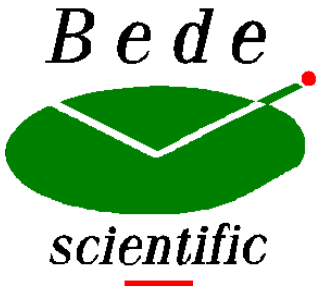
Bede Scientific Instruments



Software Office,
27 Sutton Street,
Durham,
DH1 4BW.

Title:	Bede Object Data System
Document Reference:	BODS/KERNDATSTUB /REF
Author(s):	S. Johnston
Circulation:	Programmers using object data system
Location:	\\Aslan\ \documents\allsorts\dataStub.doc

Date: 24 / 4 / 96

Version: 4.1

AMENDMENT RECORD

Date	Version	Comment	Author(s)	Approved by
14 /12/95	1.0	First draft for comment	S. Johnston	
10/1/95	2.0	Array elements and array recovery routines were incorrect	S. Johnston	
12/2/95	3.0	Mods after combination of class and data registries	S. Johnston	
9/4/96	4.0	Mods to allow for multiple instances.	S. Johnston	
24/4/96	4.1	32 bit version and file appending	S. Johnston	

1 Object Data system

In an effort to make the Bede Scientific Object Data system more portable between programming languages, the Project PANTHER Kernel datastub has been incorporated in the BD16Dobj and Bd32Dobj DLLs. This means that all programs access the object data system via successive calls to the same function `_ObjectDataStub`.

This function takes three parameters,

P0	(integer)	This should be zero, the object data system is the first DLL in the [16BIT_LIBS] or [32BIT_LIBS] sections of the kernel.ini file
P1	(integer)	A function ID number, specifies what action will be taken.
P2	(unsigned long)	An array of at least 7 elements. The data contained in each element depends on which function is being called.

This interface is identical to that used by the Project PANTHER kernel and so will provide a supported method of accessing the object data system.

The array (P2) must have at least 7 elements, P2[6] will contain the data registry handle after the call to the initialise function.

Throughout this document the individual elements of the data array (P2) are referred to as Data[0]…Data[6]. An array of four byte unsigned integers.

`Data[6]` must not be altered in any way and the data array must be passed to all functions.

2General commands.

Initialise

Description: This function initialises the data environment and must be called before the object data system can be used.

FunctionID = 0

Parameters:

Data[6] returns the registry handle.

Example:

```
unsigned long Data[7];
status = callout( 0, 0, (unsigned long)Data);
```

Close Down

Description: This function closes down the object data system and deletes all storage allocated.

FunctionID = 1

Example:

```
unsigned long Data[7];
status = callout( 0, 0, (unsigned long)Data);
```

Get Field Size

Description: Recovers information about the number of elements that may be stored within a given field. (The array size)

FunctionID = 28

Parameters:

Data[0]	(char *)	Name of the object.field
Data[1]	(long)	Returns the field size

Create object

Description: Creates an empty data object that may have fields added to it afterwards

FunctionID = 4

Parameter:

Data[0]	(char *)	Name to be given to the object

Delete Object

Description: Deletes an object from the registry and recovers all memory allocated to it. The class declaration on which the object is based is not affected.

FunctionID = 5

Parameters:

Data[0]	(char *)	Name of the object to delete.

Count Registered Objects

Description: Returns the number of data objects currently in the registry as Data[0]

FunctionID = 7

Parameters:

Data[0]	(long)	Returns the number of objects

Store Data Array

Description: This function is used to store data. It is used for all data types, and regardless of the number of elements to be stored. The data is stored within the field named. Any data stored previously within that field is destroyed. New data need not be of the same type as that previously stored within a field.

FunctionID = 8

Parameters:

Data[0]	(char *)	Object.Fieldname
Data[1]	(void *)	pointer to the data
Data[2]	(integer)	data type marker.
Data[3]	(unsigned long)	Number of data elements
Data[4]	(char *)	User defined comment string.

Store Data Element

Description: This function is used to store data. It is used for all data types, and regardless of the number of elements to be stored. The data is stored within the field named. Any data stored previously within that field is kept, except that within the array indices supplied. The data type must be the same as that already stored.

FunctionID = 9

Parameters:

Data[0]	(char *)	Object.Fieldname
Data[1]	(void *)	pointer to the data
Data[2]	(integer)	data type marker.
Data[3]	(unsigned long)	Number of data elements
Data[4]	(unsigned long)	Start position to replace from
Data[5]	(char *)	User defined comment string.

Get Data Array

Description: This function returns data to the calling routine. The user must allocate space and pass a pointer to this space for the object data system to copy the data for return. Entire arrays need not be returned.

FunctionID = 10

Parameters:

Data[0]	(char *)	Object.Fieldname
Data[1]	(void *)	pointer to the target data array
Data[2]	(integer)	data type marker.
Data[3]	(unsigned long)	Number of data elements
Data[4]	(unsigned long)	Starting index.

Store User Type

Description: This function is used to store a user defined structure.

FunctionID = 12

Parameters:

Data[0]	(char *)	Object.Fieldname
Data[1]	(void *)	pointer to the data
Data[2]	(unsigned long)	Size of the structure
Data[3]	(char *)	User defined comment string.

Get User Type

Description: This function is used to store a user defined structure.

FunctionID = 11

Parameters:

Data[0]	(char *)	Object.Fieldname
Data[1]	(void *)	pointer to the data

Lock Data Type

Description: This function locks the data type and array size for a given field. The data within may be changed.

FunctionID = 13

Parameters:

Data[0]	(char *)	Object.Fieldname

Free Data Type

Description: This function frees the data type and array size for a given field.

FunctionID = 14

Parameters:

Data[0]	(char *)	Object.Fieldname

Lock Data

Description: This function locks the data type and array size for a given field. The data within the field is made readonly.

FunctionID = 15

Parameters:

Data[0]	(char *)	Object.Fieldname

Free Data

Description: This function unlocks the data type and array size for a given field. The data within the field is made writeable.

FunctionID = 29

Parameters:

Data[0]	(char *)	Object.Fieldname

Get_Type_Status

Description: This function returns the status of the type locked flag.

FunctionID = 16

Parameters:

Data[0]	(char *)	Object.Fieldname

Get_Data_Status

Description: This function returns the status of the data locked flag.

FunctionID = 17

Parameters:

Data[0]	(char *)	Object.Fieldname

Save_As_Bede_File

Description: All objects in the registry that are tagged, will be saved to a **Bede Scientific Instruments** format file. This may be ASCII or binary format and contains all information required to reconstruct the data objects.

FunctionID = 18

Parameters:

Data[0]	(char *)	Filename
Data[1]	(char)	Mode, 0 = ASCII, 1 = binary
Data[2]	(void *)	Pointer to a header type structure Appendix 3.

Load_Bede_File

Description: This function is used to load data stored in the **Bede Scientific Instruments** file format. If the file is an ASCII file, then the objects are streamed into the registry. If the file is a binary file then only the object headers are streamed in.

FunctionID = 19

Parameters:

Data[0]	(char *)	Filename
Data[1]	(void *)	Pointer to a header stuct.

Append Bede File

Description: This function is used to append two files. Each of the two source files may be either binary or ascii and the resulting file format is chosen by the user. If only one source file is passed (Data[2] = 0) then this function can be used to convert a file between binary and ascii formats.

FunctionID = 30

Parameters:

Data[0]	(char *)	Target Filename
Data[1]	(void *)	First source filename.
Data[2]	(void *)	Second source filename.
Data[1]	(char)	Target format: 0 for ascii, 1 for binary

Tag Object

Description: Sets the **Tag** field of an object. Tagged objects can be collectively deleted, stored to, or retrieved from, a file.

FunctionID = 20

Parameters:

Data[0]	(char *)	Object name

Untag Object

Description: Clears the **Tag** field of an object. Tagged objects can be collectively deleted, stored to, or retrieved from, a file.

FunctionID = 21

Parameters:

Data[0]	(char *)	Object name

Tag All Objects

Description: Sets the **Tag** field of all objects in the registry. Tagged objects can be collectively deleted, stored to, or retrieved from, a file.

FunctionID = 22

Untag All Objects

Description: Clears the **Tag** field of all objects in the registry. Tagged objects can be collectively deleted, stored to, or retrieved from, a file.

FunctionID = 23

Delete Unloaded Objects

Description: This function removed from the registry any invalid objects, also any object headers for objects whose data is not in memory are deleted.

FunctionID = 25

Delete_Tagged_Objects

Description: Deletes all objects in the registry that have their Tag field set.

FunctionID = 26

Get_Object_Tag

Description: Returns an error code based on whether the object has its Tag field set or cleared.

FunctionID = 27

Appendix 1

Data Types.

Data type	identification number
1 byte integer (`char`)	1
2 byte integer (`short int`)	2
*** NOT USED ***	3
4 byte integer (`long`)	4
4 byte unsigned integer(`unsigned long`)	5
4 byte floating point number (`float`)	6
8 byte floating point number (double)	7
Variable length string	9

Appendix 2

Error codes returned

0 - SUCCESS
1 - RegMemAlloc, // Registry failed to malloc
2 - RegNoObject, // No object of that name
3 - RegCrFail, // Could not create the registry
4 - RegListFull, // All registry list positions allocated
5 - ObjMemAlloc, // Data object failed to malloc
6 - ObjNoDecReg, // No declaration registry
7 - ObjNoDecl, // No declaration of that name
8 - ObjInvDecl, // Declaration is invalid
9 - ObjNoField, // No field of that name
10 - ObjArrType, // Arrays have different data types
11 - ObjValRange, // Value out of range of type
12 - ObjArrSize, // Array - single value mismatch
13 - ObjUnkError, // Object passed an invalid type
14 - ObjOnFile, // Data is still on disk file
15 - ObjNoData, // No data has been set for that field
16 - ObjNotThere, // Data not found in disk file
17 - ObjInvPtr, // Expected to get a NULL pointer
18 - ObjNoStore, // No space malloced for data
19 - ObjPreTagged, // object is tagged
20 - ObjPreUntagged, // object is NOT tagged
21 - ObjChildTagged, // object has tagged children
22 - ObjTypeMismatch, // Types not compatible
23 - PimMemAlloc, // Failed to allocate memory within PIM
24 - PimNoReg, // No registry created.
25 - PimFileExists, // file exists.
26 - PimInvSubst, // Invalid registry substitution.
27 - Failure, // Failed to create registry
28 - InsufficientDiskSpace,
29 - NullParam, // expected non NULL pointer
30 - InvalidParam,
31 - StreamInError, // Error reading data file
32 - StreamOutError, // Error writing data file

```
33 - UndefinedError,        // Unknown error
34 - NoSuchClass,           // No class of this name in registry
35 - NoSuchField,           // No field of this name in class
36 - NoSuchObject,          // No object of this name in registry
37 - InvalidClassName,      // Non valid class name passed
38 - UnableToAddClass,      // Class registry full
39 - ClassRegistryError,    // Class registry corrupt
40 - ClassAllocError,       // Memory allocation error
41 - ClassIsRegistered,     // Class of this name exists
42 - ClassInstantiated,
43 - ClassNameMismatch,
44 - FrozenDeclaration,     // Cannot add fields to a class with objects
45 - InvalidFieldName,      // Illegal characters in field name
46 - UnableToAddField,      // Field cannot be added to class
47 - FieldRegistryError,    // Field registry corrupt
48 - FieldAllocError,       // memory allocation error
49 - FieldIsRegistered,     // Field of that name in the class
50 - UnableToConvert,
51 - FieldNameMismatch
52 - NoSuchFile,            // File does not exist
53 - ShortFile,             // expected more data within file
54 - BedeFile,              // File is a Bede format file
55 - NonBedeFile,           // File is not a Bede format file
56 - NoBedeFiles,           // No Bede files in that location
57 - FileOpenError,         // File locked by another user
58 - FileAlreadyExists,
59 - FileReadError,
60 - FileWriteError,
61 - FilePositionError,
62 - FileCopyError,
63 - FileEraseError,
64 - FileInsertError,
65 - FileDeleteError,
66 - MediaNotPresent,
67 - MediaNotFomatted,
68 - MediaFomatError,
```

69 - MediaError,
70 - NextDisk,
71 - WrongDisk,
72 - ArchiveMismatch,
73 - CompAppNotFound, // compression application not found
74 - CompressionError, // error during data compression
75 - FileIsCompressed, // Expected uncompressed file
76 - UncompAppNotFound, // uncompression application not found
77 - UncompressionError,
78 - FileNotCompressed,
79 - KernelNoStub,
80 - KernelNoFunction, // Invalid FunctionID passed
81 - KernelParamError, // Error in parameters passed.
82 - KernelModeCrash // Error within Panther kernel

83 - ObjFreeFail // Failed to delete previous object
84 - ObjTypeLock // field is type locked
85 - ObjDataLock // field is readonly
86 - ObjNoType // No type given for field
87 - MemFreeFail // memory deallocation failed
88 - MemLockFail // memory handle lock failed
89 - FieldLock // Field locking error
90 - FileFailOpen // File failed to open.
91 - Not8_3Filename // File is not 8.3 format when expected.
92 - KernelLibLoadFail // Kernel failed to load a DLL.
93 - End_Of_File // End of file marker reached.

Appendix 3

The Header structure as required for a Bede format file.

```
struct { char Filename[100];
         char IsBinary;
         char CreationDate[11];
         char ModificationDate[11];
         float Version;
         char IsCompressed;
         char Creator[20];
         char User[20];
         int FileType;
         char Comment[200];
         } Header;
```