Second TEA Key is the second block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Third TEA Key is the third block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Fourth TEA Key is the fourth and final block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Decrypted Text is the decrypted text of *Encrypted Text* using the supplied keys for the tiny encryption algorithm. No checks are done for whether the supplied keys were actually used originally for the encryption; rather, the decryption processing is run blind using the supplied keys and the result is then made available.

Note: the method ***CRYPT_Decrypt_TEA_x*** was added in BASh v1.5.5.

CRYPT_Decrypt_TEA_z

CRYPT_Decrypt_TEA_z *(Referenced Encrypted BLOB ; Referenced Decrypted BLOB ; First TEA Key ; Second TEA Key ; Third TEA Key ; Fourth TEA Key)*

CRYPT_Decrypt_TEA_z
(
-> *Referenced Encrypted BLOB :* Pointer
-> *Referenced Decrypted BLOB :* Pointer
-> *First TEA Key :* Longint
-> *Second TEA Key :* Longint
-> *Third TEA Key :* Longint
-> *Fourth TEA Key :* Longint
)

	Parameter	Type	Description
	Referenced Encrypted BLOB	Pointer	Referenced source BLOB which is to be decrypted
	Referenced Decrypted BLOB	Pointer	Referenced BLOB to hold source BLOB decrypted using TEA algorithm and supplied TEA keys
	First TEA Key	Longint	First TEA key which was used to encrypt text
	Second TEA Key	Longint	Second TEA key which was used to encrypt text
	Third TEA Key	Longint	Third TEA key which was used to encrypt text
	Fourth TEA Key	Longint	Fourth TEA key which was used to encrypt text
	Decrypted Text	Text	Supplied source text decrypted using TEA algorithm and supplied TEA keys

The method ***CRYPT_Decrypt_TEA_z*** will decrypt a referenced binary value using the supplied keys for the decryption. The decryption is done using the tiny encryption algorithm, documented above.

Referenced Encrypted BLOB is a pointer to the binary value which is to be decrypted using the tiny encryption algorithm.

Referenced Decrypted BLOB is pointer to the BLOB to contain the decrypted *Referenced Encrypted BLOB* using the supplied keys for the tiny encryption algorithm. No checks are done for whether the supplied keys were actually used originally for the encryption; rather, the decryption processing is run blind using the supplied keys and the result is then made available.

First TEA Key is the first block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Second TEA Key is the second block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Third TEA Key is the third block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Fourth TEA Key is the fourth and final block of 32 bits (MSB to LSB) of the TEA key used to encrypt the data using the tiny encryption algorithm.

Note: the method ***CRYPT_Decrypt_TEA_z*** was added in BASh v1.5.5.

CRYPT_Encrypt_DES_x

CRYPT_Encrypt_DES_x *(Source Text ; Key First Half ; Key Second Half) => Encrypted Text*

CRYPT_Encrypt_DES_x
(
-> *Source Text :* Text
-> *Key First Half :* Longint
-> *Key Second Half :* Longint
)
=> *Encrypted Text : Longint*

	Parameter	Type	Description
	Source Text	Text	Source text which is to be encrypted
	Key First Half	Longint	First 4 bytes of DES secret key
	Key Second Half	Longint	Second 4 bytes of DES secret key
	Encrypted Text	Text	Supplied source text encrypted using DES algorithm and supplied DES key

The method ***CRYPT_Encrypt_DES_x*** will encrypt a supplied text value using the supplied key for the encryption. The encryption is done using the DES encryption algorithm, documented above.

Source Text is the text value which is to be encrypted using the DES encryption algorithm.

Key First Half is the first block of 32 bits (MSB to LSB) of the DES key to use to encrypt the data using the DES encryption algorithm.

Key Second Half is the second block of 32 bits (MSB to LSB) of the DES key to use to encrypt the data using the DES encryption algorithm.

Encrypted Text is the encrypted *Source Text* using the supplied key for the DES encryption algorithm.

Note: the method ***CRYPT_Encrypt_DES_x*** was added in BASh v1.7.0.

CRYPT_Encrypt_DES_z

CRYPT_Encrypt_DES_z *(Referenced Source BLOB ; Referenced Encrypted BLOB ; Key First Half ; Key Second Half)*

CRYPT_Encrypt_DES_z
(
-> *Referenced Source BLOB :* Pointer
-> *Referenced Encrypted BLOB :* Pointer
-> *Key First Half :* Longint
-> *Key Second Half :* Longint
)

	Parameter	Type	Description
	Referenced Source BLOB	Pointer	Referenced source BLOB which is to be encrypted
	Referenced Encrypted BLOB	Pointer	Referenced BLOB to hold source BLOB encrypted using DES algorithm and supplied DES key
	Key First Half	Longint	First 4 bytes of DES secret key
	Key Second Half	Longint	Second 4 bytes of DES secret key

The method ***CRYPT_Encrypt_DES_z*** will encrypt a referenced BLOB using the supplied key for the decryption. The encryption is done using the DES encryption algorithm, documented above.

Referenced Source BLOB is a reference to a BLOB which is to be encrypted using the DES encryption algorithm.

Referenced Encrypted BLOB is a reference to a BLOB to contain the encrypted value of *Referenced Source BLOB*.

Key First Half is the first block of 32 bits (MSB to LSB) of the DES key to use to encrypt the data using the DES encryption algorithm.

Key Second Half is the second block of 32 bits (MSB to LSB) of the DES key to use to encrypt the data using the DES encryption algorithm.

Note: the method ***CRYPT_Encrypt_DES_z*** was added in BASh v1.7.0.

CRYPT_Encrypt_TEA_x

CRYPT_Encrypt_TEA_x *(Source Text ; First TEA Key ; Second TEA Key ; Third TEA Key ; Fourth TEA Key) => Encrypted Text*

CRYPT_Encrypt_TEA_x
(
-> Source Text : Text
-> First TEA Key : Longint
-> Second TEA Key : Longint
-> Third TEA Key : Longint
-> Fourth TEA Key : Longint
)
=> Encrypted Text : Longint

	Parameter	Type	Description
	Source Text	Text	Source text which is to be encrypted
	First TEA Key	Longint	First TEA key to use to encrypt text
	Second TEA Key	Longint	Second TEA key to use to encrypt text
	Third TEA Key	Longint	Third TEA key to use to encrypt text
	Fourth TEA Key	Longint	Fourth TEA key to use to encrypt text

	Parameter	Type	Description
	Encrypted Text	Text	Supplied source text encrypted using TEA algorithm and supplied TEA keys

The method ***CRYPT_Encrypt_TEA_x*** will encrypt a supplied text value using the supplied keys for the encryption. The encryption is done using the tiny encryption algorithm, documented above.

Source Text is the text value which is to be encrypted using the tiny encryption algorithm.

First TEA Key is the first block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Second TEA Key is the second block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Third TEA Key is the third block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Fourth TEA Key is the fourth and final block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Encrypted Source Text is the encrypted text of *Source Text* using the supplied keys for the tiny encryption algorithm.

Note: the method ***CRYPT_Encrypt_TEA_x*** was added in BASh v1.5.5.

CRYPT_Encrypt_TEA_z

CRYPT_Encrypt_TEA_z *(Referenced Source BLOB ; Referenced Encrypted BLOB ; First TEA Key ; Second TEA Key ; Third TEA Key ; Fourth TEA Key)*

CRYPT_Encrypt_TEA_z
(

-> *Referenced Source BLOB :* Pointer
-> *Referenced Encrypted BLOB :* Pointer
-> *First TEA Key :* Longint
-> *Second TEA Key :* Longint
-> *Third TEA Key :* Longint
-> *Fourth TEA Key :* Longint
)

	Parameter	Type	Description
	Referenced Source BLOB	Pointer	Referenced source BLOB which is to be encrypted
	Referenced Encrypted BLOB	Pointer	Referenced BLOB to hold source BLOB encrypted using TEA algorithm and supplied TEA keys
	First TEA Key	Longint	First TEA key to use to encrypt text
	Second TEA Key	Longint	Second TEA key to use to encrypt text
	Third TEA Key	Longint	Third TEA key to use to encrypt text
	Fourth TEA Key	Longint	Fourth TEA key to use to encrypt text

The method ***CRYPT_Encrypt_TEA_z*** will encrypt a referenced binary value using the supplied keys for the encryption. The encryption is done using the tiny encryption algorithm, documented above.

Referenced Source BLOB is a pointer to the binary value which is to be encrypted using the tiny encryption algorithm.

Referenced Encrypted BLOB is pointer to the BLOB to contain the encrypted *Referenced Source BLOB* using the supplied keys for the tiny encryption algorithm.

First TEA Key is the first block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Second TEA Key is the second block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Third TEA Key is the third block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Fourth TEA Key is the fourth and final block of 32 bits (MSB to LSB) of the TEA key to be used to encrypt the data using the tiny encryption algorithm.

Note: the method ***CRYPT_Encrypt_TEA_z*** was added in BASh v1.5.5.

CRYPT_Get_TEA_Keys_f_Text

CRYPT_Get_TEA_Keys_f_Text *(Source Text Key ; Referenced First TEA Key ; Referenced Second TEA Key ; Referenced Third TEA Key ; Referenced Fourth TEA Key)*

CRYPT_Get_TEA_Keys_f_Text
(
-> *Source Text Key :* Text
-> *Referenced First TEA Key :* Pointer
-> *Referenced Second TEA Key :* Pointer
-> *Referenced Third TEA Key :* Pointer
-> *Referenced Fourth TEA Key :* Pointer
)

	Parameter	Type	Description
	Source Text Key	Text	Source key text to be used to generated TEA encryption keys
	Referenced First TEA Key	Pointer	Referenced first TEA key calculated from supplied source text
	Referenced Second TEA Key	Pointer	Referenced second TEA key calculated from supplied source text
	Referenced Third TEA Key	Pointer	Referenced third TEA key calculated from supplied source text
	Referenced Fourth TEA Key	Pointer	Referenced fourth TEA key calculated from supplied source text

The method ***CRYPT_Get_TEA_Keys_f_Text*** will generate a set of valid tiny encryption algorithm 32 bit blocks (long-ints) from a block of supplied text for use as keys into the tiny encryption algorithm.

The keys generated are unique to the supplied source text and are fully regenerative. The keys are generated using a modi-

fied version of the MD5 hash algorithm. the generated keys are reasonably random to provide affordable security using the tiny encryption alogorithm without compromising randomness.

Source Key Text is the text value to use to generate the TEA keys. Repeated use of the exact same text value will produce the same TEA keys. *Source Key Text* must be at least 22 bytes long and can consist only of uppercase letters, lowercase letters, numerals, space, and hyphens. All other byte values are invalid.

Referenced First TEA Key is a pointer to a longint to hold the first 32 bit block (MSB to LSB) of the 128 bit TEA key generated in this method.

Referenced Second TEA Key is a pointer to a longint to hold the second 32 bit block (MSB to LSB) of the 128 bit TEA key generated in this method.

Referenced Third TEA Key is a pointer to a longint to hold the third 32 bit block (MSB to LSB) of the 128 bit TEA key generated in this method.

Referenced Fourth TEA Key is a pointer to a longint to hold the fourth and final 32 bit block (MSB to LSB) of the 128 bit TEA key generated in this method.

Note: the method ***CRYPT_Get_TEA_Keys_f_Text*** was added in BASh v1.5.5.

DATE Module

The DATE module provides very basic date operations which are currently needed in 4th Dimension. To a large extent, the DATE module acts as a bridge to other modules and functionality commonly used. Though it is the preference of the authors to utilize the DTS system exclusively for handling date and time values in 4D, the methods provided in the DATE module make this task much easier.

Note: the DATE module was added in BASh v1.5.4.

DATE_ERROR

DATE_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***DATE_ERROR*** acts as a callback method from within the DATE module for errors that may occur. Any time an error condition is detected within the DATE module, a call to the method ***DATE_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 DATE method which call the ***DATE_ERROR*** method.

The ***DATE_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: this method was added in BASh v1.8.0.

DATE_Get_Age

DATE_Get_Age *(Birth Date ; Source Date) => Age in Years*

DATE_Get_Age
(
-> *Birth Date :* Date
-> *Source Date :* Date
)
=> *Age in Years :* Longint

	Parameter	Type	Description
	Birth Date	Date	"birth" date to calculate age from
	Source Date	Date	date to calculate age to
	Age in Years	Longint	Age in years from *Birth Date* to *Source Date*

The method ***DATE_Get_Age*** returns the age in years between two specified dates.

Birth Date is the date value to calculate *Age in Years* from.

Source Date is the date value to calculate *Age in Years* to, from *Birth Date*. For example, use **Current Date** to calculate *Age in Years* from *Birth Date* to today.

Age in Years is the difference in years between *Birth Date* and *Source Date*. If *Source Date* is before *Birth Date*, *Age in Years* will be less than zero.

Note: the method ***DATE_Get_Age*** was added in BASh v1.7.0.

DATE_Get_ISO8601_GMT

DATE_Get_ISO8601_GMT *(DTS Value) => ISO 8601 DateTime Value*

DATE_Get_ISO8601_GMT
(
-> *DTS Value :* String[14]
)
=> *ISO 8601 DateTime Value :* Text

	Parameter	Type	Description
	DTS Value	String[14]	DTS value to convert to ISO 8601 formatted datetime value
	ISO 8601 DateTime Value	Text	ISO 8601 formatted datetime value

The method ***DATE_Get_ISO8601_GMT*** return a properly formatted ISO 8601 datetime stamp corresponding to a specified DTS value.

DTS Value is a valid DTS value which will be used to generate a properly formatted ISO 8601 datetime stamp.

ISO 8601 DateTime Value is a properly formatted ISO 8601 datetime stamp for the value *DTS Value*, assumed to be a GMT specified DTS value. ISO 8601 formatted values have the following format:

yyyy-mm-ddThh:mm:ssZ

Note: the method ***DATE_Get_ISO8601_GMT*** was added in BASh v1.5.8.

DATE_Get_MonthDay_x

DATE_Get_MonthDay_x *(Source Date) => Month Day String*

DATE_Get_MonthDay_x

(

-> *Source Date :* Date

)

=> *Month Day String :* Text

	Parameter	Type	Description
	Source Date	Date	Date to use to generate *Month Day String*
	Month Day String	Text	Formatted name of month and number of day for *Source Date*

The method ***DATE_Get_MonthDay_x*** will return text for the month name and day number for a specified source date. The returned "Month Day" text is the textual name of the month with the coerced numerical day value appended (a single space separating the values).

Source Date is the input date to use to generate the month day textual value.

Month Day String is the generate month day textual value corresponding to *Source Date*.

Note: the method ***DATE_Get_MonthDay_x*** was added in BASh v1.8.0.

DATE_Get_MonthEndDay_f_YM

DATE_Get_MonthEndDay_f_YM *(Year ; Month) => Last Day of Month*

DATE_Get_MonthEndDay_f_YM

(

-> *Year :* Longint

-> *Month :* Longint

)

=> *Last Day of Month :* Longint

	Parameter	Type	Description
	Year	Longint	Year value

	Parameter	Type	Description
	Month	Longint	Month value
	Last Day of Month	Longint	Last day of month specified by *Year* and *Month*

The method ***DATE_Get_MonthEndDay_f_YM*** will return the last day of a month specified by year and month values.

Year is the year value of the month to find the last day of.

Month is the month value of the month to find the last day of.

Last Day of Month is the day of the last day of the month specified by the *Year* and *Month* values.

Note: the method ***DATE_Get_MonthEndDay_f_YM*** was added in BASh v1.6.2.

DATE_Get_MonthEnd_f_Date

DATE_Get_MonthEnd_f_Date *(Source Date) => Month End Date*

DATE_Get_MonthEnd_f_Date
(
-> *Source Date :* Date
)
=> *Month End Date :* Date

	Parameter	Type	Description
	Source Date	Date	Date value from which to get last day of month
	Month End Date	Date	Date of last day of the month of *Source Date*

The method *DATE_Get_MonthEnd_f_Date* will return the date of the last day of the month the specified date.

Source Date is the date value specifying the month for which to get the last day.

Month End Date is the date of the last day of the month of *Source Date*.

Note: the method ***DATE_Get_MonthEnd_f_Date*** was added in BASh v1.6.2.

DATE_Get_MonthName_Full

DATE_Get_MonthName_Full *(Month Number) => Full Month Name*

DATE_Get_MonthName_Full
(
-> *Month Number :* Longint
)
-> *Full Month Name :* Text

	Parameter	Type	Description
	Month Number	Longint	Numeric value for the month number, between 1 and 12 inclusive
	Full Month Name	Text	Text of full month name

The method ***DATE_Get_MonthName_Full*** will return the full name of the month for a specified month number.

Month Number is the month number to return the full month name for. The month number is the sequential number of the month in the calendar year, an integer value (longint in 4D) between 1 and 12 inclusive.

Full Month Name is the full name of the month corresponding to *Month Number*. The full name of the month is retrieved from the 'STR#' resource ID 11 within the current 4D application.

Note: the method ***DATE_Get_MonthName_Full*** was added in BASh v1.8.0.

DATE_Get_MonthName_Short

DATE_Get_MonthName_Short *(Month Number) => Short Month Name*

DATE_Get_MonthName_Short
(
-> *Month Number :* Longint
)
-> *Short Month Name :* Text

	Parameter	Type	Description
	Month Number	Longint	Numeric value for the month number, between 1 and 12 inclusive
	Short Month Name	Text	Text of short month name

The method ***DATE_Get_MonthName_Short*** will return the short name of the month for a specified month number, corresponding to the value for months specified in RFC 822.

Month Number is the month number to return the short month name for. The month number is the sequential number of the month in the calendar year, an integer value (longint in 4D) between 1 and 12 inclusive.

Short Month Name is the short name of the month corresponding to *Month Number*. The short name of the month corresponds to the month values given in RFC 822 (basically, capital case English months abbreviated to only three bytes in length).

Note: the method ***DATE_Get_MonthName_Short*** was added in BASh v1.8.0.

DATE_Get_Month_f_ShortName

DATE_Get_Month_f_ShortName *(Month Short Name) => Month Number*

DATE_Get_Month_f_ShortName
(
-> *Month Short Name :* Text
)
=> *Month Number :* Longint

	Parameter	Type	Description
	Month Short Name	Text	Short textual name of month (see RFC 822)
	Month Number	Longint	Month number corresponding to *Month Short Name*

The method ***DATE_Get_Month_f_ShortName*** will return the month number for a specified textual month short name.

Month Short Name is the textual short month name to get the month number for. See RFC 822 for details on short month names, available from:

http://www.deepskytech.com/rfcs/rfc_0822.txt

Month Number is the month number corresponding to *Month Short Name*. If *Month Short Name* is not a valid short month name then *Month Number* will be set to negative one (-1).

Note: the method ***DATE_Get_Month_f_ShortName*** was added in BASh v1.5.8.

DATE_Get_RFC_GMT

DATE_Get_RFC_GMT *(Local Date ; Local Time ; Referenced GMT Date ; Referenced GMT Time) => RFC Formatted GMT Date Time Stamp*

DATE_Get_RFC_GMT
(
-> *Local Date :* Date
-> *Local Time :* Time
-> *Referenced GMT Date :* Pointer
-> *Referenced GMT Time :* Pointer
)
=> *RFC Formatted GMT Date Time Stamp :* Text

	Parameter	Type	Description
	Local Date	Date	Date value to use for calculations
	Local Time	Time	Time value to use for calculations

	Parameter	Type	Description
	Referenced GMT Date	Pointer	Reference to date variable to hold *Local Date* in GMT
	Referenced GMT Time	Pointer	Reference to time variable to hold *Local Time* in GMT
	RFC Formatted GMT Date Time Stamp	Text	RFC formatted date time stamp for for *Local Date* and *Local Time*

The method ***DATE_Get_RFC_GMT*** will format specified date and time values as a standard RFC formatted date time stamp. The time zone on the local machine will be used to convert the date and time values to GMT. The GMT converted date and time values can also be returned in separated referenced variables. Specifications for the RFC formatted date time stamps are available in RFC 822, available at:

http://www.deepskytech.com/rfcs/rfc_0822.txt

Local Date is the local date value to use for the conversions.

Local Time is the local time value to use for the conversions.

Referenced GMT Date is a reference to a date variable to contain the the GMT equivalent of *Local Date* and *Local Time*. The time zone of the local machine will be used to make the conversion. If *Referenced GMT Date* is NULL, then no converted date value will be returned.

Referenced GMT Time is a reference to a time variable to contain the the GMT equivalent of *Local Date* and *Local Time*. The time zone of the local machine will be used to make the conversion. If *Referenced GMT Time* is NULL, then no converted time value will be returned.

RFC Formatted GMT Date Time Stamp is the string of a properly formatted date time stamp (as specified by RFC 822). *Local Date* and *Local Time* will be converted to GMT according to thetime zone settings of the local machine, and then these values will be used to construct the properly formatted string.

Note: the method ***DATE_Get_RFC_GMT*** was added in BASh v1.5.4.

DATE_Make_from_Values

DATE_Make_from_Values *(Year Value ; Month Value ; Day Value) => Date Value*

DATE_Make_from_Values
(
-> *Year Value :* Longint
-> *Month Value :* Longint
-> *Day Value :* Longint
)
=> *Date Value :* Date

	Parameter	Type	Description
	Year Value	Longint	Full numeric year value
	Month Value	Longint	Full numeric month value (between one and twelve, in most cases)
	Day Value	Longint	Full numeric day value
	Date Value	Date	Properly formatted 4D date value for values specified

The method ***DATE_Make_from_Values*** will create a properly formatted 4D date variable with the values specified. The actual date format settings for the local machine will be bypassed, allowing for the creation of a properly formatted 4D date value regardless of platform, machine, and/or local user settings. The resulting date value created will not be normalized.

Year Value is the full numeric year value to be used to create the 4D date value. No abbreviations are allowed in this value, as the date variable is constructed directly from this value (e.g. 99 is interpreted as the year 99, not the year 1999).

Month Value is the full numeric month value to be used to create the 4D date value. The native 4D constant groups Days and Months can be used for values, though there is no restriction that this value be between one (1) and twelve (12).

Day Value is the full numeric day value to be used to create the 4D date value. *Day Value* is considered to be the day of

the month. Often, this value will inclusively be between one (1) and the number of days in the specified month. However, there are no restrictions placed on the value of *Day Value*.

Date Value is the properly constructed 4D date value for the values specified in *Year Value*, *Month Value*, and *Day Value*. *Date Value* is not normalized; rather, the specified values are used to directly construct the 4D date value. Machine, OS, and local user settings are bypassed, allowing for the creation of a proper 4D date value regardless of the date format settings on the local machine.

Example:

The following table shows different values passed to the method ***DATE_Make_from_Values*** and the results which are returned. The results are displayed using the format "yyyy/mm/dd" in the table. The far right column in the table contains the normalized results (see the note, below, about nromalizing 4D date values).

Y	M	D	Result	Result (Norm.)
1999	4	15	1999/04/15	1999/04/15
2000	2	29	2000/02/29	2000/02/29
2000	2	30	2000/02/30	2000/03/01
2001	12	31	2001/12/31	2001/12/31
2001	12	35	2001/12/35	2002/01/04

Note: to normalize a 4th Dimension date value, merely add zero (0) to it.

Note: the method ***DATE_Make_from_Values*** was added in BASh v1.5.4.

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 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.

Note: as of version 1.4.6 of BASh, one hundred (100) variables of each type can be used concurrently.

Note: as of version 1.5.5 of BASh, two hundred (200) variables of each of the following types can be used concurrently: text, longint, BLOB, text array, and longint array.

Note: as of version 1.8.0 of BASh, fifty (50) variables of each of the 2 dimensional array types are available from the DSS module. With this version of BASh, full support for two dimensional arrays was added to the DSS module.

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):

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

DSS_Count_Available_by_Type

DSS_Count_Available_by_Type *(Variable Type) => Available Count*

DSS_Count_Available_by_Type

(

-> *Variable Type :* Longint

)

=> *Available Count :* Longint

	Parameter	Type	Description
	Variable Type	Longint	Type of variable to count currently available variables in DSS module
	Available Count	Longint	Number of variable of given type currently available in DSS module

The method ***DSS_Count_Available_by_Type*** will return the number of currently available variables of a specified type within the the DSS module. This method can be used for snapshots of usage levels of the DSS module and to pinpoint possible leaks in usage of the DSS module.

Variable Type is the type of variable to count in the DSS module.

Available Count is the number of variables of the specified type currently available in the DSS module. Negative one (-1) will be returned if *Variable Type* is an invalid type value.

Note: the method ***DSS_Count_Available_by_Type*** was added in BASh v1.8.0.

DSS_Count_Locked_by_Type

DSS_Count_Locked_by_Type *(Variable Type ; Process ID) => Locked Count*

DSS_Count_Locked_by_Type
(
-> *Variable Type :* Longint
-> *Process ID :* Longint
)
=> *Locked Count :* Longint

	Parameter	Type	Description
	Variable Type	Longint	Type of variable to count locked DSS variables for
	Process ID	Longint	Process ID to restrict DSS locked count to (zero for no process ID restriction)

	Parameter	Type	Description
	Locked Count	Longint	Number of locked variables for given type and optional process ID within the DSS module

The method ***DSS_Count_Locked_by_Type*** will return the number of variables in the DSS module currently locked for the specified type and optional process ID values given. This method can be used to gauge the current usage of the DSS module and to possibly pinpoint leaks in the usage of the DSS module.

Variable Type is the type of DSS variable to count the locked variables in the DSS module.

Process ID is the 4D process ID of the process to restrict the count of locked DSS variables to. The DSS module tracks which process locked each DSS variable, though any process can subsequently return a DSS variable to the availability pool. Passing a value of zero (0) for this parameter will count all variables of the specified type currently locked in the DSS module.

Locked Count is the current number of variables in the DSS module of the given type and process ID restrictions which are locked. Negative one (-1) will be returned of the variable type specified is invalid.

Note: the method ***DSS_Count_Locked_by_Type*** was added in BASh v1.8.0.

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
)

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***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 called 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 Field and Variable Types)
	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_Locking_ProcessID

DSS_Get_Locking_ProcessID *(Referenced DSS Variable) => Locking Process ID*

DSS_Get_Locking_ProcessID
(

-> *Referenced DSS Variable :* Pointer
=> *Locking Process ID :* Longint
)

	Parameter	Type	Description
	Referenced DSS Variable	Pointer	Pointer to a DSS variable
	Locking Process ID	Longint	4D process ID of process that last locked the currently allocated DSS variable specified

The method ***DSS_Get_Locking_ProcessID*** will return the 4D process ID of the process which last locked the currently allocated DSS variable specified. If the DSS variable specified is not locked or is not a referenced to a DSS variable, this will be indicated in the resulting value of this method.

Referenced DSS Variable is a pointer to a DSS variable retrieved from the DSS module. This variable need not necessarily be allocated for use anymore, though use of a DSS variable after returning it to the pool of available variables in the DSS module is strongly discouraged.

Locking Process ID is the 4D process ID of the process which last locked the specified DSS variable *Referenced DSS Variable*. If *Referenced DSS Variable* is not currently locked (i.e. not currently allocated from the DSS module), this method will return zero (0) in this value. If *Referenced DSS Variable* is not a DSS variable, this method will return negative one (-1) in this value.

Note: the method ***DSS_Get_Locking_ProcessID*** was added in BASh v1.8.0.

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 Field and Variable Types)
	Referenced DSS Unary Variable	Pointer	Reference to DSS unary variable to be used matching type of *Array Type*

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

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

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.

For each variable type which can be requested, the DSS module has a pool of 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.

Note: as of version 1.5.5 of BASh, two hundred (200) variables of each of the following types can be used concurrently: text, longint, BLOB, text array, and longint array. One hundred variables of each other type are available within the DSS module.

Note: as of version 1.8.0 of BASh, support for all two dimensional arrays is available within the DSS module. Fifty (50) variables of each two dimensional array type are available for use from the DSS module.

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
$pVolumes:=DSS_Get_Variable_by_Type (Text array )
  `get all of the current volumes
FILE_Get_Volumes_Names ($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_Volumes_Names*** 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
)

	Parameter	Type	Description
	Referenced DSS Variable	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***. Variables returned to the DSS module will be automatically cleared by the DSS module, including arrays having all of their elements removed.

Referenced DSS Variable is a pointer to a valid DSS variable.

DSS_Return_Variables

DSS_Return_Variables *(Referenced DSS Variable {; Referenced DSS Variable {;… } })*

DSS_Return_Variables
(
-> *Referenced DSS Variable :* Pointer
{ -> Referenced DSS Variable : Pointer }
)

	Parameter	Type	Description
	Referenced DSS Variable	Pointer	References to one or more DSS variables to be returned to pool of available DSS variables

The method ***DSS_Return_Variables*** will return one or more variables 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***. Variables returned to the DSS module will be automatically cleared by the DSS module, including arrays having all of their elements removed.

Referenced DSS Variable is a pointer to a valid DSS variable.

Note: the method ***DSS_Return_Variables*** was added in BASh v1.6.2.

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.

DTS Values

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.

GSM Date-Time Stamp Values

A GSM Date-Time Stamp Value (hereafter referred to as a GSM DTS Value) is the standard format used for the “Short Message Peer to Peer” (SMPP) protocol specification. SMPP is commonly used for communications between SMPP enabled devices in a GSM network communication system.

Basically, this format follows the following specification for values:

YYMMDDhhmmsstnnp

where

YY is two (2) byte year value (e.g. “02”)
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”)
t is one (1) byte tenths of a second value (e.g. “4”)
nn is two (2) byte quarter hour offset count from UTC (e.g. “20”)
p is one (1) byte offset cardinality (e.g. “+” for after UTC and “-” for before UTC)

So, for example, a DTS value of “20020415183913” with no tenths of a second and in the eastern standard time zone (+04 hours) would translate to a GSM DTS Value of “020415183913016+”.

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]

	Parameter	Type	Description
	Primary DTS Value	String[14]	First DTS addend
	Secondary DTS Value	String[14]	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 DTS value wherein only the time values are normalized.

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*. Only the time portion of *Summed DTS Value* is normalized.

Example:

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

1st Addend	2nd Addend	Sum
20001122040506	00000000010101	20001122050607
20001122040506	00000014000000	20001136040506
20001031040506	00000100000000	20001131040506
20001031040506	00010000000000	20010229040506

1st Addend	2nd Addend	Sum
20001031040506	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_Add_Normalize

DTS_Add_Normalize *(Primary DTS Value; Secondary DTS Value) => Summed DTS Value*

DTS_Add_Normalize
(
-> *Primary DTS Value :* String[14]
-> *Secondary DTS Value :* String[14]
)
=> *Summed DTS Value :* String[14]

	Parameter	Type	Description
	Primary DTS Value	String[14]	First DTS addend
	Secondary DTS Value	String[14]	Second DTS addend
	Summed DTS Value	String[14]	Sum of DTS addends

The method ***DTS_Add_Normalize*** will add together two DTS values and return a DTS value wherein both the date and time values are normalized.

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*. Both the date and time portions of *Summed DTS Value* are normalized.

Example:

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

1st Addend	2nd Addend	Sum
20001122040506	00000000010101	20001122050607
20001122040506	00000014000000	20001206040506
20001031040506	00000100000000	20001201040506
20001031040506	00010000000000	20010301040506
20001031040506	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_Normalize*** method.

Note: the method ***DTS_Add_Normalize*** was added in BASh v1.5.4.

DTS_Convert_f_GSM

DTS_Convert_f_GSM *(GSM DTS Value) => DTS Value*

DTS_Convert_f_GSM

(

-> *GSM DTS Value :* String[16]

)

=> *DTS Value :* String[14]

	Parameter	Type	Description
	GSM DTS Value	String[16]	GSM date-time stamp value to be converted
	DTS Value	String[14]	Standard DTS value generated from specified GSM DTS value

The method ***DTS_Convert_f_GSM*** will convert a GSM formatted DTS value into a standard DTS value for use throughout the DTS module.

GSM DTS Value is a regular GSM formatted date-time stamp value to be converted.

DTS Value is the converted *GSM DTS Value* placed into a standard DTS format for use in the DTS module. The year is assumed to be in the 21st century, the tenths of a second and UTC time zone offset values stored in the provided *GSM DTS Value* are ignored.

Note: the method ***DTS_Convert_f_GSM*** was added in BASh v1.8.0.

DTS_Convert_f_RFC

DTS_Convert_f_RFC *(RFC Formatted DateTime Stamp) => DTS Value*

DTS_Convert_f_RFC

(

-> *RFC Formatted DateTime Stamp :* Text

)

=> *DTS Value :* String[14]

	Parameter	Type	Description
	RFC Formatted DateTime Stamp	Text	RFC formatted value (see RFC 822)
	DTS Value	String[14]	DTS value corresponding to *RFC Formatted DateTime Stamp*

The method ***DTS_Convert_f_RFC*** will convert a properly formatted RFC datetime stamp into a valid DTS value.

RFC Formatted DateTime Stamp is a properly formatted RFC datetime stamp which will be converted to a standard DTS value. See RFC 822 for details on properly formatted RFC datetime stamps, available from:

http://www.deepskytech.com/rfcs/rfc_0822.txt

DTS Value is the DTS value corresponding to the value within *RFC Formatted DateTime Stamp* .

Note: the method ***DTS_Convert_f_RFC*** was added in BASh v1.5.8.

DTS_Convert_to_GSM

DTS_Convert_to_GSM *(DTS Value ; Tenths of Second ; UTC Offset) => GSM DTS Value*

DTS_Convert_to_GSM
(
-> *DTS Value :* String[14]
-> *Tenths of Second :* Longint
-> *UTC Offset :* Longint
)
=> *GSM DTS Value :* String[16]

	Parameter	Type	Description
	DTS Value	String[14]	Standard DTS value to convert to GSM DTS value
	Tenths of Second	Longint	Tenths of a second to convert to GSM DTS value (inclusively between zero and nine)
	UTC Offset	Longint	UTC quarter hours time zone offset to convert to GSM DTS value (inclusively between negative and positive 48)

	Parameter	Type	Description
	GSM DTS Value	String[16]	GSM formatted date-time stamp value generated for provided parameter values

The method ***DTS_Convert_to_GSM*** will generate a properly formatted GSM date-time stamp value from provided parameter values.

DTS Value is the standard DTS value used to create the GSM DTS value.

Tenths of Second is the number of tenths of a second value used to create the GSM DTS value. This parameter should be an integer (Longint in 4D) with a value between zero (0) and nine (9) inclusive.

UTC Offset is the number of quarter hours for the current time zone from UTC. This parameter should be an integer (Longint in 4D) with a value between negative 48 and positive 48 inclusive.

GSM DTS Value is the GSM formatted date-time stamp value generated from the provided parameter values.

Note: the method ***DTS_Convert_to_GSM*** was added in BASh v1.8.0.

DTS_Convert_to_String_x

DTS_Convert_to_String_x *(DTS Value ; Date Format Code ; Time Format Code)* => *Date Time Stamp Text*

DTS_Convert_to_String_x
(
-> *DTS Value :* String[14]
-> *Date Format Code :* Longint
-> *Time Format Code :* Longint
)
=> *Date Time Stamp Text :* Text

	Parameter	Type	Description
	DTS Value	String[14]	DTS value to format for display
	Date Format Code	Longint	Code for format of date value of DTS
	Time Format Code	Longint	Code for format of time value of DTS
	Date Time Stamp Text	Text	Formatted date time stamp stamp ready for display

The method ***DTS_Convert_to_String_x*** will format a specified DTS value to a specified textual format and return the formatted text.

DTS Value is a standard DTS value to format for display.

Date Format Code is a longint value indicating the format to use for the date portion of *DTS Value*. The valid values are the same as the constants within the standard 4D constants group entitled **Date Display Formats**. A value of zero (0) for this parameter will suppress the inclusion of the date in the resulting text value.

Time Format Code is a longint value indicating the format to use for the time portion of *DTS Value*. The valid values are the same as the constants within the standard 4D constants group entitled **Time Display Formats**. A value of zero (0) for this parameter will suppress the inclusion of the time in the resulting text value.

Date Time Stamp Text is a text value containing the specified DTS formatted for display by the parameters specified for the data and time portions of *DTS Value*.

Note: the method ***DTS_Convert_to_String_x*** was added in BASh v1.8.2.

DTS_Get_Age

DTS_Get_Age *(Birth DTS ; Source DTS) => Age in Years*

DTS_Get_Age
(

-> *Birth DTS :* String[14]
-> *Source DTS :* String[14]
)
=> *Age in Years :* Longint

	Parameter	Type	Description
	Birth DTS	String[14]	"birth" DTS to calculate age from
	Source DTS	String[14]	DTS to calculate age to
	Age in Years	Longint	Age in years from *Birth DTS* to *Source DTS*

The method ***DTS_Get_Age*** returns the age in years between two specified DTS values.

Birth DTS is the DTS value to calculate *Age in Years* from.

Source DTS is the DTS value to calculate *Age in Years* to, from *Birth DTS*. For example, use ***DTS_Get_Current*** to calculate *Age in Years* from *Birth DTS* to today.

Age in Years is the difference in years between *Birth DTS* and *Source DTS*. If *Source DTS* is before *Birth DTS*, *Age in Years* will be less than zero.

Note: the method ***DTS_Get_Age*** was added in BASh v1.7.0.

DTS_Get_Current

DTS_Get_Current => *Current DTS Value*

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

	Parameter	Type	Description
	Current DTS Value	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 DateTime Stamp.

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
	DTS Value	String[14]	Valid DTS Value
	Date Value	Date	Valid 4D date value extracted from *DTS Value*

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
	DTS Value	String[14]	Valid DTS Value
	Referenced Date Variable	Pointer	Reference to a 4D date variable to hold the date contained within *DTS Value*
	Referenced Time Variable	Pointer	Reference to a 4D time variable to hold the time contained within *DTS Value*

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
	DTS Value	String[14]	Valid DTS Value
	Day Value	Longint	Day value as stored in supplied *DTS Value*

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
	Maximum DTS Value	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
	DTS Value	String[14]	Valid DTS Value
	Month Value	Longint	Month value as stored in supplied *DTS Value*

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_MonthDay_x

DTS_Get_MonthDay_x *(Source DTS) => Month Day String*

DTS_Get_MonthDay_x
(
-> *Source DTS :* String[14]
)
=> *Month Day String :* Text

	Parameter	Type	Description
	Source DTS	String[14]	DTS to use to generate *Month Day String*
	Month Day String	Text	Formatted name of month and number of day for *Source DTS*

The method ***DTS_Get_MonthDay_x*** will return text for the month name and day number for a specified source DTS. The returned “Month Day” text is the textual name of the month with the coerced numerical day value appended (a single space separating the values).

Source DTS is the input date to use to generate the month day textual value.

Month Day String is the generate month day textual value corresponding to *Source DTS*.

Note: the method ***DTS_Get_MonthDay_x*** was added in BASh v1.8.0.

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
)

	Parameter	Type	Description
	Reference DTS	String[14]	DTS value to reference from when calculating range
	Range Type	Longint	Type of range to calculate
	Referenced Beginning DTS	Pointer	Pointer to DTS to place beginning of calculated range into
	Referenced Ending DTS	Pointer	Pointer to DTS to place ending of calculated range into

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:

Range Type	Range Returned
1	Last week
2	Last month
3	Last calendar quarter
4	Current week
5	Current month
6	Current calendar quarter

The following rules 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
	DTS Value	String[14]	Valid DTS Value
	Time Value	Time	Valid 4D time value extracted from *DTS Value*

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
	DTS Value	String[14]	Valid DTS Value
	Year Value	Longint	Year value as stored in supplied *DTS Value*

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
	Date Value	Date	Valid 4th Dimension date value
	Time Value	Time	Valid 4th Dimension time value
	DTS Value	String[14]	Valid DTS value formed from *Date Value* and *Time Value*

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
	Years	Longint	Years value
	Months	Longint	Months value
	Days	Longint	Days value
	Hours	Longint	Hours value
	Minutes	Longint	Minutes value
	Seconds	Longint	Seconds value
	DTS Value	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 supplied values.

The method ***DTS_Make_from_Values*** does not perform any normalization upon the DTS value it creates. So, it is easy to 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"  `20 months

DTS_Make_from_Values(0;0;90;0;0;0)="00000090000000"  `90 days

DTS_Make_from_Values(0;0;0;48;0;0)="00000000480000"  `48 hours

DTS_Make_from_Values(0;0;0;0;0;90)="00000000000090"  `90 seconds
```

Note: the method ***DTS_Make_from_Values*** was added in BASh v1.4.7.

DTS_Subtract

DTS_Subtract *(Primary DTS Value ; Secondary DTS Value) => Differenced DTS Value*

DTS_Subtract
(
-> *Primary DTS Value :* String[14]
-> *Secondary DTS Value :* String[14]
)
=> *Differenced DTS Value :* String[14]

	Parameter	Type	Description
	Primary DTS Value	String[14]	First DTS subtractive in transitive subtraction
	Secondary DTS Value	String[14]	Second DTS subtractive in transitive subtraction
	Differenced DTS Value	String[14]	Difference of DTS subtractives

The method ***DTS_Subtract*** will subtract two specified DTS values and return the difference in a DTS value. The time portion of the resulting DTS value will be normalized.

Primary DTS Value is the first DTS subtractive (the value to be subtracted from) in the transitive subtraction.

Secondary DTS Value is the second DTS subtractive (the value to be subtracted) in the transitive subtraction.

Differenced DTS Value is a well formed difference of *Primary DTS Value* and *Secondary DTS Value*. Only the time portion of *Differenced DTS Value* is normalized.

Example:

The following table shows the values of two subtractives and the results which will be returned from the method ***DTS_Subtract***:

1st Addend	2nd Addend	Sum
20001122040506	00000000010101	20001222030405
20001122040506	00010203000000	19990919040506
20001031040506	00000100000000	20001131040506
20001031040506	00010000000000	19990229040506
20001031040506	00000000060000	20001221220506

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 decremented 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 above) and subtract it from the original DTS value by using the ***DTS_Subtract*** method.

Note: the method ***DTS_Subtract*** was added in BASh v1.5.4.

DTS_Subtract_Normalize

DTS_Subtract_Normalize *(Primary DTS Value ; Secondary DTS Value) => Differenced DTS Value*

DTS_Subtract_Normalize

(

-> Primary DTS Value : String[14]
-> Secondary DTS Value : String[14]

)

=> Differenced DTS Value : String[14]

	Parameter	Type	Description
	Primary DTS Value	String[14]	First DTS subtractive in transitive subtraction
	Secondary DTS Value	String[14]	Second DTS subtractive in transitive subtraction

	Parameter	Type	Description
	Differenced DTS Value	String[14]	Difference of DTS subtractives

The method ***DTS_Subtract_Normalize*** will subtract two specified DTS values and return the difference in a DTS value. The date and time portions of the resulting DTS value will be normalized.

Primary DTS Value is the first DTS subtractive (the value to be subtracted from) in the transitive subtraction.

Secondary DTS Value is the second DTS subtractive (the value to be subtracted) in the transitive subtraction.

Differenced DTS Value is a well formed difference of *Primary DTS Value* and *Secondary DTS Value*. Both the date and time portions of *Differenced DTS Value* will be normalized.

Example:

The following table shows the values of two subtractives and the results which will be returned from the method ***DTS_Subtract_Normalize***:

1st Addend	2nd Addend	Sum
20001122040506	00000000010101	20001222030405
20001122040506	00010203000000	19990919040506
20001031040506	00000100000000	20001201040506
20001031040506	00010000000000	19990301040506
20001031040506	00000000060000	20001221220506

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 decremented 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 above) and subtract it from the original DTS value by using the ***DTS_Subtract_Normalize*** method.

Note: the method ***DTS_Subtract_Normalize*** was added in BASh v1.5.4.

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_ERROR

ENV_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***ENV_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 ENV module, a call to the method ***ENV_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 ENV method which called the ***ENV_ERROR*** method.

The ***ENV_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.

ENV_Get_4DApplication_FoldPath

ENV_Get_4DApplication_FoldPath (=> *4D Application Folder Path*

ENV_Get_4DApplication_FoldPath (
=> *4D Application Folder Path :* Text

	Parameter	Type	Description
	4D Application Folder Path	Text	Full path to the folder containing the currently running 4D application

The method ***ENV_Get_4DApplication_FoldPath*** returns the full path to the folder containing the currently running 4D application.

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

Note: the method ***ENV_Get_4DApplication_FoldPath*** was added in BASh v1.6.2.

ENV_Get_4DApplication_FullPath

ENV_Get_4DApplication_FullPath (=> *4D Application Path*

ENV_Get_4DApplication_FullPath (
=> *4D Application Path :* Text

	Parameter	Type	Description
	4D Application Path	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 always returns the full path to the currently running application, even if the application is within a package.

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_4DAPPL_Pkg_FullPath

ENV_Get_4DAPPL_Pkg_FullPath (=> *4D Application Package Full Path*

ENV_Get_4DAPPL_Pkg_FullPath (
=> *4D Application Package Full Path* : Text

	Parameter	Type	Description
	4D Application Package Full Path	Text	Full path to the currently running 4D application package

The method ***ENV_Get_4DAPPL_Pkg_FullPath*** returns the full path to the current 4D application package. For environments in which there is no support for packages (4D v6.7.x on MacOS and all versions of 4D on Windows), the full path to the current 4D application is returned.

4D Application Package Full Path is the full path to the current 4D application package which is running.

Note: the method ***ENV_Get_4DAPPL_Pkg_FullPath*** was added in BASh v1.8.2.

ENV_Get_4DSerial_CompanyName

ENV_Get_4DSerial_CompanyName (=> *4D Serial Company Name*

ENV_Get_4DSerial_CompanyName (

=> *4D Serial Company Name :* Text

	Parameter	Type	Description
	4D Serial Company Name	Text	Company name specified when 4D was serialized

The method ***ENV_Get_4DSerial_CompanyName*** returns the company name used for the serialization of 4D. This is functionally equivalent to using the company name returned from the native **GET SERIAL INFORMATION** command, except no time slicing is given to the 4D application.

4D Serial Company Name is the company name used for the serialization of 4D.

Note: the method ***ENV_Get_4DSerial_CompanyName*** was added in BASh v1.6.2.

ENV_Get_4DSerial_Key

ENV_Get_4DSerial_Key (=> *4D Serial Key*

ENV_Get_4DSerial_Key (
=> *4D Serial Key :* Text

	Parameter	Type	Description
	4D Serial Key	Text	Unique product key associated with a 4D application

The method ***ENV_Get_4DSerial_Key*** returns the unique product key associated with a 4D application. This is functionally equivalent to using the key returned from the native **GET SERIAL INFORMATION** command, except no time slicing is given to the 4D application.

4D Serial Key is the unique product key associated with a 4D application.

Note: the method ***ENV_Get_4DSerial_Key*** was added in BASh v1.6.2.

ENV_Get_4DSerial_UserName

ENV_Get_4DSerial_UserName (=> *4D Serial User Name*

ENV_Get_4DSerial_UserName (
=> *4D Serial User Name :* Text

	Parameter	Type	Description
	4D Serial User Name	Text	User name specified when 4D was serialized

The method ***ENV_Get_4DSerial_UserName*** returns the user name used for the serialization of 4D. This is functionally equivalent to using the user name returned from the native **GET SERIAL INFORMATION** command, except no time slicing is given to the 4D application.

4D Serial User Name is the user name used for the serialization of 4D.

Note: the method ***ENV_Get_4DSerial_UserName*** was added in BASh v1.6.2.

ENV_Get_4DXAPPL_FolderPath

ENV_Get_4DXAPPL_FolderPath (=> *Folder Path*

ENV_Get_4DXAPPL_FolderPath (
=> *Folder Path :* Text

	Parameter	Type	Description
	Folder Path	Text	Full path to 4DX folder with the current application

The method ***ENV_Get_4DXAPPL_FolderPath*** will return the full folder path to the 4DX folder for the current platform next to the current 4D application. This method will

function correctly across platforms and in either single user or client/server environments.

This method will not confirm the actual existence of the returned full folder path. Rather, it just returns the full folder path to it for the current machine, whether it actually exists there or not.

Note: this function has little use under any version of 4D prior to 6.8.x. It was not until 4D version 6.8.x that a third 4DX folder, next to the current 4D application, was supported.

Note: under 4D v7.0.x (4D 2003), the exact location of the 4DX folders within the package has been again changed though this method handles it properly for all versions of 4D.

Folder Path will be the full path to the 4DX folder for the current platform next to the current 4D application.

Note: the method ***ENV_Get_4DXAPPL_FolderPath*** was added in BASh v1.7.0.

ENV_Get_4DXOS_FolderPath

ENV_Get_4DXOS_FolderPath (=> *Folder Path*

ENV_Get_4DXOS_FolderPath (
=> *Folder Path :* Text

	Parameter	Type	Description
	Folder Path	Text	Full path to 4DX folder within the current OS

The method ***ENV_Get_4DXOS_FolderPath*** will return the full folder path to the 4DX folder for the current platform within the current system's preferences directory. This method will function correctly across platforms and in either single user or client/server environments.

This method will not confirm the actual existence of the returned full folder path. Rather, it just returns the full

folder path to it for the current machine, whether it actually exists there or not.

Folder Path will be the full path to the 4DX folder for the current platform within the current system's preferences directory.

Note: the method ***ENV_Get_4DXOS_FolderPath*** was added in BASh v1.5.3.

ENV_Get_4DX_FolderPath

ENV_Get_4DX_FolderPath (=> *Folder Path*

ENV_Get_4DX_FolderPath (
=> *Folder Path :* Text

	Parameter	Type	Description
	Folder Path	Text	Full path to 4DX folder next to the current structure document

The method ***ENV_Get_4DX_Folder_Path*** will return the full folder path to the 4DX folder for the current platform within the structure's enclosing folder. This method will function correctly across platforms and in either single user or client/server environments. In client/server while on the client machine, the structure document's enclosing folder will be the folder containing the ".res" and ".rex" documents for the connected database.

This method will not confirm the actual existence of the returned full folder path. Rather, it just returns the full folder path to it for the current machine, whether it actually exists there or not.

Folder Path will be the full path to the 4DX folder for the current platform within the structure document's enclosing folder.

Note: the method ***ENV_Get_4DX_FolderPath*** was added in BASh v1.5.3.

ENV_Get_Application_Name

ENV_Get_Application_Name (=> *Application Name*

ENV_Get_Application_Name (
=> *Application Name :* Text

	Parameter	Type	Description
	Application Name	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
	Application Short Name	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* : Text

	Parameter	Type	Description
	Application Indicator	Text	Application type of the currently running 4D application (application types are designated by the 4D constants 4D Environemnt)

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_BASh_HardName_Long

ENV_Get_BASh_HardName_Long (=> *Long Hard Name*

ENV_Get_BASh_HardName_Long (
=> *Long Hard Name* : Text

	Parameter	Type	Description
	Long Hard Name	Text	Full long hard name of Affix BASh document

The method ***ENV_Get_BASh_HardName_Long*** returns the current full long hard name of the current Affix BASh document.

For information on the Affix BASh document, please read the section in this manual entitled Affix BASh Document.

For information on the hard file names, please read the section in the manual entitled Hard File Names.

Long Hard Name is the full long hard name of the current Affix BASh document.

Note: the method ***ENV_Get_BASh_HardName_Long*** was added in BASh v1.5.3.

ENV_Get_BASh_HardName_Short

ENV_Get_BASh_HardName_Short (=> *Short Hard Name*

ENV_Get_BASh_HardName_Short (
=> *Short Hard Name* : Text

	Parameter	Type	Description
	Short Hard Name	Text	Full short hard name of Affix BASh document

The method ***ENV_Get_BASh_HardName_Short*** returns the current full short hard name of the current Affix BASh document.

For information on the Affix BASh document, please read the section in this manual entitled Affix BASh Document.

For information on the hard file names, please read the section in the manual entitled Hard File Names.

Short Hard Name is the full short hard name of the current Affix BASh document.

Note: the method ***ENV_Get_BASh_HardName_Short*** was added in BASh v1.5.3.

ENV_Get_DataFile_FolderPath

ENV_Get_DataFile_FolderPath (=> *Folder Path*

ENV_Get_DataFile_FolderPath (
=> *Folder Path :* Text

	Parameter	Type	Description
	Folder Path	Text	Full path to the folder containing the current data file

The method ***ENV_Get_DataFile_FolderPath*** returns the full path to the folder containing the current data file. This is functionally equivalent to using the native **Data file** command within the 4D language and extracting the file name, except no time slicing is given to the 4D application.

Folder Path is the full path to the folder containing the current data file which is in use.

Note: the method ***ENV_Get_DataFile_FolderPath*** was added in BASh v1.5.3.

ENV_Get_DataFile_FullPath

ENV_Get_DataFile_FullPath (=> *Data File Path*

ENV_Get_DataFile_FullPath *(*
=> *Data File Path :* Text

	Parameter	Type	Description
	Data File Path	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_DataFile_RF_FullPath

ENV_Get_DataFile_RF_FullPath *(=> Data File Resource Fork Path*

ENV_Get_DataFile_RF_FullPath *(*
=> *Data File Resource Fork Path :* Text

	Parameter	Type	Description
	Data File Resource Fork Path	Text	Full path to the current date file resource fork

The method ***ENV_Get_DataFile_RF_FullPath*** returns the full path to the current data file resource fork. 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 and on Windows the actual document name for the resource fork of the data file has a different file extension (.4DR).

Data File Resource Fork Path is the full path to the current data file resource fork that is in use.

Note: the method ***ENV_Get_DataFile_RF_FullPath*** was added in BASh v1.8.0.

ENV_Get_DirectorySymbol

ENV_Get_DirectorySymbol (=> *Directory Symbol*

ENV_Get_DirectorySymbol (
=> *Directory Symbol* : String[1]

	Parameter	Type	Description
	Directory Symbol	String[1]	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_DirectorySymbol_by_OS

ENV_Get_DirectorySymbol_by_OS (*Platform*) => *Directory Symbol*

ENV_Get_DirectorySymbol_by_OS
(
-> *Platform* : Longint
)
=> *Directory Symbol* : String[1]

	Parameter	Type	Description
	Platform	Longint	Directory delimiter used on current platform

	Parameter	Type	Description
	Directory Symbol	String[1]	Platform indicator value (platform indicators are designated by the 4D constants Platform Properties)

The method ***ENV_Get_DirectorySymbol_by_OS*** returns the character used as the delimiter between directory items on the specified OS.

Platform is the platform indicator to get the directory sumbol for. Platform indicators are designated by the 4D constants Platform Properties.

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

Note: the method ***ENV_Get_DirectorySymbol_by_OS*** was added in BASh v1.5.3.

ENV_Get_Platform

ENV_Get_Platform (=> *Platform Indicator*

ENV_Get_Platform (
=> *Platform Indicator :* Longint

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

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_DF_FullPath

ENV_Get_Structure_DF_FullPath (=> *Full Path*

ENV_Get_Structure_DF_FullPath (
=> *Full Path* : Text

	Parameter	Type	Description
	Full Path	Text	Full path of the data fork of the current structure file as it exists on the current machine

The method ***ENV_Get_Structure_DF_FullPath*** returns the full path of the data 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 resource 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 is not a document that actually exists. This method considers the "RES" document the resource fork of the structure document and will return a full path to a corresponding data fork for the "RES" document if it were to exist.

Full Path is the full path of the data fork of the current structure on the current machine.

Note: the method ***ENV_Get_Structure_DF_FullPath*** was added in BASh v1.5.3.

ENV_Get_Structure_FolderPath

ENV_Get_Structure_FolderPath (=> *Folder Path*

ENV_Get_Structure_FolderPath (
=> *Folder Path :* Text

	Parameter	Type	Description
	Folder Path	Text	Full path of the folder containing the current structure document on the current machine

The method ***ENV_Get_Structure_FolderPath*** returns the full path of the folder containing the current structure document on the current machine.

When this method is called under 4D Client, the folder containing the current structure document is the same as the folder containing the "RES" file within the 4D folder in the OS preferences.

Full Path is the full path to the folder containing the current structure document on the current machine.

Note: the method ***ENV_Get_Structure_FolderPath*** was added in BASh v1.5.3.

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
	Structure Resource Fork File Name	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 edit-ted 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

	Parameter	Type	Description
	Structure Resource Fork Full Path	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 editted 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_qi_BASh_INITed

ENV_qi_BASh_INITed *=> qi BASh Initialized*

ENV_qi_BASh_INITed
=> qi BASh Initialized : Longint

	Parameter	Type	Description
	qi BASh Initialized	Longint	Flag for whether BASh component has been initialized successfully yet or not

The method ***ENV_qi_BASh_INITed*** will return a flag for whether the BASh component has been successfully intialized yet or not.

qi BASh INITed is a flag for whether the BASh component has been successfully initialized yet or not. This value will be set to zero (0) if BASh has not been successfully initialized and set to one (1) if BASh has been successfully initialized.

Note: the method ***ENV_qi_BASh_INITed*** was added in BASh v1.8.0.

ENV_q_Macintosh

ENV_q_Macintosh *(=> Macintosh Flag*

ENV_q_Macintosh *(*
=> Macintosh Flag : Boolean

	Parameter	Type	Description
	Macintosh Flag	Boolean	Flag for current machine running Macintosh

The method ***ENV_q_Macintosh*** returns a flag for whether the current machine is running under type of Macintosh OS, including Sytem 7, MacOS 8, or MacOS 9.

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

Note: the method ***ENV_q_Macintosh*** was added in BASh v1.5.3.

ENV_q_Windows

ENV_q_Windows (=> *Windows Flag*

ENV_q_Windows (
=> *Windows Flag :* Boolean

	Parameter	Type	Description
	Windows Flag	Boolean	Flag for current machine running Windows

The method ***ENV_q_Windows*** returns a flag for whether the current machine is running under any 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.

ENV_Set_FlushKey

ENV_Set_FlushKey (*ASCII Value*)

ENV_Set_FlushKey
(
-> *ASCII Value :* Longint
)

	Parameter	Type	Description
	ASCII Value	Longint	ASCII Value to set for the automatic Flush Buffers functionality within 4D

The method ***ENV_Set_FlushKey*** will set the keyboard equivalent value for the current **Flush Buffers** functionality available within 4D.

The value is set directly into the resource fork of the current 4D application. This includes 4D, 4D Server, 4D Client, 4D Runtime, and structure documents which have been merged with 4D Engine. This method will set the internal **Flush Buffer** keyboard equivalent for all of these applications.

The changing of the **Flush Buffers** keyboard equivalent value will not affect the current instance of 4D which is running. All instances of the 4D product line load this value from the resource fork immediately upon upon launching. Changing this value in the resource fork from code within the structure will not be affective then until the next launching of the same 4D application.

ASCII Value is the ASCII value of the character to use as the keyboard equivalent for the internal **Flush Buffers** functionality within 4D.

Note: as of BASh v1.6.1, this method will not function properly under Windows. This method relies upon the method ***RES_Open_4DApplication*** which has problems working with current releases of 4D (through 4D v6.7.2).

Note: the method ***ENV_Set_FlushKey*** was added in BASh v1.5.3.

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.

Hard File Names

Hard file names are a very old convention originally used on the Macintosh. Hard file names provide a means for specific documents to be identified exactly even if the actual document names have been changed.

Essentially, hard file names are string values which are stored in a particular location in the resource fork of a document.

Two different hard file names exist: short and long. The short hard file name is just the name to be used for identification purposes for the document. The long hard file name is the same as the short hard file name except the name also contains the current version number of the document.

For instance, the Affix BASh document contains the resources for hard file names. The short hard file name of the Affix BASh document is "Affix_BASh". The long hard file name of the Affix BASh document is "Affix_BASh_v1.6.2". These hard file names stored in the resource fork of the Affix BASh document provide a means for the methods within the BASh component to identify the Affix BASh document within the 4DX folders even if the document has been renamed. By using the commands within the FILE module for hard file names, the same functionality can be built very easily within your own 4D projects for documents which must be managed.

The actual location which the hard file names are stored is as follows: 'STR#', ID 1. Short hard file names are stored in index 1. Long hard file names are stored in index 2.

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

FILE_Convert_to_ResFork_Windows

FILE_Convert_to_ResFork_Windows *(Data Fork Full Path) => Resource Fork Full Path*

FILE_Convert_to_ResFork_Windows

(

-> Data Fork Full Path : Text

)

=> Resource Fork Full Path : Text

	Parameter	Type	Description
	Data Fork Full Path	Text	Full path to data fork of a document
	Resource Fork Full Path	Text	Full path to the resource fork of the *Data Fork Full Path* document

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_Create_Document

FILE_Create_Document *(Document Full Path ; Document Type) => Document Reference*

FILE_Create_Document

(

-> *Document Full Path :* Text

-> *Document Type :* Longint

)

=> *Document Reference :* Time

	Parameter	Type	Description
	Document Full Path	Text	Full path to document to create
	Document Type	Longint	Document type to create
	Document Reference	Time	File reference to document created

The method ***FILE_Create_Document*** will create a new document at a specified location of a specified type and return the document reference to its data fork.

If the document specified in *Document Full Path* already exists then ***FILE_Create_Document*** will generate an error and return NULL in *Document Reference*. If *Document Type* is NULL then the document will be created using the default file type ('TEXT' on Macintosh and ".txt" on Windows).

Document Full Path is the full path to the document to create.

Document Type is the type of document to be created. If *Document Type* is NULL (zero, 0) then the document will be created using the default document type ('TEXT' on Macintosh and ".txt" on Windows).

Document Reference is the document reference to the new document created. This reference is the file reference to the data fork of the new document. If there is an error creating the new document then *Document Reference* will be NULL.

Note: the method ***FILE_Create_Document*** was added in BASh v1.5.3.

FILE_Create_Folder

FILE_Create_Folder *(Folder Full Path)* => *Success*

FILE_Create_Folder
(
-> *Folder Full Path :* Text
)
=> *Success :* Longint

	Parameter	Type	Description
	Folder Full Path	Text	Full path to folder to create
	Success	Longint	Success indicator for creating folder specified in *Folder Full Path*

The method ***FILE_Create_Folder*** will create a specified folder.

If the folder specified in *Folder Full Path* already exists then this method will generate an error and will indicate failure in the *Success* indicator value returned.

Folder Full Path is the full path to the new folder to create.

Success is the indicator value for whether the new folder was successfully created in this method. *Success* will be set to one (1) if the new folder was successfully created. If the folder already exists or failed to be created then *Success* will be set to zero (0).

Note: the method ***FILE_Create_Folder*** was added in BASh v1.5.3.

FILE_Create_FullPath_Document

FILE_Create_FullPath_Document *(Full Document Path)* => *Success Status*

FILE_Create_FullPath_Document
(
-> *Full Document Path :* Text
)

=> *Success Status :* Longint

	Parameter	Type	Description
	Full Document Path	Text	Full path to document and folder hierarchy to create
	Success Status	Longint	Status code for successfully created document and folder hierarchy

The method ***FILE_Create_FullPath_Document*** will create a document in a specified location in the directory hierarchy with the specified name.

If any folders in the directory hierarchy specified do not exist, they will be created. If the document does not already exist, it will be created. If the document specifed already exists, it will remain unmodified ***FILE_Create_FullPath_Document***.

If ***FILE_Create_FullPath_Document*** creates the document, the document type will be the default document type assigned by the native 4th Dimension command **Create Document**.

Often, ***FILE_Create_FullPath_Document*** is used to confirm the existence of a particular document in a specific location. And, to create the document with the full accompanying folder hierarchy, in the event that it is needed.

Full Document Path is the full path to the document to confirm existence of and possibly create.

Success Status is the status code returned by this method. The following table lists the values which can be returned in *Success Status*:

Value	Meaning
2	document created successfully
1	document already exists
0	document does not exist
-1	not a document
-2	error creating folder path

Value	Meaning
-3	error creating document

Note: the method ***FILE_Create_FullPath_Document*** was added in BASh v1.5.4.

FILE_Create_FullPath_Folder

FILE_Create_FullPath_Folder *(Full Folder Path) => Success Status*

FILE_Create_FullPath_Folder

(

-> *Full Folder Path :* Text

)

=> *Success Status :* Longint

	Parameter	Type	Description
	Full Folder Path	Text	Full path to folder hierarchy to create
	Success Status	Longint	Status code for successfully created folder hierarchy

The method ***FILE_Create_FullPath_Folder*** will create a specified directory hierarchy. If any folders in the directory hierarchy specified do not exist, they will be created.

Often, ***FILE_Create_FullPath_Folder*** is used to confirm the existence of a particular folder hierarchy. And, to create the folder hierarchy, in the event that it is needed.

Full Folder Path is the full path to the folder hierarchy to confirm existence of and possibly create.

Success Status is the status code returned by this method. The following table lists the values which can be returned in *Success Status*:

Value	Meaning
2	document folder hierarchy created successfully

Value	Meaning
1	folder hierarchy already exists
0	folder hierarchy does not exist
-1	not a valid folder hierarchy
-2	volume does not exist
-3	error creating folder in folder hierarchy

Note: the method ***FILE_Create_FullPath_Folder*** was added in BASh v1.5.4.

FILE_ERROR

FILE_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***FILE_ERROR*** acts as a callback method from within the FILE module for errors that may occur. Any time an error condition is detected within the FILE module, a call to the method ***FILE_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 FILE method which called the ***FILE_ERROR*** method.

The ***FILE_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.

FILE_Find_FileName_Hard_Long

FILE_Find_FileName_Hard_Long *(Long Hard Name ; Full Path) => Matching File Name*

FILE_Find_FileName_Hard_Long

(

-> Long Hard Name : Text

-> Full Path : Text

)

=> Matching File Name : Text

	Parameter	Type	Description
	Long Hard Name	Text	Long hard file name to match
	Full Path	Text	Full path to folder to scan for *Long Hard Name*
	Matching File Name	Text	File name of document within *Full Path* matching *Long Hard Name*

The method ***FILE_Find_FileName_Hard_Long*** will return the file name of the first document within a specified folder matching a specified long hard file name.

Long Hard Name is the long hard file name to match.

Full Path is the full folder path to a folder to scan the contents of for *Long Hard Name*. Only the documents directly within the *Full Path* directory will be scanned. Documents contained within directories within the *Full Path* folder will not be scanned.

Matching File Name is the name of the document matching *Long Hard Name* within the folder *Full Path*. If no documents within *Full Path* match *Long Hard Name* then *Matching File Name* will be NULL.

Note: the method ***FILE_Find_FileName_Hard_Long*** was added in BASh v1.5.3.

FILE_Find_FileName_Hard_Short

FILE_Find_FileName_Hard_Short *(Short Hard Name ; Full Path) => Matching File Name*

FILE_Find_FileName_Hard_Short
(
-> *Short Hard Name :* Text
-> *Full Path :* Text
)
=> *Matching File Name :* Text

	Parameter	Type	Description
	Short Hard Name	Text	Short hard file name to match
	Full Path	Text	Full path to folder to scan for *Short Hard Name*
	Matching File Name	Text	File name of document within *Full Path* matching *Short Hard Name*

The method ***FILE_Find_FileName_Hard_Short*** will return the file name of the first document within a specified folder matching a specified short hard file name.

Short Hard Name is the short hard file name to match.

Full Path is the full folder path to a folder to scan the contents of for *Short Hard Name*. Only the documents directly within the *Full Path* directory will be scanned. Documents contained within directories within the *Full Path* folder will not be scanned.

Matching File Name is the name of the document matching *Short Hard Name* within the folder *Full Path*. If no docu-

ments within *Full Path* match *Short Hard Name* then *Matching File Name* will be NULL.

Note: the method ***FILE_Find_FileName_Hard_Short*** was added in BASh v1.5.3.

FILE_Find_Plugin_Hard_Long

FILE_Find_Plugin_Hard_Long *(Long Hard Name)* => *Full Path*

FILE_Find_Plugin_Hard_Long
(
-> *Long Hard Name :* Text
)
=> *Full Path :* Text

	Parameter	Type	Description
	Long Hard Name	Text	Long hard file name to match
	Full Path	Text	Full path to first matching document found in any currently valid 4DX folder

The method ***FILE_Find_Plugin_Hard_Long*** will return the full path of the first document within a currently valid 4DX folder matching a specified long hard file name. The 4DX folders valid for the current platform and environment will be scanned in the standard 4D load order for plugins to find the first document with a matching long hard file name.

Long Hard Name is the long hard file name to match.

Full Path is the full document path to the first document found matching *Long Hard Name* within any of the currently valid 4DX folders.

Note: the method ***FILE_Find_Plugin_Hard_Long*** was added in BASh v1.7.0.

FILE_Force_to_FullPath

FILE_Force_to_FullPath *(Path to Check ; Default Path)* => *Full Path*

FILE_Force_to_FullPath
(
-> *Path to Check :* Text
-> *Default Path :* Text
)
=> *Full Path :* Text

	Parameter	Type	Description
	Path to Check	Text	Possible relative path to make into a full path
	Default Path	Text	Default path to use to create full path
	Full Path	Text	Full path created

The method ***FILE_Force_to_FullPath*** will take a relative document or directory path and combine it with a default directory path, returning a full path for the relative document or directory. No checks are done on the validity of any parts of the paths.

Path to Check is the relative path which is the be checked. If it is a relative path and not a full path (i.e. it starts with a directory symbol), then it will be combined with the supplied *Default Path* and returned. If *Path to Check* is already a full path (i.e. it does not start with a directory symbol), then it is returned unaltered.

Default Path is the directory path to be combined and returned with *Path to Check* when *Path to Check* is found to be a relative path.

Full Path is the resulting full path returned from the operations of this method.

Note: the method ***FILE_Force_to_FullPath*** was added in BASh v1.6.0

FILE_Force_to_RelativePath

FILE_Force_to_RelativePath *(Path to Check ; Default Directory Path) => Relative Path*

FILE_Force_to_RelativePath

(

-> *Path to Check :* Text

-> *Default Directory Path :* Text

)

=> *Relative Path :* Text

	Parameter	Type	Description
	Path to Check	Text	Possible full path to make into a relative path
	Default Directory Path	Text	Default directory path to use to create full path
	Relative Path	Text	Relative path created

The method ***FILE_Force_to_RelativePath*** will take a full document or directory path and turn it into a relative directory path, returning a relative path from the default directory path for the full document or directory path. No checks are done on the validity of any parts of the paths.

Path to Check is the full path which is to be checked. If it is a full path and not a relative path (i.e. it does not start with a directory symbol), then it will be made into a path relative to the supplied *Default Directory Path* and returned. If *Path to Check* is already a relative path (i.e. it does not start with a directory symbol), then it is returned unaltered.

Default Directory Path is the directory path to make *Path to Check* relative to when *Path to Check* is found to be a full path.

Relative Path is the resulting relative path returned from the operations of this method.

Note: the method ***FILE_Force_to_RelativePath*** was added in BASh v1.7.0.

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
	Full Path	Text	Full path to a document
	File Name	Text	File name of document specified in *Full Path*

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.

FILE_Get_Document_List

FILE_Get_Document_List *(Folder Full Path ; Referenced Document Names)* => *Document Count*

FILE_Get_Document_List

(

-> *Folder Full Path :* Text

-> *Referenced Document Names :* Pointer

)

=> *Document Count :* Longint

	Parameter	Type	Description
	Folder Full Path	Text	Full path to folder
	Referenced Document Names	Pointer	Pointer to text array to hold document names
	Document Count	Longint	Number of documents returned in *Referenced Document Names*

The method ***FILE_Get_Document_List*** returns the number of documents and document names within a specified directory.

Folder Full Path is the full path to a directory to get the directory listing of documents from.

Referenced Document Names is a pointer to a text array which will hold the names of all of the documents within *Folder Full Path*.

Document Count is the number of documents contained within *Folder Full Path*. This will be the same number of elements in *Referenced Document Names* after this method is called.

Note: the method ***FILE_Get_Document_List*** was added in BASh v1.5.3.

FILE_Get_Extension

FILE_Get_Extension *(Document Name or Path) => Document Extension*

FILE_Get_Extension

(

-> *Document Name or Path :* Text

)

=> *Document Extension :* Text

	Parameter	Type	Description
	Document Name or Path	Text	Full document name, relative path to document, or full path to document to extra document extension from
	Document Extension	Text	Document extension extracted from *Document Name or Path*

The method ***FILE_Get_Extension*** will extract the document extension from a provided document name, document relative path, or document full path. No checking is done for the proper formatting of the provided document name or path and the existence of the document is not checked at all.

Document Name or Path is a provided document name, document relative path, or document full path to extract the document extension from. The proper formatting of this value or the actual existence of the document or path is not checked at all in this method.

Document Extension is the document extension extracted from *Document Name or Path*. The extension of the document is extracted merely by returning all text following the final period (".") in *Document Name or Path*.

Note: the method ***FILE_Get_Extension*** was added in BASh v1.8.0.

FILE_Get_FPaths_HFolders

FILE_Get_FPaths_HFolders *(Target Folder Full Path ; Referenced Full Paths ; Depth)*
=> *Return Count*

FILE_Get_FPaths_HFolders
(
-> *Target Folder Full Path :* Text
-> *Referenced Full Paths :* Pointer
-> *Depth :* Longint
)
=> *Return Count :* Longint

	Parameter	Type	Description
	Target Folder Full Path	Text	Full path to target folder to scan
	Referenced Full Paths	Pointer	Pointer to text array to contain full paths to all folders
	Depth	Longint	Number of folders to traverse within target folder to retrieve folders listing
	Return Count	Longint	Number of full paths to folders returned by calling this routine

The method ***FILE_Get_FPaths_HFolders*** will scan a hierarchical folder set to a specified depth from a specified starting target folder and return the full paths of all folders contained within.

This method is ideal for traversing a directory structure to find a particular folder of document within a hierarchical forlder structure in the file system.

Target Folder Full Path is a text value containing the full path of the target folder to begin scanning from. Only full folder paths within this specified target folder will be returned.

Referenced Full Paths is a pointer to a text array to contain the full paths of all folders found in this method. The full paths will be in no particular order within this array but will all be within the specified target folder up to including all folders traversed to the depth specified.

Depth is a longint value containing depth of the number of enclosing folders traverse when building the list of full paths to enclosing folders. A value of one (1) indicates to only return the full paths of folders directly within the target folder. A value of two (2) indicates to only return the full paths of folders directly within the target folder and full paths for all of the folders directly within the those. Any positive value can be passed for this parameter. A value of MAXLONG for this parameter indicates scanning is to be done for all folders within the target folder regardless of depth; obviously, beware of amount of value that may potentially returned when calling this routine with a very high depth value or a target folder containing a complex and expansive directory structure.

Return Count is a number of full paths returned in the *Referenced Full Paths* array.

Note: the method ***FILE_Get_FPaths_HFolders*** was added in BASh v1.8.2.

FILE_Get_Path_Parent

FILE_Get_Path_Parent *(Full Path ; Platform) => Full Parent Path*

FILE_Get_Path_Parent
(
-> *Full Path :* Text
-> *Platform :* Longint
)
=> *Full Parent Path :* Text

	Parameter	Type	Description
	Full Path	Text	Full path to folder or document
	Platform	Longint	Platform indicator value which *Full Path* is valid upon (platform indicators are designated by the 4D constants Platform Properties)
	Full Parent Path	Text	Full path to parent of *Full Path*

The method ***FILE_Get_Path_Parent*** will return the full path to the enclosing directory for a specified full path to a document or directory. This method will operate correctly for paths on either Macintosh or Windows and it will automatically detect whether the full path specified is to a folder or a document.

Full Path is the full path to either a document or directory on either Macintosh or Windows.

Platform is the platform indicator value which *Full Path* is valid upon. Platform indicators are designated by the 4D constants Platform Properties.

Full Parent Path returns the full path to the parent directory of *Full Path*. *Full Parent Path* can be either the full path to

the directory containing the directory specified by *Full Path* or *Full Parent Path* can be the full path to the directory containing the document specified by *Full Path*.

Note: the method ***FILE_Get_Path_Parent*** was added in BASh v1.5.3.

FILE_Get_Platform_for_Path

FILE_Get_Platform_for_Path *(Full or Relative Path) => Path Platform*

FILE_Get_Platform_for_Path

(

-> *Full or Relative Path :* Text

)

=> *Path Platform :* Longint

	Parameter	Type	Description
	Full or Relative Path	Text	Full or relative path to determine the platform for
	Path Platform	Longint	Platform for path provided

The method ***FILE_Get_Platform_for_Path*** will return the platform which a specified path is formatted for. The platform for the provided path is determined by the first path delimiter found (after skipping any Windows drive designator which may exist in a Window full path value).

Full or Relative Path is a full or relative path to determine the platform it is formatted for.

Path Platform is the path *Full or Relative Path* is formatted for. The value returned will correspond to Power Macintosh and Windows constants from the native **Platform Properties** constants group. If the platform for the provided path can not be determined, this value will be set to zero (0).

Note: the method ***FILE_Get_Platform_for_Path*** was added in BASh v1.8.0.

FILE_Get_Volumes_Names

FILE_Get_Volumes_Names *(Referenced Text Array)*

FILE_Get_Volumes_Names
(
-> *Referenced Text Array :* Text
)

	Parameter	Type	Description
	Referenced Text Array	Pointer	Referenced array to contain list of volumes available on the current, local machine

The method ***FILE_Get_Volumes_Names*** will return the list of volumes available on the current, local machine.

FILE_Get_Volumes_Names acts similar to the native 4th Dimension command **VOLUME LIST**, with the exception that both platforms will return volumes names as fully qualified directory paths, including the trailing item delimiter. This correction allows for completely seamless integration of this command across platforms in 4D.

Referenced Text Array is a referenced text or string array which is to contain the list of volumes available on the current, local machine.

Note: the method ***FILE_Get_Volumes_Names*** was added in BASh v1.5.4.

FILE_qi_Document_Exists

FILE_qi_Document_Exists *(Full Document Path)* => *Exists*

FILE_qi_Document_Exists
(
-> *Full Document Path :* Text
)
=> *Exists :* Longint

	Parameter	Type	Description
	Full Document Path	Text	Full path to document to check for existence
	Exists	Longint	Existence indicator value

The method ***FILE_qi_Document_Exists*** will return an indicator value for whether a specified document at a specified location actually exists.

Full Document Path is the full path to a document on the current machine.

Exists is an indicator value for whether the document *Full Document Path* exists. If *Exists* is one (1) then the document *Full Document Path* actually exists on the current machine. If *Exists* is zero (0) then the document *Full Document Path* does not exist on the current machine.

Note: the method ***FILE_qi_Document_Exists*** was added in BASh v1.5.3.

FILE_qi_Folder_Exists

FILE_qi_Folder_Exists *(Full Folder Path)* => *Exists*

FILE_qi_Folder_Exists
(
-> *Full Folder Path :* Text
)
=> *Exists :* Longint

	Parameter	Type	Description
	Full Folder Path	Text	Full path to directory to check for existence
	Exists	Longint	Existence indicator value

The method ***FILE_qi_Folder_Exists*** will return an indicator value for whether a specified directory at a specified location actually exists.

Full Folder Path is the full path to a directory on the current machine.

Exists is an indicator value for whether the directory *Full Folder Path* exists. If *Exists* is one (1) then the directory *Full Folder Path* actually exists on the current machine. If *Exists* is zero (0) then the directory *Full Folder Path* does not exist on the current machine.

Note: the method ***FILE_qi_Folder_Exists*** was added in BASh v1.5.3.

FILE_qi_Volume_Exists

FILE_qi_Volume_Exists *(Named Volume)* => *Exists*

FILE_qi_Volume_Exists
(
-> *Named Volume :* Text
)
=> *Exists :* Longint

	Parameter	Type	Description
	Named Volume	Text	Volume name to confirm existence of
	Exists	Longint	qi for whether volume named exists or not

The method ***FILE_qi_Volume_Exists*** will confirm whether a named volume actually exists on the current, local machine.

Named Volume is a named volume to check for existence on the current, local machine.

Exists is qi for whether *Named Volume* exists on the current local machine. If *Named Volume* exists, *Exists* will be set to one (1); if the named volume does not exist, *Exists* will be set to zero (0).

Note: the method ***FILE_qi_Volume_Exists*** was added in BASh v1.5.4.

FILE_Rename_Document

FILE_Rename_Document *(Document Full Path ; New Document Name)*

FILE_Rename_Document
(
-> *Document Full Path :* Text
-> *Document New Name :* String[32]
)

	Parameter	Type	Description
	Document Full Path	Text	Full path to document to rename
	Document New Name	String[32]	New name of document

The method ***FILE_Rename_Document*** will rename an existing specified document to a new specified name. This is done using the native 4D command **MOVE DOCUMENT**.

Document Full Path is the full path to a document to rename.

Document New Name is the new name to give the specified document.

Note: the method ***FILE_Rename_Document*** was added in BASh v1.8.0.

FILE_Resolve_Alias

FILE_Resolve_Alias *(Full Path to Resolve) => Resolved Full Path*

FILE_Resolve_Alias
(
-> *Full Path to Resolve :* Text
)
=> *Resolved Full Path :* Text

	Parameter	Type	Description
	Full Path to Resolve	Text	Full path to document to resolve all aliases for
	Resolved Full Path	Text	Full path to document without any aliases

The method ***FILE_Resolve_Alias*** will check every directory and document within a specified full path and resolve any aliases, returning a full path without any aliases. This method will work for both full paths to directories and full paths to documents.

Full Path to Resolve is the full path to a directory or document which is to have all aliases resolved.

Resolved Full Path is the full path to the target directory or document *Full Path to Resolve* without any aliases anywhere in the path hierarchy.

Note: the method ***FILE_Resolve_Alias*** was added in BASh v1.6.0.

FILE_Translate_Path_by_OS

FILE_Translate_Path_by_OS *(Source Path ; Target Platform ; Windows Drive Letter)*
=> Translated Path

FILE_Translate_Path_by_OS
(
-> Source Path : Text
-> Target Platform : Longint
-> Windows Drive Letter : String[1]
)
=> Translated Path : Text

	Parameter	Type	Description
	Source Path	Text	Original path to transate
	Target Platform	Longint	
	Windows Drive Letter	String[1]	

	Parameter	Type	Description
	Translated Path	Text	

The method ***FILE_Translate_Path_by_OS*** will attempt to translate a full or relative path from one platform to another. No considerations are made to the actual existence of the directories or documents referenced by the paths, nor is much consideration even made for the valid formatting of the paths. Rather, this method utilizes an advanced algorithm to merely translate the characters of a path assuming similar directory structures between the platforms.

Source Path is the original full or relative path to be translated. This can be a path to a directory or a document.

Target Platform is the platform to translate the supplied path to. This value should be either Power Macintosh or Windows, constants available within the **Platform Properties** native 4D constants group.

Windows Drive Letter is the single letter of the drive to use when translating a full path to Windows. In all other circumstances, this value is ignored and can just be left empty.

Translated Path is the specified path translated to the specified target platform.

Note: the method ***FILE_Translate_Path_by_OS*** was added in BASh v1.8.0.

FMAP Module

The FMAP module provides simple access to document and MIME type mappings between Macintosh and Windows. At its core, the FMAP module provides a means to map Macintosh document type codes, to Windows document extensions, and MIME types to each other. When interchanging documents, especially online, this mapping provides a simple means to coordinate different document types. Default values for each environment can be set and entries can be named for simplified storage, identification, and retrieval. As well, mapping to a Macintosh document creator code is provided for each entry to provide a means to differentiate documents of the same type though having different applications that were used to create them.

Note: the FMAP module was initially added in BASh v1.8.0.

FMAP Storage Format

The basic format for storage of the FMAP entries in 4D follows closely the storage format for the 'fMap' custom resource (see the RES module, later in this manual). In 4D this is accomplished using a series of related array variables (a stack). The FMAP stack consists of the following entries, one for each row in the stack:

Column Title	Data Type
Entitled	Text
Macintosh Type Code	Longint
Windows Type Code	String [5]
Macintosh Creator Code	Longint
MIME Type	Text
Active on Macintosh	Boolean
Active on Windows	Boolean
Default for Type	Boolean

Each row in the FMAP stack contains a field for the storage of each of these items. Macintosh type and creator codes can be handled as strings in 4D by first converting using the CONV module. The active values indicate whether the current row of values are valid for each of the applicable platforms; with these booleans, entries can be made for just one platform in a row, as needed. The default type boolean value indicates the row entry is the default entries for the Macintosh document type and Windows extension values given.

FMAP Stacks

The FMAP stack consists of entries at both a process level and interprocess level in 4D. This means, there are in actuality two stacks being maintained internally within the FMAP module. All of the FMAP routines provide transparency for the maintenance and usage of both the process level and interprocess level FMAP stacks, so there is no need to provide extended functionality in your code to account for this.

This does become important though when creating your own entries for inclusion in the FMAP stack. The method ***FMAP_Load_Stack_s*** provides a means for custom FMAP entries to be loaded into the FMAP stack. When loading custom entries, it must be indicated when using the above routine which stack they should be loaded into, the process level stack or the interprocess level stack.

For use in the FMAP module, all entries in the process level FMAP stack are searched first. If no matching entry is found, then the interprocess level stack is used to find a matching entry. This provides a means for custom, process level dependent, FMAP usage throughout your 4D applications. This is exceptionally powerful when combined, for instance, with a web server in 4D; different process within 4D can be using the FMAP module while each process has maintained different values within the FMAP stack. Keep in mind though that the process level stack in the FMAP module is cleared when a process finishes running; all entries in the process level FMAP stack will be removed at this point, leaving entries for all other FMAP stacks for other processes still running.

The default values in the interprocess FMAP stack are loaded when BASh is initialized. These values correspond directly the 'fMap' resources within the Affix BASh document.

FMAP_Clear_Stack_s

FMAP_Clear_Stack_s *(Stack Scope)*

FMAP_Clear_Stack_s
(
-> *Stack Scope :* Longint
)

	Parameter	Type	Description
	Stack Scope	Longint	Indicator for process or interprocess level stack entries

The method ***FMAP_Clear_Stack_s*** will clear all entries from either the process or interprocess level FMAP stack as indicated by the provided flag.

Stack Scope is an indicator for whether entries at the process level or interprocess level of the FMAP stack should be cleared. Setting this value to one (1) will clear all entries at the interprocess level; setting this value to two (2) will clear all entries at the process level.

Note: the method ***FMAP_Clear_Stack_s*** was added in BASh v1.8.0.

FMAP_ERROR

FMAP_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***FMAP_ERROR*** acts as a callback method from within the FMAP module for errors that may occur. Any time an error condition is detected within the FMAP module, a call to the method ***FMAP_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 FMAP method which call the ***FMAP_ERROR*** method.

The ***FMAP_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 ***FMAP_ERROR*** was added in BASh v1.8.0.

FMAP_Get_DCrCode_by_MType_s

FMAP_Get_DCrCode_by_MType_s *(Mac Document Type Code) => Default Mac Document Creator Code*

FMAP_Get_DCrCode_by_MType_s

(

-> *Mac Document Type Code :* Longint

)

=> *Default Mac Document Creator Code :* Longint

	Parameter	Type	Description
	Mac Document Type Code	Longint	Macintosh document type code to lookup in FMAP stack
	Default Mac Document Creator Code	Longint	Default Macintosh document creator code matching search criteria within FMAP stack

The method ***FMAP_Get_DCrCode_by_MType_s*** will return the default Macintosh document creator code for a specified Mac document type code from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Mac Document Type Code is the Macintosh document type value to search for in the FMAP stack. Only entries flagged as default entries for the type specified will be considered valid search results within the FMAP stack.

Mac Document Creator Code is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_DCrCode_by_MType_s*** was added in BASh v1.8.0.

FMAP_Get_DMType_by_MType_s

FMAP_Get_DMType_by_MType_s *(Mac Document Type Code) => Default Mac Document Type Code*

FMAP_Get_DMType_by_MType_s
(
-> *Mac Document Type Code :* Longint
)
=> *Default Mac Document Type Code :* Longint

	Parameter	Type	Description
	Mac Document Type Code	Longint	Macintosh document type code to lookup in FMAP stack

	Parameter	Type	Description
	Default Mac Document Type Code	Longint	Default Macintosh document type code matching search criteria within FMAP stack

The method ***FMAP_Get_DMType_by_MType_s*** will return the default Mac document type code for a specified Mac document type code from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Mac Document Type Code is the Macintosh document type value to search for in the FMAP stack. Only entries flagged as default entries for the type specified will be considered valid search results within the FMAP stack.

Default Mac Document Type Code is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_DMType_by_MType_s*** was added in BASh v1.8.0.

FMAP_Get_DMType_by_Title_s

FMAP_Get_DMType_by_Title_s *(Row Title) => Default Mac Document Type Code*

FMAP_Get_DMType_by_Title_s

(

-> *Row Title :* Text

)

=> *Default Mac Document Type Code :* Longint

	Parameter	Type	Description
	Row Title	Text	Row title to lookup in FMAP stack
	Default Mac Document Type Code	Longint	Default Macintosh document type code matching search criteria within FMAP stack

The method ***FMAP_Get_DMType_by_Title_s*** will return the default Mac document type code for a specified row title from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Row Title is the title of the row entry to search for in the FMAP stack. Only entries flagged as default entries for the type specified will be considered valid search results within the FMAP stack.

Default Mac Document Type Code is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_DMType_by_Title_s*** was added in BASh v1.8.0.

FMAP_Get_DSuffix_by_MType_s

FMAP_Get_DSuffix_by_MType_s *(Mac Document Type Code) => Default Windows Suffix*

FMAP_Get_DSuffix_by_MType_s

(

-> *Mac Document Type Code :* Longint

)

=> *Default Windows Suffix :* String [5]

	Parameter	Type	Description
	Mac Document Type Code	Longint	Macintosh document type code to lookup in FMAP stack
	Default Windows Suffix	String [5]	Default Windows extension matching search criteria within FMAP stack

The method ***FMAP_Get_DSuffix_by_MType_s*** will return the default Windows extension for a specified Macinotsh document type code from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Mac Document Type Code is the Macintosh document type value to search for in the FMAP stack. Only entries flagged as default entries for the type specified will be considered valid search results within the FMAP stack.

Default Windows Suffix is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_DSuffix_by_MType_s*** was added in BASh v1.8.0.

FMAP_Get_DTitles_All_s

FMAP_Get_DTitles_All_s *(Referenced Mac Document Type Codes ; Referenced Row Titles) => Row Count*

FMAP_Get_DTitles_All_s

(

-> *Referenced Mac Document Type Codes :* Pointer
-> *Referenced Row Titles :* Pointer

)

=> *Row Count :* Longint

	Parameter	Type	Description
	Referenced Mac Document Type Codes	Pointer	Pointer to longint array to hold Macintosh document type codes
	Referenced Row Titles	Pointer	Pointer to text array to hold all row titles for default entries in FMAP stack
	Row Count	Longint	Number of rows returned in referenced parameters

The method ***FMAP_Get_DTitles_All_s*** will populate two arrays with the default row titles and their corresponding Macintosh document type codes for all entries in the FMAP stack flagged as default entries. This routine is useful for providing interface for selecting document types by users when the document type may not be easily determined within user intervention.

Referenced Mac Document Type Codes is a pointer to a longint array to be populated with all of the Macintosh document type codes in the FMAP stack marked as defaults.

Referenced Row Titles is a pointer to a text array to be populated with all of the row titles for entries in the FMAP stack marked as defaults. These values correspond to entries in *Referenced Mac Document Type Codes* as a well formed stack.

Row Count is the number of rows returned in the referenced parameters.

Note: the method ***FMAP_Get_DTitles_All_s*** was added in BASh v1.8.0.

FMAP_Get_MIME_by_MType_s

FMAP_Get_MIME_by_MType_s *(Mac Document Type Code) => MIME Type*

FMAP_Get_MIME_by_MType_s

(

-> *Mac Document Type Code :* Longint

)

=> *MIME Type :* Text

	Parameter	Type	Description
	Mac Document Type Code	Longint	Macintosh document type code to lookup in FMAP stack
	MIME Type	Text	MIME type matching search criteria within FMAP stack

The method ***FMAP_Get_MIME_by_MType_s*** will return the MIME type for a specified Macinotsh document type code from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Mac Document Type Code is the Macintosh document type value to search for in the FMAP stack.

MIME Type is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_MIME_by_MType_s*** was added in BASh v1.8.0.

FMAP_Get_MIME_by_Suffix_s

FMAP_Get_MIME_by_Suffix_s *(Windows Suffix) => MIME Type*

FMAP_Get_MIME_by_Suffix_s
(
-> *Windows Suffix :* String [5]
)
=> *MIME Type :* Text

	Parameter	Type	Description
	Windows Suffix	String [5]	Windows document suffix to lookup in FMAP stack
	MIME Type	Text	MIME type matching search criteria within FMAP stack

The method ***FMAP_Get_MIME_by_Suffix_s*** will return the MIME type for a specified Windows document suffix from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Windows Suffix is the Windows document suffix to search for in the FMAP stack.

MIME Type is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_MIME_by_Suffix_s*** was added in BASh v1.8.0.

FMAP_Get_Title_by_MType_s

FMAP_Get_Title_by_MType_s *(Mac Document Type Code) => Row Title*

FMAP_Get_Title_by_MType_s
(
-> *Mac Document Type Code :* Longint
)
=> *Row Title :* Text

	Parameter	Type	Description
	Mac Document Type Code	Longint	Macintosh document type code to lookup in FMAP stack
	Row Title	Text	Row title matching search criteria within FMAP stack

The method ***FMAP_Get_Title_by_MType_s*** will return the row title for a specified Macinotsh document type code from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Mac Document Type Code is the Macintosh document type value to search for in the FMAP stack.

Row Title is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_Title_by_MType_s*** was added in BASh v1.8.0.

FMAP_Get_Title_by_Suffix_s

FMAP_Get_Title_by_Suffix_s *(Windows Suffix) => Row Title*

FMAP_Get_Title_by_Suffix_s
(
-> *Windows Suffix :* String [5]
)

=> *Row Title :* Text

	Parameter	Type	Description
	Windows Suffix	String [5]	Windows document suffix to lookup in FMAP stack
	Row Title	Text	Row title matching search criteria within FMAP stack

The method ***FMAP_Get_Title_by_Suffix_s*** will return the row title for a specified Windows document suffix from the FMAP stack. Process level entries in the FMAP stack will be scanned before interprocess level entries in the FMAP stack.

Windows Suffix is the Windows document suffix to search for in the FMAP stack.

Row Title is the first entry found within the FMAP stack corresponding to the search criteria for this method. If no matching entries are found in the FMAP stack, a NULL value will be returned.

Note: the method ***FMAP_Get_Title_by_Suffix_s*** was added in BASh v1.8.0.

FMAP_Load_Stack_s

FMAP_Load_Stack_s *(Stack Scope ; Resource Fork File Reference ; qi Clear Stack) => New Row Count*

FMAP_Load_Stack_s
(
-> *Stack Scope :* Longint
-> *Resource Fork File Reference :* Time
-> *qi Clear Stack :* Longint
)
=> *New Row Count :* Longint

	Parameter	Type	Description
	Stack Scope	Longint	Indicator for whether to load to the process or interprocess level FMAP stack

	Parameter	Type	Description
	Resource Fork File Reference	Time	Document reference for already open resource fork containing 'fMap' resource to load
	qi Clear Stack	Longint	Indicator for whether current entries in the scoped FMAP stack should be cleared before loading new entries
	New Row Count	Longint	Numbe of new entries loaded into FMAP stack

The method ***FMAP_Load_Stack_s*** will load new entries into the FMAP stack from a specified open resource document. New entries are loaded from 'fMap' resources in the specified open resource document. New entries can be loaded into either the process or interprocess level FMAP stack and the scoped stack can optionally be cleared before new entries are loaded. The number of new entries loaded is returned from this method call.

Stack Scope is an indicator for whether the process level or interprocess level FMAP stack should be used within this method. A value of one (1) for this parameter will operate on the interprocess level stack and a value of two (2) will operate on the process level stack. This value will effect which stack scope is used to load the new entries into and optionally which stack will be cleared before the new values are loaded.

Resource Fork File Reference is a standard 4D document reference for the resource document containing the 'fMap' resources to load. All resources of type 'fMap' will be loaded from the specified document. This method assumes the specified resource document is already open and ready to be read.

qi Clear Stack is an indicator for whether the scoped stack should be cleared of all entries before loading new entries. If this values is set to zero (0), the new values will merely be prepended to the scoped FMAP stack. If this value is one (1), the scoped FMAP stack will first be cleared and then new entries will be loaded.

New Row Count is number of rows loaded with this routine. A value of less than zero indicates an error occurred and no new entries were loaded.

Note: the method ***FMAP_Load_Stack_s*** was added in BASh v1.8.0.

IB Module

The Indexed BLOB (IB) module provides a convenient structured storage mechanism for storage space in memory. When plain variable storage is not structure enough and a more organized storage mechanism would be convenient, the IB module provides the simple and direct means to organize data stored in RAM.

Basically, the IB module consists of routines for handle storage of data into an indexed BLOB. The BLOB to be used to store data is set to contain a fixed number of index entries upon its creation. Once created, then, values can be stored into the BLOB by specifying the index entry into the BLOB to store the individual values under. Data can obviously be retrieved and removed from the indexed BLOB at a future time. When memory starts to become a little tight, routines are provided to automatically compact an indexed BLOB, as well, thereby defragmenting the memory occupied by the indexed BLOB and recovering memory. Once an indexed BLOB is no longer needed, it can then be completely deconstructed and cleared.

Routines in the IB module make all of this a fairly simple matter of course. The IB module handles and construction, storage management, internal indexing, compaction, and deconstruction of indexed BLOBs (IBs). The developer need only use the routines in the IB module to manage the data to be stored and retrieved in IBs.

Note: the IB module was added in BASh v1.8.2.

IB_Append_Variable_by_Key

IB_Append_Variable_by_Key *(Referenced IB ; Key ; Referenced Variable) => Bytes Appended*

IB_Append_Variable_by_Key
(
-> *Referenced IB :* Pointer
-> *Key :* Longint
-> *Referenced Variable :* Pointer
)
=> *Bytes Appended :* Longint

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable
	Key	Longint	Key to value in IB variable
	Referenced Variable	Pointer	Referenced variable to append value into existing IB index
	Bytes Appended	Longint	Number of bytes appended to contain specified variable and value

The method ***IB_Append_Variable_by_Key*** appends a specified variable to the value stored in a specified key value in a specified IB variable. The native **VARIABLE TO BLOB** command is used to stored the variable, so all variable typing and sizing information is stored with the variable value. The number of bytes appended to hold the specified variable and its value is returned from this routine.

The value stored in the specified key in the specified IB variable is often made larger by use of this command. This can result in a movement of the block of memory in the IB variable used to store the value. If this occurs, the previous block of memory is not available for use by the IB variable until the it is compacted. Frequent use of this command should be accompanied by occasional use of the ***IB_Compact_IB*** routine.

Referenced IB is a pointer to an IB variable.

Key is a longint variable containing the key to a value in the specified IB variable.

Referenced Variable is a pointer to a variable to append into *Key* value of *Referenced IB*. The type of the referenced variable can be any type except a pointer, an array of pointers, or a two dimensional array.

Bytes Appended is a longint value set to contain the total number of bytes occupied by the variable and value appended. If there is an error appending the variable or its value, this will be set to negative one (-1).

Note: the method ***IB_Append_Variable_by_Key*** was added in BASh v1.8.2.

IB_Clear_Value_by_Key

IB_Clear_Value_by_Key *(Referenced IB ; Key)*

IB_Clear_Value_by_Key
(
-> *Referenced IB :* Pointer
-> *Key :* Longint
)

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable
	Key	Longint	Key to value in IB variable

The method ***IB_Clear_Value_by_Key*** will clear the value stored in a specified key in a specified IB variable. The key will not be set as free (i.e. not in use) but the value will be set to NULL.

Referenced IB is a pointer to an IB variable.

Key is a longint variable containing the key to a value in the specified IB variable.

Note: the method ***IB_Clear_Value_by_Key*** was added in BASh v1.8.2.

IB_Compact_IB

IB_Compact_IB *(Referenced IB)*

IB_Compact_IB
(
-> *Referenced IB :* Pointer
)

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable

The method ***IB_Compact_IB*** will compact the space used by a specified IB variable. This effectively defragments the memory used within an IB variable.

To speed the operations performed within an IB variable, on the fly degragmenting and compaction of an IB variable was not implemented. The result is that some operation can cause the memory used within an IB variable to become fragmented, resulting in unused memory within an IB variable. To recover this memory, occasional use of this compaction routine will defragment the memory used in an IB variable.

The commands within the IB module that can cause fragmentation of memory, and under what conditions this occurs, are noted in each individual command. Depending on the exact usage of an IB variable, compaction may or may not be needed. As well, different usages of an IB variable may increase or decrease the frequency in which compaction of an IB variable is needed.

Referenced IB is a pointer to an IB variable.

Note: the method ***IB_Compact_IB*** was added in BASh v1.8.2.

IB_ERROR

IB_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

IN_ERROR

(

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

)

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***IB_ERROR*** acts as a callback method from within the IB module for errors that may occur. Any time an error condition is detected within the IB module, a call to the method ***IB_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 IB method which call the ***IB_ERROR*** method.

The ***IB_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 ***IB_ERROR*** was added in BASh v1.8.2.

IB_Get_Key_Free

IB_Get_Key_Free *(Referenced IB) => Key*

IB_Get_Key_Free
(
-> *Referenced IB :* Pointer
)
=> *Key :* Longint

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable
	Key	Longint	Key to value in IB variable

The method ***IB_Get_Key_Free*** will return a key that has is currently free and not in used within a specified IB variable. Whenever an key into an IB variable is needed for storing a value, this routine should be used to retrieve a free key for this use.

Referenced IB is a pointer to an IB variable.

Key is a longint variable containing a free key in the specified IB variable. If there are no available keys to return or some other error occurs, *Key* will be set to negative one (-1). All value key values in an IB variable are positive longints.

Note: the method ***IB_Get_Key_Free*** was added in BASh v1.8.2.

IB_Get_One

IB_Get_One *(Index Total Count) => Referenced IB*

IB_Get_One
(
-> *Index Total Count :* Longint
)
-> *Referenced IB :* Pointer

	Parameter	Type	Description
	Index Total Count	Longint	Total number of index entries (keys) in IB variable to create
	Referenced IB	Pointer	Pointer to IB variable

The method ***IB_Get_One*** will create a new IB variable with a specified number of index entries and return a pointer to the newly created IB variable.

Index Total Count is a longint value indicating the total number of index entries (keys) to have available in the IB variable to be created. The number of index entries in an IB variable is set when the IB variable is created and can not be changed. The number of index entries can be between one (1) and 65,535 (notice this is an unsigned integer [unsigned decimal word], a two byte value).

Referenced IB is a pointer to an IB variable.

Note: the method ***IB_Get_One*** was added in BASh v1.8.2.

IB_Get_Value_by_Key

IB_Get_Value_by_Key *(Referenced IB ; Key ; Referenced BLOB) => Value Bytes*

IB_Get_Value_by_Key
(
-> *Referenced IB :* Pointer
-> *Key :* Longint
-> *Referenced BLOB :* Pointer
)
=> *Value Bytes :* Longint

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable
	Key	Longint	Key to value in IB variable
	Referenced BLOB	Pointer	Referenced BLOB to contain value retrieved
	Value Bytes	Longint	Number of bytes of return value

The method ***IB_Get_Value_by_Key*** will retrieve and return a value for a specified key in a specified IB variable. The value will be returned in a specified referenced BLOB and the number of bytes return will also be returned.

Referenced IB is a pointer to an IB variable.

Key is a longint variable containing the key to a value in the specified IB variable.

Referenced BLOB is a pointer to a BLOB to contain the value retrieved. The BLOB is cleared before the retrieved value is set into it directly.

Value Bytes is a longint value returned containing the number of bytes of the returned value. If there is an error in this routine, this value will be set to negative one (-1).

Note: the method ***IB_Get_Value_by_Key*** was added in BASh v1.8.2.

IB_Return_One

IB_Return_One *(Referenced IB)*

IB_Return_One

(

-> *Referenced IB :* Pointer

)

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable

The method ***IB_Return_One*** will return a specified IB variable to the pool of available IB variable. This method should be called whenever a previously retrieve IB variable is no longer to be used. This makes it available for future use by calling ***IB_Get_One***.

Referenced IB is a pointer to an IB variable.

Note: the method ***IB_Return_One*** was added in BASh v1.8.2.

IB_Set_Key_Free

IB_Set_Key_Free *(Referenced IB ; Key)*

IB_Set_Key_Free
(
-> *Referenced IB :* Pointer
-> *Key :* Longint
)

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable
	Key	Longint	Key to value in IB variable

The method ***IB_Set_Key_Free*** set free a specified key in a specified IB variable. Setting a key free in an IB variable makes it available for use again by subsequent calls to ***IB_Get_Key_Free***.

The value stored in the specified key in the specified IB variable will be cleared and the space will not be available again without compaction of the IB variable (the key is available for future use by the memory previously used is not recovered without compaction).

Referenced IB is a pointer to an IB variable.

Key is a longint variable containing the key to a value in the specified IB variable.

Note: the method ***IB_Set_Key_Free*** was added in BASh v1.8.2.

IB_Set_Value_by_Key

IB_Set_Value_by_Key *(Referenced IB ; Key ; Referenced BLOB)*

IB_Set_Value_by_Key
(
-> *Referenced IB :* Pointer

-> *Key :* Longint
-> *Referenced BLOB :* Pointer
)

	Parameter	Type	Description
	Referenced IB	Pointer	Pointer to IB variable
	Key	Longint	Key to value in IB variable
	Referenced BLOB	Pointer	Pointer to BLOB containing value to store in specified key of specified IB variable

The method ***IB_Set_Value_by_Key*** will set a specified value into a specified key of a specified IB variable. Any previous contents stored in the same location will be lost.

The value stored in the specified key in the specified IB variable is often made larger by use of this command. This can result in a movement of the block of memory in the IB variable used to store the value. If this occurs, the previous block of memory is not available for use by the IB variable until the it is compacted. Frequent use of this command should be accompanied by occasional use of the ***IB_Compact_IB*** routine.

Referenced IB is a pointer to an IB variable.

Key is a longint variable containing the key to a value in the specified IB variable.

Referenced BLOB is a pointer to a BLOB containing the value to be stored in the location specified in the IB variable. The complete contents of this BLOB will be stored in the IB variable.

Note: the method ***IB_Set_Value_by_Key*** was added in BASh v1.8.2.

INIT Module

The INIT module is the first module which needs to be dealt with in the BASh component. At this time, it consists of only a single routine. But, this single routine provides initialization for all of the modules within the BASh component.

It is **strongly** recommended that all initialization be done early in the **On Startup** and **On Server Startup** database methods.

And, it is also strongly recommended to not consider any of the methods in the INIT module as reentrant during any single use of the application without it being clearly stated in this documentation.

Note: the INIT module was added in BASh v1.5.4.

INIT_BASh

INIT_BASh *({SEQ Path })*

INIT_BASh
(
-> *SEQ Path :* Text [optional]
)

	Parameter	Type	Description
	SEQ Path	Text	Full or relative path to document resource fork to use for SEQ module storage

The method ***INIT_BASh*** will initialize the whole BASh component.

Before any of the methods within the BASh component can function, ***INIT_BASh*** should be called. It is best to call ***INIT_BASh*** early in the **On Startup** and **On Server Startup** database methods.

There is no need to call ***INIT_BASh*** twice during the operation of the database. And, it is actually discouraged to call the method repeatedly during use of the application.

SEQ Path is a full or relative path to a resource document to use for storage in the SEQ module in BASh. If a relative path is supplied for this parameter, the path is assumed to be relative to the directory containing the current structure document. This parameter is optional and defaults to the resource fork of the current data file if no value is supplied for this parameter.

Note: the method ***INIT_BASh*** was added in BASh v1.5.4.

Note: the *SEQ Path* parameter was added in BASh v1.8.0.

INT Module

The INT module, also known as the interruption module, handles some of the basic interruption processing needed within 4th Dimension. Specifically, the INT module provides stacking functionality for the native **ON ERR CALL** command. By using the commands in the INT module, a 4D developer can easily implement a FILO (first in, last out) stack for methods used by **ON ERR CALL**; this method stack has been dubbed the "OEC Stack". Remember, the **ON ERR CALL** handler is different for each process currently running within 4th Dimension. The INT module maintains a separate OEC Stack for each process.

For usage, it is worth noting that the INT module is best used in conjunction with the PROS module. The INT module uses process variables in 4D which are initialized in processes managed using the PROS module. Whether a process is spawned using ***PROS_New_Process*** or added during operation with the command ***PROS_Beginning_Process*** is irrelevent; the important factor is that one of these two PROS methodologies are used with processes using the INT module.

Note: the INT module was added in BASh v1.6.0.

INT_ERR_Clear_All

INT_ERR_Clear_All *(*

INT_ERR_Clear_All *(*

	Parameter	Type	Description
	(none)	n/a	n/a

The method ***INT_ERR_Clear_All*** clears completely all entries from the OEC Stack and clears the current setting to the **ON ERR CALL** interruption handler for the current process.

Note: the method ***INT_ERR_Clear_All*** was added in BASh v1.6.0.

INT_ERR_Pop_Method

INT_ERR_Pop_Method *(*

INT_ERR_Pop_Method *(*

	Parameter	Type	Description
	(none)	n/a	n/a

The method ***INT_ERR_Pop_Method*** clears the current method installed (pops) within the **ON ERR CALL** interruption handler, removes the last installed entry within the EOC Stack, and installs the next previous method within the OEC Stack as the current **ON ERR CALL** interruption handler method. All of this is done solely in the current process. If before calling ***INT_ERR_Pop_Method*** there is only one method in the OEC Stack then no new **ON ERR CALL** interruption handler will be be installed.

Note: the method ***INT_ERR_Pop_Method*** was added in BASh v1.6.0.

INT_ERR_Push_Method

INT_ERR_Push_Method *(Error Handling Method Name)*

INT_ERR_Push_Method
(
-> Error Handling Method Name : String[32]
)

	Parameter	Type	Description
	Error Handling Method Name	String[32]	Name of method to install as last error handling method

The method ***INT_ERR_Push_Method*** pushes a new method onto the OEC Stack and installs it as the current **ON ERR CALL** interruption handler for the current process. The previously installed **ON ERR CALL** interruption handler method, if any, will no longer be in use.

Error Handling Method Name is the name of the method to be installed as the current **ON ERR CALL** interruption handler method.

Note: the method ***INT_ERR_Push_Method*** was added in BASh v1.6.0

INT_PROS_Error_Callback

INT_PROS_Error_Callback

INT_PROS_Error_Callback (

	Parameter	Type	Description
	(none)	n/a	n/a

The method ***INT_PROS_Error_Callback*** is the default **ON ERR CALL** handler installed when the PROS module has been used to launch a process in 4th Dimension. This method is installed as the first and only **ON ERR CALL** interruption handler in the OEC Stack for the current process.

Other methods can be installed using the ***INT_ERR_*** methods within the INT module of BASh. This method is fully editable for the developer to customize as needed.

Note: the method ***INT_PROS_Error_Callback*** was added in BASh v1.6.0

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:

Type	Value
BLOB	Blob Size is 0
Boolean	False
Date	!00/00/00!
Integer	0
Longint	0
Picture	(*0, empty picture)
Pointer	Nil
Real	0
Text	“ “
Time	†00:00:00†
DTS (String[14]	“00000000000000“

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 errorneous operation of the code with the BASh component.

NULL_Dereference_by_Type

NULL_Dereference_by_Type *(Variable Type) => Referenced NULL Variable*

NULL_Dereference_by_Type
(
-> *Variable Type :* Longint
)
=> *Referenced NULL Variable :* Pointer

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

The method ***NULL_Dereference_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
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***NULL_ERROR*** acts as a callback method from within the NULL 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 NULL method which called 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
	Referenced Variable	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. For referenced arrays, the arrays are resized to zero elements.

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
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 NVP method which called 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.

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
	NVP Name	Text	Named value to extract
	Referenced Names Array	Pointer	Pointer to names array
	Referenced Values Array	Pointer	Pointer to values array
	NVP Value	Text	Value matching NVP Name

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

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

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 packed into *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.

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

	Parameter	Type	Description
	Referenced Names Array	Pointer	Pointer to names array of NVP stack
	Referenced Values Array	Pointer	Pointer to values array of NVP stack
	Column Delimiter ASCII Value	Longint	ASCII Value of the column delimiter to use to pack the NVP stack into text

	Parameter	Type	Description
	Row Delimiter ASCII Value	Longint	ASCII Value of the row delimiter to use to pack the NVP stack into text
	Trailing Row Delimiter Flag	Boolean	Flag to include the trailing row delimiter value in the packed NVP stack
	Packed NVP Text	Text	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

	Parameter	Type	Description
	Packed NVP Text	Text	NVP stack packed into text
	Referenced Names Array	Pointer	Pointer to array to use as names array for unpacked NVP pairings
	Referenced Values Array	Pointer	Pointer to array to use as values array for unpacked NVP pairings
	Column Delimiter ASCII Value	Longint	ASCII Value of the column delimiter used to pack the NVP stack into text
	Row Delimiter ASCII Value	Longint	ASCII Value of the row delimiter used to pack the NVP stack into text
	Comment ASCII Value	Longint	ASCII Value of the comment character used in the packed NVP stack
	NVP Count	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.

NVP_Set_Value_by_Name_s

NVP_Set_Value_by_Name_s *(Name ; Value ; Referenced Names Array ; Referenced Values Array ; qi Add Element)* => *Element Index*

NVP_Set_Value_by_Name_s

(

-> *Name :* Text
-> *Value :* Text
-> *Referenced Names Array :* Pointer
-> *Referenced Values Array :* Pointer
-> *qi Add Element :* Longint

)

=> *Element Index :* Longint

	Parameter	Type	Description
	Name	Text	Name of element to edit or insert value for
	Value	Text	Value to set
	Referenced Names Array	Pointer	Pointer to text array containing names
	Referenced Values Array	Pointer	Pointer to text array containing values

	Parameter	Type	Description
	qi Add Element	Longint	Flag for whether to insert new element in stack when *Name* is not found
	Element Index	Longint	Index of element edited or inserted

The method ***NVP_Set_Value_by_Name_s*** will set the value of or optionally insert a new row for a specified name in referenced names and values arrays.

Name is name of the element to edit. If a row in the referenced arrays is not found matching *Name*, this method can optionally insert a new row and then set *Name* and *Value* into these elements.

Value is the value to set into the found element or new row in the referenced arrays.

Referenced Names Array is a pointer to the text array containing the names.

Referenced Values Array is a pointer to the text array containing the values.

qi Add Element is a flag for whether to insert a new element in the referenced arrays when *Name* is not found to match any elements in *Referenced Names Array*. If this value is set to one (1), a new element will be added to the end of the referenced arrays under the above conditions. If this value is set to zero (0), a new element will not be added under the above conditions.

Element Index is the index of the element in the referenced arrays matching the *Name* and thereby edited. If a new element is added to the referenced arrays, this value will be set to the index of the new elements. If no element was edited and no element was added to the referenced arrays, this value will be set to negative one (-1).

Note: the method ***NVP_Set_Value_by_Name_s*** was added in BASh v1.8.0.

PROS Module

The PROS module provides very basic functionality involving 4th Dimension processes. A few holes in the native functionality have been filled in and some functionality natively available has been more conveniently packages by the methods in the PROS module.

Note: the PROS module was added in BASh v1.5.4.

Changes as of BASh v1.6.0

As of BASh v1.6.0, the PROS module has been expanded considerably. It now has all of the functionality needed for a complete process management system within 4th Dimension. Much of the mundane coding needed to handle multiple processes within 4D is now handled directly within the PROS module. As well, many hooks are now available within the PROS module for additional features coming to the BASh component.

It is highly recommended that developers begin implementing and using the methods available within the PROS module. These methods provide the means for many of the process level initialization and cleanup routines needed by other modules within BASh, and other 4D components based upon BASh, to be properly handled.

Implementing the PROS Module

The process module can be implemented in one of two different ways. Both means provide for full compatibility of existing and new process level functionality within DSTi's line of 4D components. There are only slight differences between each implementation method.

First, a 4D developer can rely on the PROS module for handling the launching of all processes within 4D. Using the method ***PROS_New_Process***, a 4D developer has an analogous mechanism for procedurally launching processes. This method should be used to replace all calls to the native **New Process** command in 4D when implementing the PROS module in this fashion. Read the description of the ***PROS_New_Process*** method for details on creating a new process with the PROS module.

Second, a 4D developer can rely on the existing, native implementation for launching new processes in 4D. The PROS module can still be compatible with such cases by making a couple of calls within the new processes handler method. Immediately at the beginning of a new process launched without use of the PROS module, a call to the PROS method ***PROS_Beginning_Process*** should be made; this will initialize the

process within the PROS module and any other 4D components using the PROS module's process level initialization hooks. At the end of the same process handler method, a call to ***PROS_Ending_Process*** should be made; this will clean up all process level data structures used by the PROS module and any other 4D components using the PROS module's process level clean up hooks.

Either of these process handling methods can be used. And, a developer can even choose to implement a mixing of these two process handling methodologies within the same structure; for instance, a new process being developed can be handled using the ***PROS_New_Process*** method while existing processes within the same structure use the ***PROS_Beginning_Process*** and ***PROS_Ending_Process*** methods. Read the appropriate sections for each of these commands for details about how different methods and the options for implementation.

The PROS Stack

Whenever a new process is launched within 4D and the PROS module is made aware of it (either from the PROS module launching it with the method ***PROS_New_Process*** or by initialization within the handler method of the new process with a call to ***PROS_Beginning_Process***), the PROS module will keep track of the process and many of its attributes.

This information is stored in a data structure entitled the "PROS Stack".

The PROS Stack maintains a listing of process names, process IDs, unique process IDs, primary window names, close box method names, and control method names. When

When the BASh component is initialized, the PROS Stack is initialized and the default 4D process, commonly called the "User/Custom Menus Process" is placed into the PROS Stack; in the PROS Stack, this process is entitled "Default_Process".

All methods within the PROS module that contain the stack designator in their titles (i.e. ending with "_s") rely on the PROS Stack for their information.

The 4D developer using BASh need not do anything to initialize, maintain, or clear the PROS Stack. This is handled completely internally within the PROS module. As long as the PROS module is notified of new processes properly, the PROS Stack and all of the functionality of the

PROS module will be available to the 4D developer throughout their 4D project.

Process Control Methods

A process control method is the installed method which is initially called when a new process is launched in 4D. When using the native 4D command **New Process**, this method is commonly called the “Process Method”.

The use of the Process Control Method within the PROS module is completely analogous to the native implementation in 4D. The only differences involve the wrapping of the Process Control Method with code used by the PROS module for initializing process level data structures and cleaning up once a process is going to end.

Process Name Fragments

Every 4D developer should already be familiar with process names as used by 4th Dimension. The 4D Language Reference provides fairly complete documentation on the use of process names within 4D.

It has been found though that additional functionality involving process names can be extremely helpful to a 4D developer. The PROS module within BASh address many of these useful pieces of functionality through the use of "Process Name Fragments".

Basically, a Process Name Fragment is just the beginning characters of a process name. Often, a process name fragment is the complete process name. But, within the PROS module, process name fragments can provide additional features. Almost exclusively, the additional functionality afforded by process name fragments within the PROS module are available solely through the method ***PROS_New_Process***. This method takes a process name fragment as a parameter.

Within the PROS Stack, every process must have a fully unique name. This is not true of the native implementation of 4D processes. This provides the benefit to the 4D developer of implementing more powerful and flexible multiprocess user interfaces with minimal effort. For instance, when a user selects a menu item entitled “Customers” from anywhere within the application, a list of customers would normally be displayed. If the user were to again select the Customers menu item, another process with the same name, and another window with the same interface, would by default be presented to the user. This can often be confusing to a user. To make the application check for whether the second selection of the Customers menu item would be launching another

process requires additional code on the part of the programmer. Using the PROS module, no additional code is required; rather, selecting the menu item Customers by the user would call the same code that contains a call to ***PROS_New_Process*** ; this call determines whether the Customers process already exists and brings it to the front, instead of launching a new process, automatically for the developer. This is a huge savings in repetitive code for the 4D developer.

In the event that multiple processes with the same functional elements are needed, the PROS module provides a mechanism for this as well. Whenever a process name fragment ends with an underscore ("_"), the PROS module assumes that the process being created is a "repeated" process that can be running multiple copies of itself. The method ***PROS_New_Process*** will automatically scan the PROS Stack for other instances of the process that exist and will find the "lowest" instance in which the process name fragment is not in existence. This will then be used, as a formatted, three (3) digit numeral, to construct the full process name of the new process created.

For instance, if a new process with a process name fragment of "Background_Reporter_" is being created for the very first time, then the PROS module will determine that the full process name of the new process should be "Background_Reporter_001". Before this process ends, if another instance of the same process is create with the same process name fragment, the PROS module will give it a full process name of "Background_Reporter_002". Again, this is a tremendous savings in code to the 4D developer as multiple processes of the same type can be created with very little effort. And, the advantages of unique process names is then realised throughout the application.

PROS_Beginning_Process

PROS_Beginning_Process *(Process Name ; Window Name ; Close Box Method Name)*

PROS_Beginning_Process
(
-> *Process Name :* String[32]
-> *Window Name :* String[80]
-> *Close Box Method Name :* String[32]
)

	Parameter	Type	Description
	Process Name	String[32]	Name of current process
	Window Name	String[80]	Window name for current process
	Close Box Method Name	String[32]	Name of method for close boxes for current process

The method ***PROS_Beginning_Process*** will register a new process which was not created using the PROS module. This registration of the current process with the PROS module will basically be the addition of the new process to the PROS Stack and the initialization of all process level data structures used by BASh (and possibly other 4D components).

It is important that this method be called **very** early in the handler method for the new process. Ideally, the very first line of code in the new process should be a call to this method.

Note: if the current process was launched with the PROS module, for instance by using the method ***PROS_New_Process***, then this method should not be used in the new process.

Process Name is the full name of the new process. If the current process is a local process, *Process Name* should include the dollar sign "$" at the beginning of it's name.

Window Name is the name of the primary window to be used within the new process.

Note: full implementation of this parameter and it's value is not available in the current version of BASh. But, it is worth using this value now as it will become more useful with future version of the BASh component.

Close Box Method Name is the name of the method which should be called whenever a close box in a window within the process is clicked.

Note: full implementation of this parameter and it's value is not available in the current version of BASh. But, it is worth using this value now as it will become more useful with future version of the BASh component.

Note: the method ***PROS_Beginning_Process*** was added in BASh v1.6.0

PROS_Delay_Current

PROS_Delay_Current *(Delay Tick Count)*

PROS_Delay_Current
(
-> *Delay Tick Count :* Longint
)

	Parameter	Type	Description
	Delay Tick Count	Longint	Ticks to delay current process

The method ***PROS_Delay_Current*** will delay the current process for the specified number of ticks.

4D kernel processes, which can not be directly delayed, are held synchronously within this routine until the timeout has been reached. During this time, the process state will not read as delayed.

Delay Tick Count is the number of ticks to delay the current process.

Note: the method ***PROS_Delay_Current*** was added in BASh v1.5.4.

PROS_Ending_Process

PROS_Ending_Process

PROS_Ending_Process *(*

	Parameter	Type	Description
	(none)	n/a	n/a

The method ***PROS_Ending_Process*** removes registration of a dying process which was not created using the PROS module. This registration removal of the current process with the PROS module will basically be the removal of the process from the PROS Stack and the orderly clean up and removal of all process level data structures used by BASh (and possibly other 4D components).

It is important that this method be called very late in the handler method for the process. Ideally, the very last line of code in the process should be a call to this method.

Note: if the current process was launched with the PROS module, for instance by using the method ***PROS_New_Process***, then this method should not be used in the process.

Note: the method ***PROS_Ending_Process*** was added in BASh v1.6.0

PROS_ERROR

PROS_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***PROS_ERROR*** acts as a callback method from within the PROS module for errors that may occur. Any time

an error condition is detected within the PROS module, a call to the method ***PROS_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 PROS method which called the ***PROS_ERROR*** method.

The ***PROS_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.

PROS_Get_CloseBoxMethod_s

PROS_Get_CloseBoxMethod_s *(Process ID)* => *Close Box Method*

PROS_Get_CloseBoxMethod_s

(

-> *Process ID :* Longint

)

=> *Close Box Method :* String[32]

	Parameter	Type	Description
	Process ID	Longint	Process ID of process to retrieve information about from Process Stack
	Close Box Method	String[32]	Name of method for close boxes for *Process ID*

The method ***PROS_Get_CloseBoxMethod_s*** will return the name of the close box method associated with a particular process ID as stored in the PROS Stack.

Process ID is the process ID to look up in the PROS stack.

Close Box Method is the name of the close box method associated with *Process ID*. If *Process ID* is not found in the PROS Stack, *Close Box Method* will be set to NULL (empty).

Note: the method ***PROS_Get_CloseBoxMethod_s*** was added in BASh v1.6.0

PROS_Get_ControlMethod_s

PROS_Get_ControlMethod_s *(Process ID)* => *Control Method*

PROS_Get_ControlMethod_s
(
-> *Process ID :* Longint
)
=> *Control Method :* String[32]

	Parameter	Type	Description
	Process ID	Longint	Process ID of process to retrieve information about from Process Stack
	Control Method	String[32]	Name of method for controlling process for *Process ID*

The method ***PROS_Get_ControlMethod_s*** will return the name of the main control method associated with a particular process ID as stored in the PROS Stack.

Process ID is the process ID to look up in the PROS stack.

Control Method is the name of the control method associated with *Process ID*. If *Process ID* is not found in the PROS Stack, *Control Method* will be set to NULL (empty).

Note: the method ***PROS_Get_ControlMethod_s*** was added in BASh v1.6.0

PROS_Get_ProcessID_s

PROS_Get_ProcessID_s *(Process Name)* => *Process ID*

PROS_Get_ProcessID_s
(

-> *Process Name :* String[32]
)
=> *Process ID :* Longint

	Parameter	Type	Description
	Process Name	String[32]	Name of process to lookup
	Process ID	Longint	Process ID found matching *Process Name*

The method ***PROS_Get_ProcessID_s*** will return the process ID associated with a particular process name as stored in the PROS Stack.

Process Name is the full name of the process to look up in the PROS stack.

Process ID is the ID of the process associated with *Process Name*. If *Process Name* is not found in the PROS Stack, *Process ID* will be set to NULL (zero, 0).

Note: the method ***PROS_Get_ProcessID_s*** was added in BASh v1.6.0

PROS_Get_ProcessIDs_s

PROS_Get_ProcessIDs_s *(Process Name Fragment ; Referenced Process IDs)* =>
Found Count

PROS_Get_ProcessIDs_s
(
-> *Process Name Fragment :* String[32]
-> *Referenced Process IDs :* Pointer
)
=> *Found Count :* Longint

	Parameter	Type	Description
	Process Name Fragment	String[32]	Process name fragment to search for
	Referenced Process IDs	Pointer	Pointer to longint array to hold found processes IDs

	Parameter	Type	Description
	Found Count	Longint	Number of processes found matching *Process Name Fragment*

The method ***PROS_Get_ProcessIDs_s*** will return the process IDs of all processes matching a specified process name fragment from within the PROS Stack. Any process in the PROS Stack which has a name beginning with the specified process name fragment will be returned from this method.

Process Name Fragment is the name fragment to be used for scanning process names stored within the PROS Stack. Process names in the PROS Stack that begin with *Process Name Fragment* will be considered matches during this scan and will have their associated process IDs returned by this method.

Referenced Process IDs is a pointer to a longint array to contain the process IDs of all found rows within the PROS Stack which have matches for *Process Name Fragment*. After calling this method, *Referenced Process IDs* will have the same number of rows as *Found Count*.

Found Count is the number of rows within the PROS Stack which were found to have process names beginning with *Process Name Fragment*. This is also the number of rows within *Referenced Process IDs* upon returning from a call to this method.

Note: the method ***PROS_Get_ProcessIDs_s*** was added in BASh v1.6.0

PROS_Get_ProcessName_s

PROS_Get_ProcessName_s *(Process ID) => Process Name*

PROS_Get_ProcessName_s
(
-> *Process ID :* Longint
)
=> *Process Name :* String[32]

	Parameter	Type	Description
	Process ID	Longint	Process ID of process to retrieve information about from Process Stack
	Process Name	String[32]	Name process for *Process ID*

The method ***PROS_Get_ProcessName_s*** will return the full process name associated with a particular process ID as stored in the PROS Stack. Using this method will be much faster than native implementations in 4D for retrieving the same data.

Process ID is the process ID to look up in the PROS stack.

Process Name is the full name of the process associated with *Process ID*. If *Process ID* is not found in the PROS Stack, *Process Name* will be set to NULL (empty).

Note: the method ***PROS_Get_ProcessName_s*** was added in BASh v1.6.0

PROS_Get_ProcessNames_s

PROS_Get_ProcessNames_s *(Process Name Fragment ; Referenced Process Names)*
=> Found Count

PROS_Get_ProcessNames_s
(
-> Process Name Fragment : String[32]
-> Referenced Process Names : Pointer
)
=> Found Count : Longint

	Parameter	Type	Description
	Process Name Fragment	String[32]	Process name fragment to search for
	Referenced Process Names	Pointer	Pointer to string or text array to hold found processes names
	Found Count	Longint	Number of processes found matching *Process Name Fragment*

The method ***PROS_Get_ProcessNames_s*** will return the full process names of all processes matching a specified process name fragment from within the PROS Stack. Any process in the PROS Stack which has a name beginning with the specified process name fragment will be returned from this method.

Process Name Fragment is the name fragment to be used for scanning process names stored within the PROS Stack. Process names in the PROS Stack that begin with *Process Name Fragment* will be considered matches during this scan and will have their associated full process names returned by this method.

Referenced Process Names is a pointer to a string or text array to contain the full process names of all found rows within the PROS Stack which have matches for *Process Name Fragment*. After calling this method, *Referenced Process Names* will have the same number of rows as *Found Count*.

Found Count is the number of rows within the PROS Stack which were found to have process names beginning with *Process Name Fragment*. This is also the number of rows within *Referenced Process Names* upon returning from a call to this method.

Note: the method ***PROS_Get_ProcessNames_s*** was added in BASh v1.6.0

PROS_Get_State

PROS_Get_State *(Process ID) => Process State*

PROS_Get_State
(
-> *Process ID :* Longint
)
=> *Process State :* Longint

	Parameter	Type	Description
	Process ID	Longint	Process ID of process to retrieve information about

	Parameter	Type	Description
	Process State	Longint	Current process state for *Process ID*

The method ***PROS_Get_State*** will return the process state of a specified process ID directly from 4D.

Process ID is the process ID of a process to check the state of.

Process State is the state of the process *Process ID*. Values for *Process State* correspond to the values in the 4D constants group Process state, available both in the 4D language reference and within 4D under the constants tab in the Explorer.

Note: the method ***PROS_Get_State*** was added in BASh v1.6.0.

PROS_Get_Type

PROS_Get_Type *(Process ID) => Process Type*

PROS_Get_Type
(
-> *Process ID :* Longint
)
=> *Process Type :* Longint

	Parameter	Type	Description
	Process ID	Longint	4D process ID to get the the type of
	Process Type	Longint	Process type of *Process ID*

The method ***PROS_Get_Type*** will return the process type of the specified 4D process ID. Process types are also known as the process origin.

Process ID is the 4D process ID for the process to get the type of. *Process ID* must always be with respect to the current, local machine; this is not the actual process ID of remote processes on other machines.

Process Type is the process type code of the specified process *Process ID*. Valid values for *Process Type* are as follows:

Value	Meaning
-11	Contextual Web Process
-10	Other 4D Process
-9	External Task
-8	Event Manager
-7	Apple Event Manager
-6	Serial Port Manager
-5	Indexing Process
-4	Cache Manager
-3	Noncontextual Web Process
-2	Design Process
-1	User or Custom Menus Process
0	None
1	Created from Programming
2	Created from Menu Command
3	Created from User Mode
4	Other User Process

Note: the method ***PROS_Get_Type*** was added in BASh v1.5.4.

PROS_Get_UniqueID

PROS_Get_UniqueID (*Process ID*) => *Process Unique ID*

PROS_Get_UniqueID
(
-> *Process ID :* Longint
)

=> *Process Unique ID :* Longint

	Parameter	Type	Description
	Process ID	Longint	4D process ID to get the unique ID of
	Process Unique ID	Longint	Unique process ID of *Process ID*

The method ***PROS_Get_UniqueID*** will return the 4D unique process ID for a specified 4D process ID. Using this method will be much faster than native implementations in 4D for retrieving the same data.

Process ID is the 4D process ID for the process to get the type of. *Process ID* must always be with respect to the current, local machine; this is not the actual process ID of remote processes on other machines.

Process Unique ID is the 4D unique process ID of the specified 4D process *Process ID*.

Note: due to a bug in 4D v6.7.0, this command does not return the correct value when used with this version of 4D. This bug appears to be fixed as of 4D v6.7.1.

Note: the method ***PROS_Get_UniqueID*** was added in BASh v1.5.4.

PROS_Get_UniqueID_s

PROS_Get_UniqueID_s *(Process ID) => Process Unique ID*

PROS_Get_UniqueID_s
(
-> *Process ID :* Longint
)
=> *Process Unique ID :* Longint

	Parameter	Type	Description
	Process ID	Longint	4D process ID to get the unique ID of
	Process Unique ID	Longint	Unique process ID of *Process ID*

The method ***PROS_Get_UniqueID_s*** will return the unique process ID associated with a particular process ID as stored in the PROS Stack.

Process ID is the process ID to look up in the PROS stack.

Process Unqiue ID is the unqiue ID of the process associated with *Process ID*. If *Process ID* is not found in the PROS Stack, *Process Unqiue ID* will be set to NULL (zero, 0).

Note: the method ***PROS_Get_UniqueID_s*** was added in BASh v1.6.0

PROS_Get_Visible

PROS_Get_Visible *(Process ID) => Process Visible Flag*

PROS_Get_Visible
(
-> *Process ID :* Longint
)
=> *Process Visible Flag :* Boolean

	Parameter	Type	Description
	Process ID	Longint	Process ID of process to retrieve information about
	Process Visible Flag	Boolean	Flag for visible state of *Process ID*

The method ***PROS_Get_Visible*** will a boolean value for the visible state of a specified process ID directly from 4D.

Process ID is the process ID of a process to check the visibility of.

Process Visible Flag is the current state of visibility of the process *Process ID*. If *Process ID* is current visible, *Process Visible Flag* will be set to **True**. Otherwise, *Process Visible Flag* will be set to **False**.

Note: the method ***PROS_Get_Visible*** was added in BASh v1.6.0.

PROS_Get_WindowName_s

PROS_Get_WindowName_s *(Process ID) => Window Name*

PROS_Get_WindowName_s
(
-> *Process ID :* Longint
)
=> *PROS_Get_WindowName_s :* String[80]

	Parameter	Type	Description
	Process ID	Longint	Process ID of process to retrieve information about from Process Stack
	Window Name	String[80]	Name of window for *Process ID*

The method ***PROS_Get_WindowName_s*** will return the window name associated with a particular process ID as stored in the PROS Stack.

Process ID is the process ID to look up in the PROS stack.

Window Name is the name of the window associated with *Process ID*. If *Process ID* is not found in the PROS Stack, *Window Name* will be set to NULL (empty).

Note: the method ***PROS_Get_WindowName_s*** was added in BASh v1.6.0

PROS_New_Process

PROS_New_Process *(Process Name Fragment ; Process KB ; Control Method Name ; Window Name ; Close Box Method Name) => Process ID*

PROS_New_Process
(
-> *Process Name Fragment :* String[32]
-> *Process KB :* Longint
-> *Control Method Name :* String[32]

-> *Window Name :* String[80]
-> *Close Box Method Name :* String[32]
)
=> *Process ID :* Longint

	Parameter	Type	Description
	Process Name Fragment	String[32]	Fragment of name for new process to be created
	Process KB	Longint	Number of kilobytes to assign to new process
	Control Method Name	String[32]	Name of method for controlling new process
	Window Name	String[80]	Name of window to be used in new process
	Close Box Method Name	String[32]	Name of method to be used for close boxes in new process
	Process ID	Longint	ID of new process

The method ***PROS_New_Process*** will either launch a new process with the given parameters or bring to the front an existing process with the same full name. The nature of the process name fragment supplied and checking whether it exists already will determine whether a new process will be launched or not.

When a new process is launched, the parameters provided to this method will be used to launch the new process. The new process will be automatically added to the PROS Stack, as well. Within the new process, all process level initialization needed for BASh will be automatically done. As well, the method ***INT_PROS_Error_Callback*** will be installed automatically as the first OEC handler within the INT module (see the documentation for the INT module, above, for more information about the OEC Stack and OEC interruption handlers). All clean up will be automatically handled for process level BASh data structures within the new process, as well, just before the process ends.

Process Name Fragment is fragment of a process name to used for the new process to be created or brought to the front. If *Process Name Fragment* is determined to be a full process name, then it will be checked for first in the PROS Stack; if the full process name *Process Name Fragment* is found within the PROS Stack, no new process will be created

and the existing process matching *Process Name Fragment* will instead be brought to the front. If the full process name *Process Name Fragment* is not found to already exist in the PROS Stack, then a new process will be created with the provided parameters. If *Process Name Fragment* is determined to be a process name fragment and not a full process name, then the PROS Stack is scanned to make certain that the full process name used for the new process to be created is unique. See the **Process Control Methods** section above for more details about process name fragments.

Process KB is the stack size in kilobytes to assign to the new process. It is recommended that a value of at least 64 be passed for the stack size, with a value of 128 being more than enough for practically any circumstances.

Control Method Name is the name of the method to install as the control method for the new process to be created. See the **Process Control Methods** section above for more details about process control methods.

Window Name is the full name of the main window within the new process to be created. Passing a NULL (empty) value for this parameter is allowed.

Close Box Method Name is the full name of the method to be called whenever a close box within a window is clicked within this process. Passing a NULL (empty) value for this parameter is allowed.

Note: the full implementation of the functionality associated with this parameter is not yet available within BASh

Note: the method ***PROS_New_Process*** was added in BASh v1.6.0

PROS_Set_CloseBoxMethod_s

PROS_Set_CloseBoxMethod_s *(Process ID ; Close Box Method)*

PROS_Set_CloseBoxMethod_s
(
-> *Process ID :* Longint

```
        -> Close Box Method : String[32]
)
```

	Parameter	Type	Description
	Process ID	Longint	ID of process to set close box method for within Process Stack
	Close Box Method	String[32]	New close box method to store within Process Stack for *Process ID*

The method ***PROS_Set_CloseBoxMethod_s*** will set the associated close box method for a particular process within the PROS Stack. This allows for the developer to change the name of a close box method procedurally for a given process managed by the PROS module.

Process ID is the process ID of the process within the PROS Stack to change the close box method of.

Close Box Method is the close box method name to set as assiciated to *Process ID* within the PROS module. Any previously associated close box method for *Process ID* will be discarded.

Note: the method ***PROS_Set_CloseBoxMethod_s*** was added in BASh v1.6.1.

PROS_Set_WindowName_s

PROS_Set_WindowName_s *(Process ID ; Window Name)*

PROS_Set_WindowName_s

```
(
        -> Process ID : Longint
        -> Window Name : String[80]
)
```

	Parameter	Type	Description
	Process ID	Longint	ID of process to set window name for within Process Stack

	Parameter	Type	Description
	Window Name	String[80]	New window name to store within Process Stack for *Process ID*

The method ***PROS_Set_WindowName_s*** will set the associated window name value for a particular process within the PROS Stack. This allows for the developer to change the name of a window procedurally for a given process managed by the PROS module.

Process ID is the process ID of the process within the PROS Stack to change the window name of.

Window Name is the full window name to set as assiciated to *Process ID* within the PROS module. Any previously associated window name for *Process ID* will be discarded.

Note: the method ***PROS_Set_WindowName_s*** was added in BASh v1.6.0

PROS_Wait_End_Complete

PROS_Wait_End_Complete *(Process ID)*

PROS_Wait_End_Complete

```
(
        -> Process ID : Longint
)
```

	Parameter	Type	Description
	Process ID	Longint	ID of 4D process to wait to end

The method ***PROS_Wait_End_Complete*** will wait for a specified process indicated by 4D process ID to end completely before returning. This method can be used with operations that require a separate process controlling functionality to complete before continuing.

Process ID is the 4D process ID to wait to complete. If the specified process does not exist, this method returns immediately.

Note: the method ***PROS_Wait_End_Complete*** was added in BASh v1.8.0.

PTEXT Module

The PTEXT module provides simple parameter text replacement routines for use within 4th Dimension. With the use of parameter text values it is a simple matter to create general replacement and formatted routines for common values used in 4D based applications.

Note: the PTEXT module was added in BASh v1.5.7.

Parameter Text Values

Parameter text (PText) values are simple text or binary values which have particular series of byte values embedded within them for use as placeholders. These placeholders are meant to be replaced with parameters which using the text or binary values within an application.

Placeholders within a PText value have the following format:

```
^nn
```

where nn is a number formatted to two digits. The following string is a valid, raw PText value containing two different placeholders:

```
The ^01 jumped over the ^02.
```

The following is a second example of a valid, raw PText value, again containing two different placeholders:

```
<A HREF="mailto:^01">^02, ^01</A>
```

If the second, raw PText value, above, were set into the variable $xRawPText, then the following function call:

```
PTEXT_Set_Values_x($xRawPText;"foo@goo.com";"me")
```

would return the value:

```
<A HREF="mailto:foo@goo.com">me, foo@goo.com</A>
```

Since placeholders contain numbers, the maximum number of unique placeholders allowed within a raw PText value is 99 (from ^01 to ^99).

PTEXT_Set_Values_ax

PTEXT_Set_Values_ax *(Raw PText ; Referenced Values)* => *Populated PText*

PTEXT_Set_Values_ax
(
-> *Raw PText* : Text
-> *Referenced Values* : Pointer
)
=> *Populated PText* : Text

	Parameter	Type	Description
	Raw PText	Text	Text value containing raw PText
	Referenced Values	Pointer	Referenced text array containing values to substitute into *Raw PText*
	Populated PText	Text	Text value with substitutions listed in *Referenced Values* placed into *Raw PText*

The method ***PTEXT_Set_Values_ax*** will populate a specified raw PText with the indexed entries in a referenced text array and return the resulting text value.

Raw PText is the raw PText value in textual format which contains placeholders to be replaced.

Referenced Values is a pointer to a text array containing indexed entries to use as substitutes into *Raw PText* value.

Populated PText is the resulting *Raw PText* value with all of the substitutions values from *Referenced Values* placed.

Note: the method ***PTEXT_Set_Values_ax*** was added in BASh v1.5.7.

PTEXT_Set_Values_az

PTEXT_Set_Values_az *(Referenced PText ; Referenced Values)*

PTEXT_Set_Values_az
(

-> *Referenced PText :* Pointer
-> *Referenced Values :* Pointer
)

	Parameter	Type	Description
	Referenced PText	Pointer	Pointer to BLOB containing PText value to perform substitution upon
	Referenced Values	Pointer	Referenced text array containing values to substitute into *Raw PText*

The method ***PTEXT_Set_Values_az*** will populate a referenced PText value within a BLOB with the indexed entries in a referenced text array.

Referenced PText is a pointer to a BLOB containing the PText value which is to have substitutions performed upon.

Referenced Values is a pointer to a text array containing indexed entries to use as substitutes into *Referenced PText* value.

Note: the method ***PTEXT_Set_Values_az*** was added in BASh v1.5.7.

PTEXT_Set_Values_x

PTEXT_Set_Values_x *(Raw PText ; First Value { ; Second Value { ; ... } })* => *Populated PText*

PTEXT_Set_Values_x
(
-> *Raw PText :* Text
-> *First Value :* Text
[-> *Second Value :* Text }
)
=> *Populated PText :* Text

	Parameter	Type	Description
	Raw PText	Text	Text value containing raw PText

	Parameter	Type	Description
	First Value	Text	Value to use as replacement into *Raw PText*
	Populated PText	Text	Text value with substitutions listed in *Referenced Values* placed into *Raw PText*

The method ***PTEXT_Set_Values_x*** will populate a specified raw PText with one or more specified replacement values.

Raw PText is the raw PText value in textual format which contains placeholders to be replaced.

First Value is a text value to use as a replacement value into *Raw PText* value. One of more values may be passed to this method, as needed by the particular PText value.

Populated PText is the resulting *Raw PText* value with all of the substitutions values placed.

Note: the method ***PTEXT_Set_Values_x*** was added in BASh v1.5.7.

PTEXT_Set_Values_z

PTEXT_Set_Values_z *(Referenced PText ; First Value { ; Second Value { ; ... } }) => Populated PText*

PTEXT_Set_Values_z
(
-> *Referenced PText :* Pointer
-> *First Value :* Text
{ -> *First Value :* Text }
)

	Parameter	Type	Description
	Referenced PText	Pointer	Pointer to BLOB containing PText value to perform substitution upon
	First Value	Text	Value to use as replacement into *Raw PText*

The method ***PTEXT_Set_Values_z*** will populate a referenced PText value within a BLOB with one or more specified replacement values.

Referenced PText is a pointer to a BLOB containing the PText value which is to have substitutions performed upon.

First Value is a text value to use as a replacement value into *Referenced PText* value. One of more values may be passed to this method, as needed by the particular PText value.

Note: the method ***PTEXT_Set_Values_z*** was added in BASh v1.5.7.

QUIT Module

The QUIT module is used to de-initialize the BASh component before quitting 4D. At this time, it consists of only a single routine. But, this single routine handles shutting down the BASh component.

It is **strongly** recommended that all initialization be done late in the **On Exit** database method. This method should also be called in the **On Server Shutdown** database method.

And, it is also strongly recommended to not consider any of the methods in the QUIT module as reentrant during any single use of the application without it being clearly stated in this documentation.

Note: the QUIT module was added in BASh v1.6.2.

QUIT_BASh

QUIT_BASh *(*

QUIT_BASh *(*

	Parameter	Type	Description
	(none)	n/a	n/a

The method ***QUIT_BASh*** will de-initialize the whole BASh component.

After ***QUIT_BASh*** is called, no other methods in the BASh component should be used. It is best to call ***QUIT_BASh*** late in the **On Exit** and **On Server Shutdown** database methods.

There are no parameters or returned values to ***QUIT_BASh***.

Note: the method QUIT_BASh was added in BASh v1.6.2.

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 expend significantly.

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

'bYt#' Resource

The 'bYt#' resource type is a custom resource type developed for inclusion in the BASh component. This resource type provides storage of a counted list of single byte values. This custom resource type is ideal for use in conversion tables and byte values lists that require a structured storage location and format.

The template resource, of type 'TMPL', for the 'bYt#' resource is in the Affix BASh plugin document. Opening the Affix document with a resource editor will reveal the resource definition for the 'bYt#' resource. For a complete listing of the resource field types in the 'bYt#' resource, please refer to the 'TMPL' resource in the Affix BASh plugin document.

The method **RES_Load_bYt_List** provides direct access to load and parse resources of type 'bYt#'.

Note: support for the 'bYt#' resource type was added in BASh v1.8.2.

'fMap' Resource

The 'fMap' resource type is a custom resource type developed for inclusion in the BASh component. This resource type provides mapping information between Macintosh and Windows document types and standard MIME types. Such mapping of document types provides a means for sharing common document types between platforms and online. This is especially true when serving documents over the web.

The template resource, of type 'TMPL', for the 'fMap' resource is in the Affix BASh plugin document. Opening the Affix document with a resource

editor will reveal the resource definition for the 'fMap' resource. This resource structure definition contains the following resource fields:

Resource Field Type	Resource Field Type Description	Field Title
PSTR	Pascal String	Entitled
TNAM	Type Name	Macintosh Type Code
P006	Pascal String, max. 5 bytes	Windows Type Code
TNAM	Type Name	Macintosh Creator Code
PSTR	Pascal String	MIME Type
LBIT [31]	Long, Bit 31	Active on Macintosh
LBIT [30]	Long, Bit 30	Active on Windows
LBIT [29]	Long, Bit 29	Default for Type
LBIT [28-0]	Long, Bits 28 thru 0	(reserved)

The method **RES_Load_fMap** provides direct access to load and parse resources of type 'fMap'. Routines in the FMAP module of BASh provide functionality for resources of this type.

'HTbl' Resource

The 'HTbl' resource type is a custom resource type developed for inclusion in the BASh component. This resource type provides storage of all of the common HTML table parameters. It also includes storage for repeated row and column formatting options commonly used when constructing HTML tables.

The template resource, of type 'TMPL', for the 'HTbl' resource is in the Affix BASh plugin document. Opening the Affix document with a resource editor will reveal the resource definition for the 'fMap' resource. For a complete listing of the resource field types in the 'HTbl' resource, please refer to the 'TMPL' resource in the Affix BASh plugin document.

The method **RES_Load_HTbl** provides direct access to load and parse resources of type 'HTbl'.

Note: routines for utilizing the 'HTbl' format for storage and manipulation of HTML table descriptors will be made available in a future component from Deep Sky Technologies, Inc.

'iXrf' Resource

The 'iXrf' resource type is a custom resource type developed for inclusion in the BASh component. This resource type provides a simple means to store and retrieve parallel arrays of longint values for use throughout your 4D based applications. When loaded, a 'iXrf' resource is just a pair of longint arrays of the values stored in the 'iXrf' resource. This type of structure is commonly used to provide relational functionality between objects identified unique, longint values.

The template resource, of type 'TMPL', for the 'iXrf' resource is in the Affix BASh plugin document. Opening the Affix document with a resource editor will reveal the resource definition for the 'iXrf' resource. This resource structure definition contains the following resource fields:

Resource Field Type	Resource Field Type Description	Field Title
LCNT	Counted List Count	Row Count
LSTC	Counted List Begin	n/a
DLNG	Double Long	Column 1 Value
DLNG	Double Long	Column 2 Value
LSTE	Counted List End	n/a

The method **RES_Load_iXrf** provides direct access to load and parse resources of type 'iXrf'.

'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.

'rSr#' Resource

The 'rSr#' resource type is a custom resource type developed for inclusion in the BASh component. This resource type provides storage of a counted list of resource references. Each entry in a resource reference contains the resource type and resource ID of the referenced resource. This custom resource type is ideal for providing linkage data between multiple resources that are designed and coded to work in conjunction with each other.

The template resource, of type 'TMPL', for the 'rSr#' resource is in the Affix BASh plugin document. Opening the Affix document with a resource editor will reveal the resource definition for the 'rSr#' resource. For a complete listing of the resource field types in the 'rSr#' resource, please refer to the 'TMPL' resource in the Affix BASh plugin document.

The method **RES_Load_rSr_List** provides direct access to load and parse resources of type 'rSr#'. The method **RES_Load_rSr_List_Data** provides a mechanism to follow referenced resources and load their data with a single, convenient method call.

Note: support for the 'rSr#' resource type was added in BASh v1.8.2.

'TXT#' Resource

The 'TXT#' resource type is a custom resource type developed for inclusion in the BASh component. This resource type provides a simple means to store and retrieve multiple text values for use throughout your 4D based applications. When loaded, a 'TXT#' resource is just a text array of the values stored in the 'TXT#' resource.

The template resource, of type 'TMPL', for the 'TXT#' resource is in the Affix BASh plugin document. Opening the Affix document with a resource editor will reveal the resource definition for the 'TXT#' resource. This resource structure definition contains the following resource fields:

Resource Field Type	Resource Field Type Description	Field Title
LCNT	Counted List Count	Row Count
LSTC	Counted List Begin	n/a
LSTR	Text	Indexed Text
LSTE	Counted List End	n/a

The method **RES_Load_TXT_List** provides direct access to load and parse resources of type 'TXT#'.

'WAGr' Resource

Documentation for the 'WAgr' resource will be provided in a future release of the BASh component.

'WPGr' Resource

Documentation for the 'WPgr' resource will be provided in a future release of the BASh component.

RES_Close

RES_Close *(Resource Fork File Reference)*

RES_Close
(
-> *Resource Fork File Reference :* Time
)

	Parameter	Type	Description
	Resource Fork File Reference	Time	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) => Resource Fork File Reference*

RES_Create_File
(
-> *Full Path :* Text
-> *File Type :* Longint
)
=> *Resource Fork File Reference :* Time

	Parameter	Type	Description
	Full Path	Text	Full path to new resource document to create
	File Type	Longint	File type of new resource document
	Resource Fork File Reference	Time	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 defaul 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
	Resource Fork File Reference	Time	Resource fork file reference to delete from
	Resource Type	Longint	Resource type to delete
	Resource ID	Longint	Resource ID to delete
	Success Indicator	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
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***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 RES method which called 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_Fork_Size

RES_Get_Fork_Size *(Document Full Path)* => *Resource Fork Bytes*

RES_Get_Fork_Size
(
-> *Document Full Path :* Text
)
=> *Resource Fork Bytes :* Longint

	Parameter	Type	Description
	Document Full Path	Text	Full path to document
	Resource Fork Bytes	Longint	Number of bytes in the resource fork of specified document

The method ***RES_Get_Fork_Size*** returns the number of bytes in the resource fork of a specified document. On Windows, it retrieves the size of the resource document, if any, equivalent to the resource fork of a single document on Macintosh.

Document Full Path is the full path to the document to get the resource fork size of.

Resource Fork Bytes is the number of bytes in the resource fork of the document *Document Full Path*. If the document does not exist, *Resource Fork Bytes* will be set to negative one (-1). If the document specified in *Document Full Path* is a resource document and the current OS is Windows, *Resource Fork Bytes* will be set to negative two (-2).

Note: the method ***RES_Get_Fork_Size*** was added in BASh v1.5.6.

RES_Get_Free_ResourceID

RES_Get_Free_ResourceID *(Resource Document Reference ; Resource Type ; Starting Resource ID) => Next Free Resource ID*

RES_Get_Free_ResourceID
(
-> *Resource Document Reference :* Time
-> *Resource Type :* Longint
-> *Starting Resource ID :* Longint
)
=> *Next Free Resource ID :* Longint

	Parameter	Type	Description
	Resource Document Reference	Time	Reference document reference
	Resource Type	Longint	Resource type
	Starting Resource ID	Longint	Starting resource ID to begin scanning
	Next Free Resource ID	Longint	Next available resource ID

The method ***RES_Get_Free_ResourceID*** returns the next free resource ID for the specified resource type above the specified starting point.

Resource Document Reference is the reference to the resource document returned by **Open Resource File**.

Resource Type is the resource type to scan for the next available resource ID.

Starting Resource ID is the ID to start scanning for the next available ID. Set *Starting Resource ID* to MAXLONG to use any available resource ID.

Next Free Resource ID is the next available resource ID above *Starting Resource ID* for *Resource Type*. If no resource IDs are available, *Next Free Resource ID* is set to MAXLONG.

Note: the method ***RES_Get_Free_ResourceID*** was added in BASh v1.7.0.

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
)

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

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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Resource Text Value	Text	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_4DK_List

RES_Load_4DK_List *(Resource Fork File Reference ; Resource ID ; Referenced Constants Names ; Referenced Constants Values ; Referenced Constants Types) => Success Indicator*

RES_Load_4DK_List
(
-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced Constants Names :* Pointer
-> *Referenced Constants Values :* Pointer
-> *Referenced Constants Types :* Pointer
)
=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference of resource fork to load from
	Resource ID	Longint	Resource ID of '4DK#' resource to load
	Referenced Constants Names	Pointer	Pointer to text array to contain names of constants loaded
	Referenced Constants Values	Pointer	Pointer to text array to contain values of constants loaded
	Referenced Constants Types	Pointer	Pointer to longint array to contains types of constants loaded
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_4DK_List*** will load and parse a specified '4DK#' resource. This method is ideal for presenting choice lists, popups, and other interface widgets for selections of values related directly to constants.

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 Constants Names is a pointer to a text array to load the constants names into. Passing a NULL pointer for this parameter will skip the loading of the constants names.

Referenced Constants Values is a pointer to a text array to load the constants values into. Passing a NULL pointer for this parameter will skip the loading of the constants values.

Referenced Constants Types is a pointer to a longint array to load the constants types into. Passing a NULL pointer for this parameter will skip the loading of the constants types. The type values are set according to the constants available in the native 4D **Field and Variable Types** constants group.

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 successully loaded.

Note: the method ***RES_Load_4DK_List*** was added in BASh v1.8.0.

RES_Load_bYt_List

RES_Load_bYt_List *(Resource Fork File Reference ; Resource ID ; Referenced Byte List)*
=> Success Indicator

RES_Load_bYt_List
(
-> Resource Fork File Reference : Time
-> Resource ID : Longint
-> Referenced Byte List : Pointer
)
=> Success Indicator : Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference of resource fork to load from
	Resource ID	Longint	Resource ID of 'bYt#' resource to load
	Referenced Byte List	Pointer	Pointer to BLOB, longint array, or integer array to contain the loaded byte values
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_bYt_List*** will load and parse a specified 'bYt#' resource. 'bYt#' resources are best used for storing variable sized lists of byte values used for conversions, encoding and decoding, and projection routines.

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 Byte List is a pointer to a BLOB, longint array, or integer array to load the byte values into. If referencing a BLOB, the byte values loaded will be the complete contents of the referenced BLOB; the number of bytes in the BLOB will be the same as the number of entries in the resource list. If referencing an array type, the byte values loaded will the inserted one value per array element; the number of elements in the array will be the same as the number of entries in the resource list.

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 successully loaded.

Note: the method ***RES_Load_bYt_List*** was added in BASh v1.8.2.

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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Picture	Pointer	Pointer to picture variable to load resource into
	Success Indicator	Longint	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 successully 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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Title	Pointer	Pointer to text variable to hold specified title
	Referenced Macintosh Types	Pointer	Pointer to text variable to hold Macintosh document type code
	Referenced Windows Types	Pointer	Pointer to text variable to hold Windows document extension
	Referenced Macintosh Creators	Pointer	Pointer to text variable to hold Macintosh application creator code
	Referenced MIME Types	Pointer	Pointer to text variable to hold MIME type definition
	Referenced Parm Bits	Pointer	Pointer to longint variable to hold parameter flags
	Success Indicator	Longint	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.

Referenced Title is a pointer to a text variable to hold the title of the specified 'fMap' resource being read.

Referenced Macintosh Type is a pointer to a text variable to hold the Macintosh document type code (usually four bytes) read from the specified 'fMap' resource..

Referenced Windows Type is a pointer to a text variable to hold the Windows document type extension (usually three bytes) read from the specified 'fMap' resource.

Referenced Macintosh Creator is a pointer to a text variable to hold the Macintosh application creator code (usually four bytes) read from the specified 'fMap' resource.

Referenced MIME Type is a pointer to a text variable to hold the MIME type read from the specified 'fMap' resource.

Referenced Parm Bits is a pointer to a longint variable to hold the parameter bits (flags) read from the specified 'fMap' resource. Bit 31 is always the flag for active on Macintosh. Bit 30 is always the flag for active on Windows. Bit 29 is always the flag for a default 'fMap' entry.

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 successully loaded.

Note: the method ***RES_Load_fMap*** was added in BASh v1.5.1.

RES_Load_HTbl

RES_Load_HTbl *(Resource File Reference ; Resource ID ; Referenced Resource)* => *Success*

RES_Load_HTbl

(

-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced Resource :* Pointer

)

=> *Success :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with

	Parameter	Type	Description
	Resource ID	Longint	Resource ID to work with
	Referenced Resource	Pointer	Referenced BLOB to load 'HTbl' resource into
	Success	Longint	qi for loaded successfully

The method ***RES_Load_HTbl*** retrieves the contents of a 'HTbl' 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 Resource is a pointer to a BLOB to contain the resource retrieved.

Success is an indicator value for whether the loading was successful. If *Success* is zero (0) then the resource was not loaded; if *Success* is one (1) then the resource was successfully loaded.

Note: the method ***RES_Load_HTbl*** was added in BASh v1.5.5.

RES_Load_iXrf

RES_Load_iXrf *(Resource File Reference ; Resource ID ; Referenced Ones Column ; Referenced Many Column) => Success Indicator*

RES_Load_iXrf
(
-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced Ones Column :* Pointer
-> *Referenced Many Column :* Pointer
)
=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Ones Column	Pointer	Pointer to longint array to load ones column into
	Referenced Many Column	Pointer	Pointer to longint array to load many column into
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_iXrf*** loads and parses the contents of a 'iXrf' 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 Ones Column is a pointer to a longint array to load the ones column from the specified resource into.

Referenced Many Column is a pointer to a longint array to load the many column from the specified resource 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 successully loaded.

Note: the method ***RES_Load_iXrf*** was added in BASh v1.8.0.

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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Lock Pin Names	Pointer	not available at this time
	Referenced Low Flags	Pointer	not available at this time
	Referenced High Flags	Pointer	not available at this time
	Success Indicator	Longint	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 documention 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 successully 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
```

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced MENU Resource IDs	Pointer	not available at this time
	Success Indicator	Longint	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 successully 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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Menu Item Titles	Pointer	Pointer to array to hold menu item titles
	Referenced Menu Item States	Pointer	Pointer to array to hold menu item states
	Referenced Menu Item Styles	Pointer	Pointer to array to hold menu item styles
	Referenced Menu Item Keys	Pointer	Pointer to array to hold menu items keys
	Referenced Menu Item Marks	Pointer	Pointer to array to hold menu items marks
	Success Indicator	Longint	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 successully 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_MENV

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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Menu Item Names	Pointer	not available at this time
	Referenced Method Names	Pointer	not available at this time
	Success Indicator	Longint	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 documention 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 successully 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
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Picture	Pointer	Pointer to picture variable to load resource into
	Success Indicator	Longint	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 successully loaded.

Note: the method ***RES_Load_PICT*** was added in BASh v1.5.1.

RES_Load_rSr_List

RES_Load_rSr_List *(Resource Fork File Reference ; Resource ID ; Referenced Resource Types ; Referenced Resource IDs) => Success Indicator*

RES_Load_rSr_List
(
-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced Resource Types :* Pointer

-> *Referenced Resource IDs :* Pointer

)

=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference of resource fork to load from
	Resource ID	Longint	Resource ID of 'rSr#' resource to load
	Referenced Resource Types	Pointer	Pointer to longint array to contain the list of resource types loaded
	Referenced Resource IDs	Pointer	Pointer to longint array to contain the list of resource IDs loaded
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_rSr_List*** will load and parse a specified 'rSr#' resource. 'rSr#' resources are best used for storing linkage and grouping information for other resources that work together by design and coding.

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 Resource Types is a pointer to a longint array to contain the list of resource types loaded. The elements in this array will be paired, element for element, with the resource IDs loaded and returned in *Referenced Resource IDs*.

Referenced Resource IDs is a pointer to a longint array to contain the list of resource IDs loaded. The elements in this array will be paired, element for element, with the resource types loaded and returned in *Referenced Resource Types*.

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 successully loaded.

Note: the method ***RES_Load_rSr_List*** was added in BASh v1.8.2.

RES_Load_rSr_List_Data

RES_Load_rSr_List_Data *(Resource Fork File Reference ; Resource ID ; Referenced Data Construct) => Success Indicator*

RES_Load_rSr_List_Data

(

-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced Data Construct :* Pointer

)

=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference of resource fork to load from
	Resource ID	Longint	Resource ID of 'rSr#' resource to load referenced data from
	Referenced Data Construct	Pointer	Pointer to pointer array referencing variables to load referenced resources data into
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_rSr_List_Data*** will load the referenced resources from a specified 'rSr#' resource. All entries in the specified 'rSr#' resource will be loaded and sequentially placed in the specified referenced variables.

Note: currently, only referenced resources of type 'bYt#', 'STR#', and 'TXT#' are supported for loading from this routine.

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 Data Construct is a pointer to an array of pointers. The array of pointers must have the same number of elements as there are items in the specified 'rSr#' resource. Each pointer in *Referenced Data Construct* must reference a variable of an acceptable type for the referenced resource in the equivalent element in the 'rSr#' resource specified.

The following table lists the resource types supported within this routine and the matching 4D variable types that can be used to load data from each support resource type:

Resource Type	Resource Type Description	4D Variable Type
bYt#	Counted Byte List	BLOB, longint array, integer array
STR#	Counted String List	String Array, Text Array
TXT#	Counted Text List	Text Array

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 successully loaded.

Note: the method ***RES_Load_rSr_List_Data*** was added in BASh v1.8.2.

RES_Load_SEQn

RES_Load_SEQn *(Resource Fork File Reference; Resource ID; Referenced TF Code ; Referenced Next Number ; Referenced Description) => Success Indicator*

RES_Load_SEQn

(

-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced TF Code :* Pointer
-> *Referenced Next Number :* Pointer
-> *Referenced Description :* Pointer

)

=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced TF Code	Pointer	Reference to longint variable to receive Table Field Code (TFC)
	Referenced Next Number	Pointer	Reference to longint variable to receive next sequence number
	Referenced Description	Pointer	Reference to text variable to receive sequence number description
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_SEQn*** retrieves the contents of a 'SEQn' resource from a specified resource document. The SEQn resource contains sequence number information.

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 TF Code is a reference to a longint variable to receive the Table Field Code (TFC).

Referenced Next Number is a reference to a longint variable to receive the next sequence number.

Referenced Description is a reference to a text variable to receive the description of the sequence number.

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 successully loaded.

Note: the method ***RES_Load_SEQn*** was added in BASh v1.6.2.

RES_Load_SEQr

RES_Load_SEQr *(Resource Fork File Reference; Resource ID; Referenced TF Code ; Referenced Recycled Numbers) => Success Indicator*

RES_Load_SEQr

(

-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced TF Code :* Pointer
-> *Referenced Recycled Numbers :* Pointer

)

=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced TF Code	Pointer	Reference to longint variable to receive Table Field Code (TFC)
	Referenced Recycled Numbers	Pointer	Reference to longint array to receive recycled sequence numbers
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_SEQr*** retrieves the contents of a 'SEQr' resource from a specified resource document. The SEQr resource contains sequence number information.

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 TF Code is a reference to a longint variable to receive the Table Field Code (TFC).

Referenced Recycled Numbers is a reference to a longint array to receive the recycled sequence numbers.

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 successully loaded.

Note: the method ***RES_Load_SEQr*** was added in BASh v1.6.2.

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
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Template Item Titles	Pointer	Pointer to text array to hold template item titles which are loaded
	Referenced Template Field Types	Pointer	Pointer to longint array to hold template field types which are loaded
	Success Indicator	Longint	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 successully 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_TXT_List

RES_Load_TXT_List *(Resource Fork File Reference ; Resource ID ; Referenced Texts)* => *Success Indicator*

RES_Load_TMPL

(

-> *Resource Fork File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced Texts :* Pointer

)

=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Texts	Pointer	Pointer to text array to hold texts loaded from resource
	Success Indicator	Longint	qi for loaded successfully

The method ***RES_Load_TXT_List*** retrieves the contents of a 'TXT#' 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 Texts is a pointer to a text array which will hold the text values that 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 successully loaded.

Note: the method ***RES_Load_TXT_List*** was added in BASh v1.8.0.

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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Lock Code	Pointer	not available at this time
	Referenced Table Name	Pointer	not available at this time
	Referenced Action Titles	Pointer	not available at this time

	Parameter	Type	Description
	Referenced Action Values	Pointer	not available at this time
	Referenced Method Names	Pointer	not available at this time
	Referenced Link Titles	Pointer	not available at this time
	Referenced Resource IDs	Pointer	not available at this time
	Referenced Pin Indices	Pointer	not available at this time
	Referenced Page Flags	Pointer	not available at this time
	Success Indicator	Longint	qi for loaded successfully

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 documention 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 successully 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

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Global Lock Code	Pointer	not available at this time
	Referenced Global Pin Index	Pointer	not available at this time
	Referenced File Names	Pointer	not available at this time
	Referenced Resource IDs	Pointer	not available at this time
	Referenced Pin Indices	Pointer	not available at this time
	Referenced Link Titles	Pointer	not available at this time
	Referenced Method Names	Pointer	not available at this time
	Referenced Low Product Flags	Pointer	not available at this time
	Referenced High Product Flags	Pointer	not available at this time
	Success Indicator	Longint	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 documention 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 successully 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
	Referenced BLOB	Pointer	Pointer to BLOB to contain 'TMPL' resource created
	Referenced Template Item Titles	Pointer	Pointer to text array containing template item titles
	Referenced Template Field Types	Pointer	Pointer to longint array containing template field types
	Success Indicator	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 successully 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
	Resource Fork File Reference	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: as of BASh v1.6.1, this method will not function properly under Windows. Through 4D v6.7.2, there is bug in 4D on Windows in which the 4D application will crash of the resource fork of the current 4D application is opened. So, this method has been disabled when run under Windows to prevent this bug from appearing when quitting.

Note: the method ***RES_Open_4DApplication*** was added in BASh v1.5.1.

RES_Open_BASh

RES_Open_BASh *=> Resource Fork File Reference*

RES_Open_BASh *(*
=> Resource Fork File Reference : Time

	Parameter	Type	Description
	Resource Fork File Reference	Time	Resource fork file reference opened

The method ***RES_Open_BASh*** will open the resource fork of the current BASh Affix document (it should be in one of the 4DX directories) 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_BASh*** was added in BASh v1.5.3.

RES_Open_DataFile

RES_Open_DataFile *=> Resource Fork File Reference*

RES_Open_DataFile *(*
=> Resource Fork File Reference : Time

	Parameter	Type	Description
	Resource Fork File Reference	Time	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
	Resource Fork File Reference	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_HTbl_Cells

RES_Parse_HTbl_Cells *(Referenced Resource ; Beginning Offset ; Referenced Cell IDs ; Referenced Cell Column Starts ; Referenced Cell Column Ends ; Referenced Cell Nth Columns ; Referenced Cell Nth Column Offsets ; Referenced Cell Horizontal Alignments ; Referenced Cell Vertical Alignments ; Referenced Cell Column Spans ; Referenced Cell Row Spans ; Referenced Cell Widths ; Referenced Cell Heights ; Referenced Cell Background Colors ; Referenced Cell Height Percentage Indicators ; Referenced Cell NoWrap Indicators) => Offset*

RES_Parse_HTbl_Cells

(

-> *Referenced Resource :* Pointer
-> *Beginning Offset :* Longint
-> *Referenced Cell IDs :* Pointer
-> *Referenced Cell Column Starts :* Pointer
-> *Referenced Cell Column Ends :* Pointer
-> *Referenced Cell Nth Columns :* Pointer
-> *Referenced Cell Nth Column Offsets :* Pointer
-> *Referenced Cell Horizontal Alignments :* Pointer
-> *Referenced Cell Vertical Alignments :* Pointer
-> *Referenced Cell Column Spans :* Pointer
-> *Referenced Cell Row Spans :* Pointer
-> *Referenced Cell Widths :* Pointer
-> *Referenced Cell Heights :* Pointer
-> *Referenced Cell Background Colors :* Pointer
-> *Referenced Cell Height Percentage Indicators :* Pointer
-> *Referenced Cell NoWrap Indicators :* Pointer

)

=> *Offset :* Longint

	Parameter	Type	Description
	Referenced Resource	Pointer	Referenced BLOB containing 'HTbl' resource
	Beginning Offset	Longint	asd
	Referenced Cell IDs	Pointer	asd
	Referenced Cell Column Starts	Pointer	asd
	Referenced Cell Column Ends	Pointer	asd
	Referenced Cell Nth Columns	Pointer	asd
	Referenced Cell Nth Column Offsets	Pointer	asd
	Referenced Cell Horizontal Alignments	Pointer	asd
	Referenced Cell Vertical Alignments	Pointer	asd
	Referenced Cell Column Spans	Pointer	asd
	Referenced Cell Row Spans	Pointer	asd
	Referenced Cell Widths	Pointer	asd
	Referenced Cell Heights	Pointer	asd
	Referenced Cell Background Colors	Pointer	asd
	Referenced Cell Height Percentage Indicators	Pointer	asd
	Referenced Cell NoWrap Indicators	Pointer	asd
	Offset	Longint	Ending offset after parsing 'HTbl' resource

The method ***RES_Parse_HTbl_Cells*** will parse HTML table cell descriptors from a referenced 'HTbl' resource.

asd

Note: the method ***RES_Parse_HTbl_Cells*** was added in BASh v1.5.5.

RES_Parse_HTbl_Rows

RES_Parse_HTbl_Rows *(Referenced Resource ; Beginning Offset ; Referenced Row Starts ; Referenced Row Ends ; Referenced Row Nths ; Referenced Row Nths Offsets ; Referenced Row Horizontal Alignments ; Referenced Row Vertical Alignments ; Referenced Row Background Colors ; Referenced Row Heights ; Referenced Row Heights Percentage Indicators ; Referenced Row Packed Column IDs) => Offset*

RES_Parse_HTbl_Rows

(

-> *Referenced Resource :* Pointer
-> *Beginning Offset :* Longint
-> *Referenced Row Starts :* Pointer
-> *Referenced Row Ends :* Pointer
-> *Referenced Row Nths :* Pointer
-> *Referenced Row Nths Offsets :* Pointer
-> *Referenced Row Horizontal Alignments :* Pointer
-> *Referenced Row Vertical Alignments :* Pointer
-> *Referenced Row Background Colors :* Pointer
-> *Referenced Row Heights :* Pointer
-> *Referenced Row Heights Percentage Indicators :* Pointer
-> *Referenced Row Packed Column IDs :* Pointer

)

=> *Offset :* Longint

	Parameter	Type	Description
	Referenced Resource	Pointer	Referenced BLOB containing 'HTbl' resource
	Beginning Offset	Longint	asd
	Referenced Row Starts	Pointer	asd
	Referenced Row Ends	Pointer	asd
	Referenced Row Nths	Pointer	asd
	Referenced Row Nths Offsets	Pointer	asd
	Referenced Row Horizontal Alignments	Pointer	asd
	Referenced Row Vertical Alignments	Pointer	asd

	Parameter	Type	Description
	Referenced Row Background Colors	Pointer	asd
	Referenced Row Heights	Pointer	asd
	Referenced Row Heights Percentage Indicators	Pointer	asd
	Referenced Row Packed Column IDs	Pointer	asd
	Offset	Longint	Ending offset after parsing 'HTbl' resource

The method ***RES_Parse_HTbl_Rows*** will parse HTML table row descriptors from a referenced 'HTbl' resource.

asd

Note: the method ***RES_Parse_HTbl_Rows*** was added in BASh v1.5.5.

RES_Parse_HTbl_Table

RES_Parse_HTbl_Table *(Referenced Resource ; Referenced Table Border ; Referenced Table Width ; Referenced Table Height ; Referenced Table Cell Padding ; Referenced Table Cell Spacing ; Referenced Table Background Color ; Referenced Table Horizontal Alignment ; Referenced Table Height Pecentage Indicator ; Referenced Table Width Percentage Indicator)*

RES_Parse_HTbl_Table
(
-> *Referenced Resource :* Pointer
-> *Referenced Table Border :* Pointer
-> *Referenced Table Width :* Pointer
-> *Referenced Table Height :* Pointer
-> *Referenced Table Cell Padding :* Pointer
-> *Referenced Table Cell Spacing :* Pointer
-> *Referenced Table Background Color :* Pointer
-> *Referenced Table Horizontal Alignment :* Pointer
-> *Referenced Table Height Pecentage Indicator :* Pointer
-> *Referenced Table Width Percentage Indicator :* Pointer
)

	Parameter	Type	Description
	Referenced Resource	Pointer	Referenced BLOB containing 'HTbl' resource
	Referenced Table Border	Pointer	asd
	Referenced Table Width	Pointer	asd
	Referenced Table Height	Pointer	asd
	Referenced Table Cell Padding	Pointer	asd
	Referenced Table Cell Spacing	Pointer	asd
	Referenced Table Background Color	Pointer	asd
	Referenced Table Horizontal Alignment	Pointer	asd
	Referenced Table Height Pecentage Indicator	Pointer	asd
	Referenced Table Width Percentage Indicator	Pointer	asd

The method ***RES_Parse_HTbl_Table*** will parse HTML table descriptors from a referenced 'HTbl' resource.

asd

Note: the method ***RES_Parse_HTbl_Table*** was added in BASh v1.5.5.

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
	Referenced BLOB	Pointer	Pointer to BLOB containing 'TMPL' resource to parse
	Referenced Template Item Titles	Pointer	Pointer to text array to contain template item titles
	Referenced Template Field Types	Pointer	Pointer to longint array to contain template field types
	Success Indicator	Longint	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 successully parsed.

Note: the method ***RES_Parse_TMPL*** was added in BASh v1.5.1.

RES_qi_Resource_Exists

RES_qi_Resource_Exists *(Resource File Reference ; Resource Type ; Resource ID)* => *qi Resource Exists*

RES_qi_Resource_Exists
(

-> *Resource File Reference :* Time
-> *Resource Type :* Longint
-> *Resource ID :* Longint
)
=> *qi Resource Exists :* Longint

	Parameter	Type	Description
	Resource File Reference	Time	Resource fork file referenced to work with
	Resource Type	Longint	Resource type to check
	Resource ID	Longint	Resource ID to check
	qi Resource Exists	Longint	qi for whether resource exists or not

The method ***RES_qi_Resource_Exists*** returns a qi for whether the specified resource exists.

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 check.

Resource ID is the resource ID to check.

qi Resource Exists is a qi indicating whether the specified resource exists. If the specified resource exists, *qi Resource Exists* is set to one (1); if not, *qi Resource Exists* is set to zero (0).

Note: the method ***RES_qi_Resource_Exists*** was added in BASh v1.6.2.

RES_Set_HTbl_Resource

RES_Set_HTbl_Resource *(Resource File Reference ; Resource ID ; Referenced Resource) => Success*

RES_Set_HTbl_Resource
(
-> *Resource File Reference :* Time
-> *Resource ID :* Longint
-> *Referenced Resource :* Pointer

)

=> *Success :* Longint

	Parameter	Type	Description
	Resource File Reference	Time	Resource fork file referenced to work with
	Resource ID	Longint	Resource ID to work with
	Referenced Resource	Pointer	Referenced BLOB to set into indicated 'HTbl' resource
	Success	Longint	qi for set successfully

The method ***RES_Set_HTbl_Resource*** sets the contents of a 'HTbl' resource within 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 Resource is a pointer to a BLOB containing the 'HTbl' resource to write to the resource fork of the specified document.

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

Note: the method ***RES_Set_HTbl_Resource*** was added in BASh v1.5.5.

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 File Reference :* Time
-> *Resource Type :* Longint
-> *Resource ID :* Longint

-> *Resource Name :* String[255]

)

=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource 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 Name	String[255]	Value to set the resource name to
	Success Indicator	Longint	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 successully 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 File Reference :* Time
-> *Resource Type :* Longint
-> *Resource ID :* Longint
-> *Resource Properties :* Longint
)
=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource 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 successully set.

Note: the method ***RES_Set_Resource_Properties*** was added in BASh v1.5.1.

RES_Set_SEQn

RES_Set_SEQn *(Resource File Reference; Resource ID; TF Code ; Next Sequence Number ; Description) => Success Indicator*

RES_Set_SEQn

(

-> *Resource File Reference :* Time
-> *Resource ID :* Longint
-> *TF Code :* Longint
-> *Next Sequence Number :* Longint
-> *Description :* Text

)

=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource 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
	TF Code	Longint	Table Field Code (TFC)
	Next Sequence Number	Longint	Next sequence number
	Description	Text	Description for sequence number
	Success Indicator	Longint	qi for set successfully

The method ***RES_Set_SEQn*** sets the contents of a 'SEQn' resource within a specified resource document. The SEQn resource contains sequence number information.

Resource File Reference is the reference to the resource fork of the resource document to work with.

Resource ID is the resource ID to work with.

TF Code is the Table Field Code (TFC) to set in this resource.

Next Sequence Number is the next sequence number to set in this resource.

Description is the description for the sequence number to set in this 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 successully set.

Note: the method ***RES_Set_SEQn*** was added in BASh v1.6.2.

RES_Set_SEQr

RES_Set_SEQr *(Resource File Reference; Resource ID; TF Code ; Referenced Recycled Numbers) => Success Indicator*

RES_Set_SEQr
(
-> *Resource File Reference :* Time
-> *Resource ID :* Longint
-> *TF Code :* Longint
-> *Referenced Recycled Numbers :* Pointer
)
=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource 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
	TF Code	Longint	Table Field Code (TFC)
	Referenced Recycled Numbers	Pointer	Reference to longint array containing recycled sequence numbers
	Success Indicator	Longint	qi for set successfully

The method ***RES_Set_SEQr*** sets the contents of a 'SEQr' resource within a specified resource document. The SEQr resource contains sequence number information.

Resource File Reference is the reference to the resource fork of the resource document to work with.

Resource ID is the resource ID to work with.

TF Code is the Table Field Code (TFC) to set in this resource.

Referenced Recycled Numbers is a reference to a longint array containing recycled sequence numbers.

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 successully set.

Note: the method ***RES_Set_SEQr*** was added in BASh v1.6.2.

RES_Set_TEXT_Resource

RES_Set_TEXT_Resource *(Resource Fork File Reference; Resource ID; Resource Text Value) => Success Indicator*

RES_Set_TEXT_Resource
(
-> *Resource File Reference :* Time
-> *Resource ID :* Longint
-> *Resource Text Value :* Text
)
=> *Success Indicator :* Longint

	Parameter	Type	Description
	Resource 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 Text Value	Text	Value to set the resource contents to
	Success Indicator	Longint	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 successully set.

Note: the method ***RES_Set_TEXT_Resource*** was added in BASh v1.5.1.

RW Module

The RW module provides some simple utility functions useful for handling the read/write state of tables in 4D. None of these functions work specifically with any structure, rather they provide useful mechanisms for use throughout any 4D based project.

Note: the RW module was initially added in BASh v1.8.0.

RW_Set_ReadOnly_All

RW_Set_ReadOnly_All

RW_Set_ReadOnly_All

	Parameter	Type	Description
	none	n/a	n/a

The method ***RW_Set_ReadOnly_All*** will set all tables in the current database to read only for the current process. This is extremely useful when the program will be controlling the exact access priviledges to records within the 4D database.

Note: the method ***RW_Set_ReadOnly_All*** was added in BASh v1.8.0.

RW_Wait_for_RecordLoad_Unlocked

RW_Wait_for_RecordLoad_Unlocked *(Referenced Table { ; Ticks to Wait { ; Ticks Between Loads } }) => Current Record Load State*

RW_Wait_for_RecordLoad_Unlocked

(

-> *Referenced Table :* Pointer
-> *Ticks to Wait :* Longint [optional]
-> *Ticks Between Loads :* Longint [optional]

)

=> *Current Record Load State :* Longint

	Parameter	Type	Description
	Referenced Table	Pointer	Pointer to table to load current record in read/write mode
	Ticks to Wait	Longint	Total number of ticks to wait for recrod to be loaded and locked in current process
	Ticks Between Loads	Longint	Number of ticks to wait between each load attempt of current record
	Current Record Load State	Longint	Indicator for state of record at completion of this method

The method ***RW_Wait_for_RecordLoad_Unlocked*** will attempt to repeatedly load the current record for a specified table until the record is loaded and unlocked to the current process.

Referenced Table is a pointer to a table to load the current record for. There must already be a current record for the referenced table, though it need not be either loaded or unlocked in the current process.

Ticks to Wait is the total number of ticks to attempt to load the current record unlocked before this method gives up. This parameter is optional and will default to wait indefinitely for the record to be loaded unlocked in the current process.

Ticks Between Loads is the number of ticks to wait between each load attempt of the current record. This parameter is optional and defaults to three (3) ticks when not provided. Proper setting of this value is critical to many systems, as repeated loading of the record can put a heavy load on the data engine. For online systems in which records are locked for short periods of time, setting this value low is probably ideal. For client/server systems with users in the native GUI, setting this value higher is probably a better choice.

Current Record Load State is an indicator returned by this method for the state of the current record. This value will be set to negative one (-1) if the record was deleted before loaded and locked in the current process. This value will be set to zero (0) if the method was not able to load the record unlocked in the current process before timing out. This value

will be set to one (1) if the record was loaded unlocked in the current process (assumedly, successful completion of this method as the record is then ready for editing within the current process).

Note: the method ***RW_Wait_for_RecordLoad_Unlocked*** was added in BASh v1.8.0.

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
	Semaphore Name	String[32]	Name of semaphore to clear

The method ***SEM_Clear_One*** will clear a named semaphore.

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.

Semaphore Name is the name of the semaphore which should be cleared.

SEM_ERROR

SEM_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

SEM_ERROR
(

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***SEM_ERROR*** acts as a callback method from within the SEM module for errors that may occur. Any time an error condition is detected within the SEM module, a call to the method ***SEM_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 SEM method which called the ***SEM_ERROR*** method.

The ***SEM_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 ***SEM_ERROR*** was added in BASh v1.5.4.

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

	Parameter	Type	Description
	Semaphore Name	String[32]	Name of the semaphore to set
	Ticks Between Checks	Longint	Number of ticks to wait between checking for the availability of the semaphore to the current process
	Total Ticks to Wait	Longint	Total number of ticks to wait for the semaphore in the current process until returning from the method
	Semaphore Status	Longint	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.

Semaphore Name is the name of the semaphore which should be set.

Ticks Between Checks 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 iteration of the 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.

Total Ticks to Wait 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 *Total Ticks to Wait* 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 *Total Ticks to Wait* 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.

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 (1) 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.

SEM_Test_One

SEM_Test_One *(Semaphore Name)* => *Semaphore Status*

SEM_Test_One
(
-> *Semaphore Name :* String[32]
)
=> *Semaphore Status :* Longint

	Parameter	Type	Description
	Semaphore Name	String[32]	Name of semaphore to check
	Semaphore Status	Longint	qi for whether semaphore is currently set

The method ***SEM_Test_One*** will return the status of a named semaphore.

Semaphore Name is the name of a semaphore to check the status of.

Semaphore Status is qi for whether the *Semaphore Name* is currently set. *Semaphore Status* will be set to one (1) if *Semaphore Name* is currently set; *Semaphore Status* will be set to zero (0) if *Semaphore Name* is not currently set.

Note: the method ***SEM_Test_One*** was added in BASh v1.5.4.

SEQ Module

The SEQ module is a mechanism for providing sequence numbers that does not rely on the native **Sequence Number** command or special tables to hold this information. The SEQ module works with both standalone applications and in client-server mode. Sequence numbers can be recycled, making them available to be used again.

The SEQ module organizes sequence numbers by Table Field Code (TFC). This TFC will usually be generated by a pointer to a field. This allows you to have multiple sequence numbers per table if desired. In addition, if a sequence number was needed that was not tied to a specific field, a special TFC could be made up and the sequence number could be accessed using that special code.

By default, the stoarge of sequence numbers and recycled sequence numbers is done in the resource fork of the current data file. This default storage location guarantees that the sequence numbers are associated directly with a particular data file.

As of BASh v1.8.0, the storage location for all items in the SEQ module can be procedurally set when the BASh module is initialized with the ***INIT_BASh*** method. Any document resource fork can be specified for use within the SEQ module using this new optional parameter.

Note: the SEQ module was initially added in BASh v1.6.2.

SEQ_ERROR

SEQ_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

SEQ_ERROR

(

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

)

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number

	Parameter	Type	Description
	Special Error Text	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 ***SEQ_ERROR*** acts as a callback method from within the SEQ module for errors that may occur. Any time an error condition is detected within the SEQ module, a call to the method ***SEQ_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 SEQ method which called the ***SEQ_ERROR*** method.

The ***SEQ_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.

SEQ_Get_Description

SEQ_Get_Description *(TF Code) => Description*

SEQ_Get_Description
(
-> *TF Code :* Longint
)
=> *Description :* Text

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Description	Text	Description for specified TFC

The method ***SEQ_Get_Description*** will return the SEQ module's default description for a given Table Field Code (TFC).

TF Code is the Table Field Code (TFC) for which to generate the SEQ module's default description.

Description is the generated default description for the specified Table Field Code (TFC).

Note: the method ***SEQ_Get_Description*** was added in BASh v1.6.2.

SEQ_Get_Many

SEQ_Get_Many *(TF Code ; Count ; Referenced Sequence Numbers) => qi Success*

SEQ_Get_Many
(
-> *TF Code :* Longint
-> *Count :* Longint
-> *Referenced Sequence Numbers :* Pointer
)
=> *qi Success :* Longint

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Count	Longint	Number of sequence numbers to get
	Referenced Sequence Numbers	Pointer	Reference to longint array to receive sequence numbers
	qi Success	Longint	qi for successfully retrieved sequence numbes

The method ***SEQ_Get_Many*** will return many unused sequence numbers for the specified Table Field Code (TFC).

TF Code is the Table Field Code (TFC) for which to get many unused sequence numbers.

Count is the number of sequence numbers to return.

Referenced Sequence Numbers is a reference to a longint array to receive the sequence numbers.

qi Success is an indicator value for whether this command was successful. If *qi Success* is zero (0) then the sequence numbers could not be retrieved; if *qi Success* is one (1) then the sequence numbers were successfully retreived.

Note: the method ***SEQ_Get_Many*** was added in BASh v1.6.2.

SEQ_Get_Many_NewOnly

SEQ_Get_Many_NewOnly *(TF Code ; Count ; Referenced Sequence Numbers) => qi Success*

SEQ_Get_Many_NewOnly
(
-> *TF Code :* Longint
-> *Count :* Longint
-> *Referenced Sequence Numbers :* Pointer
)
=> *qi Success :* Longint

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Count	Longint	Number of sequence numbers to get
	Referenced Sequence Numbers	Pointer	Reference to longint array to receive sequence numbers
	qi Success	Longint	qi for successfully retrieved sequence numbes

The method ***SEQ_Get_Many_NewOnly*** will return many new sequence numbers, in order, for the specified Table Field Code (TFC). These sequence numbers will not include any recycled sequence numbers.

TF Code is the Table Field Code (TFC) for which to get many unused sequence numbers.

Count is the number of sequence numbers to return.

Referenced Sequence Numbers is a reference to a longint array to receive the sequence numbers.

qi Success is an indicator value for whether this command was successful. If *qi Success* is zero (0) then the sequence numbers could not be retrieved; if *qi Success* is one (1) then the sequence numbers were successfully retreived.

Note: the method ***SEQ_Get_Many_NewOnly*** was added in BASh v1.6.2.

SEQ_Get_One

SEQ_Get_One *(TF Code) => Sequence Number*

SEQ_Get_One
(
-> *TF Code :* Longint
)
=> *Sequence Number :* Longint

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Sequence Number	Longint	Value of next sequence number

The method ***SEQ_Get_One*** will return an unused sequence number for the specified Table Field Code (TFC).

TF Code is the Table Field Code (TFC) for which to get an unused sequence number.

Sequence Number is an unused sequence number for the specified TFC. If *Sequence Number* could not be retrieved because of an error, *Sequence Number* will be set to negative one (-1).

Note: the method ***SEQ_Get_One*** was added in BASh v1.6.2.

SEQ_Get_One_NewOnly

SEQ_Get_One_NewOnly *(TF Code { ; qi No Increment }) => Sequence Number*

SEQ_Get_One_NewOnly

(

-> *TF Code :* Longint
-> *qi No Increment :* Longint [optional]

)

=> *Sequence Number :* Longint

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Sequence Number	Longint	Value of next sequence number
	qi No Increment	Longint	qi flag for whether the new sequence number to be returned should be read and incremented (default) or just read without incrementing.

The method ***SEQ_Get_One_NewOnly*** will return the next sequence number, in order, for the specified Table Field Code (TFC). This method will not access recycled sequence numbers.

TF Code is the Table Field Code (TFC) for which to get an unused sequence number.

Sequence Number is the next sequence number, in order, for the specified TFC. If *Sequence Number* could not be retrieved because of an error, *Sequence Number* will be set to negative one (-1).

qi No Increment is an optional parameter that will allows for the next new sequence number to be read without incrementing it. If this parameter is set to zero (0), the next new sequence number is read and incremented. If this parameter is set to one (1), the next new sequence number is read but not incremented. The default behavior when an invalid value is passed for this parameter or the parameter is not passed at all is the increment the stored next new sequence number.

Note: the method ***SEQ_Get_One_NewOnly*** was added in BASh v1.6.2.

Note: the optional parameter *qi No Increment* was added in BASh v1.8.0.

SEQ_Make_TFCode_by_Ref

SEQ_Make_TFCode_by_Ref *(Referenced Table or Field) => TF Code*

SEQ_Make_TFCode_by_Ref
(
-> *Referenced Table or Field :* Pointer
)
=> *TF Code :* Longint

	Parameter	Type	Description
	Referenced Table or Field	Pointer	Referenced table or field to use to generate Table Field Code (TFC)
	TF Code	Longint	Generated Table Field Code (TFC)

The method ***SEQ_Make_TFCode_by_Ref*** will return a Table Field Code (TFC), generated from the specified table or field.

Referenced Table or Field is a pointer to a table or field to use to generate a TFC.

TF Code is the Table Field Code (TFC) generated from Referenced Table or Field.

Note: the method ***SEQ_Make_TFCode_by_Ref*** was added in BASh v1.6.2.

SEQ_Recycle_Many

SEQ_Recycle_Many *(TF Code ; Referenced Sequence Numbers)*

SEQ_Recycle_Many
(
-> *TF Code :* Longint
-> *Referenced Sequence Numbers :* Pointer
)

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)

	Parameter	Type	Description
	Referenced Sequence Numbers	Pointer	Reference to longint array containing sequence numbers to be recycled

The method ***SEQ_Recycle_Many*** will recycle many sequence numbers for the specified Table Field Code (TFC). Recycled sequence numbers are then available to be given out again.

TF Code is the Table Field Code (TFC) for which to recycle many sequence numbers.

Referenced Sequence Numbers is a reference to a longint array containing the sequence numbers to be recycled.

Note: the method ***SEQ_Recycle_Many*** was added in BASh v1.6.2.

SEQ_Recycle_One

SEQ_Recycle_One *(TF Code ; Sequence Number)*

SEQ_Recycle_One
(
-> *TF Code :* Longint
-> *Sequence Number :* Longint
)

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Sequence Number	Longint	Sequence number to be recycled

The method ***SEQ_Recycle_One*** will recycle a sequence number for the specified Table Field Code (TFC). Recycled sequence numbers are then available to be given out again.

TF Code is the Table Field Code (TFC) for which to recycle a sequence number.

Sequence Number is the sequence number to be recycled.

Note: the method ***SEQ_Recycle_One*** was added in BASh v1.6.2.

SEQ_Set_Description

SEQ_Set_Description *(TF Code ; Description)*

SEQ_Set_Description
(
-> *TF Code :* Longint
-> *Description :* Text
)

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Description	Text	New description for the sequence specified by *TF Code*

The method ***SEQ_Set_Description*** will set the description of the sequence specified by the Table Field Code (TFC).

TF Code is the Table Field Code (TFC) for which to set the description.

Description is the new description for the sequence specified by *TF Code*.

Note: the method ***SEQ_Set_Description*** was added in BASh v1.6.2.

SEQ_Set_Next

SEQ_Set_Next *(TF Code ; Sequence Number)*

SEQ_Set_Next
(
-> *TF Code :* Longint
=> *Sequence Number :* Longint

)

	Parameter	Type	Description
	TF Code	Longint	Table Field Code (TFC)
	Sequence Number	Longint	Value of next sequence number

The method ***SEQ_Set_Next*** will set the next unused sequence number for the specified Table Field Code (TFC).

TF Code is the Table Field Code (TFC) for which to set the next unused sequence number.

Sequence Number is the next unused sequence number to set.

Note: the method ***SEQ_Set_Next*** was added in BASh v1.8.0.

SERNO Module

The SERNO module is useful for serializing applications or functionality within an application. It provides for encoded storage locations within the serial numbers generated for a product code, user count, major and minor versions, beta flag, and year and month expiration dates.

The routines in the SERNO module merely provide a mechanism for generating and decoding serial numbers. The actions a particular application are to take with respect to serial numbers is determined by the developer.

The serial numbers created by the SERNO module are twenty (20) byte strings of alphanumeric values. All serial numbers are case sensitive. And, it is a simple enough matter to format serial numbers with embedded hyphens for easier handling by users.

The data contained in generated serial numbers is encoded across all of the byte values within the serial number. No particular byte value corresponds to any single serial number parameters, thereby assuring reasonable security for application developers. Also, random bytes of data are spread throughout the contents of generated serial numbers and checksums are inherently encoded, as well.

Greater security for serial numbers can be had by combining the SERNO module methods with the CRYPT module and CODEC module. The CRYPT module can be used to provide a 128 bit encryption on top of a serial number and the CODEC module can then encode the encrypted serial number so that it is still easily handled by users.

There are limitations to the numerical values of all parameters encoded within a serial number. The following table lists the valid value ranges for each value encoded in a serial number:

Parameter	Low	High
Product Code	0	63
Users	0	65535
Major Version	0	63
Minor Version	0	63
Beta Flag	n/a	n/a
Month	0	16
Year	0	15

Keep in mind, though the different parameter values are named particularly, these names are merely for convenience in documentation and referencing. In essense, the differen values are named arbitrarily and can be treated as such. So, the developer chooses completely the effect each parameter, and its values, have upon the application they are being used within.

Note: the SERNO module was initially added in BASh v1.5.5.

SERNO_Create_SerialNumber

SERNO_Create_SerialNumber *(Product Code ; Users ; Major Version ; Minor Version ; Beta Flag ; Month ; Year) => Serial Number*

SERNO_Create_SerialNumber
(
-> *Product Code :* Longint
-> *Users :* Longint
-> *Major Version :* Longint
-> *Minor Version :* Longint
-> *Beta Flag :* Boolean
-> *Month :* Longint
-> *Year :* Longint
)
=> *Serial Number :* Text

	Parameter	Type	Description
	Product Code	Longint	Unique identifier code for product
	Users	Longint	Number of users
	Major Version	Longint	Major version number for product
	Minor Version	Longint	Minor version number for product
	Beta Flag	Longint	Flag to indicate beta version for product
	Month	Longint	Expiration month for product
	Year	Longint	Expiration Year for product
	Serial Number	Text	Generated serial number containing discrete data elements specified

The method ***SERNO_Create_SerialNumber*** will generate a serial number from the values provided. Subsequent calls to

this method will produce different serial numbers each time even with the parameters remaining the same. All parameters are encoded into the serial number and can be extracted later using the other methods in the SERNO module.

Product Code is the product code to encode into the generated serial number.

Users is the user count to encode into the generated serial number.

Major Version is the major version code to encode into the generated serial number.

Minor Version is the minor version code to encode into the generated serial number.

Beta Flag is the boolean beta flag to encode into the generated serial number.

Month is the expiration month value to encode into the generated serial number.

Year is the expiration year value to encode into the generated serial number.

Serial Number is the twenty (20) byte serial number randomly generated containing the encoded parameters provided.

Note: the method ***SERNO_Create_SerialNumber*** was added in BASh v1.5.5.

SERNO_Decode_SerialNumber

SERNO_Decode_SerialNumber *(Serial Number ; Referenced Product Code ; Referenced Users ; Referenced Major Version ; Referenced Minor Version ; Referenced Beta Flag ; Referenced Month ; Referenced Year) => Valid Serial Flag*

SERNO_Decode_SerialNumber
(

-> *Serial Number :* Text
-> *Referenced Product Code :* Pointer
-> *Referenced Users :* Pointer

-> *Referenced Major Version :* Pointer
-> *Referenced Minor Version :* Pointer
-> *Referenced Beta Flag :* Pointer
-> *Referenced Month :* Pointer
-> *Referenced Year :* Pointer
)
=> *Valid Serial Flag :* Boolean

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Referenced Product Code	Pointer	Referenced longint to hold product code
	Referenced Users	Pointer	Referenced longint to hold users
	Referenced Major Version	Longint	Referenced longint to hold major version
	Referenced Minor Version	Longint	Referenced longint to hold minor version
	Referenced Beta Flag	Longint	Referenced boolean to hold beta flag
	Referenced Month	Longint	Referenced longint to hold expiration month
	Referenced Year	Longint	Referenced longint to hold expiration year
	Valid Serial Flag	Boolean	q for whether serial number is valid and well formed

The method ***SERNO_Decode_SerialNumber*** will decode a serial number and return the values in referenced parameters.

Serial Number is the twenty (20) byte serial number to decode.

Referenced Product Code is a pointer to a longint to contain the product code decoded and extracted from *Serial Number*. The parameter can be passed a NULL value to skip.

Referenced Users is a pointer to a longint to contain the user count decoded and extracted from *Serial Number*. The parameter can be passed a NULL value to skip.

Referenced Major Version is a pointer to a longint to contain the major version code decoded and extracted from *Serial Number*. The parameter can be passed a NULL value to skip.

Referenced Minor Version is a pointer to a longint to contain the minor version code decoded and extracted from *Serial Number*. The parameter can be passed a NULL value to skip.

Referenced Beta Flag is a pointer to a boolean to contain the beta flag to decoded and extracted from *Serial Number*. The parameter can be passed a NULL value to skip.

Referenced Month is a pointer to a longint to contain the expiration month value decoded and extracted from *Serial Number*. The parameter can be passed a NULL value to skip.

Referenced Year is a pointer to a longint to contain the expiration year value decoded and extracted from *Serial Number*. The parameter can be passed a NULL value to skip.

Note: the method ***SERNO_Decode_SerialNumber*** was added in BASh v1.5.5.

SERNO_Get_Beta

SERNO_Get_Beta *(Serial Number) => Beta Flag*

SERNO_Get_Beta
(
-> *Serial Number :* Text
)
=> *Beta Flag :* Boolean

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Beta Flag	Boolean	Beta flag contained in supplied serial number

The method ***SERNO_Get_Beta*** will decode a serial number and return a particular encoded parameter within it.

Serial Number is the twenty (20) byte serial number to decode.

Beta Flag is the beta flag decoded and extracted from *Serial Number*.

Note: the method ***SERNO_Get_Beta*** was added in BASh v1.5.5.

SERNO_Get_MajorVersion

SERNO_Get_MajorVersion *(Serial Number) => Major Version*

SERNO_Get_MajorVersion
(
-> *Serial Number :* Text
)
=> *Major Version :* Longint

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Major Version	Longint	Major version contained in supplied serial number

The method ***SERNO_Get_MajorVersion*** will decode a serial number and return a particular encoded parameter within it.

Serial Number is the twenty (20) byte serial number to decode.

Major Version is the major version value decoded and extracted from *Serial Number*.

Note: the method ***SERNO_Get_MajorVersion*** was added in BASh v1.5.5.

SERNO_Get_MinorVersion

SERNO_Get_MinorVersion *(Serial Number) => Minor Version*

SERNO_Get_MinorVersion
(
-> *Serial Number :* Text
)

=> *Minor Version :* Longint

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Minor Version	Longint	Minor version contained in supplied serial number

The method ***SERNO_Get_MinorVersion*** will decode a serial number and return a particular encoded parameter within it.

Serial Number is the twenty (20) byte serial number to decode.

Minor Version is the minor version value decoded and extracted from *Serial Number*.

Note: the method ***SERNO_Get_MinorVersion*** was added in BASh v1.5.5.

SERNO_Get_Month

SERNO_Get_Month *(Serial Number) => Month*

SERNO_Get_Month
(
-> *Serial Number :* Text
)
=> *Month :* Longint

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Month	Longint	Expiration month contained in supplied serial number

The method ***SERNO_Get_Month*** will decode a serial number and return a particular encoded parameter within it.

Serial Number is the twenty (20) byte serial number to decode.

Month is the expiration month value decoded and extracted from *Serial Number*.

Note: the method ***SERNO_Get_Month*** was added in BASh v1.5.5.

SERNO_Get_ProductCode

SERNO_Get_ProductCode *(Serial Number) => Product Code*

SERNO_Get_ProductCode
(
-> *Serial Number :* Text
)
=> *Product Code :* Longint

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Product Code	Longint	Product code contained in supplied serial number

The method ***SERNO_Get_ProductCode*** will decode a serial number and return a particular encoded parameter within it.

Serial Number is the twenty (20) byte serial number to decode.

Product Code is the product code value decoded and extracted from *Serial Number*.

Note: the method ***SERNO_Get_ProductCode*** was added in BASh v1.5.5.

SERNO_Get_Users

SERNO_Get_Users *(Serial Number) => Users*

SERNO_Get_Users

(

-> *Serial Number :* Text

)

=> *Users :* Longint

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Users	Longint	Users contained in supplied serial number

The method ***SERNO_Get_Users*** will decode a serial number and return a particular encoded parameter within it.

Serial Number is the twenty (20) byte serial number to decode.

Users is the user count value decoded and extracted from *Serial Number*.

Note: the method ***SERNO_Get_Users*** was added in BASh v1.5.5.

SERNO_Get_Year

SERNO_Get_Year *(Serial Number)* => *Year*

SERNO_Get_Year

(

-> *Serial Number :* Text

)

=> *Year :* Longint

	Parameter	Type	Description
	Serial Number	Text	Serial number to decode
	Year	Longint	Expiration year contained in supplied serial number

The method ***SERNO_Get_Year*** will decode a serial number and return a particular encoded parameter within it.

Serial Number is the twenty (20) byte serial number to decode.

Year is the expiration year value decoded and extracted from *Serial Number*.

Note: the method ***SERNO_Get_Year*** was added in BASh v1.5.5.

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
	Text Value	Text	Text value to clean
	Cleansed Text Value	Text	Cleansed text value

The method ***STR_Clean_EmailAddress*** will remove all invalid byte values for a properly formatted 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 bytes removed.

Note: the method ***STR_Clean_EmailAddress*** was added in BASh v1.5.1.

STR_Clean_EmailUsername

STR_Clean_EmailUsername *(Text Value)* => *Cleansed Text Value*

STR_Clean_EmailUsername
(
-> *Text Value :* Text
)
=> *Cleansed Text Value :* Text

	Parameter	Type	Description
	Text Value	Text	Text value to clean
	Cleansed Text Value	Text	Cleansed text value

The method ***STR_Clean_EmailUsername*** will removed all invalid byte values for a properly comprised email username.

Valid values for email username include: uppercase letters, lowercase letters, numbers, hyphen, period, and underscore. 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 username bytes removed.

Note: the method ***STR_Clean_EmailUsername*** was added in BASh v1.5.3.

STR_Compare_Bytes

STR_Compare_Bytes *(First Text Value ; Second Text Value)* => *qi Equal*

STR_Compare_Bytes
(
-> *First Text Value :* Text
-> *Second Text Value :* Text

)

=> *qi Equal :* Longint

	Parameter	Type	Description
	First Text Value	Text	Text value to compare for equality
	Second Text Value	Text	Text value to compare for equality
	qi Equal	Longint	qi indicator for byte comparison of text values being exactly equal

The method ***STR_Compare_Bytes*** will do a byte by byte comparison of two text values and return an indicator for their exact equality. This method looks for exact byte equality between the two text values.

First Text Value is a text value to compare.

Second Text Value is a text value to compare.

qi Equal is the indicator returned for whether the two supplied text values are exactly equal. This value will be set to zero (0) if the two text values are not exactly equal and set to one (1) if the two values are exactly equal.

Note: the method ***STR_Compare_Bytes*** was added in BASh v1.8.0.

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

	Parameter	Type	Description
	Referenced Concatenated Text	Pointer	Pointer to text variable to hold concatenated text
	Overflow Option	Longint	Overflow option setting
	Referenced Text Value	Pointer	One or more pointer to text values to concatenate
	Byte Count	Longint	Number of bytes in resulting *Referenced Concatenated Text*

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

	Parameter	Type	Description
	Text Value	Text	Text value to scan
	ASCII Value	Longint	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

	Parameter	Type	Description
	Text Value	Text	Text value to scan
	Match Text	Text	Text value to match for counting
	Occurrence Count	Longint	Count of occurrences of *Match Text* within *Text Value*

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

STR_Get_CharPosition *(Source Text ; Search Text ; Starting Position)* => *Found Position*

STR_Get_CharPosition
(
-> *Source Text :* Text
-> *Search Text :* Text
-> *Starting Position :* Longint
)
=> *Found Position :* Longint

	Parameter	Type	Description
	Source Text	Text	Text value to search
	Search Text	Text	Text to search for in Source Text
	Starting Position	Longint	Starting position for search within *Text Value*
	Found Position	Longint	First position of *Search Text* found within *Source Text*.

The method ***STR_Get_CharPosition*** returns the position of the search text found within the specified text block. The search starts at the starting position, and will continue either until the search text is found, or the end of the text to search.

Source Text is the text block to search.

Search Text is the text to search for within *Source Text*.

Starting Position is the starting position to begin the search from.

Found Position is the position of the first occurrence of *Search Text* found within *Source Text* after *Starting Position*. The following table summarizes result values for *Found Position* after calling this method:

Found Position	Indicates
> 0	position of text match
- 1	no match found

Note: the method ***STR_Get_CharPosition*** was added in BASh v1.6.1.

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

	Parameter	Type	Description
	Text Value	Text	Text value to search
	ASCII Value	Longint	ASCII value of byte to scan for
	Start Byte	Longint	Starting byte position within *Text Value*
	Direction Option	Longint	Direction option for whether to scan forward or backwards within *Text Value*
	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 doen 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:

Found Position	Indicates
> 0	position of next match
0	no match found
- 1	text blob to scan is empty
- 2	ASCII value is invalid
- 3	start byte is out of range

Note: since *Direction Option* is actually the looping parameter within this method, scanning can 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_Position_by_ASCII_Range

STR_Get_Position_by_ASCII_Range *(Source Text ; Starting Position ; Scan Length ; ASCII Low Value ; ASCII High Value) => Found Position*

STR_Get_Position_by_ASCII_Range
(

-> *Source Text :* Text
-> *Starting Position :* Longint
-> *Scan Length :* Longint
-> *ASCII Low Value :* Longint
-> *ASCII High Value :* Longint
)
=> *Found Position :* Longint

	Parameter	Type	Description
	Source Text	Text	Text to scan
	Starting Position	Longint	Starting position for search
	Scan Length	Longint	Length of text to search
	ASCII Low Value	Longint	Low end of ASCII range to look for
	ASCII High Value	Longint	High end of ASCII range to look for
	Found Position	Longint	Position of first byte found in *Source Text* within the ASCII range specified

The method ***STR_Get_Position_by_ASCII_Range*** returns the byte position of the first byte found within the text block that is within the specified ASCII range. The text can be searched either forward or backward from any starting position within the text block.

The ASCII range is inclusive, meaning that the specified low and high values will be searched for as well as all values between them.

Source Text is the text block to scan.

Starting Position is the position within the *Source Text* to begin searching for a character within the specified ASCII value range. Pass a value of one (1) to begin searching at the first character, or a value of MAXLONG to begin searching at the end of the *Source Text*.

Scan Length is the number of characters following *Starting Position* within *Source Text* which will be searched. Pass a positive value to search forward, or a negative value to search backward. Pass MAXLONG to search to the end of the text, or negative MAXLONG (-MAXLONG) to search to the beginning of the text.

ASCII Low Value is the low end of the ASCII range to search for.

ASCII High Value is the high end of the ASCII range to search for.

Found Position is the position within *Source Text* of the first character found within the specified ASCII value range. If no matching character was found within the specified offset range, *Found Position* will be set to negative one (-1).

Note: the method ***STR_Get_Position_by_ASCII_Range*** was added in BASh v1.6.1.

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
	Text Value	Text	Text block to scan
	First Line Text	Text	Text of first line within *Text Value*

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
)

	Parameter	Type	Description
	Referenced Text Value	Pointer	Pointer to text value to pad
	Preferred Length	Longint	Preferred minimum length of *Referenced Text Value*
	ASCII Value	Longint	ASCII value of bytes used for padding
	Padding Flag	Boolean	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

	Parameter	Type	Description
	Referenced Text Array	Pointer	Pointer to text array to hold parsed values
	Text Value	Text	Text block to parse
	Delimiter ASCII Value	Longint	ASCII value of value delimiter within *Text Value*
	Element Count	Longint	Number of elements parsed from *Text Value*

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:

Element Count	Indicates
> 0	element count
- 1	reference is 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; Delimter Text) => Element Count*

STR_Parse_to_Array_by_Str
(
-> *Referenced Text Array :* Pointer
-> *Text Value :* Text
-> *Delimiter Text :* Text
)
=> *Element Count :* Longint

	Parameter	Type	Description
	Referenced Text Array	Pointer	Pointer to text array to hold parsed values
	Text Value	Text	Text block to parse
	Delimiter Text	Text	Text value of delimiter within *Text Value*
	Element Count	Longint	Number of elements parsed from *Text Value*

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.

Delimter 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:

Element Count	Indicates
> 0	element count
- 1	reference is 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_Position_NonAlpha

STR_Position_NonAlpha *(Text Value ; Starting Position ; Byte Count) => Found Position*

STR_Position_NonAlpha
(
-> *Text Value :* Text
-> *Starting Position :* Longint
-> *Byte Count :* Longint
)
=> *Found Position :* Longint

	Parameter	Type	Description
	Text Value	Text	Text to scan
	Starting Position	Longint	Starting position in text to begin scanning from
	Byte Count	Longint	Maximum number of bytes within text to scan
	Found Position	Longint	Position within text of first non-alpha byte found

The method ***STR_Position_NonAlpha*** returns the first byte position within a specified range of a specified text value which is not a standard ASCII character value.

Text Value is the text which is to be scanned.

Starting Position is the first byte position within *Text Value* which is to be scanned.

Byte Count is the maximum number of bytes to scan after *Starting Position* within *Text Value*. No bytes after will be scanned. To always scan to the end of *Text Value*, pass a value of MaxLong for *Byte Count*.

Found Position is the first position within *Text Value* after *Starting Position* which contains a non-alpha value. If there are no bytes found within the specified range which are non-alpha, *Found Position* will be set to zero (0).

Note: the method ***STR_Position_NonAlpha*** was added in BASh v1.5.7.

STR_Position_NonAlphaNumeric

STR_Position_NonAlphaNumeric (*Text Value* ; *Starting Position* ; *Byte Count*) => *Found Position*

STR_Position_NonAlphaNumeric
(
-> *Text Value :* Text
-> *Starting Position :* Longint

-> *Byte Count :* Longint
)
=> *Found Position :* Longint

	Parameter	Type	Description
	Text Value	Text	Text to scan
	Starting Position	Longint	Starting position in text to begin scanning from
	Byte Count	Longint	Maximum number of bytes within text to scan
	Found Position	Longint	Position within text of first non-alphanumeric byte found

The method ***STR_Position_NonAlphaNumeric*** returns the first byte position within a specified range of a specified text value which is not a standard ASCII character or numeral value.

Text Value is the text which is to be scanned.

Starting Position is the first byte position within *Text Value* which is to be scanned.

Byte Count is the maximum number of bytes to scan after *Starting Position* within *Text Value*. No bytes after will be scanned. To always scan to the end of *Text Value*, pass a value of MaxLong for *Byte Count*.

Found Position is the first position within *Text Value* after *Starting Position* which contains a non-alphanumeric value. If there are no bytes found within the specified range which are non-alphanumeric, *Found Position* will be set to zero (0).

Note: the method ***STR_Position_NonAlphaNumeric*** was added in BASh v1.5.7.

STR_Position_NonNumeric

STR_Position_NonNumeric (*Text Value* ; *Starting Position* ; *Byte Count*) => *Found Position*

STR_Position_NonNumeric

```
(
        -> Text Value : Text
        -> Starting Position : Longint
        -> Byte Count : Longint
)
        => Found Position : Longint
```

	Parameter	Type	Description
	Text Value	Text	Text to scan
	Starting Position	Longint	Starting position in text to begin scanning from
	Byte Count	Longint	Maximum number of bytes within text to scan
	Found Position	Longint	Position within text of first non-numeric byte found

The method ***STR_Position_NonNumeric*** returns the first byte position within a specified range of a specified text value which is not a standard ASCII numeric value.

Text Value is the text which is to be scanned.

Starting Position is the first byte position within *Text Value* which is to be scanned.

Byte Count is the maximum number of bytes to scan after *Starting Position* within *Text Value*. No bytes after will be scanned. To always scan to the end of *Text Value*, pass a value of MaxLong for *Byte Count*.

Found Position is the first position within *Text Value* after *Starting Position* which contains a non-numeric value. If there are no bytes found within the specified range which are non-numeric, *Found Position* will be set to zero (0).

Note: the method ***STR_Position_NonNumeric*** was added in BASh v1.5.7.

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 Wildcard ASCII Value :* Longint
-> *Single Wildcard ASCII Value :* Longint

)

=> *Match Successful :* Longint

	Parameter	Type	Description
	Source Text Value	Text	Text to compare with a particular filter and filter criteria
	Filter	Text	Filter to compare against *Source Text Value*
	Multiple Wildcard ASCII Value	Longint	ASCII value of multiple wildcard byte within *Filter*
	Single Wildcard ASCII Value	Longint	ASCII value of single wildcard byte within *Filter*
	Match Successful	Longint	qi for whether *Source Text Value* 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_qi_Valid_EmailAddress

STR_qi_Valid_EmailAddress *(Email Address)* => *Valid Indicator*

STR_qi_Valid_EmailAddress
(
-> *Email Address :* Text
)
=> *Valid Indicator :* Longint

	Parameter	Type	Description
	Email Address	Text	Email address to check for validity in formatting and construct
	Valid Indicator	Longint	Indicator valid for whether specified email address is well formed

The method ***STR_qi_Valid_EmailAddress*** will do basic formatting and content checks on a supplied email address and indicate whether it is well formed or not.

This routine will check for invalid byte values within the supplied email address. Also, the email address must contain an at sign (“@”) in a valid position and a period (“.”) within a valid position. Obviously, no checks are done as to whether the email address exists or even if the domain of the email address exists or is even valid. Rather, this routine is checking the formatting and validity of byte values within the supplied email address.

Email Address is the text of the email address to check.

Valid Indicator is a longint indicator of the validity of the content and formatting of the supplied email address. This

will be set to zero (0) if the supplied email address is not valid, one (1) if the supplied email address appears well formed and formatted, or two (2) if the supplied email address parameter is empty.

Note: the method ***STR_qi_Valid_EmailAddress*** was added in BASh v1.8.2.

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

	Parameter	Type	Description
	Text Value	Text	Text block to truncate
	Delimiter ASCII Value	Longint	ASCII value to truncate after
	Truncated Text Value	Text	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

	Parameter	Type	Description
	Text Value	Text	Text block to scan
	Truncated Text Value	Text	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

	Parameter	Type	Description
	Text Value	Text	Text value to filter
	Filtered Text Value	Text	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
	Text Value	Text	Text value to trim
	Trimmed Text Value	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_Pre *(Text Value) => Trimmed Text Value*

STR_Remove_Spaces_Pre
(
-> *Text Value :* Text
)
=> *Trimmed Text Value :* Text

	Parameter	Type	Description
	Text Value	Text	Text value to trim
	Trimmed Text Value	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
	Text Value	Text	Text value to trim
	Trimmed Text Value	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
	Text Value	Text	Text value to scan
	Old ASCII Value	Longint	ASCII value to be replaced
	New ASCII Value	Longint	ASCII value to replace with
	New Text Value	Text	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_Strip_Between

STR_Strip_Between *(Text to be Stripped ; Starting Delimiter Text Value ; Ending Delimiter Text Value ; Strip Delimiters Code) => Stripped Text*

STR_Strip_Between
(
-> Text to be Stripped : Text
-> Starting Delimiter Text Value : Text
-> Ending Delimiter Text Value : Text
-> Strip Delimiters Code : Longint
)
=> Stripped Text : Text

	Parameter	Type	Description
	Text to be Stripped	Text	Text value to be scanned for stripping
	Starting Delimiter Text Value	Text	Text value of starting delimiter for stripping
	Ending Delimiter Text Value	Text	Text value of ending delimiter for stripping
	Strip Delimiters Code	Longint	Code for whether delimiters are to be stripped along with contents between delimiters

	Parameter	Type	Description
	Stripped Text	Text	Text value of contents and possibly delimiters stripped

The method ***STR_Strip_Between*** will remove delimited contents from a specified text value. The delimiter values can be specified, as well as whether the delimiters themselves will be removed.

When dealing with delimiters of a single byte, the method ***STR_Strip_Between_by_ASCII*** will be much faster, as the comparisons used within this routine are much less efficient.

Text to be Stripped is a text value to be scanned for delimited data.

Starting Delimiter Text Value is the text value of the beginning delimiter to search for within *Text to be Stripped*.

Ending Delimiter Text Value is the text value of the ending delimiter to search for within *Text to be Stripped*.

Strip Delimiters Code is a longint value code for indicating whether the delimiter bytes are to be removed with the content between any and all found delimiters. If *Strip Delimiters Code* is set to one (1) then the delimiter bytes will also be removed. If *Strip Delimiters Code* is set to zero (0) then the delimiter bytes will not be removed. All other values for *Strip Delimiters Code* will be consider the same as passing zero (0) for this parameter.

Stripped Text is *Text to be Stripped* with delimited contents removed.

Note: the method ***STR_Strip_Between*** was added in BASh v1.6.0

STR_Strip_Between_by_ASCII

STR_Strip_Between_by_ASCII *(Text to be Stripped ; Starting Delimiter ASCII Value ; Ending Delimiter ASCII Value ; Strip Delimiters Code) => Stripped Text*

STR_Strip_Between_by_ASCII

```
(
        -> Text to be Stripped : Text
        -> Starting Delimiter ASCII Value : Longint
        -> Ending Delimiter ASCII Value : Longint
        -> Strip Delimiters Code : Longint
)
        => Stripped Text : Text
```

	Parameter	Type	Description
	Text to be Stripped	Text	Text value to be scanned for stripping
	Starting Delimiter ASCII Value	Longint	ASCII value of starting delimiter byte for stripping
	Ending Delimiter ASCII Value	Text	ASCII value of ending delimiter byte for stripping
	Strip Delimiters Code	Longint	Code for whether delimiters are to be stripped along with contents between delimiters
	Stripped Text	Text	Text value of contents and possibly delimiters stripped

The method ***STR_Strip_Between_by_ASCII*** will remove delimited contents from a specified text value. The delimiter byte values can be specified, as well as whether the delimiter bytes themselves will be removed.

When dealing with delimiters of a single byte, this routine will be much faster than ***STR_Strip_Between***, as the comparisons used within this routine are much more efficient.

Text to be Stripped is a text value to be scanned for delimited data.

Starting Delimiter ASCII Value is the byte value of the beginning delimiter to search for within *Text to be Stripped*.

Ending Delimiter ASCII Value is the byte value of the ending delimiter to search for within *Text to be Stripped*.

Strip Delimiters Code is a longint value code for indicating whether the delimiter bytes are to be removed with the content between any and all found delimiters. If *Strip Delimiters Code* is set to one (1) then the delimiter bytes will also

be removed. If *Strip Delimiters Code* is set to zero (0) then the delimiter bytes will not be removed. All other values for *Strip Delimiters Code* will be consider the same as passing zero (0) for this parameter.

Stripped Text is *Text to be Stripped* with delimited contents removed.

Note: the method ***STR_Strip_Between_by_ASCII*** was added in BASh v1.6.0

STR_Unwrap_by_ASCII

STR_Unwrap_by_ASCII(*Wrapped Text* ; *Wrapped Byte ASCII Value*) => *Unwrapped Text*

STR_Unwrap_by_ASCII
(
-> *Wrapped Text :* Text
-> *Wrapped Byte ASCII Value :* Longint
)
=> *Unwrapped Text :* Text

	Parameter	Type	Description
	Wrapped Text	Text	Text value to be unwrapped
	Wrapped Byte ASCII Value	Longint	ASCII byte value of character to be removed at beginning and end of *Wrapped Text*
	Unwrapped Text	Text	Text value after unwrapping

The method ***STR_Unwrap_by_ASCII*** will unwrap a specified text value of a specified byte value if the text value both begins and ends with the specified byte value. This method is ideal for removing formatting characters from text; for instance, it is a simple means to remove double quotes from columnar data being imported where each value is wrapped in double quotes.

Wrapped Text is a text value to check for being wrapped and to unwrap if by the specifed wrap byte value.

Wrapped Byte ASCII Value is a longint containing the ASCII value of the wrapping byte to check as a wrapper. If this value is both the beginning and ending character in *Wrapped Text* then it will be removed in this routine.

Unwrapped Text is the unwrapped text value after being unwrapped within this routine.

Note: the method ***STR_Unwrap_by_ASCII*** was added in BASh v1.8.2.

STR_Wrap_in_DoubleQuotes

STR_Wrap_in_DoubleQuotes *(Text Value) => Wrapped Text Value*

STR_Wrap_in_DoubleQuotes
(
-> *Text Value :* Text
)
=> *Wrapped Text Value :* Text

	Parameter	Type	Description
	Text Value	Text	Text value to wrap
	Wrapped Text Value	Text	*Text Value* 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
	First Time Addend	Time	Time value to add
	Second Time Addend	Time	Time value to add
	Normalized Time Sum	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 examplifies 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
	Time Value	Time	Supplied time value
	Number of Hours	Longint	Number of hours in supplied time value *Time Value*

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
	Time Value	Time	Supplied time value
	Number of Minutes	Longint	Number of minutes past hour in supplied time value *Time Value*

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
	Time Value	Time	Supplied time value
	Number of Seconds	Longint	Number of seconds past minute in supplied time value *Time Value*

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 examplifies 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.

2D Array Types

The field and variable types native to 4D are incomplete. There is no support for differentiation between different 2D array types. The constants group **TYPE, Array 2D, Variable Types**, documented earlier in this document, was added to BASh to compensate for this deficiency. These extended variable types can aid significantly in development in 4D.

Full support for these 2D array types is available throughout the BASh component. Understanding the format and values used in the support of different 2D array types is invaluable to 4D development.

Note: support for the extended 2D array types was added in BASh v1.8.0.

Loose Typing

Many different field and variable types in 4D actually can coexist in expressions with impugnity. For instance, comparing a 4D Integer value with a 4D Longint value is completely fine in the 4D language. Loosely, these two types of variables can be considered equivalent.

The “Loose Typing” introduced in the TYPE module of BASh makes allowances for such behavior in 4D. For all variable and field types available in 4D, the TYPE module in BASh considers there is a loose type equivalent. These loose types are used throughout BASh where appropriate.

The following table lists all of the native 4D field and variable types and the extended types for 2D arrays used in the TYPE moduel and shows their equivalent loose type:

4D/TYPE 2D Type	Loose Constant	Loose Value
Is Alpha Field	Is Text	2
Is Real	Is Real	1
Is Text	Is Text	2
Is Picture	Is Picture	3
Is Date	Is Date	4
Is Boolean	Is Boolean	6
Is Integer	Is Longint	9
Is Longint	Is Longint	9
Is Time	Is Time	11
Real Array	Real Array	14
Integer Array	Longint Array	16
Longint Array	Longint Array	16
Date Array	Date Array	17
Text Array	Text Array	18
Picture Array	Picture Array	19
Pointer Array	Pointer Array	20
String Array	Text Array	18
Boolean Array	Boolean Array	22
Is Pointer	Is Pointer	23
Is String Var	Is Text	2
Is BLOB	Is BLOB	30
Type_A2D_Real	TYPE_A2D_Real	114
TYPE_A2D_Integer	TYPE_A2D_Longint	116
TYPE_A2D_Longint	TYPE_A2D_Longint	116
TYPE_A2D_Date	TYPE_A2D_Date	117
TYPE_A2D_Text	TYPE_A2D_Text	118
TYPE_A2D_Picture	TYPE_A2D_Picture	119
TYPE_A2D_Pointer	TYPE_A2D_Pointer	120
TYPE_A2D_String	TYPE_A2D_Text	118

4D/TYPE 2D Type	Loose Constant	Loose Value
TYPE_A2D_Boolean	TYPE_A2D_Boolean	122

Note: support for loose typing was added in BASh v1.8.0.

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:

Unary	Array
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

	Parameter	Type	Description
	Referenced Variable	Pointer	Reference to a variable which will have its type checked
	Variable Type	Longint	Variable type to compare to (variable types are designated by the 4D constants Field and Variable Types)
	Match State	Longint	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 a indicator for whether the types match or an error condition.

Referenced Variable is a pointer to any variable whose type is going to be checked.

Variable Type is a variable type value to check against the type of *Referenced Variable*. The list of valid type values are available as native constants in 4th Dimension under the Field and Variable Types grouping.

Match State is the indicator for the match validity of *Referenced Variable* and *Variable Type*. Valid values for *Match State* are as follows:

Value	Meaning
1	Types Match
0	Types to not match
- 2	*Referenced Variable* is Nil
- 2	*Referenced Variable* is not a pointer

TYPE_Compare_UnaryTypes_Loose

TYPE_Compare_UnaryTypes_Loose *(First Type Value ; Second Type Value) => qi Equal*

TYPE_Compare_UnaryTypes_Loose

(

-> *First Type Value :* Longint
-> *Second Type Value :* Longint

)

=> *qi Equal :* Longint

	Parameter	Type	Description
	First Type Value	Longint	Type value to do loose comparison on
	Second Type Value	Longint	Type value to do loose comparison on
	qi Equal	Longint	qi indicator for whether type values are loosely equal

The method ***TYPE_Compare_UnaryTypes_Loose*** will do a loose comparison on two supplied type values and return whether they are loosely equal or not.

First Type Value is a type value to do a loose comparison with.

Second Type Value is a type value to do a loose comparison with.

qi Equal is the indicator returned for whether the two supplied type values are loosely equal. This value is set to zero (0) if they are not loosely equal and set to one (1) if they are loosely equal.

Note: the method ***TYPE_Compare_UnaryTypes_Loose*** was added in BASh v1.8.0.

TYPE_ERROR

TYPE_ERROR *(BASh Error Number; Special Error Text; Calling Method Name)*

TYPE_ERROR

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

	Parameter	Type	Description
	BASh Error Number	Longint	Internal BASh error number
	Special Error Text	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 ***TYPE_ERROR*** acts as a callback method from within the TYPE module for errors that may occur. Any time an error condition is detected within the TYPE module, a call to the method ***TYPE_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 TYPE method which call the ***TYPE_ERROR*** method.

The ***TYPE_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: this method was added in BASh v1.8.0.

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 Field and Variable Types)
	Array Type	Longint	Equivalent array data type (variable types are designated by the 4D constants Field and Variable Types)

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_Type

TYPE_Get_Type *(Referenced Field or Variable) => Extended Type*

TYPE_Get_Type

(

-> *Referenced Field or Variable :* Pointer

)

=> *Extended Type :* Longint

	Parameter	Type	Description
	Referenced Field or Variable	Pointer	Pointer to field or variable to check the type of

	Parameter	Type	Description
	Extended Type	Longint	Type value of supplied referenced parameter (see **TYPE, Array 2D, Variable Types** custom constants group for extended variable types)

The method ***TYPE_Get_Type*** will return the type of a referenced field or variable passed to it. The type value returned will be an extended type for 2 dimensional arrays. This method should be used as a replacement for the native **Type** command in 4D, as apprropriate.

Referenced Field or Variable is a pointer to a field or variable to get the type of.

Extended Type is the type of the value referenced by *Referenced Field or Variable*. If the referenced parameter is a 2 dimensional array, the extended types will be used as the result of this method.

Note: the method ***TYPE_Get_Type*** was added in BASh v1.8.0.

TYPE_Get_Type_Loose

TYPE_Get_Type_Loose *(Type Value) => Loose Type Value*

TYPE_Get_Type_Loose
(
-> *Type Value :* Longint
)
-> *Loose Type Value :* Longint

	Parameter	Type	Description
	Type Value	Longint	Type value to retrieve the loose type value of
	Loose Type Value	Longint	Loose type value of supplied type value

The method ***TYPE_Get_Type_Loose*** will return the loose type value for a given type value.

Type Value is the given type value to get the loose type value of.

Loose Type Value is the loose type value equivalent of *Type Value*. If the supplied *Type Value* is invalid, this will be set to negative one (-1).

Note: the method ***TYPE_Get_Type_Loose*** was added in BASh v1.8.0.

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 Field and Variable Types)
	Unary Type	Longint	Equivalent array data type (variable types are designated by the 4D constants Field and Variable Types)

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 Field and Variable Types)
	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.

Variable Type 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.

Array Match State is an indicator of whether *Variable Type* is an array type. If *Array Match State* is zero (0) then *Variable Type* is not an array. If *Array Match State* is one (1) then *Variable Type* is an array of some type.

TYPE_qi_Array_1D

TYPE_qi_Array_1D *(Type Value) => qi 1D Array*

TYPE_qi_Array_1D
(
-> *Type Value :* Longint
)

=> *qi 1D Array :* Longint

	Parameter	Type	Description
	Type Value	Longint	Type value to check
	qi 1D Array	Longint	Indicator for whether supplied value is a 1D array

The method ***TYPE_qi_Array_1D*** will return an indicator for whether a supplied type value is a one dimensional array type.

Type Value is the type value to check.

qi 1D Array is an indicator for whether *Type Value* is a one dimensional array type. This value will be set to zero (0) if *Type Value* is not a one dimensional array and set to one (1) if it is.

Note: the method ***TYPE_qi_Array_1D*** was added in BASh v1.8.0.

TYPE_qi_Array_2D

TYPE_qi_Array_2D *(Type Value) => qi 2D Array*

TYPE_qi_Array_2D
(
-> *Type Value :* Longint
)
=> *qi 2D Array :* Longint

	Parameter	Type	Description
	Type Value	Longint	Type value to check
	qi 2D Array	Longint	Indicator for whether supplied value is a 2D array

The method ***TYPE_qi_Array_2D*** will return an indicator for whether a supplied type value is a two dimensional array type.

Type Value is the type value to check.

qi 2D Array is an indicator for whether *Type Value* is a two dimensional array type. This value will be set to zero (0) if *Type Value* is not a two dimensional array and set to one (1) if it is.

Note: the method ***TYPE_qi_Array_2D*** was added in BASh v1.8.0.

TYPE_qi_BLOB

TYPE_qi_BLOB *(Variable Type) => BLOB Match State*

TYPE_qi_BLOB

(

-> *Variable Type :* Longint

)

=> *BLOB Match State :* Longint

	Parameter	Type	Description
	Variable Type	Longint	Variable type to check (variable types are designated by the 4D constants Field and Variable Types)
	BLOB Match State	Longint	State of type match check

The method ***TYPE_qi_BLOB*** will check whether a supplied variable type is a BLOB.

Variable Type 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 a BLOB.

BLOB Match State is an indicator of whether *Variable Type* is of BLOB type. If *BLOB Match State* is zero (0) then *Variable Type* is not a BLOB. If *BLOB Match State* is one (1) then *Variable Type* is a BLOB.

TYPE_qi_Unary

TYPE_qi_Unary *(Variable Type) => Unary Match State*

TYPE_qi_Unary

(

-> *Variable Type :* Longint

)

=> *Unary Match State :* Longint

	Parameter	Type	Description
	Variable Type	Longint	Variable type to check (variable types are designated by the 4D constants Field and Variable Types)
	Unary Match State	Longint	State of type match check

The method ***TYPE_qi_Unary*** will check whether a supplied variable type is any form of unary variable.

Variable Type 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 unary variable or field type, regardless of the exact type of unary.

Unary Match State is an indicator of whether *Variable Type* is an array type. If *Unary Match State* is zero (0) then *Variable Type* is not a unary. If *Unary Match State* is one (1) then *Variable Type* is a unary of some type.

Note: the method ***TYPE_qi_Unary*** was added in BASh v1.8.0.

URL Module

The URL module comprises basic routines for handling and manipulating fully formatted, RFC compliant URLs. The routines within the URL module allow for the creation and parsing of properly formatted URLs. The current version of the URL module supports nine (9) different URL schemes. As needed, more URL schemes will be supported in future releases of the BASh component.

Note: the URL module was initially added in BASh v1.5.7.

Fully Formatted URLs

RFC compliant URL formatting is discussed in RFC 1738. You can get a copy of this RFC from:

http://www.deepskytech.com/rfcs/rfc_1738.txt

The basic format for URLs is as follows:

<scheme><username>:<password>@<host>:<port>/<path>$<directarg>?<searchargs>

The rules for which values are allowed for a particular scheme are available within RFC 1738. As well, the rules for the inclusion of different separator values are dependant upon the inclusion of other variable values within a particular URL scheme.

<scheme> is often known as the protocol. It is an indicator, often ending with a colon (":") or a colon followed by two slashes ("://"). Common values for schems include "http://", "ftp://", and "mailto:". The URL scheme constants provided within the BASh module provide a means to indicate different scheme values more compactly and efficient. Routines within the URL module provide a means for translating back and forth between the textual value of a scheme and its constant value.

The other values that comprise an URL can be learned about by reading RFC 1738, referenced above.

URL_Extract_DirectParam

URL_Extract_DirectParam *(Fully Formatted URL) => Direct Parameter*

URL_Extract_DirectParam

(
    -> *Fully Formatted URL :* Text
)
    => *Direct Parameter :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Direct Parameter	Text	Direct parameter value extracted from *Fully Formatted URL*

The method ***URL_Extract_DirectParam*** will return the direct parameter value extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Direct Parameter is the direct parameter value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_DirectParam*** was added in BASh v1.5.7.

URL_Extract_Host

URL_Extract_Host *(Fully Formatted URL)* => *Host*

URL_Extract_Host
(
    -> *Fully Formatted URL :* Text
)
    => *Host :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Host	Text	Host value extracted from *Fully Formatted URL*

The method ***URL_Extract_Host*** will return the host name value extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Host is the host name value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Host*** was added in BASh v1.5.7.

URL_Extract_Password

URL_Extract_Password *(Fully Formatted URL) => Password*

URL_Extract_Password
(
-> *Fully Formatted URL :* Text
)
=> *Password :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Password	Text	Password value extracted from *Fully Formatted URL*

The method ***URL_Extract_Password*** will return the password value extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Password is the password value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Password*** was added in BASh v1.5.7.

URL_Extract_Path

URL_Extract_Path *(Fully Formatted URL)* => *Path*

URL_Extract_Path
(
-> *Fully Formatted URL :* Text
)
=> *Path :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Path	Text	Path value extracted from *Fully Formatted URL*

The method ***URL_Extract_Path*** will return the path value extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Path is the path value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Path*** was added in BASh v1.5.7.

URL_Extract_Path_w_Params

URL_Extract_Path_w_Params *(Fully Formatted URL)* => *Path with Parameters*

URL_Extract_Path_w_Params
(
-> *Fully Formatted URL :* Text
)
=> *Path with Parameters :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Path with Parameters	Text	Path value with attached direct and search parameters extracted from *Fully Formatted URL*

The method ***URL_Extract_Path_w_Params*** will return the path value, direct parameter, and search parameters extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Path with Parameters is the path value with the subsequent direct parameter and search arguments, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Path_w_Params*** was added in BASh v1.5.7.

URL_Extract_Port

URL_Extract_Port *(Fully Formatted URL)* => *Port*

URL_Extract_Port
(
-> *Fully Formatted URL :* Text
)
=> *Port :* Longint

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Port	Longint	Port value extracted from *Fully Formatted URL*

The method ***URL_Extract_Port*** will return the path value extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Port is the port value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Port*** was added in BASh v1.5.7.

URL_Extract_Scheme_i

URL_Extract_Scheme_i *(Fully Formatted URL) => Scheme Constant*

URL_Extract_Scheme_i
(
-> *Fully Formatted URL :* Text
)
=> *Scheme Constant :* Longint

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Scheme Constant	Longint	Scheme constant value extracted from *Fully Formatted URL*

The method ***URL_Extract_Scheme_i*** will return the scheme constant value for the scheme extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Scheme Constant is the scheme constant value for the scheme value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Scheme_i*** was added in BASh v1.5.7.

URL_Extract_Scheme_x

URL_Extract_Scheme_x *(Fully Formatted URL)* => *Scheme*

URL_Extract_Scheme_x
(
-> *Fully Formatted URL :* Text
)
=> *Scheme :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Scheme	Text	Scheme constant value extracted from *Fully Formatted URL*

The method ***URL_Extract_Scheme_i*** will return the scheme constant value for the scheme extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Scheme is the textual scheme value for the scheme value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Scheme_x*** was added in BASh v1.5.7.

URL_Extract_SearchParams

URL_Extract_SearchParams *(Fully Formatted URL)* => *Search Parameters*

URL_Extract_SearchParams
(
-> *Fully Formatted URL :* Text
)
=> *Search Parameters :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Search Parameters	Text	Search parameters value extracted from *Fully Formatted URL*

The method ***URL_Extract_SearchParams*** will return the search parameters value extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Search Parameters is the search parameters value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_SearchParams*** was added in BASh v1.5.7.

URL_Extract_Username

URL_Extract_Username *(Fully Formatted URL) => Username*

URL_Extract_Username
(
-> *Fully Formatted URL :* Text
)
=> *Username :* Text

	Parameter	Type	Description
	Fully Formatted URL	Text	Fully formatted, RFC compliant URL to extract discrete value from
	Username	Text	User name value extracted from *Fully Formatted URL*

The method ***URL_Extract_Username*** will return the user name value extracted from a fully formatted URL.

Fully Formatted URL is a full, RFC compliant URL which is to have data extracted from it.

Username is the user name value, if any, extracted from the specified *Fully Formatted URL*.

Note: the method ***URL_Extract_Username*** was added in BASh v1.5.7.

URL_Get_Scheme_i

URL_Get_Scheme_i *(Scheme) => Scheme Constant*

URL_Get_Scheme_i
(
-> *Scheme :* Text
)
=> *Scheme Constant :* Longint

	Parameter	Type	Description
	Scheme	Text	Textual scheme value
	Scheme Constant	Longint	Scheme constant value matching supplied textual scheme value

The method ***URL_Get_Scheme_i*** will return scheme constant value for a specified textual scheme value.

Scheme is the textual scheme value to get the scheme constant for.

Scheme Constant is the scheme constant value matching the supplied textual scheme value, *Scheme*. *Scheme Constant* will be set to zero (0) if the *Scheme* is not a valid scheme.

Note: the method ***URL_Get_Scheme_i*** was added in BASh v1.5.7.

URL_Get_Scheme_x

URL_Get_Scheme_x *(Scheme Constant)* => *Scheme*

URL_Get_Scheme_x
(
-> *Scheme Constant :* Longint
)
=> *Scheme :* Text

	Parameter	Type	Description
	Scheme Constant	Longint	Scheme constant value
	Scheme	Text	Textual scheme value matching supplied scheme constant value

The method ***URL_Get_Scheme_x*** will return textual scheme value for a specified scheme constant value.

Scheme Constant is the scheme constant value to get the textual scheme value for.

Scheme is the textual scheme value matching the supplied scheme constant value, *Scheme Constant*. *Scheme* will be set to NULL (empty) if the *Scheme Constant* is not a valid scheme constant value.

Note: the method ***URL_Get_Scheme_x*** was added in BASh v1.5.7.

URL_Make_URL

URL_Make_URL *(Scheme ; Username ; Password ; UnPw Flags ; Host ; Port ; Path ; Direct Parameter ; Referenced Search Arguments Names ; Referenced Search Arguments Values)* => *URL*

URL_Make_URL
(
-> *Scheme :* Longint
-> *Username :* Text
-> *Password :* Text
-> *UnPw Flags :* Longint
-> *Host :* Text
-> *Port :* Longint
-> *Path :* Text

-> *Direct Parameter :* Text
-> *Referenced Search Arguments Names :* Pointer
-> *Referenced Search Arguments Values :* Pointer
)
=> *URL :* Text

	Parameter	Type	Description
	Scheme	Longint	Scheme constant value for URL to be created
	Username	Text	Username for URL to be created
	Password	Text	Password for URL to be created
	UnPw Flags	Longint	Coded username and password flags for URL to be created
	Host	Text	Host name for URL to be created
	Port	Longint	Port for URL to be created
	Path	Text	Path for URL to be created
	Direct Parameter	Text	Direct Parameter for URL to be created
	Referenced Search Arguments Names	Pointer	Pointer to text array containing search arguments names for URL to be created
	Referenced Search Arguments Values	Pointer	Pointer to text array containing search arguments values for URL to be created
	URL	Text	Fully formatted, RFC compliant URL

The method ***URL_Make_URL*** will a properly formatted, RFC compliant URL for the values specified. See RFC 1738, or the method comments in the 4D Explorer window in 4D, for details on which URL values are available with different URL schemes.

Scheme is the scheme constant value for the URL to be created. Valid scheme constant values are listed in the URL Schemes constants group included with the BASh component.

Username is the user name value, if any, for the URL to be created.

Password is the password value, if any, for the URL to be created.

UnPw Flags is a flags value to indicate whether a username and possibly password is to be forced for inclusion with an URL even if the individual values are empty. The following tables lists the valid values, and their meaning, for this parameter:

Value	Meaning
0	do not force either value
1	force include password only
2	force include both username and password

Host is the host name value, if any, for the URL to be created.

Port is the port value, if any, for the URL to be created.

Path is the path value, if any, for the URL to be created. Note, the path value does not have a leading slash in most cases; a single slash in the URL is a delimiter and is not part of the *Path* value.

Direct Parameter is the direct parameter value, if any, for the URL to be created.

Referenced Search Arguments Names is a pointer to a text array containing search arguments names, if any, for the URL to be created. This value can be set to NULL to skip.

Referenced Search Arguments Values is a pointer to a text array containing search arguments values, if any, for the URL to be created. This value can be set to NULL to skip.

URL is the fully formatted, RFC compliant URL created from the specified values.

Note: the method ***URL_Make_URL*** was added in BASh v1.5.7.

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
	Referenced Variable	Pointer	Referenced variable to get the name of
	Variable Name	Text	Actual name of *Referenced Variable*

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

	Parameter	Type	Description
	Reference Value	Pointer	Any form of a referenced (pointer)
	nil Match State	Longint	Indicator for whether *Reference Value* is a NULL pointer

The method ***VAR_qi_Null_Pointer*** will check the pointer passed and see if it is a NULL pointer. This method is equivalent to the native 4th Dimension command Nil.

Reference Value is a pointer to a variable.

nil Match State is an indicator for whether *Reference Value* is a NULL pointer. If *Reference Value* is a NULL pointer, *nil Match State* will be set to one (1); if *Reference Value* is not a NULL pointer, *nil Match State* will be set to zero (0).

WORD Module

The WORD module provides operations which can be performed on four byte word data types. These differ often from 4D native longint data types in the actual meanings of the encoded values.

For instance, one common type of four byte word value is the unsigned longint. Where a native 4D longint value has of values from -2 ^ 31 to (2 ^ 31) -1, an unsigned longint has a range of 0 to 2 ^ 32.

Different words require that operation be performed upon them differently. The basic routines available in the WORD module provide a small basis for operating on words of different types. As well, basic bit operations have been included in the WORD module to increase the functionality which is not already included natively in 4D's native bit operations.

Note: the WORD module was added in BASh v1.5.4.

WORD_Add_Unsigned

WORD_Add_Unsigned *(First Addend ; Second Addend) => Word Sum*

WORD_Add_Unsigned
(
-> *First Addend :* Longint
-> *Second Addend :* Longint
)
=> *Word Sum :* Longint

	Parameter	Type	Description
	First Addend	Longint	First addend for unsigned longint addition
	Second Addend	Longint	Second addend for unsigned longint addition
	Word Sum	Longint	Unsigned longint sum of *First Addend* and *Second Addend*

The method ***WORD_Add_Unsigned*** will perform an unsigned longint addition upon two supplied unsigned longint values. The unsigned longint sum will be returned.

First Addend and *Second Addend* are the two unsigned longint values which are to be added.

Word Sum is the unsigned longint sum of *First Addend* and *Second Addend*.

Note: the method ***WORD_Add_Unsigned*** was added in BASh v1.5.4.

WORD_Clear_Bits

WORD_Clear_Bits *(Word { ; Bit to Clear { ; ... } }) => Word Result*

WORD_Clear_Bits
(
-> *Word :* Longint
{ -> *Bit to Clear :* Longint }
)
=> *Word Result :* Longint

	Parameter	Type	Description
	Word	Longint	Word value to clear bits within
	Bit to Clear	Longint	Zero, one, or more bit numbers to clear
	Word Result	Longint	Reuslting word with specified bits cleared

The method ***WORD_Clear_Bits*** will clear one or more numbered bits within a longint and return the resulting longint.

Word is the longint value to have one or more bits cleared within.

Bit to Clear is the bit number to clear within *Word*. Multiple bit numbers can be passed to this method. Valid bit numbers are between zero (0) and thirty-one (31).

Word Result is the longint value of *Word* with the specified bits cleared.

Note: the method WORD_Clear_Bits was added in BASh v1.5.5.

WORD_Get_Bit_Range

WORD_Get_Bit_Range *(Word ; Start Extract Bit Number ; End Extract Bit Number ; Start Result Bit Number) => Word Result*

WORD_Get_Bit_Range
(
-> *Word :* Longint
-> *Start Extract Bit Number :* Longint
-> *End Extract Bit Number :* Longint
-> *Start Result Bit Number :* Longint
)
=> *Word Result :* Longint

	Parameter	Type	Description
	Word	Longint	Word value to extract bit range from
	Start Extract Bit Number	Longint	Bit number in supplied word to begin extraction from
	End Extract Bit Number	Longint	Bit number in supplied word to end extraction from
	Start Result Bit Number	Longint	Bit number in resulting word to place extraction into
	Word Results	Longint	Resulting word for extracted bit range place in specified location

The method ***WORD_Get_Bit_Range*** will extract a range of bit values from a supplied longint and return it in a particular location within a longint.

Word is the longint value to extract a bit range from.

Start Extract Bit Number is the starting bit number to extract from *Word*. Valid bit numbers are between zero (0) and thirty-one (31).

End Extract Bit Number is the end bit number to extract from *Word*. Valid bit numbers are between zero (0) and thirty-one (31).

Start Result Bit Number is the starting bit number within the result to place the extracted bits into. This allows for the result of the extracted bits to automatically be shifted in the result for the convenience of the developer. Valid bit numbers are between zero (0) and thirty-one (31).

Word Result is the longint value of the extracted bits located at the bit location within the value specified in the method call.

Note: the method ***WORD_Get_Bit_Range*** was added in BASh v1.5.5.

WORD_Rotate_Left

WORD_Rotate_Left *(Word Value ; Left Shift Count) => Rotated Word*

WORD_Rotate_Left
(
-> *Word Value :* Longint
-> *Left Shift Count :* Longint
)
=> *Rotated Word :* Longint

	Parameter	Type	Description
	Word Value	Longint	Word value to operate on
	Left Shift Count	Longint	Bit count to rotate left
	Rotated Word	Longint	Resulting word value rotated left *Left Shift Count*

The method ***WORD_Rotate_Left*** will do a left rotating bit shift on a specified word value for a specified number of bits.

Rotated bits from the left end of a word will be attached as new bits on the right end of the word, maintaining the 32 bit integrity of the word value.

Word Value is the four byte word value which is to be left shifted rotationally.

Left Shift Count is the number of bits *Word Value* will be rotationally shifted left.

Rotated Word is the resulting *Word Value* after is has been rotationally shifted left *Left Shift Count* .

Note: the method ***WORD_Rotate_Left*** was added in BASh v1.5.4.

WORD_Rotate_Left_Range

WORD_Rotate_Left_Range *(Word Value ; Left Shift Count ; Start Bit ; End Bit)* => *Rotated Word*

WORD_Rotate_Left_Range
(
-> *Word Value :* Longint
-> *Left Shift Count :* Longint
-> *Start Bit :* Longint
-> *End Bit :* Longint
)
=> *Rotated Word :* Longint

	Parameter	Type	Description
	Word Value	Longint	Word value to operate on
	Left Shift Count	Longint	Bit count to rotate left
	Start Bit	Longint	Starting bit position for rotation
	End Bit	Longint	Ending bit position for rotation
	Rotated Word	Longint	Resulting word value rotated left *Left Shift Count*

The method ***WORD_Rotate_Left_Range*** will do a left rotating bit shift on a specified word value for a specified number of bits, within a bit range. Rotated bits from the left end of the range will be attached as new bits on the right end of the range, maintaining the 32 bit integrity of the word value.

Word Value is the four byte word value which is to be left shifted rotationally.

Left Shift Count is the number of bits *Word Value* will be rotationally shifted left.

Start Bit is the starting bit position for rotation. *Start Bit* is included in the range to be rotated.

End Bit is the ending bit position for rotation. *End Bit* is included in the range to be rotated. 4D's bits are numbered 0-31, right (least significant) to left (most significant), and *End Bit* should be higher numerically than *Start Bit*.

Rotated Word is the resulting *Word Value* after is has been rotationally shifted left *Left Shift Count* .

Note: the method ***WORD_Rotate_Left_Range*** was added in BASh v1.7.0.

WORD_Set_Bits

WORD_Set_Bits *(Word { ; Bit to Set { ; ... } }) => Word Result*

WORD_Set_Bits
(
-> *Word :* Longint
{ -> *Bit to Set :* Longint }
)
=> *Word Result :* Longint

	Parameter	Type	Description
	Word	Longint	Word value to set bits within
	Bit to Set	Longint	Zero, one, or more bit numbers to set
	Word Result	Longint	Reuslting word with specified bits set

The method ***WORD_Set_Bits*** will set one or more numbered bits within a longint and return the resulting longint.

Word is the longint value to have one or more bits set within.

Bit to Set is the bit number to set within *Word* . Multiple bit numbers can be passed to this method. Valid bit numbers are between zero (0) and thirty-one (31).

Word Result is the longint value of *Word* with the specified bits set.

Note: the method ***WORD_Set_Bits*** was added in BASh v1.5.5.

XML Module

The XML module in BASh provides a simple means to parse and store parsed XML data. The routines in the XML module provide a means to easily parse XML data into a format convenient and efficient to access in 4D. This storage format is called an XML Construct, and routines in the XML module then allow easy access to the data within an XML Construct. As with many areas of 4D components, the code in the XML module benefits significantly in performance when the 4D application is compiled.

Note: the XML module was initially added in BASh v1.8.0.

XML Construct Structure

An XML Construct has a very specific structure in 4D. This structure is expected and handled internally by the routines in the XML module.

An XML Construct handle in 4D is basically a pointer to a pointer array. This pointer array has five (5) elements in it, each element being a pointer to an array of values (hence the term “handle”). These arrays referenced in the XML Construct handle are a well formed stack used to store the XML document in an easily referenced format in 4D. The arrays that form each XML Construct stack are dynamically retrieved, managed, and returned from the DSS module by the XML module.

The arrays in an XML Construct handle consist of the following items:

Array Type	Array Title
Text	Tag Names
Text	Tag Values
Longint	Tag Value Lengths
Longint	Indent Levels
Text	Tag Properties NVPs

Each element in the XML Construct handle arrays correspond sequentially to each opening XML tag in the parsed XML document; for each

opening XML tag in the parsed XML document, a new element is added to the end of the XML Construct stack.

The Tag Names array contains the tag name for each opening tag in the XML document.

The Tag Values array contains the value containing directly between the opening and closing tag within the XML document. The value will include any and all bytes between the opening and closing XML tag, including tabs, carriage returns, and linefeeds.

The Tag Value Lengths array contains the number of bytes in the Tag Values array for each element. This is used primarily to determine whether a value exists and is empty or whether a value merely does not exist. If a tag value has no children tag and has no value, the length will be set to negative one (-1). For all tags that have children, there should be no value so the length will be set to zero (0).

The Indent Levels array contains a tag hierarchy level. Tag levels are counting from one and go up for each success, child opening tag. For each closing tag, the indent level decreases by one. With the indent level values, the hierarchy of the XML data is maintained in the XML Construct.

The Tag Properties NVPs contains a packed, tab and carriage return delimited text value for all XML tag properties in XML opening tags. Using the routines in the NVP module of BASh provides a simple mechanism for parsing and retrieving this data.

XML Construct Examples

For the first example, we will consider the following simple XML document:

```
<RatingServiceSelectionRequest xml:lang="en-US"><CR>
        <tab><Request><CR>
        <tab>   <tab><CustomerContext><CR>
        <tab>   <tab>   <tab>Rating and Service<CR>
        <tab>   <tab></CustomerContext><CR>
        <tab></Request>
</RatingServiceSelectionRequest><CR>
```

Please recognize that the tab and CR indicators in the above example have been inserted to help with understanding exactly how the XML document is parsed by the XML module.

When this document is parsed, the resulting XML Construct would have values equivalent to the following array and element values listing:

Tag Names	Tag Values	Value Len.	Indent	Properties
RatingServiceSelectionRequest		0	1	xml:lang<tab >en-US<CR>
Request		0	2	
CustomerContext	<CR><tab><tab ><tab>Rating and Ser-vice<CR><tab><tab>	25	3	

Notice in particular the value for the CustomerContext tag. It contains all of the leading and trailing tabs and carriage returns in the value. When retrieving a value from the XML Construct, this must be a consideration. Using the methods in the STR module provide a simple means to strip these characters assuming they are not wanted in the actual value.

The following code is a good example of loading an XML document from a document, retrieving an XML Construct handle, parsing the XML document, and then cleaning up all of the objects used:

```
C_POINTER($pHandle)
C_POINTER($pBLOB)
C_LONGINT($iError)
  `get temp BLOB for loading XML document
$pBLOB:=DSS_Get_Variable_by_Type (Is BLOB )
  `get XML Construct handle for parsing XML
$pHandle:=XML_Create_Handle
  `load document from disk
$iError:=BLOB_Document_to_BLOB ($pBLOB;"MacHD:my.xml")
  `parse XML document into XML Construct
XML_Parse_to_Arrays ($pBLOB;$pHandle)
  `return BLOB to free memory
DSS_Return_Variable ($pBLOB)
  `do whatever code here to work with XML
  `return XML handle to free memory
XML_Return_Handle ($pHandle)
```

XML_Count_Children

XML_Count_Children *(XML Construct Handle ; Parent Index) => Children Count*

XML_Count_Children
(
-> *XML Construct Handle :* Pointer
-> *Parent Index :* Longint
)
=> *Children Count :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Parent Index	Longint	Index into XML Construct element containing parent XML tag
	Children Count	Longint	Number of direct children counted

The method ***XML_Count_Children*** will count the number of direct children of a specified XML tag in a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Parent Index is the index into *XML Construct Handle* of the element to count all direct children of.

Children Count is a longint value returned containing the number of direct children counted.

Note: the method ***XML_Count_Children*** was added in BASh v1.8.2.

XML_Count_Descendants

XML_Count_Descendants *(XML Construct Handle ; Parent Index) => Descendants Count*

XML_Count_Descendants

(

-> *XML Construct Handle :* Pointer

-> *Parent Index :* Longint

)

=> *Descendants Count :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Parent Index	Longint	Index into XML Construct element containing parent XML tag
	Descendants Count	Longint	Number of descendants counted

The method ***XML_Count_Descendants*** will count the number of descendants of a specified XML tag in a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Parent Index is the index into *XML Construct Handle* of the element to count all descendants of.

Descendants Count is a longint value returned containing the number of descendants counted.

Note: the method ***XML_Count_Descendants*** was added in BASh v1.8.2.

XML_Count_Siblings

XML_Count_Siblings *(XML Construct Handle ; Sibling Index) => Siblings Count*

XML_Count_Siblings

(

-> *XML Construct Handle :* Pointer

-> *Sibling Index :* Longint

)

=> *Siblings Count :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Sibling Index	Longint	Index into XML Construct element containing sibling XML tag
	Siblings Count	Longint	Number of siblings counted

The method ***XML_Count_Siblings*** will count the number of siblings of a specified XML tag in a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Sibling Index is the index into *XML Construct Handle* of the element to count all siblings of.

Siblings Count is a longint value returned containing the number of siblings counted.

Note: the method ***XML_Count_Siblings*** was added in BASh v1.8.2.

XML_Create_Handle

XML_Create_Handle => *XML Construct Handle*

XML_Create_Handle
=> *XML Construct Handle :* Pointer

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to array of pointer with elements pointing to XML Construct arrays

The method ***XML_Create_Handle*** will return a valid XML Construct handle. This handle references XML Construct arrays that are empty (have no elements).

Note: make certain every handle retrieved with this method is eventually returned using the method ***XML_Return_Handle***.

XML Construct Handle is a pointer to an array of pointers. This array of pointers contains five elements, each element being a pointer to one of the arrays that comprise the XML Construct arrays.

Note: the method ***XML_Create_Handle*** was added in BASh v1.8.0.

XML_Get_Ancestors_Indices

XML_Get_Ancestors_Indices *(XML Construct Handle ; Child Index ; Ancestors Indices)*

XML_Get_Ancestors_Indices
(
-> *XML Construct Handle :* Pointer
-> *Child Index :* Longint
-> *Ancestors Indices :* Pointer
)

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Child Index	Longint	Index into XML Construct element containing child XML tag
	Ancestors Indices	Longint	Pointer to longint array to contain indices of all ancestors of specified child tag

The method ***XML_Get_Ancestors_Indices*** will return the indices of all direct ancestors of a specified XML tag in a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Child Index is the index into *XML Construct Handle* of the element to return all ancestors for.

Ancestors Indices is a pointer to a longint array to contain the indices of all of the direct ancestors of *Child Index* within *XML Construct Handle*. The array is begun with the immediate parent tag index and proceeds to progressively further ancestors as the elements in the array increase. The top (zero) level of *XML Construct Handle* is not returned as a valid ancestor from this routine.

Note: the method ***XML_Get_Ancestors_Indices*** was added in BASh v1.8.2.

XML_Get_Descendent_Last_Index

XML_Get_Descendent_Last_Index *(XML Construct Handle ; Parent Index) => Last Descendant Index*

XML_Get_Descendent_Last_Index
(
-> *XML Construct Handle :* Pointer
-> *Parent Index :* Longint
)
=> *Last Descendant Index :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Parent Index	Longint	Index into XML Construct element containing parent XML tag
	Last Descendant Index	Longint	Index into XML Construct of last descendant beneath specified *Parent Index*

The method ***XML_Get_Descendent_Last_Index*** will return the index of the last descendant in a specified XML Construct of a specified parent index.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Parent Index is the index into *XML Construct Handle* of the element to find the last descendant for.

Last Descendant Index is the index of the last descendant in *XML Construct Handle* of the tag in the element specified by *Parent Index*. This value will be set to negative one (-1) if an error of any kind occurs during the scan of *XML Construct Handle*.

Note: the method ***XML_Get_Descendent_Last_Index*** was added in BASh v1.8.0.

XML_Get_Indent_by_Index

XML_Get_Indent_by_Index *(XML Construct Handle ; Tag Index) => Indent Level*

XML_Get_Indent_by_Index
(
-> *XML Construct Handle :* Pointer
-> *Tag Index :* Longint
)
=> *Indent Level :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Tag Index	Longint	Index into XML Construct element containing XML tag to work from
	Indent Level	Longint	Indent level of specified tag in specified XML Construct

The method ***XML_Get_Indent_by_Index*** will return the indent level of a specified XML tag by index within a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Tag Index is the index into *XML Construct Handle* of the element to work from.

Indent Level is a longint value returned containing the indent level of the specified tag in the specified XML Construct. This value will be set to zero (0) for the version and meta information of the XML Construct. This value will be set to

negative one (-1) if the specified XML tag index is out of range for the specified XML Construct.

Note: the method ***XML_Get_Indent_by_Index*** was added in BASh v1.8.2.

XML_Get_Index_by_Name

XML_Get_Index_by_Name *(XML Construct Handle ; XML Tag Name ; Begin Index ; End Index) => Found Index*

XML_Get_Index_by_Name

(

-> *XML Construct Handle :* Pointer
-> *XML Tag Name :* Text
-> *Begin Index :* Longint
-> *End Index :* Longint

)

=> *Found Index :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to valid XML Construct
	XML Tag Name	Text	XML tag name to scan for
	Begin Index	Longint	First index in XML Construct to begin scanning for specified tag name
	End Index	Longint	Last index in XML Construct to scan for specified tag name
	Found Index	Longint	Index of first XML Construct entry matching specified parameters

The method ***XML_Get_Index_by_Name*** will return the first index in a specified XML Construct containing an XML tag matching a specified tag name. The range can be restricted to a specified range of elements within the XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

XML Tag Name is the name of the XML tag to scan for in the specified XML Construct.

Begin Index is the index of the first element in the specified XML Construct to begin scanning for the specified XML tag name. To scan from the beginning of the XML Construct, a value of zero (0) can be passed for this parameter.

End Index is the index of the last element in the specified XML Construct to scan for the specified XML tag name. To scan to the end of the XML Construct, a value of MAXLONG can be passed for this parameter.

Found Index is the index of the first element in the specified XML Construct within the range of elements specified containing an XML tag matching the specified XML tag name. If no matching element is found, this value will be set to negative one (-1).

Note: the method ***XML_Get_Index_by_Name*** was added in BASh v1.8.0.

XML_Get_Index_Parent_by_Index

XML_Get_Index_Parent_by_Index *(XML Construct Handle ; Tag Index) => Parent Level*

XML_Get_Index_Parent_by_Index
(
-> *XML Construct Handle :* Pointer
-> *Tag Index :* Longint
)
=> *Parent Level :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Tag Index	Longint	Index into XML Construct element containing XML tag to work from
	Parent Level	Longint	Index of parent of specified tag

The method ***XML_Get_Index_Parent_by_Index*** will return the index of the parent of a specified XML tag by index within a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Tag Index is the index into *XML Construct Handle* of the element to work from.

Parent Index is a longint value returned containing the index of the parent XML tag of *Tag Index* within *XML Construct Handle*. This value will be set to negative one (-1) if the parent is not found or if *Tag Index* is out of range for *XML Construct Handle*.

Note: the method ***XML_Get_Index_Parent_by_Index*** was added in BASh v1.8.2.

XML_Get_Indices_by_NameIL

XML_Get_Indices_by_NameIL *(XML Construct Handle ; XML Tag Name ; Indent Level ; Begin Index ; End Index ; Referenced Found Indices) => Found Indices Count*

XML_Get_Indices_by_NameIL

(

-> *XML Construct Handle :* Pointer
-> *XML Tag Name :* Text
-> *Indent Level :* Longint
-> *Begin Index :* Longint
-> *End Index :* Longint
-> *Referenced Found Indices :* Pointer

)

=> *Found Indices Count :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to valid XML Construct
	XML Tag Name	Text	XML tag name to scan for
	Indent Level	Longint	Indent level to restrict results to
	Begin Index	Longint	First index in XML Construct to begin scanning for specified tag name
	End Index	Longint	Last index in XML Construct to scan for specified tag name

	Parameter	Type	Description
	Referenced Found Indices	Pointer	Pointer to longint array to hold indices of all found elements matching scan criteria
	Found Indices Count	Longint	Count of number of elements found matching scan criteria

The method ***XML_Get_Indices_by_NameIL*** will scan and return the element indices within a specified valid XML Construct matching a specified XML tag name at a specified indent level. The range can be restricted to a specified range of elements within the XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

XML Tag Name is the name of the XML tag to scan for in the specified XML Construct.

Indent Level is the indent level value to restrict the scan to returning valid results for.

Begin Index is the index of the first element in the specified XML Construct to begin scanning for the specified XML tag name at the specified indent level. To scan from the beginning of the XML Construct, a value of zero (0) can be passed for this parameter.

End Index is the index of the last element in the specified XML Construct to scan for the specified XML tag name at the specified indent level. To scan to the end of the XML Construct, a value of MAXLONG can be passed for this parameter.

Referenced Found Indices is a pointer to a longint array to contain the indices of all elements found matching the scan criteria.

Found Indices Count is the number of elements found to match the scan criteria. This is the same as the number of elements in *Referenced Found Indices* when a call to this method is completed.

Note: the method ***XML_Get_Indices_by_NameIL*** was added in BASh v1.8.0.

XML_Get_Lineage_by_Index

XML_Get_Lineage_by_Index *(XML Construct Handle ; Tag Index ; Delimiter ASCII Value) => Lineage*

XML_Get_Lineage_by_Index
(
-> *XML Construct Handle :* Pointer
-> *Tag Index :* Longint
-> *Delimiter ASCII Value :* Longint
)
=> *Lineage :* Text

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Tag Index	Longint	Index into XML Construct element containing XML tag to work from
	Delimiter ASCII Value	Longint	ASCII value of delimiter to use in returned lineage text
	Lineage	Text	Tag names of all direct parents of XML tag in XML Construct

The method ***XML_Get_Lineage_by_Index*** will return a delimited list of the names of all tags of the direct parents of a specified XML tag by index in a specified XML Construct. The parent's tag names are given in order from the closest to further direct lineage.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Tag Index is the index into *XML Construct Handle* of the element to work from.

Delimiter ASCII Value is a longint value containing the ASCII value to use as the delimiter between parent tag names in the returned *Lineage* value.

Lineage is a text value set to the delimited names of direct parents of the specified XML tag within the specified XML

Construct. The lineage is returned from closest to furthest direct parentage.

Note: the method ***XML_Get_Lineage_by_Index*** was added in BASh v1.8.2.

XML_Get_Name_by_Index

XML_Get_Name_by_Index *(XML Construct Handle ; Tag Index) => Tag Name*

XML_Get_Name_by_Index
(
-> *XML Construct Handle :* Pointer
-> *Tag Index :* Longint
)
=> *Tag Name :* Text

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Tag Index	Longint	Index into XML Construct element containing XML tag to work from
	Tag Name	Text	Text of XML tag

The method ***XML_Get_Name_by_Index*** will return the XML tag name of a specified XML tag by index within a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Tag Index is the index into *XML Construct Handle* of the element to work from.

Tag Name is a text value returned containing the XML tag name of *Tag Index* within *XML Construct Handle*. This will be set to an empty text value if the specified XML tag index is invalid or out of range for the specified XML Construct.

Note: the method ***XML_Get_Name_by_Index*** was added in BASh v1.8.2.

XML_Get_Properties_by_Index

XML_Get_Properties_by_Index *(XML Construct Handle ; Tag Index ; Referenced Properties Names ; Referenced Properties Values)*

XML_Get_Properties_by_Index

(

-> *XML Construct Handle :* Pointer
-> *Tag Index :* Longint
-> *Referenced Properties Names :* Pointer
-> *Referenced Properties Values :* Pointer

)

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Tag Index	Longint	Index into XML Construct element containing XML tag to work from
	Referenced Properties Names	Pointer	Referenced text array to contain all properties names
	Referenced Properties Value	Pointer	Referenced text array to contain all properties values

The method ***XML_Get_Properties_by_Index*** will return all properties names and values for a specified XML tag by index within a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Tag Index is the index into *XML Construct Handle* of the element to work from.

Referenced Properties Names is a pointer to a text array to contain the properties name of *Tag Index* within *XML Construct Handle*. The elements in this returned array will match by index the properties values returned in *Referenced Properties Values*.

Referenced Properties Values is a pointer to a text array to contain the properties values of *Tag Index* within *XML Construct Handle*. The elements in this returned array will match

by index the properties names returned in *Referenced Properties Names*.

Note: the method ***XML_Get_Properties_by_Index*** was added in BASh v1.8.2.

XML_Get_ValueLength_by_Index

XML_Get_ValueLength_by_Index *(XML Construct Handle ; Tag Index) => Value Length*

XML_Get_ValueLength_by_Index
(
-> *XML Construct Handle :* Pointer
-> *Tag Index :* Longint
)
=> *Value Length :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct
	Tag Index	Longint	Index into XML Construct element containing XML tag to work from
	Value Length	Longint	Number of bytes comprising value of specified index in specified XML Construct

The method ***XML_Get_ValueLength_by_Index*** will return the number of bytes of the value of a specified XML tag by index within a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Tag Index is the index into *XML Construct Handle* of the element to work from.

Value Length is a longint value returned containing the number of bytes of the value of *Tag Index* within *XML Construct Handle*. This will be set to negative one (-1) if the tag has no specified value (which is different than having an empty value). This will be set to negative two (-2) if the specified

index is invalid or out of range for the specified XML Construct.

Note: the method ***XML_Get_ValueLength_by_Index*** was added in BASh v1.8.2.

XML_Get_Value_by_Index

XML_Get_Value_by_Index *(XML Construct Handle ; XML Index) => XML Value*

XML_Get_Value_by_Index
(
-> *XML Construct Handle :* Pointer
-> *XML Index :* Longint
)
=> *XML Value :* Text

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to valid XML Construct
	XML Index	Longint	Index of element in XML Construct
	XML Value	Text	Value stored in specified index of XML Construct

The method ***XML_Get_Value_by_Index*** will return the value of a specified index in a specified XML Construct.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

XML Index is an index of an element in the specified XML Construct to retrieve the value of.

XML Value is the value stored in the specified index of the specified XML Construct.

Note: the method ***XML_Get_Value_by_Index*** was added in BASh v1.8.0.

XML_Parse_to_Arrays

XML_Parse_to_Arrays *(Referenced XML ; XML Construct Handle)*

XML_Parse_to_Arrays
(
-> *Referenced XML :* Pointer
-> *XML Construct Handle :* Pointer
)

	Parameter	Type	Description
	Referenced XML	Pointer	Pointer to BLOB containing raw XML to be parsed
	XML Construct Handle	Pointer	Pointer to valid XML Construct

The method ***XML_Parse_to_Arrays*** will parse raw XML and populate a specified XML Construct with the parsed values.

Referenced XML is a pointer to a BLOB containing the raw, well formatted XML to be parsed.

XML Construct Handle is a pointer to a valid XML Construct to store the parsed XML into. Any previously parsed XML in the specified XML Construct will be replaced with the newly parsed values.

Note: the method ***XML_Parse_to_Arrays*** was added in BASh v1.8.0.

XML_qi_Valid_Handle

XML_qi_Valid_Handle *(XML Construct Handle) => qi Valid XML Construct Handle*

XML_qi_Valid_Handle
(
-> *XML Construct Handle :* Pointer
)
=> *qi Valid XML Construct Handle :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct to check for validity
	qi Valid XML Construct Handle	Longint	Indicator for whether specified XML Construct is valid and well formed

The method ***XML_qi_Valid_Handle*** will return an indicator for whether a specified XML Construct is valid and well formed.

Validity checks will be done on the XML Construct handle array, target XML Construct arrays' types, and the well formed nature of the XML Construct arrays themselves.

XML Construct Handle is a pointer to an XML Construct to check for validity.

qi Valid XML Construct Handle is an indicator for the valid and well formed state of the specified XML Construct. If valid and well formed, this value will be set to one (1). If not valid or well formed, this value will be set to zero (0).

Note: the method ***XML_qi_Valid_Handle*** was added in BASh v1.8.0.

XML_qi_Value_Exists_by_Index

XML_qi_Value_Exists_by_Index *(XML Construct Handle ; Tag Index) => Value Exists Indicator*

XML_qi_Value_Exists_by_Index
(
-> *XML Construct Handle :* Pointer
-> *Tag Index :* Longint
)
=> *Value Exists Indicator :* Longint

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to XML Construct

	Parameter	Type	Description
	Tag Index	Longint	Index into XML Construct element containing XML tag to work from
	Value Exists Indicator	Longint	qi indicator for whether the specified index within the specified XML Construct has a value that exists

The method ***XML_qi_Value_Exists_by_Index*** will an indicator value for whether a value exists for a specified XML tag by index within a specified XML Construct.

Note: an empty value for a specified XML tag is different than a value that does not exist.

XML Construct Handle is a pointer to a valid, parsed XML Construct.

Tag Index is the index into *XML Construct Handle* of the element to work from.

Value Exists Indicator is a longint value returned to indicate whether the values exists for *Tag Index* within *XML Construct*. If a value exists this will be set to one (1); a value does not exist this value will be set to zero (0).

Note: the method ***XML_qi_Value_Exists_by_Index*** was added in BASh v1.8.2.

XML_Return_Handle

XML_Return_Handle *(XML Construct Handle)*

XML_Return_Handle

(

-> *XML Construct Handle :* Pointer

)

	Parameter	Type	Description
	XML Construct Handle	Pointer	Pointer to valid XML Construct to return to available pool

The method ***XML_Return_Handle*** will return and clean up after a specified XML Construct. Once an XML Construct has been returned, it should no longer be used in your code.

XML Construct Handle is a pointer to a valid XML Construct previously retrieved with ***XML_Create_Handle*** that is to be returned to the pool of available XML Constructs.

Note: the method ***XML_Return_Handle*** was added in BASh v1.8.0.

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 v1.8.2

released Monday, March 10th, 2003

Changes:

Added conditional check in method ***BLOB_Replace_by_Length_z*** to allow for a source BLOB that is empty to be passed and used without error.

Corrected bug in ***CODEC_Encode_QuotedPrintable_z*** in which bytes encoded in the last couple of bytes of the maximum line length could prevent a soft break from being inserted; correction involves using a soft break at any point past 70 bytes in a line.

Forced ***CODEC_Encode_QuotedPrintable_z*** to encode all periods (ASCII 46) to avoid problems with lines in email beginning with a period.

Corrected bug in ***DTS_Subtract_Normalize*** in which resulting month rollings of a single month past a year boundary could result in the wrong DTS result.

Added ISO and HTML Entity encoding/decoding methods.

Changed quoted-printable encoding/decoding to use BASh routines for ISO-8859-1 translation as native routines in 4D are incorrect.

Added resource loading methods for 'bYt#' and 'rSr#' resource types; currently, 'rSr#' can only reference 'bYt#', 'STR#', and 'TXT#' methods.

Modified method ***CODEC_Decode_Base64_z*** to skip extraneous carriage returns that have no matching linefeeds in the bytes being decoded; also changed code that removes trailing line ending to handle CR line ending as well as CRLF.

Modified method ***CODEC_Decode_Base64_z*** to skip extraneous trailing bytes if forming a proper grouping of four bytes with padding.

Removed initial calculation of decoded BLOB size from method ***CODEC_Decode_Base64_z*** and changed it to just trim extra bytes to the destination offset of the last bytes set; this accounts for skipping invalid carriage returns, linefeeds, and trailing bytes in encoded BLOB.

Fixed bug in ***CODEC_Encode_Base64_z*** where space was not being allocated for final CRLF line ending; when the last line was full length, there was not room in the BLOB for the final CRLF, resulting in a range check error.

Corrected bug in ***NVP_Parse_to_Arrays*** in which empty lines could cause an out of range error.

Corrected misnamed IP variable in PROS module for handling stack sizes.

Cleaned out all declared local variables not being used in methods.

Added 'cfrg' resource to all affix documents for compatibility with 4D v70x.

Added distribution of 4D v70x compatible versions of component and plugins.

Corrected methods ***ENV_Get_4DApplication_FoldPath*** and ***ENV_Get_4DApplication_FullPath*** to handle changes made in native commands under 4D v70x on any MacOS environment (packages, basically).

Modified ***ENV_Get_4DXAPPL_FolderPath*** to handle new location for 4DX folder within MacOS package under 4D v70x.

Added IB module to component, including following methods:
  protected:
    IB_Append_Variable_by_Key
    IB_Clear_Value_by_Key
    IB_Compact_IB
    IB_Get_Key_Free
    IB_Get_One
    IB_Get_Value_by_Key
    IB_Return_One
    IB_Set_Key_Free
    IB_Set_Value_by_Key
  public:
    IB_ERROR

Added protected methods:
  ARR_Insert_Elements_f_Array
  ARR_Insert_Element_Binary
  CODEC_Decode_HTML_Entities_x

CODEC_Decode_HTML_Entities_z
CODEC_Decode_ISO_x
CODEC_Decode_ISO_z
CODEC_Encode_HTML_Entities_x
CODEC_Encode_HTML_Entities_z
CODEC_Encode_ISO_x
CODEC_Encode_ISO_z
DTS_Convert_to_String_x
ENV_Get_4DAPPL_Pkg_FullPath
FILE_Get_FPaths_HFolders
RES_Load_bYt_List
RES_Load_rSr_List
RES_Load_rSr_List_Data
STR_qi_Valid_EmailAddress
STR_Unwrap_by_ASCII
XML_Count_Children
XML_Count_Descendants
XML_Count_Siblings
XML_Get_Ancestors_Indices
XML_Get_Indent_by_Index
XML_Get_Index_Parent_by_Index
XML_Get_Lineage_by_Index
XML_Get_Name_by_Index
XML_Get_Properties_by_Index
XML_Get_ValueLength_by_Index
XML_qi_Value_Exists_by_Index

Upgraded to 4D Pack v6.8.2 for use under 4D v68x.

BASh v1.8.1

released Friday, October 25th, 2002

Changes:

Corrected resource ID conflicts within platforms of BASh Affix documents; made certain all affix documents coincide with the new plugin ID for BASh (29600).

Made Affix document on Mac under 4D v67x a true plugin stub.

Corrected issue with bug in 4D v68x on Windows in which resource names can cause conflicts.

Changed method ***CODEC_Decode_Hex_x*** to just call ***CODEC_Decode_Hex_z***.

Corrected bug in ***CODEC_Decode_Hex_z*** in which zero entry doublets were being decoded incorrectly.

Added the protected methods:
CODEC_Convert_ShiftJIS_to_JIS_x
CODEC_Convert_ShiftJIS_to_JIS_z

BASh v1.8.0

released Thursday, September 12th, 2002

Changes:

Finished full documentation for ***RES_Load_fMap*** method in manual.

Corrected bug in ***BLOB_Count_Occurrences_of_ASCII*** in which the first byte was not being within the specified offset range was not be checked.

Corrected bug in ***BLOB_Count_Occurrences_of_ASCII*** in which the ending offset was not being handled properly to limit the byte range parameters for the searching.

Added optional directory symbol parameter to the protected method ***FILE_FullPath_To_FileName*** [thanks to Richard Babillis].

Modified method ***FILE_Force_to_FullPath*** to assume that if parameter for path to force is not a relative path then if checks whether there are any directory symbols within the value; if no directory symbols are within the value, it assumes the value is a document name and will then prepend the supplied default directory path.

Changed all DSS routines for storage of DSS variables in use to not only indicate whether particular DSS variables are in use but to also store which process number originally locked each variable in use.

Corrected bugs in method ***DTS_Subtract_Normalize*** wherein not all normalization on dates were being done properly.

Added the constants group **TYPE, Array 2D, Variable Types** for the typing of two dimensional arrays.

Added support for 2D arrays throughout DSS module; all 2D array types in the DSS module have a total of 50 variables for use throughout the DSS system.

Added support for **TYPE, Array 2D, Variable Types** constants for use within the methods ***TYPE_Get_Unary_by_Array*** and ***TYPE_qi_Array***.

Cleaned out usage of all calls to ***TYPE_qi_Array*** to be consistent with full support for 2D array types throughout BASh.

Cleaned out usage of all calls to **Type** 4D command in component to support fully new 2D array types within BASh.

Rewrote following methods to provide significant speed improvements both interpreted and compiled:

CODEC_Decode_UUEncode_z
CODEC_Encode_Base64_z
CODEC_Encode_UUEncode_z
CODEC_Decode_Base64_z

Added FMAP module to component, including following methods:

protected:

FMAP_Clear_Stack_s
FMAP_Get_DCrCode_by_MType_s
FMAP_Get_DMType_by_MType_s
FMAP_Get_DMType_by_Title_s
FMAP_Get_DSuffix_by_MType_s
FMAP_Get_DTitles_All_s
FMAP_Get_MIME_by_MType_s
FMAP_Get_MIME_by_Suffix_s
FMAP_Get_Title_by_MType_s
FMAP_Get_Title_by_Suffix_s
FMAP_Load_Stack_s

public:

FMAP_ERROR

Added many 'fMap' resources to the Affix BASh documents for use by developers, if desired.

Added 'TMPL' resource for 'fMap' resource in Affix BASh documents.

Increased option types for the method ***DTS_Get_Range*** to include values for current period as well as last period.

Added new fields to 'HTbl' resource template; this includes 9) Table CSS Class Name, 52), Row CSS Class Name, and 104) Cell CSS Class Name; all new fields in 'TMPL' are of type PSTR.

Added 'HTbl' template resource to Affix BASh documents.

Added support for CSS Class Name value to methods ***RES_Parse_HTbl_Table***, ***RES_Parse_HTbl_Rows***, and ***RES_Parse_HTbl_Cells***.

Added 'iXrf' template resource to Affix BASh documents.

Updated method ***SEQ_Make_TFCode_by_Ref*** to check that referenced parameter is actually a table or field.

Modified the method ***RES_Open_DataFile*** to always call 4D for the resource fork file reference of the data file as 4D Backup has a bug in it that it closes the resource fork of the data file when a backup is run and reopens it once the backup is done, effectively changing the resource fork file reference under these circumstances.

Modified method ***NULL_Set_Variables*** to now accept referenced arrays for clearing.

Added parameter to ***SEQ_Get_One_NewOnly*** to allow for not incrementing stored next sequence number, thereby allowing next new value to merely be read without actually using it.

Fixed bug in ***CODEC_Decode_Base64_z*** in which the bit shifting with bytes to convert in the source could cause a range check error if the MSB on any byte was set.

Modifed ***CODEC_Decode_Base64_x*** to just call ***CODEC_Decode_Base64_z*** for efficiency of maintaining the code base.

Corrected bug in ***CODEC_Decode_Base64_z*** to no longer generate range checking error when the source BLOB ends with a carriage return/line feed pair.

Added **IDLE** call to tight decode loop within ***CODEC_Decode_Base_64_z*** which is hit for every single carriage return/line feed pair (each line, in other words) encountered in the source BLOB.

Corrected bug in stored procedure for SEQ module in which the job sent flag was not being cleared just before the job complete flag was being set.

Added optional parameter to ***INIT_BASh*** for setting the document full path to use in the SEQ module.

Added code through SEQ module to support custom setting for document full path to use for storage within the SEQ module.

Modified ***QUIT_BASh*** routine to close resource fork of document being used in SEQ module.

Modified ***CODEC_Encode_Base64_x*** and ***CODEC_Encode_Base64_z*** methods to take an optional parameter for whether linebreaks are to be included in the encoded value; modified all methods that call these to now include the new, optional parameter.

Changed ***CODEC_Encode_Base64_x*** method to instead just call ***CODEC_Encode_Base64_z*** method for encoding.

Changed ***CODEC_Decode_Base64_x*** method to instead just call ***CODEC_Decode_Base64_z*** method for decoding.

Made certain all methods in BASh component are invisible.

Cleaned up all type checking in methods ***ARR_Convert_Type_to_Longint***, ***ARR_Find_in_Range***, ***PROS_Get_ProcessNames_s***, and ***RES_Load_LoCK***.

Added RW module to component, including following methods:
  protected:
    RW_Set_ReadOnly_All
    RW_Wait_for_RecordLoad_Unlocked

Added XML module to component, including following methods:
  protected:
    XML_Create_Handle
    XML_Get_Descendant_Last_Index
    XML_Get_Index_by_Name
    XML_Get_Indices_by_NameIL
    XML_Get_Value_by_Index
    XML_Parse_to_Arrays
    XML_qi_Valid_Handle
    XML_Return_Handle

Added the protected methods:
    ARR_Set_Values *
    BLOB_Append_BLOB
    BLOB_Parse_to_Arrays_x
    BLOB_Replace_by_Length_z
    CONV_Hex6RGB_to_Longint_x
    CONV_Longint_to_Hex6RGB_x
    DATE_Get_MonthDay_x
    DATE_Get_MonthName_Full
    DATE_Get_MonthName_Short
    DSS_Count_Available_by_Type **
    DSS_Count_Locked_by_Type **

DSS_Get_Locking_ProcessID **
DTS_Convert_f_GSM
DTS_Convert_to_GSM
DTS_Get_MonthDay_x
ENV_Get_DataFile_RF_FullPath
ENV_qi_BASh_INITed
FILE_Get_Extension
FILE_Get_Platform_for_Path ***
FILE_Rename_Document
FILE_Translate_Path_by_OS ***
NVP_Set_Value_by_Name_s ****
PROS_Wait_End_Complete
RES_Load_4DK_List
RES_Load_iXrf
RES_Load_TXT_List
SEQ_Set_Next
STR_Compare_Bytes
TYPE_Compare_UnaryTypes_Loose
TYPE_Get_Type
TYPE_Get_Type_Loose
TYPE_qi_Array_1D
TYPE_qi_Array_2D
TYPE_qi_Unary

*** Thanks to Justin Leavens for the ideas for these methods.**
**** Thanks to Larry Klein for the ideas for these methods.**
***** Thanks to Richard Babillis for the ideas for these methods.**
****** Thanks to Jerome Pupier for the ideas for these methods.**

Added the public methods:
DATE_ERROR
TYPE_ERROR

Changed name of method ***RES_Set_Resource_Properites*** to ***RES_Set_Resource_Properties*** so that it is now spelled correctly.

Updated 4D Pack to v6.8.1.

BASh v1.7.0

released 20020222

Changes:

Fixed bug in the method ***CRYPT_Decrypt_TEA_DWRD*** which could fail with too short parameters or local variables if the last byte was a possible double byte value under double byte OSes; this was not completely done correctly in the previous release. [Special thanks to Ken Ishimoto for this.]

Fixed a bug in the method ***FILE_Force_to_FullPath*** in which passing a path to a director for the default path would make the method fail.

Corrected bug in ***DTS_Convert_f_RFC*** in which many properly RFC formatted date-time stamp values were not being converted at all to DTS values.

Corrected bugs with the native implementation of the **Date** function, changing all uses to instead use the BASh method ***DATE_Make_from_Values***, in the methods ***DTS_Get_Date_Time*** and ***DTS_Get_Range***.

Added an optional "bytes to search" parameter to the method ***BLOB_Find_Between_x***.

Fixed a bug in the methods ***ARR_Coerce_to_Text*** and ***ARR_Coerce_from_Text*** in which 2D arrays passed as source for unary typing would cause a crash after an appropriate _ERROR call for type incompatibility; now, the routines just failed to operate when passed a 2D array as source for unary typing (still after an appropriate _ERROR call for type incompatibility).

Added support for new 4D plugin architecture under 4D v6.8.x in which plugins can be located next to the current application in any environment.

Changed MD5 encoding algorithm slightly to be much faster; this involved not using pointers where it wasn't necessary.

Changed the name of the Mac affix document from "Affix_BASh" to "Affix_BASh.4DX".

Changed the name of the Mac 4D Pack plugin document from "4dp_v671" to "4D_Pack_v671.4DX".

Changed the name of the Win 4D Pack plugin document from "4dp_v671" to "4D_Pack_v671".

Created a carbon affix document entitled "Affix_BASh.4CX".

Included a carbon 4D Pack plugin entitled "4D_Pack_v676b01.4DX".

Added the protected methods:

ARR_Remove_Duplicates
DATE_Get_Age [Special thanks to Ken Ishimoto for this.]
DTS_Get_Age [Special thanks to Ken Ishimoto for this.]
ENV_Get_4DXAPPL_FolderPath
FILE_Delete_Folder
FILE_Find_Plugin_Hard_Long
FILE_Force_to_RelativePath
FILE_Get_Folder_List
RES_Get_Free_ResourceID

Implemented MD4 encoding algorithm; this algorithm is similar to MD5, but faster at the expense of less security; see RFC 1320 for details; added protected methods:

CODEC_Encode_MD4_x
CODEC_Encode_MD4_z

Implemented DES encryption algorithm; added protected methods:

BLOB_Set_Bytes
CRYPT_Decrypt_DES_x
CRYPT_Decrypt_DES_z
CRYPT_Encrypt_DES_x
CRYPT_Encrypt_DES_z
WORD_Rotate_Left_Range

BASh v1.6.2

released 20020104

Changes:

Changed the name of the Windows affix document from "Afx_BASh" to "Affix_BASh".

Modified the method ***PROS_New_Process*** to make calls to **RESUME PROCESS** and **SHOW PROCESS** for processes which already exist. Before, it only made a call to **BRING TO FRONT** which would not resume processes which were previously paused and not make visible processes which were previously hidden.

Fixed bugs in ***BLOB_Find_Text_Case*** and ***BLOB_Find_Text_NonCase*** methods where searches were not resetting completely after finding a partial match. For example, if you were looking for "aaabc" in "aaaaabc", it would not be found.

Rewrote methods ***BLOB_Replace_Text_Case*** and ***BLOB_Replace_Text_NonCase*** to fix a couple of bugs and speed them up.

Corrected the comments in the 4D Explorer comments window for the method ***BLOB_Count_Occurrences_of_ASCII***.

Corrected the method ***DTS_Get_Date*** to now support all possible date formats. [Special thanks to Ken Ishimoto for this.]

Added new CONV routines:

CONV_ArrLong_to_LSTC
CONV_LSTC_to_ArrLong

Added new DATE routines:

DATE_Get_MonthEndDay_f_YM
DATE_Get_MonthEnd_f_Date

Added new DSS routine:

DSS_Return_Variables

Added new ENV routines:

ENV_Get_4DApplication_FoldPath

ENV_Get_4DSerial_CompanyName
ENV_Get_4DSerial_Key
ENV_Get_4DSerial_UserName

Added new RES routines:

RES_Load_SEQn
RES_Load_SEQr
RES_qi_Resource_Exists
RES_Set_SEQn
RES_Set_SEQr

Added the complete SEQ module, which includes the methods:

SEQ_Get_Description
SEQ_Get_Many
SEQ_Get_Many_NewOnly
SEQ_Get_One
SEQ_Get_One_NewOnly
SEQ_Make_TFCode_by_Ref
SEQ_SEQ_Recycle_Many
SEQ_Recycle_One
SEQ_Set_Description

Added the complete QUIT module, which includes the method:

QUIT_BASh

Having a call to ***INIT_BASh*** is now required not only early in the **On Startup** database method but also early in the **On Server Startup** database method.

Adding a call to ***QUIT_BASh*** is now required late in the **On Exit** and **On Server Shutdown** database methods.

Fixed a bug in the method **STR_Get_CharPosition_by_ASCII** in which a zero ASCII value could not be found. [Special thanks to Ken Ishimoto for this.]

Fixed a bug in the method **CRYPT_Decrypt_TEA_x** in which double byte language systems would fail on some serials as the **Position** command would not always find NULL bytes. [Special thanks to Ken Ishimoto for this.]

Fixed bugs in the methods **CRYPT_Decrypt_TEA_DWRD**, **CRYPT_Decrypt_TEA_x**, **CRYPT_Encrypt_TEA_DWRD**,

CRYPT_Encrypt_TEA_x which could fail with too short parameters or local variables if the last byte was a possible double byte value under double byte OSes. [Special thanks to Ken Ishimoto for this.]

BASh v1.6.1

released 20011109

Changes:

Fixed a bug in the method ***PROS_Ending_Process*** where it was calling the wrong method.

Fixed a bug in the PROS module in which it could try to read the control method before it was set in the PROS stack.

Fixed a bug in the method ***DTS_Convert_f_RFC***; after the RFC date was parsed the number of words found should be 6 not 7.

Fixed a bug in the method ***BLOB_Find_Byte*** wherein searching a portion of a BLOB would sometimes return an incorrect result. Fixing this involved completely rewriting the routine to add a good speed boost to it, as well.

Fixed a bug in the methods ***CONV_Longint_to_Type*** and ***CONV_Type_to_Longint*** where conversions from or to negative numbers wouldn't work. There may have also been problems with these methods with double-byte languages.

Completely rewrote ***BLOB_Find_Text_Case*** and ***BLOB_Find_Text_NonCase*** to avoid copying blobs when doing partial searches. Also, changed the way the searching loop worked, which doubled the speed of searching in interpreted mode. This rewrite also fixed a couple of bugs related to searching backwards in BLOBs when using these two methods.

Modified the method ***NULL_Set_Variables*** to generate an error for each parameter reference that is NULL.

The method ***RES_Open_4DApplication*** has been changed to only open the 4D application's resource fork when running on MacOS. when running on Windows, it would cause the 4D application to crash when quitting if the application's resource fork had been opened. This affects directly the operation of the method ***ENV_Set_FlushKey*** under Windows, also.

Added new BLOB routines:

BLOB_Count_Occurrences_of_ASCII
BLOB_Find_Byte_by_ASCII_Range

Added new CODEC routines:

CODEC_Decode_MIME_Words_x
CODEC_Decode_QuotedPrintable_x
CODEC_Decode_QuotedPrintable_z
CODEC_Encode_MIME_Words_x
CODEC_Encode_QuotedPrintable_x
CODEC_Encode_QuotedPrintable_z

Added new PROS routines:

PROS_Set_CloseBoxMethod_s

Added new STR routines:

STR_Get_CharPosition
STR_Get_Position_by_ASCII_Range

BASh v1.6.0

released 20010904

Changes:

Fixed a bug in the method ***ARR_Find_First_Binary*** in which an array with only one element would never have a found index.

Fixed a bug in the method ***ARR_Find_First_Binary*** where if the value was equal to the first or last element, it would return not found; also, fixed a problem with odd sized arrays by using Round on a calculation.

Fixed a bug in the method ***SERNO_Decode_SerialNumber*** where length of serial number wasn't being checked prior to operating on it; if serial number was too long, it would cause array index error.

Made all of the methods in the BASh component invisible so they will not show up in the User environment or within any 4D native windows.

Rewrote the method ***WORD_Add_Unsigned*** to increase speed significantly; it now runs approximately five (5) times faster, both interpreted and compiled.

Removed all integers from the DSS module and mapped them to longints.

Fixed a bug in the method ***ENV_Get_Structure_DF_FullPath*** where when run under 4D Client would return an incorrect value; this had a trickling affect on many other routines that called this method, as well.

Fixed a bug in the method ***ENV_Get_4DXOS_FolderPath*** where when run under 4D Client would return an incorrect value; this had a trickling affect on many other routines that called this method, as well.

Fixed bitshifting bugs in relation to two byte languages in the following methods (major thanks to Ken Ishimoto for finding and correcting this problem for DSTi):

CODEC_Convert_to_Base64_Bytes
CODEC_Decode_Base64_x
CODEC_Decode_Hex_x
CODEC_Decode_Hex_z

CONV_Hex8_to_Text

Changed the methods ***RES_Load_TMPL***, ***RES_Parse_TMPL***, and ***RES_Make_TMPL_f_Arrays*** to read and write values directly as 4 byte integers instead of as text converted from/to integers.

Modified the methods ***BLOB_Find_Text_Case*** and ***BLOB_Find_Text_NonCase*** to do searches not only towards the ending but also towards the beginning.

Modified the method ***CRYPT_Decrypt_TEA_DWRD*** to use the new method ***CRYPT_Decrypt_TEA_DWRD_z*** to provide a compatible means for TEA to decrypt under double byte operating systems.

Fixed a bug in the method ***ENV_Get_Structure_RF_FullPath*** in which under 4D Client the path was incomplete.

Fixed a bug in the method ***FILE_Convert_to_ResFork_Windows*** in which the proper extension was not being appended when running under 4D Client.

Added the method ***CRYPT_Decrypt_TEA_DWRD_z*** to provide a compatible means for TEA to decrypt under double byte operating systems.

Added methods to the ARR module:

ARR_Find_in_Range

Added methods to the BLOB module:

BLOB_Remove_Bytes_by_ASCII
BLOB_Replace_Text_by_Offset
BLOB_Strip_Between_by_ASCII
BLOB_Strip_Between_x

Added methods to the CRYPT module:

CRYPT_Convert_TEA_x_to_z

Added methods to the FILE module:

FILE_Force_to_FullPath
FILE_Resolve_Alias

Added the INT module to the BASh component. Added methods to the INT module:

INT_PROS_Error_Callback
INT_ERR_Clear_All
INT_ERR_Pop_Method
INT_ERR_Push_Method

Added methods to the PROS module:

PROS_Beginning_Process
PROS_Ending_Process
PROS_ERROR
PROS_Get_CloseBoxMethod_s
PROS_Get_ControlMethod_s
PROS_Get_ProcessID_s
PROS_Get_ProcessIDs_s
PROS_Get_ProcessName_s
PROS_Get_ProcessNames_s
PROS_Get_State
PROS_Get_UniqueID_s
PROS_Get_Visible
PROS_Get_WindowName_s
PROS_New_Process
PROS_Set_WindowName_s

Added methods to the STR module:

STR_Strip_Between
STR_Strip_Between_by_ASCII

BASh v1.5.9

released 20010706

Changes:

Added methods to the ARR module:

ARR_Find_First_Binary
ARR_Find_Last_Same

Added methods to the BLOB module:

BLOB_Change_Case
BLOB_XOR_Bytes

Added methods to the CODEC module:

CODEC_Decode_UUEncode_File
CODEC_Decode_UUEncode_x
CODEC_Decode_UUEncode_z
CODEC_Encode_CRAMMD5_z
CODEC_Encode_KeyedMD5_z
CODEC_Encode_UUEncode_File
CODEC_Encode_UUEncode_x
CODEC_Encode_UUEncode_z

Set all _ERROR methods in all modules to contain a TRACE command to assist in debugging.

BASh v1.5.8

released 20010625

Changes:

Added method to the BLOB module:

BLOB_Find_Byte

Added methods to the CODEC module:

CODEC_Decode_Hex_x
CODEC_Decode_Hex_z
CODEC_Encode_Hex_x
CODEC_Encode_Hex_z

Added methods to the DATE module:

DATE_Get_ISO8601_GMT
DATE_Get_Month_f_ShortName

Added method to the DTS module:

DTS_Convert_f_RFC

Fixed a bug in the method ***ENV_Get_BASh_RF_FullPath*** in which erroneously the system's 4DX directory would be scanned for the Affix document even if the Affix document was already found next to the application in the 4DX directory.

BASh v1.5.7

released 20010604

Changes:

Added the PTEXT module to the BASh component. Added methods to the PTEXT module:

PTEXT_Set_Values_ax
PTEXT_Set_Values_az
PTEXT_Set_Values_x
PTEXT_Set_Values_z

Added the URL module to the BASh component. Added methods to the URL module:

URL_Extract_DirectParam
URL_Extract_Host
URL_Extract_Password
URL_Extract_Path
URL_Extract_Path_w_Params
URL_Extract_Port
URL_Extract_Scheme_i
URL_Extract_Scheme_x
URL_Extract_SearchParams
URL_Extract_Username
URL_Get_Scheme_i
URL_Get_Scheme_x
URL_Make_URL

Add URL schemes constants to the Affix document for use in the URL module.

Added URL scheme 'STR#' resource to the Affix document for use in translating URL schemes from constants to their textual equivalents.

Added method to the BLOB module:

BLOB_Find_Text_Case
BLOB_Replace_Text_Case
BLOB_Replace_Text_NonCase

Added method to the STR module:

STR_Position_NonAlpha
STR_Position_NonAlphaNumeric
STR_Position_NonNumeric

BASh v1.5.6

released 20010516

Changes:

Added method to the RES module:

RES_Get_Fork_Size

Fixed a bug in the initialization code for the component wherein the searching for the Affix BASh document could fail if there was a document in the 4DX folder that did not have a resource fork.

Fixed a bug in the method ***CRYPT_Get_TEA_Keys_f_Text*** in which some of the variable initialization was performed incorrectly (thanks JM).

BASh v1.5.5

released 20010425

Changes:

Added methods to the ARR module:

ARR_Coerce_to_Text
ARR_Coerce_from_Text

Added method to the BLOB module:

BLOB_Replace_Byte

Added methods to the CODEC module:

CODEC_Decode_URL_z
CODEC_Encode_URL_z

Added the CRYPT module to the BASh component. Added methods to the CRYPT module:

CRYPT_Decrypt_TEA_x
CRYPT_Decrypt_TEA_z
CRYPT_Encrypt_TEA_x
CRYPT_Encrypt_TEA_z
CRYPT_Get_TEA_Keys_f_Text

Added methods to the RES module:

RES_Load_HTbl
RES_Parse_HTbl_Cells
RES_Parse_HTbl_Rows
RES_Parse_HTbl_Table
RES_Set_HTbl_Resource

Added the SERNO module to the BASh component. Added methods to the SERNO module:

SERNO_Create_SerialNumber
SERNO_Decode_SerialNumber
SERNO_Get_Beta
SERNO_Get_MajorVersion
SERNO_Get_MinorVersion
SERNO_Get_Month
SERNO_Get_ProductCode

SERNO_Get_Users
SERNO_Get_Year

Added methods to the WORD module:

WORD_Clear_Bits
WORD_Get_Bit_Range
WORD_Set_Bits

Upgraded 4D Pack from v6.7.0 to v6.7.1.

For future compatibility and avoidance of particular bugs under Windows, the Affix documents on Windows have been renamed to no longer begin with "Affix_" and instead begin with "afx_".

Fixed a bug in ***DTS_Get_Current*** in which, under rare circumstances, the date could be one date and the time from the following day. 4D needs to provide a way to atomically get both the current date and the current time. [Thanks to TK of IRGi for the bug report and bug fix.]

Fixed a crashing bug when compiled under Windows in which calls to some methods with a referenced pointer array would fail. This is the old 4D bug involving assigning a referenced NULL pointer to a referenced pointer variable value (4D SA has still not fixed this bug since 4D v3.5.x). The bug fix is to just not make the assignment when compiled under Windows in the offending methods.

Fixed bugs within the **CONV_IP_to_Longint** and ***CONV_Longint_to_IP*** methods in which the values which would convert to negative 4D longint values were being handled incorrectly.

Fixed a bug in applying the padding to the encoding object within the method ***CODEC_Encode_MD5_z*** .

Fixed a bug in the method ***ENV_Get_Application_Name_Short*** in which the resource fork of the structure was not being opened or referenced correctly.

Increased DSS variable space from 100 to 200 objects each for text, longint, BLOB, text array, and longint array. Remaining data types were left at 100 objects of each type.

BASh v1.5.4

released 20010212

Changes:

Added methods to the ARR module:

ARR_Set_Sizes

Added methods to the BLOB module:

BLOB_BLOB_to_Document
BLOB_Document_to_BLOB
BLOB_Find_Between_Folding_x
BLOB_Find_Between_x
BLOB_Find_Text_NonCase
BLOB_Finds_Between_Folding_x

Added methods to the CODEC module:

CODEC_Encode_MD5_x
CODEC_Encode_MD5_z
CODEC_ERROR

Added the DATE module to the BASh component. Added methods to the DATE module:

DATE_Get_RFC_GMT
DATE_Make_from_Values

Added methods to the DTS module:

DTS_Add_Normalize
DTS_Subtract
DTS_Subtract_Normalize

Added methods to the ENV module:

ENV_ERROR

Added the INIT module to the BASh component. Added methods to the INIT module:

INIT_BASh

Added the PROS module to the BASh component. Added methods to the PROS module:

PROS_Delay_Current
PROS_Get_Type
PROS_Get_UniqueID

Added methods to the SEM module:

SEM_ERROR
SEM_Test_One

Added the WORD module to the BASh component. Added methods to the WORD module:

WORD_Add_Unsigned
WORD_Rotate_Left

Made the 4D Pack plugin a requirement to use the BASh component; this is not enforced in the code, though.

Fixed a bug in the method ***SEM_Set_One*** in which the determination of whether the method actually set a semaphore which was previously set in the same process was not returning the correct result.

Fixed a bug in the initialization of the RES module in which previously initializing the ENV module would make the RES module think that it too had been initialized.

Fixed bugs in the ***SEM_Set_One*** method wherein the wrong result was returned if the semaphore being set was already set in the current process.

Fixed a bug in ***SEM_Set_One*** in which the delay between check ticks being non-zero could result in an infinite loop waiting for the semaphore, as the timeout checking was reversed incorrectly.

Rebuilt the initialization routines for the ENV module.

Rebuilt the initialization routines for the CODEC module.

Rebuilt the initialization routines for the DSS module.

Changed the way the DSS stack was locked and unlocked in nested calls within the same process.

Rebuilt the initialization routines for the NULL module.

Rebuilt the initialization routines for the RES module.

Fixed a bug wherein kernel processes were not delaying.

Fixed a bug in the method ***CODEC_Decode_URL_x*** in which an encoded plus byte "+" would be decoded to a space if the "plus to space" option was selected.

BASh v1.5.3

released 20001204

Changes:

Fixed a bug in the method ***FILE_Convert_to_ResFork_Windows*** in which the resource fork of a 4D data file was not calculated correctly.

Created the Affix BASh document for handling all resources and data structures essential to the BASh components functionality. Added support for the Affix BASh document throughout the BASh component. Including full long hard file name checking for the Affix BASh document, to be located in the 4DX directory, across all relevant BASh component methods.

Moved all needed resources (only one so far) within the BASh component to instead be distributed in the Affix BASh document, meant to be placed in the 4DX folder. This will facilitate distribution of custom resource types and resources which are not supported directly within components by 4D.

Modified the method ***CODEC_Encode_URL_x*** to instead pull its needed resource from the Affix BASh file instead of from the resource fork of the structure.

Updated the documentation for the method ***CODEC_Encode_URL_x***. The details for the storage location of the "Allowable ASCII Values for URLs" resource changed from the structure document to the new Affix BASh document.

Added another index entry into the Affix BASh document for URL encoded within the CODEC module. This value was for the number 4 (value 52) which was previously missing in the "Allowable ASCII Values for URLs" resource.

Added documentation for Hard File Names within the FILE module. This includes details on what hard file names are, the differences between long and short hard file names, and the storage location for hard file names. This new section is entitled Hard File Names.

Updated the installation and updating instructures to account for the Affix BASh document. This section of the developer's manual now includes a section describing in details the Affix BASh document. This new section is entitled Affix BASh Document.

Added the BLOB module to the BASh component. This included the following methods:

BLOB_Append_Text
BLOB_Clear

Added methods to the ENV module:

ENV_Get_4DXOS_FolderPath
ENV_Get_4DX_FolderPath
ENV_Get_BASh_HardName_Long
ENV_Get_BASh_HardName_Short
ENV_Get_BASh_RF_FullPath
ENV_Get_DataFile_FolderPath
ENV_Get_DirectorySymbol_by_OS
ENV_Get_Structure_DF_FullPath
ENV_Get_Structure_FolderPath
ENV_q_Macintosh
ENV_Set_FlushKey

Added methods to the FILE module:

FILE_Create_Document
FILE_Create_Folder
FILE_ERROR
FILE_Find_FileName_Hard_Long
FILE_Find_FileName_Hard_Short
FILE_Get_Document_List
FILE_Get_Path_Parent
FILE_qi_Document_Exists
FILE_qi_Folder_Exists

Added method to the RES module:

RES_Open_BASh

Added method to the STR module:

STR_Clean_EmailUsername

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, 'WPGr' 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_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.

Errors

The listing of error codes and conditions is obviously a continuously updated process. With practically every new version of BASh, the error codes and conditions can and do change. Though it has been a long time coming, we are now documenting much of the error conditions that can occur in BASh.

Different methods in the BASh component can generate errors when used incorrectly or when used under the wrong circumstances. When an error condition is encountered, the methods within the BASh component will call the applicable “_ERROR” method. The “_ERROR” methods are documented above. The first parameter sent to an “_ERROR” method is the error code identifying the unique error condition that occurred.

With future versions of the BASh component (and other components to come), the management and handling of error conditions will become much better documented, much clearer to understand, and easier to handle in your code. For now, though, reading this manual thoroughly is the best single source of understanding for the error conditions that can arise in using the BASh component.

Error Codes

When an "_ERROR" method is called in the BASh component, the first parameter is always an error code. This error code is always a 7 digit integer indicating the unique error condition which was detected in the code.

These 7 digit numbers actually consist of two pieces for uniquely identifying the error code and condition. The first five digits is an internal code for the module which an error occured within. The last two digits identifies the unique error condition from within a module that has been detected.

The following is a quick listing of all of the error codes, grouped by the module which the error codes are from. Each error code includes the textual description of the error condition.

RES: 29000

01 Document path parameter is empty
02 Failed to open resource fork
03 Variable is not of type picture
04 Referenced variable is not an array of type text
05 Referenced variable is not an array of type boolean
06 Referenced variable is not an array of type longint
07 Referenced variable is not an array of type string
08 Failed to read resource
09 Referenced variable is not of type picture
10 Resource failed to be loaded or may be empty
11 Referenced variable is of type string array or text array
12 Referenced variable is not of type BLOB
13 Referenced array has no elements
14 Element counts of referenced array indicate array stack is not well formed
15 Path to resource fork file is empty
16 Path to resource fork file is empty
17 Failed to mark RES module as initialized
18 Resource improperly formatted
19 Referenced data structure does match size of 'rSr#' resource
20 Unsupported resource type element in 'rSr#' resource

ARR: 29004

01 Parameter is not of type pointer
02 Variable is not an array
03 Parameter for number of elements to add is less than zero
04 Parameter for number of elements to remove is less than zero
05 Parameter for number of elements to remove is more than the number of elements in the array
06 Parameter for new size of array is less than zero
07 Unknown condition found
08 Referenced variable does not match referenced array
09 Starting index is less than zero
10 Starting index is beyond the number of elements in the array
11 Ending index is less than starting index
12 Invalid parameter count
13 Referenced parameter is not an array

14 Referenced parameters not of matching types
15 Parameter value is out of range

SEQ: 29005

01 Unable to find specified TFCode
02 Referenced parameter is not of type array
03 Referenced parameter array is not of type longint
04 Invalid parameter value
05 Failed to lock Nexts stack
06 'SEQr' resource not found
07 Failed to load 'SEQr' resource
08 Failed to launch BASh_SEQ_n stored procedure
09 Failed to set new job SP semaphore
10 Stored procedure not found
11 Unknown action code received by stored procedure
12 Failed to lock access to stored procedure
13 Method should not be called from this application type
14 Wrong number of parameters in calling current method
15 Parameter is not of type pointer
16 Pointer is NULL
17 Failed to open specified resource fork
18 Specified document does not exist

PROS: 29006

01 Process number was not found in the process tracking arrays
02 Process number has no close box method stored in the tracking arrays
03 Dereferenced parameter is not an array
04 Dereferenced array is not of type longint
05 The current process is not in the process tracking arrays
06 There are too many parameters on the process parameter stack
07 Process parameter stack overload
08 Failed to start new process
09 Failed to lock PROS stack

CONV: 29008

01 TYPE is not a full length of four (4) bytes
02 Dereferenced parameter is not an array
03 Dereferenced array is not of type string
04 Dereferenced array is not of type longint
05 Dereferenced parameter is not of type blob
06 Parameter is not a list
07 Array has no elements
08 Arrays have different number of elements
09 Dereference is NULL
10 Dereferenced parameter is an array
11 Unknown condition found

NULL: 29011

01 Undefined type value
02 Offscreen area ID not found in storage arrays
03 Failed to mark NULL module as initialized

NVP: 29017
01 NVP names and values arrays have different element counts
02 No NVP values to pack

FILE: 29018
01 Document specified by full path already exists
02 Folder specified by full path already exists
03 Specified platform unknown
04 Windows drive designator out of range
05 Drive designator is empty

DSS: 29023
01 Failed to mark DSS module as initialized
02 Failed to lock DSS stack
03 Invalid variable type value
04 Referenced value failed to resolve
05 No variables matching type passed

SEM: 29030
01 Named semaphore is empty
02 Named semaphore is not currently set
03 Named semaphore was not set by the current process
04 Named semaphore failed to be cleared for unknown reason
05 Failed to lock SEM stack semaphore
06 BASh semaphore schema functions only with local semaphores

DATE: 29063
01 Month number out of range

ENV: 29064
01 Byte value out of range
02 4D application’s resource reference invalid

FMAP: 29065
01 Invalid scope value

TYPE: 29066
01 Failed to copy zero array
02 Parameter not of type pointer
03 Invalid parameter count

IB: 29078

01 Failed to retrieve index total count
02 IB not large enough to contain full index
03 Key out of range
04 Failed to pack referenced variable to BLOB
05 Failed to retrieve IB version
06 IB too small to retrieve index total count
07 Failed to retrieve new IB
08 Index total count out of range
09 Length offset for value out of range
10 Value offset out of range
11 IB too small ro retrieve version

CODEC: 29079

01 Parameter not of type pointer
02 Reference is NULL
03 Referenced parameter not of type pointer
04 BLOB size not even
05 Invalid decoding resource
06 Invalid encoding resource

Method Listing

The following is a listing of all of the methods within the BASh component (COMPILER methods are not included in this listing), including all private methods which are not directly available in the API when the BASh component is installed. Following each method is a list of the error codes which each method can generate.

ARR_Add_Elements_to_End
- 2900401
- 2900402
- 2900403

ARR_Add_Elements_to_Top
- 2900401
- 2900402
- 2900403

ARR_Clear
- 2900401
- 2900402

ARR_Coerce_from_Text

ARR_Coerce_to_Text

ARR_Convert_Longint_to_Text

ARR_Convert_Text_to_Longint

ARR_Convert_Type_to_Longint
- 2900802
- 2900803
- 2900804

ARR_Cycle_Stack

ARR_ERROR

ARR_Find_First_Binary

ARR_Find_in_Range
- 2900402
- 2900408
- 2900409
- 2900410
- 2900411

ARR_Find_Last_Same

ARR_Insert_Elements_f_Array

ARR_Insert_Element_Binary

ARR_Order_to_Match
- 2900407

ARR_Pack_to_Text

ARR_Populate_Text_Array
- 2900402

ARR_Remove_Duplicates

ARR_Remove_Elements_from_End
- 2900401
- 2900402
- 2900404
- 2900405

ARR_Remove_Elements_from_Top
- 2900401
- 2900402
- 2900404
- 2900405

ARR_Set_Size
- 2900401

        2900402
        2900406
        2900407
ARR_Set_Sizes
        2900401
        2900402
        2900406
        2900407
ARR_Set_Values
        2900401
        2900412
        2900413
        2900414
        2900415
BLOB_Append_BLOB
BLOB_Append_Text
BLOB_BLOB_to_Document
BLOB_Change_Case
BLOB_Clear
BLOB_Count_Occurrences_of_ASCII
BLOB_Document_to_BLOB
BLOB_Finds_Between_Folding_x
BLOB_Find_Between_Folding_x
BLOB_Find_Between_x
BLOB_Find_Byte
BLOB_Find_Byte_by_ASCII_Range
BLOB_Find_Text_Case
BLOB_Find_Text_NonCase
BLOB_Parse_to_Arrays_x
BLOB_Remove_Bytes_by_ASCII
BLOB_Replace_Byte
BLOB_Replace_by_Length_z
BLOB_Replace_Text_by_Offset
BLOB_Replace_Text_Case
BLOB_Replace_Text_NonCase
BLOB_Set_Bytes
BLOB_Strip_Between_by_ASCII
BLOB_Strip_Between_x
BLOB_XOR_Bytes
CODEC_Convert_from_Base64_Byte
CODEC_Convert_f_UUEncode_Byte
CODEC_Convert_ShiftJIS_to_JIS_x
CODEC_Convert_ShiftJIS_to_JIS_z
        2907901
        2907902
        2907903
        2907904
CODEC_Convert_to_Base64_Byte
CODEC_Convert_to_Base64_Bytes
CODEC_Convert_to_UUEncode_Byte
CODEC_Decode_Base64_x
CODEC_Decode_Base64_z
CODEC_Decode_CLE_x
CODEC_Decode_Hex_x
CODEC_Decode_Hex_z
CODEC_Decode_HTML_Entities_x
CODEC_Decode_HTML_Entities_z
        2907905
CODEC_Decode_ISO_x

CODEC_Decode_ISO_z
CODEC_Decode_MIME_Words_x
CODEC_Decode_QuotedPrintable_x
CODEC_Decode_QuotedPrintable_z
CODEC_Decode_URL_x
CODEC_Decode_URL_z
CODEC_Decode_UUEncode_File
CODEC_Decode_UUEncode_x
CODEC_Decode_UUEncode_z
CODEC_Do_MD4_Transformations
CODEC_Do_MD4_Transform_1
CODEC_Do_MD4_Transform_2
CODEC_Do_MD4_Transform_3
CODEC_Do_MD5_Transformations
CODEC_Do_MD5_Transform_1
CODEC_Do_MD5_Transform_2
CODEC_Do_MD5_Transform_3
CODEC_Do_MD5_Transform_4
CODEC_Encode_Base64_x
CODEC_Encode_Base64_z
CODEC_Encode_CLE_x
CODEC_Encode_CRAMMD5_z
CODEC_Encode_Hex_x
CODEC_Encode_Hex_z
CODEC_Encode_HTML_Entities_x
CODEC_Encode_HTML_Entities_z
	2907906
CODEC_Encode_ISO_x
CODEC_Encode_ISO_z
CODEC_Encode_KeyedMD5_z
CODEC_Encode_MD4_x
CODEC_Encode_MD4_z
CODEC_Encode_MD5_x
CODEC_Encode_MD5_z
CODEC_Encode_MIME_Words_x
CODEC_Encode_QuotedPrintable_x
CODEC_Encode_QuotedPrintable_z
CODEC_Encode_URL_x
CODEC_Encode_URL_z
CODEC_Encode_UUEncode_File
CODEC_Encode_UUEncode_x
CODEC_Encode_UUEncode_z
CODEC_ERROR
CODEC_INIT
CONV_ArrLong_to_LSTC
CONV_ASCII_to_Hex2
CONV_ASCII_to_PrintableText
CONV_Coerce_from_Text
	2900809
	2900810
	2900811
CONV_Coerce_to_Text
	2900809
	2900810
	2900811
CONV_ERROR
CONV_Hex6RGB_to_Longint_x
CONV_Hex8_to_Longint
CONV_Hex8_to_Text

CONV_IP_to_Longint
CONV_Longint_to_Hex6RGB_x
CONV_Longint_to_Hex8
CONV_Longint_to_IP
CONV_Longint_to_Type
CONV_LSTC_to_ArrLong
CONV_Text_to_Hex8
CONV_Text_to_Longint
CONV_Text_to_Real
CONV_Type_to_Longint
	2900801
CRYPT_Convert_TEA_x_to_z
CRYPT_Decrypt_DES_x
CRYPT_Decrypt_DES_z
CRYPT_Decrypt_TEA_DWRD
CRYPT_Decrypt_TEA_DWRD_z
CRYPT_Decrypt_TEA_x
CRYPT_Decrypt_TEA_z
CRYPT_Do_DES_Data_Cipher
CRYPT_Do_DES_Data_FP
CRYPT_Do_DES_Data_IP
CRYPT_Encrypt_DES_x
CRYPT_Encrypt_DES_z
CRYPT_Encrypt_TEA_DWRD
CRYPT_Encrypt_TEA_x
CRYPT_Encrypt_TEA_z
CRYPT_Get_DES_Keys
CRYPT_Get_DES_SBox
CRYPT_Get_TEA_Keys_f_Text
DATE_ERROR
DATE_Get_Age
DATE_Get_ISO8601_GMT
DATE_Get_MonthDay_x
DATE_Get_MonthEndDay_f_YM
DATE_Get_MonthEnd_f_Date
DATE_Get_MonthName_Full
	2906301
DATE_Get_MonthName_Short
	2906301
DATE_Get_Month_f_ShortName
DATE_Get_RFC_GMT
DATE_Make_from_Values
DSS_Count_Available_by_Type
	2902302
	2902303
DSS_Count_Locked_by_Type
	2902302
	2902303
DSS_ERROR
DSS_Get_Array_by_Unary_Type
	2902303
DSS_Get_Locking_ProcessID
	2900401
	2900403
	2900404
	2900405
DSS_Get_Unary_by_Array_Type
	2900403
DSS_Get_Variable_by_Type

2900403
2900405
2900406
2902302
DSS_INIT
DSS_INIT_Variables
DSS_Lock_Stack
2902302
DSS_Return_Variable
2900401
2900403
2900404
2900405
2902302
DSS_Return_Variables
DSS_Unlock_Stack
DTS_Add
DTS_Add_Normalize
DTS_Convert_f_GSM
DTS_Convert_f_RFC
DTS_Convert_to_GSM
DTS_Convert_to_String_x
DTS_Get_Age
DTS_Get_Current
DTS_Get_Date
DTS_Get_Date_Time
DTS_Get_Day
DTS_Get_Maximum
DTS_Get_Month
DTS_Get_MonthDay_x
DTS_Get_Range
DTS_Get_Time
DTS_Get_Year
DTS_Make_from_DateTime
DTS_Make_from_Values
DTS_Subtract
DTS_Subtract_Normalize
ENV_ERROR
ENV_Get_4DApplication_FoldPath
ENV_Get_4DApplication_FullPath
ENV_Get_4DSerial_All
ENV_Get_4DSerial_CompanyName
ENV_Get_4DSerial_Key
ENV_Get_4DSerial_UserName
ENV_Get_4DXAPPL_FolderPath
ENV_Get_4DXOS_FolderPath
ENV_Get_4DX_FolderPath
ENV_Get_Application_Name
ENV_Get_Application_Name_Short
ENV_Get_Application_Type
ENV_Get_BASh_HardName_Long
ENV_Get_BASh_HardName_Short
ENV_Get_BASh_RF_FullPath
ENV_Get_DataFile_FolderPath
ENV_Get_DataFile_FullPath
ENV_Get_DataFile_RF_FullPath
ENV_Get_DirectorySymbol
ENV_Get_DirectorySymbol_by_OS
ENV_Get_Platform

ENV_Get_Structure_DF_FullPath
ENV_Get_Structure_FolderPath
ENV_Get_Structure_RF_FileName
ENV_Get_Structure_RF_FullPath
ENV_INIT
ENV_qi_BASh_INITed
ENV_q_Macintosh
ENV_q_Windows
ENV_Set_FlushKey
    2906401
    2906402
FILE_Convert_to_ResFork_Windows
FILE_Create_Document
    2901801
FILE_Create_Folder
    2901802
FILE_Create_FullPath_Document
FILE_Create_FullPath_Folder
FILE_Delete_Folder
FILE_ERROR
FILE_Find_FileName_Hard_Long
FILE_Find_FileName_Hard_Short
FILE_Find_Plugin_Hard_Long
FILE_Force_to_FullPath
FILE_Force_to_RelativePath
FILE_FullPath_to_FileName
FILE_Get_Document_List
FILE_Get_Extension
FILE_Get_Folder_List
FILE_Get_FPaths_HFolders
FILE_Get_Path_Parent
FILE_Get_Platform_for_Path
FILE_Get_Volumes_Names
FILE_qi_Document_Exists
FILE_qi_Folder_Exists
FILE_qi_Volume_Exists
FILE_Rename_Document
FILE_Resolve_Alias
FILE_Translate_Path_by_OS
    2901803
    2901804
    2901805
FMAP_Clear_Stack_s
    2906501
FMAP_ERROR
FMAP_Get_DCrCode_by_MType_s
FMAP_Get_DMType_by_MType_s
FMAP_Get_DMType_by_Title_s
FMAP_Get_DSuffix_by_MType_s
FMAP_Get_DTitles_All_s
FMAP_Get_MIME_by_MType_s
FMAP_Get_MIME_by_Suffix_s
FMAP_Get_Title_by_MType_s
FMAP_Get_Title_by_Suffix_s
FMAP_INIT
FMAP_INIT_p
FMAP_Load_Stack_s
    2906501
IB_Append_Variable_by_Key

2907801
2907802
2907803
2907804
IB_Clear_Value_by_Key
2907801
2907802
2907803
IB_Compact_IB
2907801
2907802
2907805
IB_ERROR
IB_Get_Index_TotalCount
2907806
IB_Get_Key_Free
2907801
2907802
IB_Get_One
2907807
2907808
IB_Get_Value_by_Key
2907801
2907802
2907803
2907809
2907810
IB_Get_Version
2907811
IB_Return_One
IB_Set_Key_Free
2907801
2907802
2907803
IB_Set_Value_by_Key
2907801
2907802
2907803
INIT_BASh
INT_ERR_Clear_All
INT_ERR_INIT
INT_ERR_Pop_Method
INT_ERR_Push_Method
INT_PROS_Error_Callback
NULL_Dereference_by_Type
2901101
NULL_ERROR
NULL_Get_DTS
NULL_Get_Pointer
NULL_INIT
NULL_Set_Variables
2901102
NVP_ERROR
NVP_Extract_Value_by_Name_s
NVP_Extract_Value_by_Name_x
NVP_Pack_to_Text
2901701
2901702
NVP_Parse_to_Arrays

NVP_Set_Value_by_Name_s
PROS_Add_s
PROS_Beginning_Process
    2900609
PROS_Beginning_Process_p
PROS_Delay_Current
PROS_Delete_s
    2900605
PROS_Ending_Process
PROS_Ending_Process_p
    2900609
PROS_ERROR
PROS_Get_CloseBoxMethod_s
    2900601
    2900602
PROS_Get_ControlMethod_s
    2900601
    2900602
PROS_Get_ProcessIDs_s
    2900603
    2900604
PROS_Get_ProcessID_s
PROS_Get_ProcessNames_s
    2900603
    2900604
PROS_Get_ProcessName_s
PROS_Get_State
PROS_Get_Type
PROS_Get_UniqueID
PROS_Get_UniqueID_s
PROS_Get_Visible
PROS_Get_WindowName_s
PROS_Handler_c_p
PROS_INIT
PROS_Lock_Arrays
    2900609
PROS_New_Process
    2900609
PROS_Set_CloseBoxMethod_s
    2900609
PROS_Set_WindowName_s
    2900609
PROS_Unlock_Arrays
PROS_Wait_End_Complete
PROS_Wait_Launch_Complete_s
PTEXT_Set_Values_ax
PTEXT_Set_Values_az
PTEXT_Set_Values_x
PTEXT_Set_Values_z
QUIT_BASh
RES_Close
RES_Create_File
RES_Delete_Resource
RES_ERROR
RES_Get_Fork_Size
RES_Get_Free_ResourceID
RES_Get_Resource_List
    2900004
    2900006

RES_Get_TEXT_Resource
RES_INIT
RES_Load_4DK_List
	2900002
	2900008
	2900018
RES_Load_bYt_List
	2900002
	2900008
	2900018
RES_Load_cicn
	2900009
RES_Load_fMap
	2900010
RES_Load_HTbl
	2900008
	2900010
RES_Load_iXrf
	2900006
	2900010
RES_Load_LoCK
	2900010
	2900011
RES_Load_MBAR
	2900006
	2900010
RES_Load_MENU
	2900004
	2900005
	2900006
	2900008
RES_Load_MENV
	2900010
	2900011
RES_Load_PICT
	2900009
RES_Load_rSr_List
	2900002
	2900008
	2900018
RES_Load_rSr_List_Data
	2900019
	2900020
RES_Load_SEQn
	2900010
RES_Load_SEQr
	2900010
RES_Load_TMPL
	2900004
	2900006
	2900008
RES_Load_TXT_List
	2900002
	2900010
	2900018
RES_Load_WAGr
	2900010
RES_Load_WPGr
	2900010

RES_Make_TMPL_f_Arrays
	2900004
	2900007
	2900012
	2900013
	2900014
RES_Open
	2900002
	2900015
	2900016
RES_Open_4DApplication
RES_Open_BASh
RES_Open_DataFile
RES_Open_Structure
RES_Parse_HTbl_Cells
	2900008
	2900010
RES_Parse_HTbl_Rows
	2900008
	2900010
RES_Parse_HTbl_Table
	2900008
	2900010
RES_Parse_TMPL
	2900004
	2900007
	2900012
RES_qi_Resource_Exists
RES_Set_HTbl_Resource
	2900010
RES_Set_Resource_Name
RES_Set_Resource_Properties
RES_Set_SEQn
RES_Set_SEQr
RES_Set_TEXT_Resource
RW_Set_ReadOnly_All
RW_Wait_for_RecordLoad_Unlocked
SEM_Clear_One
	2903001
	2903002
	2903003
SEM_ERROR
SEM_Set_One
	2903001
SEM_Test_One
	2903001
SEQ_Add_Row_s
	2900505
SEQ_c
	2900511
SEQ_CleanUp_NewJob
	2900510
	2900513
SEQ_ERROR
SEQ_Get_Description
	2900501
	2900509
SEQ_Get_Many
	2900502

2900503
2900504
2900509
SEQ_Get_Many_NewOnly
2900502
2900503
2900504
2900505
2900509
SEQ_Get_NewJob_Results
2900510
2900513
SEQ_Get_One
2900505
2900506
2900507
2900509
SEQ_Get_One_NewOnly
2900505
2900509
SEQ_INIT
SEQ_Load_TFCodes_s
SEQ_Lock_Nexts_s
2900505
SEQ_Lock_SP
2900512
SEQ_Make_TFCode_by_Ref
2900514
2900515
2900516
2900517
SEQ_n
2900508
SEQ_Preload_Stacks_s
SEQ_QUIT
2900509
2900510
SEQ_Recycle_Many
2900502
2900503
2900509
SEQ_Recycle_One
2900501
2900505
2900509
SEQ_Send_NewJob_to_SP
2900510
2900513
SEQ_Set_Description
2900501
2900505
2900509
SEQ_Set_Next
2900501
2900505
2900509
SEQ_Set_Preferences
2900517
2900518

SEQ_Unlock_Nexts_s
SEQ_Unlock_SP
SEQ_Wait_NewJob_Results
    2900510
    2900513
SERNO_Create_SerialNumber
SERNO_Decode_Character
SERNO_Decode_SerialNumber
SERNO_Encode_Character
SERNO_Get_Beta
SERNO_Get_MajorVersion
SERNO_Get_MinorVersion
SERNO_Get_Month
SERNO_Get_ProductCode
SERNO_Get_Users
SERNO_Get_Year
STR_Clean_EmailAddress
STR_Clean_EmailUsername
STR_Compare_Bytes
STR_Concatenate_Text
STR_Count_Occurrences_of_ASCII
STR_Count_Occurrences_of_String
STR_Get_CharPosition
STR_Get_CharPosition_by_ASCII
STR_Get_Line_First
STR_Get_Position_by_ASCII_Range
STR_Pad_String
STR_Parse_to_Array_by_ASCII
STR_Parse_to_Array_by_Str
STR_Position_NonAlpha
STR_Position_NonAlphaNumeric
STR_Position_NonNumeric
STR_qi_Match_Contains_Filter
STR_qi_Match_Filter_NonCase
STR_qi_Match_Single_Filter
STR_qi_Valid_EmailAddress
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_Strip_Between
STR_Strip_Between_by_ASCII
STR_Unwrap_by_ASCII
STR_Wrap_in_DoubleQuotes
TIME_Add_Normalize
TIME_Get_Difference_Offset
TIME_Get_Hours
TIME_Get_Minutes
TIME_Get_Seconds
TIME_Get_Sum_Offset
TIME_Subtract_Normalize
TYPE_Compare_Dereferenced_Type
TYPE_Compare_UnaryTypes_Loose
TYPE_ERROR
TYPE_Get_Array_by_Unary
TYPE_Get_Type

2906601
2906602
2906603
TYPE_Get_Type_Loose
TYPE_Get_Unary_by_Array
TYPE_qi_Array
TYPE_qi_Array_1D
TYPE_qi_Array_2D
TYPE_qi_BLOB
TYPE_qi_Unary
URL_Extract_DirectParam
URL_Extract_Host
URL_Extract_Password
URL_Extract_Path
URL_Extract_Path_w_Params
URL_Extract_Port
URL_Extract_Scheme_i
URL_Extract_Scheme_x
URL_Extract_SearchParams
URL_Extract_Username
URL_Get_Scheme_i
URL_Get_Scheme_x
URL_INIT
URL_Make_URL
VAR_Get_Variable_Name
VAR_qi_Null_Pointer
WORD_Add_Unsigned
WORD_Clear_Bits
WORD_Get_Bit_Range
WORD_Rotate_Left
WORD_Rotate_Left_Range
WORD_Set_Bits
XML_Count_Children
XML_Count_Descendants
XML_Count_Siblings
XML_Create_Handle
XML_Get_Ancestors_Indices
XML_Get_Descendant_Last_Index
XML_Get_Indent_by_Index
XML_Get_Index_by_Name
XML_Get_Index_Parent_by_Index
XML_Get_Indices_by_NameIL
XML_Get_Lineage_by_Index
XML_Get_Name_by_Index
XML_Get_Properties_by_Index
XML_Get_Tag_Properties
XML_Get_ValueLength_by_Index
XML_Get_Value_by_Index
XML_Parse_to_Arrays
XML_Parse_to_Arrays_p
XML_qi_Valid_Handle
XML_qi_Value_Exists_by_Index
XML_Return_Handle