

RETURN TO MAIN MENU

# **Recital/SDK**

## **Recital Software Developers Kit**

Recital Corporation,  
100 Cummings Center, Suite 318J  
Beverly, MA 01915

Recital may have patents and/or patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.  
COPYRIGHT ©1988-2005 Recital Corporation. All rights reserved. All Recital products are trademarks or registered trademarks of Recital Corporation, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Last Updated June, 2005

## INDEX

---

<b>OVERVIEW</b>	1
<b>FUNCTION LEVELS</b>	2
<b>INCLUDE FILE</b>	3
<b>LINKING C FUNCTIONS</b>	9
<b>SHARED LIBRARY FUNCTIONS</b>	5
<b>DYNAMIC LINK LIBRARY FUNCTIONS</b>	7
<b>FUNCTION TABLE</b>	11
<b>Function Level 1</b>	14
_parc()	15
_parclen()	17
_parcsiz()	19
_pards()	21
_parinfo()	23
_parinfo()	25
_parl()	27
_parnd()	29
_parni()	30
_parnl()	31
_paro()	32
_parts()	34
_parys()	36
_ret()	38
_retc()	39
_retclen()	41
_retcls()	43
_retl()	44
_retnd()	45
_retni()	46
_retnl()	47
_reto()	48
_retts()	49
_retys()	50
ALENGTH()	51
ISARRAY()	52
ISCHAR()	54
ISCURRENCY()	54
ISDATE()	55
ISDATETIME()	56
ISLOG()	57
ISMEMO()	58
ISNUM()	59
ISOBJECT()	60
PCOUNT	61
<b>Function Level 2</b>	
COMMAND()	62
ERROR()	63
EXPRESSION()	64

<b>Function Level 3</b>	
ARRAY_ALEN()	66
ARRAY_DEFINE()	67
ARRAY_LOOKUP()	69
ARRAY_UPDATE()	71
BLOB_READ()	74
BLOB_RECLAIM()	75
BLOB_SIZE()	76
BLOB_UPDATE()	77
BLOB_WRITE()	78
CHAR_ALLTRIM()	79
CHAR_LOWER()	80
CHAR_LPAD()	81
CHAR_LTRIM()	82
CHAR_RPAD()	83
CHAR_RTRIM()	84
CHAR_STR()	85
CHAR_UPPER()	86
CURR_STOY()	87
CURR_YTOS()	88
DATE_AMPM()	89
DATE_CDOW()	90
DATE_CMONTH()	91
DATE_CTOD()	92
DATE_DATE()	94
DATE_DATETIME()	95
DATE_DAY()	96
DATE_DOW()	97
DATE_DTOC()	98
DATE_DTOS()	99
DATE_MONTH()	100
DATE_STOD()	101
DATE_STOT()	102
DATE_TIME()	103
DATE_TTOS()	104
DATE_YEAR()	105
DBF_ALIAS()	106
DBF_APPEND()	107
DBF_DBF()	108
DBF_DELETE()	109
DBF_DELETED()	110
DBF_FETCH()	111
DBF_FILTER()	113
DBF_FMT()	114
DBF_GATHER()	115
DBF_GOTO()	117
DBF_INDEXKEY()	118
DBF_INDEXORDER()	119
DBF_ISREADONLY()	123
DBF_LOCKR()	125
DBF_NDX()	126
DBF_READ()	127
DBF_RECALL()	129

DBF_RECBUFFER()	130
DBF_RECCOUNT()	131
DBF_RECNO()	132
DBF_RECSIZE()	133
DBF_SCATTER()	134
DBF_SEEK()	136
DBF_SELECT()	137
DBF_SKIP()	138
DBF_UNLOCKF()	139
DBF_UNLOCKR()	140
DBF_UPDATE()	141
DBF_USED()	143
FIELD_COUNT()	144
FIELD_LOOKUP()	145
FIELD_NAME()	147
FIELD_UPDATE()	149
FIELD_VALUE()	151
MEMO_MLCOUNT()	153
MEMO_MLINE()	154
MEMO_READ()	156
MEMO_RECLAIM()	158
MEMO_SIZE()	159
MEMO_UPDATE()	160
MEMO_WRITE()	162
MEMVAR_DEFINE()	163
MEMVAR_LOOKUP()	166
MEMVAR_UPDATE()	168
<b>Function level 4</b>	170
DEFINE_CLASS()	171
DEFINE_METHOD()	173
DEFINE_PROPERTYGET()	175
DEFINE_PROPERTYSET()	177
DISPATCH_FACTORY()	179
DISPATCH_METHOD()	181
DISPATCH_PROPGET()	183
DISPATCH_PROPSET()	185
OBJECT_ASSIGN()	187
OBJECT_DELETE()	189
OBJECT_GETARG()	191
OBJECT_GETARGC()	193
OBJECT_GETDATA()	195
OBJECT_GETOBJECT()	196
OBJECT_GETPARAMETER()	198
OBJECT_GETPROPERTY()	200
OBJECT_GETTYPE()	202
OBJECT_GETVALUE()	204
OBJECT_NEW()	206
OBJECT_RETERROR()	208
OBJECT_RETPROPERTY()	210
OBJECT_RETRESULT()	211
OBJECT_SETARG()	213
OBJECT_SETDATA()	215

OBJECT_SETPROPERTY()	217
<b>Appendix A</b>	219

## GENERAL OVERVIEW

---

The RECITAL/SDK (Software Developers Kit) provides developers with an Applications Programming Interface (API) allowing access to the internals of Recital. This gives developers the ability to write User Defined Functions (UDF) and User Defined Classes (UDC) in 'C' and then link them into the Recital product executables or access them via shared libraries (.so files on Linux/Unix) or dynamic link libraries (Windows DLLs).

Even though the RECITAL/4GL language is rich with functionality, the RECITAL/SDK allows developers to extend the RECITAL/4GL language by building 'C' libraries which may be used to access operating system dependent information and perform application specific functions that the Recital language does not provide. Such operations may include low-level calls to communications devices, the operating system and network facilities. The RECITAL/SDK provides the means to build presentation graphics and Xwindows functions for displaying the data accessed by Recital. In fact there is no limit for what you may use the RECITAL/SDK.

Functions for use with BLOBS (binary large objects) are included in the API. Any data type including picture and sound may be stored in Recital's Memo files and accessed with the blob functions.

The RECITAL/SDK can be used to make UDFs and UDCs accessible from the following Recital products:

Product	Linked	Shared Libraries	DLL
Recital Terminal Developer for Linux	√	√	
Recital Terminal Developer for UNIX	√	√	
Recital Terminal Developer for OpenVMS	√		
Recital Universal Application Server for Linux	√	√	
Recital Universal Application Server for UNIX	√	√	
Recital Universal Application Server for Windows			√

## FUNCTION LEVELS

---

The API (Applications Programming Interface) provides a library of 'C' functions that are divided into four different levels.

Functions available at level 1 are for use with parameter passing between Recital and your C function. These are used to check the type and number of parameters passed and to convert Recital data types to C data types. A 'C' UDF which uses the RECITAL/API can have a variable number of parameters passed to it from Recital. For example:

```
store "RECITAL" to m_string
store 7.0 to m_value
m_result = dbapi_udf ( m_string, m_value)
```

Functions available at level 2 are high-level functions used to execute RECITAL commands and evaluate expressions.

Functions available at level 3 are low-level functions separated into the following categories dependant usage.

TYPE	DESCRIPTION
ARRAY	Recital array access
BLOB	Operations on binary large objects
CHARACTER	Character string functions
CURRENCY	Currency functions
DATE	Date, datetime and time functions
DBF	Database I/O operations
FIELD	Field I/O operations
MEMO	Memo I/O operations
MEMVAR	Recital memory variable operations

Functions available at level 4 are for use with building classes and managing objects.

## INCLUDE FILE

---

The include file “dbapi.h” contains definitions of variable types, data structures and macro synonyms which are used by many of the Recital/API functions.

### API Variable Types:

TYPE NAME	USED FOR	DATATYPE
API_ATYPE	Array variable	int (512)
API_BOTTOM	DBF_GOTO( API_BOTTOM )	int (-2)
API_CLASS	User defined class (UDC)	int (1)
API_CTYPE	Character variable	int (1)
API_DTYPE	Date variable	int (8)
API_FUNCTION	User defined function (UDF)	int (0)
API_LTYPE	Logical variable	int (4)
API_NTTYPE	Numeric variable	int (2)
API_OTYPE	Object variable	int (256)
API_PRIVATE	Private memory variable	int (1)
API_PUBLIC	Public memory variable	int (1)
API_REFPAR	Passed by reference with @	int (32)
API_TOP	DBF_GOTO( API_TOP )	int (-1)
API_TTYPE	DateTime variable	int (64)
API_UTYPE	Undefined variable	int (0)
API_YTYPE	Currency variable	int (128)
CHARACTER	Character string	char
CURRENCY	Recital currency	Currency structure
DATE	Recital date	unsigned long
DATETIME	Recital datetime	DateTime structure
LOGICAL	Logical data type	int
MEMO	Recital memo	Operating system dependant
NUMERIC	Numeric data type	double
OBJARG	Object argument	Recital object argument format
OBJECT	Recital object	Pointer to an object

### API Data Structures:

The API\_FUNCTION\_TABLE data structure below is used to store information relating the C UDF to its name within Recital. This structure is used when statically linking in the function.

```
struct  API_FUNCTION_TABLE {  
    char          *name;           /* Pointer to recital function name      */  
    int           (*address)();    /* Address of C support function        */  
};
```



The API\_SHARED\_FUNCTION\_TABLE data structure below is used to store information relating the C UDF or UDC to its name within Recital. This structure is used when dynamically loading in the function or class.

```
struct  API_SHARED_FUNCTION_TABLE {
    char    *name;           /* Pointer to recital function name      */
    char    *apiname;       /* Pointer to C support function name    */
    int     type;           /* Function type 0 function 1 class      */
};
```

The API\_EXPRESSION\_RESULT data structure below is used to store the result which is returned from the api\_expression() function.

```
struct  API_EXPRESSION {
    int     erro;           /* RECITAL error() number                */
    char    type;          /* Data type of result                   */
    int     width;         /* Width of result                       */
    int     decimal;       /* Decimal places                        */
    char    character;     /* CHARACTER result                      */
    double  number;        /* NUMERIC RESULT                        */
    char    logical;       /* LOGICAL result                       */
    unsigned long date;    /* Recital DATE result                   */
    DATETIME datetime;    /* DATETIME result                      */
    CURRENCY currency;    /* CURRENCY result                      */
};
```

The API\_MEMVAR data structure below is used to describe the contents of a Recital memory variable.

```
union   API_UNION {
    char    *info_character; /* Char memory variable                  */
    char    info_logical;   /* Logical memory variable                */
    double  info_number;    /* Numeric memory variable                */
    unsigned long info_date; /* Date memory variable                   */
    DATETIME info_datetime; /* DateTime memory variable               */
    CURRENCY info_currency; /* Currency memory variable               */
};

struct  API_MEMVAR {
    char    type;           /* Data type of the memory                */
    union   API_UNION value; /* Value of the memory variable           */
    int     width;         /* Display width                          */
    int     decimal;       /* Display # of decimal places            */
};
```

The API\_FIELD data structure is used to describe the structure of a Recital field.

```
struct  API_FIELD {
    int     fieldno;        /* Field number                           */
    char    type;          /* Data type of field                     */
    int     size;          /* Storage size in the record             */
    int     width;         /* Display width                          */
    int     decimal;       /* # decimal places for numerics          */
    char    *value;        /* Pointer to data in current record      */
};
```

## SHARED LIBRARY FUNCTIONS

---

'C' Functions defined in shared libraries can be used with the following Recital products:

Product	Executables
Recital Terminal Developer for Linux	<path>/db.exe
Recital Terminal Developer for UNIX	<path>/db.exe
Recital Terminal Developer for Windows	<path>/db.exe
Recital Server for Linux	<path>/db_netserver <path>/db_recserver
Recital Server for UNIX	<path>/db_netserver <path>/db_recserver

### Defining the C functions:

Samples to demonstrate the usage of shared libraries are included in the following directories:

Product	Default API Linked Directory
Recital Terminal Developer for Linux	<path>/sdk/api/shared
Recital Terminal Developer for UNIX	<path>/sdk/api/shared
Recital Terminal Developer for Windows	<path>/sdk/api/shared
Recital Server for Linux	<path>/api/shared
Recital Server for UNIX	<path>/api/shared

The shared library must include the following four elements:

1. The include file "dbapi.h"

```
#include "dbapi.h"
```

2. The API\_SHARED\_FUNCTION\_TABLE structure, used to define the 'C' functions and objects included in the library:

```
// Recital API function address table
static struct API_FUNCTION_ADDRESS_TABLE *api_function_address_table = NULL;

// Add all functions and objects to this structure as follows:
//      Recital Name,      C Function Name      Type
//
// Make sure the last entry is NULL.
//
static struct API_SHARED_FUNCTION_TABLE api_function_table[7] = {
    {"schar",      "fnSamplesCharacter",    API_FUNCTION},
    {"stype",      "fnSamplesType",        API_FUNCTION},
    {"slog",       "fnSamplesLogical",      API_FUNCTION},
    {"snum",       "fnSamplesNumeric",      API_FUNCTION},
    {"sopen",      "fnSamplesOpen",        API_FUNCTION},
    {"myclass",    "clsMyClass",           API_CLASS},
    {NULL,        NULL,                    -1}
};
```

3. The initAPI() function:

```
// This function is used to define function addresses for API calls.
int initAPI(struct          API_FUNCTION_ADDRESS_TABLE *function_address_table)
{
    api_function_address_table = function_address_table;
    return 0;
}
```

4. The getFunctions() function:

```
// This function is used to return the function names of the API.
struct API_SHARED_FUNCTION_TABLE *getFunctions(void)
{
    return api_function_table;
}
```

**Using the ‘C’ functions and classes:**

Shared libraries are loaded using the SET LIBRARY TO <library> [ADDITIVE] command. By default library files are accessed from the directory defined by the DB\_LIBDIR environment variable. This is set in the ‘profile’ file of the Recital product:

Product	Profile
Recital Terminal Developer for Linux	<path>/profile.db
Recital Terminal Developer for UNIX	<path>/profile.db
Recital Terminal Developer for Windows	<path>/profile.db
Recital Server for Linux	<path>/profile.uas
Recital Server for UNIX	<path>/profile.uas

By default, DB\_LIBDIR is set to the sharedlib sub-directory of the recital root. If full path information is specified, the shared library files can be loaded from other directories, e.g.

```
// pdf.so is in the DB_LIBDIR directory
set library to pdf
```

```
//mylib.so is in /usr/recital/myapplibs
set library to /usr/recital/myapplibs/mylib
```

The Recital/4GL function LOADLIBRARY(<library>) can also be used to load additional shared libraries. The name of the shared library file including the full path and file extension must be specified.

The SET LIBRARY TO or RELEASE LIBRARY <library> commands are used to close all or specified shared libraries.

The Recital/4GL LIST PROCEDURE and DISPLAY PROCEDURE commands include loaded shared library function names in their listings and the Recital/4GL LIST CLASSES and DISPLAY CLASSES commands include loaded shared library class names in their listings.

Loaded shared library functions can be used in the same way as an internal Recital function. Loaded shared library classes can be used in the same way as Recital system classes.

## DYNAMIC LINK LIBRARY FUNCTIONS

---

'C' Functions and classes defined in dynamic link libraries (DLLs) can be used with the following Recital products:

Product	Executables
Recital Server for Windows	<path>\db_netserver.exe

### Defining the C functions:

Samples to demonstrate the usage of DLLs are included in the sharedlib\Samples directory below the Recital root directory.

The main source file for the DLL must include the following four elements:

1. The include file "dbapi.h"

```
#include "dbapi.h"
```

2. The API\_SHARED\_FUNCTION\_TABLE structure, used to define the 'C' functions and classes included in the library:

```
// Recital API function address table
static struct API_FUNCTION_ADDRESS_TABLE *api_function_address_table = NULL;

// Add all functions to this structure as follows:
//      Recital Name,      C Function Name ,      Type
//
// Make sure the last entry is NULL.
//
static struct API_SHARED_FUNCTION_TABLE api_function_table[7] = {
    {"schar",      "fnSamplesCharacter",  API_FUNCTION},
    {"stype",      "fnSamplesType",      API_FUNCTION},
    {"slog",       "fnSamplesLogical",    API_FUNCTION},
    {"snum",       "fnSamplesNumeric",    API_FUNCTION},
    {"sopen",      "fnSamplesOpen",       API_FUNCTION},
    {"myclass",    "clsMyClass",          API_CLASS},
    {NULL,         NULL,                  -1}
};
```

3. The initAPI() function:

```
// This function is used to define function addresses for API calls.
// C++ example
extern "C" int WINAPI EXPORT initAPI(struct API_FUNCTION_ADDRESS_TABLE *function_address_table)
{
    api_function_address_table = function_address_table;
    return 0;
}
```

#### 4. The getFunctions() function:

```
// This function is used to return the function names of the API.  
// C++ example  
extern "C" struct API_SHARED_FUNCTION_TABLE *getFunctions(void)  
{  
    return api_function_table;  
}
```

#### Using the ‘C’ functions and classes:

Shared libraries are loaded using the SET LIBRARY TO <library> [ADDITIVE] command. By default library files are accessed from the directory defined by the DB\_LIBDIR environment variable. This is set in the Recital Server Manager *Settings*.

By default, DB\_LIBDIR is set to the sharedlib sub-directory of the recital root. If full path information is specified, the shared library files can be loaded from other directories, e.g.

```
// pdf.dll is in the DB_LIBDIR directory  
set library to pdf
```

```
//mylib.dll is in C:\Program Files\Recital\Myapplibs\myapplibs  
set library to "C:\Program Files\Recital\Myapplibs\mylib"
```

The Recital/4GL function LOADLIBRARY(<library>) can also be used to load additional shared libraries. The name of the shared library file including the full path and file extension must be specified.

The SET LIBRARY TO or RELEASE LIBRARY <library> commands are used to close all or specified shared libraries.

The Recital/4GL LIST PROCEDURE and DISPLAY PROCEDURE commands include loaded shared library function names in their listings and the Recital/4GL LIST CLASSES and DISPLAY CLASSES commands include loaded shared library class names in their listings.

Loaded shared library functions can be used in the same way as an internal Recital function. Loaded shared library classes can be used in the same way as Recital system classes.

## LINKING 'C' FUNCTIONS

---

External 'C' Functions can be linked into the following Recital product executables:

Product	Relinked Executables
Recital Terminal Developer for Linux	<path>/db.exe
Recital Terminal Developer for UNIX	<path>/db.exe
Recital Terminal Developer for OpenVMS	[<path>]DB_XX.EXE
Recital Universal Application Server for Linux	<path>/db_netserver <path>/db_recserver
Recital Universal Application Server for UNIX	<path>/db_netserver <path>/db_recserver

### Defining the C functions:

External 'C' functions are defined in the RECITAL/API definition file **db.api** located in the 'api linked' directory.

Product	Default API Linked Directory
Recital Terminal Developer for Linux	<path>/sdk/api/linked
Recital Terminal Developer for UNIX	<path>/sdk/api/linked
Recital Terminal Developer for OpenVMS	[<path>].SDK.API
Recital Universal Application Server for Linux	<path>/api/linked
Recital Universal Application Server for UNIX	<path>/api/linked

The first column contains the RECITAL function name and the second column contains the name of the 'C' function to be linked with RECITAL. When writing functions in 'C', Recital recommends that they are prefixed with "dbapi\_" so that they do not conflict with any internal Recital functions. The dbmake linking utility will automatically compile and link in all the \*.c files in the 'api linked' directory.

<u>RECITAL NAME</u>	<u>C FUNCTION NAME</u>
apitest	dbapi_apitest

The Recital function name cannot be longer than 10 characters.

### Linking in 'C' functions:

Once all the 'C' functions are defined in the **db.api** file the **dbmake** utility is used to link them into the Recital executables as defined above.

Product	dbmake
Recital Terminal Developer for Linux	<path>/sdk/api/linked/dbmake
Recital Terminal Developer for UNIX	<path>/sdk/api/linked/dbmake
Recital Terminal Developer for OpenVMS	[<path>].SDK.API]DBMAKE.COM
Recital Universal Application Server for Linux	<path>/api/linked/dbmake
Recital Universal Application Server for UNIX	<path>/api/linked/dbmake

**Using the ‘C’ functions:**

Once the functions have been linked into Recital they can be used in the same way that a Recital function can be used.

If you do not specify the correct number of parameters to the ‘C’ function Recital will display the error message “Invalid parameter”. If the incorrect data type is passed, Recital will display the error message “Data type mismatch in parameter list.”

## FUNCTION TABLE

---

The following table is a quick reference to all of the functions available in the RECITAL/API.

Function Level 1	Description
_parc()	Pass a pointer to a Recital character string
_parclen()	Length of a character string
_parcsiz()	Size of a character string passed by reference
_pards()	Pass a character pointer to a Recital date
_parinfo()	Parameter checking of arrays
_parinfo()	Parameter checking
_parl()	Pass a Recital logical as an integer
_parnd()	Pass a Recital numeric as a double
_parni()	Pass a Recital numeric as an integer
_parnl()	Pass a Recital numeric as a long
_paro()	Pass an object pointer to a Recital object
_parts()	Pass a character pointer to a Recital datetime
_parys()	Pass a character pointer to a Recital currency
_ret()	Return to Recital
_retc()	Return a character string to Recital
_retclen()	Return a character string and its length to Recital
_retds()	Return a date string to Recital
_retl()	Return an integer to a Recital logical
_retnd()	Return a double to a Recital numeric
_retni()	Return an integer to a Recital numeric
_retnl()	Return a long to a Recital numeric
_reto()	Return an object to Recital
_retts()	Return a datetime string to Recital
_retys()	Return a currency string to Recital
ALENGTH()	Number of array elements
ISARRAY()	Is parameter an array
ISCHAR()	Is parameter a character string
ISCURRENCY()	Is parameter a currency
ISDATE()	Is parameter a date
ISDATETIME()	Is parameter a datetime
ISLOG()	Is parameter a logical
ISMEMO()	Is parameter a memo
ISNUM()	Is parameter a numeric
ISOBJECT()	Is parameter an object
PCOUNT()	Number of parameters passed
<b>Function level 2</b>	
COMMAND()	Execute a Recital command
ERROR()	Call the Recital error handler.
EXPRESSION()	Evaluate a RECITAL expression
<b>Function level 3</b>	<b>Description</b>
ARRAY_ALEN()	Length of the array
ARRAY_DEFINE()	Define an array element
ARRAY_LOOKUP()	Look up an array element
ARRAY_UPDATE()	Update an element in an array
BLOB_READ()	Read a blob from a memo file



BLOB_RECLAIM()	Reclaim space in a memo file
BLOB_SIZE()	Size of a blob in a memo file
BLOB_UPDATE()	Update a blob in a memo file
BLOB_WRITE()	Write a blob to a memo file
CHAR_ALLTRIM()	All trim a character string
CHAR_LOWER()	Convert a string to lower case
CHAR_LPAD()	Left pad a string
CHAR_LTRIM()	Left trim a character string
CHAR_RPAD()	Left pad a string
CHAR_RTRIM()	Right trim a character string
CHAR_STR()	Convert a numeric to a string
CHAR_UPPER()	Convert a string to upper case
CURR_STOY()	String to a currency conversion
CURR_YTOS()	Currency to string conversion
DATE_AMPM()	Current time in AM/PM format
DATE_CDOW()	Character day of the week
DATE_CMONTH()	Character month of date
DATE_CTOD()	String to date conversion
DATE_DATE()	Current date as a character
DATE_DATETIME()	Current date and time as a character
DATE_DAY()	Number of day
DATE_DOW()	Number of day of week
DATE_DTOC()	Date to string conversion
DATE_DTOS()	Date to string conversion
DATE_MONTH()	Number of month
DATE_STOD()	String to date conversation
DATE_STOT()	String to datetime conversion
DATE_TIME()	Current time
DATE_TTOS()	Datetime to string conversion
DATE_YEAR()	Number of year
DBF_ALIAS()	Current database alias name
DBF_APPEND()	Append a record
DBF_DBF()	Current database name
DBF_DELETE()	Delete a record
DBF_DELETED()	Is the record deleted
DBF_FETCH()	Fetch a record
DBF_FILTER()	Current database filter condition
DBF_FMT()	Current format file for database
DBF_GATHER()	Gather an array into a database record
DBF_GOTO()	Goto record number
DBF_INDEXKEY()	Index key expression
DBF_INDEXORDER()	Return master index
DBF_ISBOF()	Is record pointer at beginning of file
DBF_ISEOF()	Is record pointer at end of file
DBF_ISEXCLUSIVE()	Is database opened exclusive
DBF_ISREADONLY()	Is database read-only
DBF_LOCKF()	Lock a database file
DBF_LOCKR()	Lock the current record
DBF_NDX()	Current index for database
DBF_READ()	Read a record
DBF_RECALL()	Recall a record
DBF_RECBUFFER()	Flush a database record buffer
DBF_RECCOUNT()	Current database record count

DBF_RECNO()	Current database record number
DBF_RECSIZE()	Database record size
DBF_SCATTER()	Scatter a database record into an array
DBF_SEEK()	Seek a key
DBF_SELECT()	Select a database work area
DBF_SKIP()	Skip a record
DBF_UNLOCKF()	Unlock a database file
DBF_UNLOCKR()	Unlock the current record
DBF_UPDATE()	Update a record
DBF_USED()	Is a database open in the current work area
FIELD_COUNT()	Number of fields in a database
FIELD_LOOKUP()	Look up a field definition
FIELD_NAME()	Specified field name
FIELD_UPDATE()	Update specified field
FIELD_VALUE()	Specified field value
MEMO_MLCOUNT()	Number of lines in the memo field
MEMO_MLINE()	Return a memo line
MEMO_READ()	Read a memo into a string
MEMO_RECLAIM()	Reclaim space in the memo file
MEMO_SIZE()	Size of the memo field
MEMO_UPDATE()	Update a memo field
MEMO_WRITE()	Write a string to a memo
MEMVAR_DEFINE()	Define a RECITAL memory variable
MEMVAR_LOOKUP()	Lookup a RECITAL memory variable
MEMVAR_UPDATE()	Update a RECITAL memory variable
<b>Function level 4</b>	<b>Description</b>
DEFINE_CLASS()	Define a class
DEFINE_PROPERTYGET()	Define a get property
DEFINE_PROPERTYSET()	Define a set property
DEFINE_METHOD()	Define a method
DISPATCH_FACTORY()	Dispatch a factory method
DISPATCH_PROPGET()	Dispatch a get property
DISPATCH_PROPSET()	Dispatch a set property
DISPATCH_METHOD()	Dispatch a method
OBJECT_ASSIGN()	Assign a new object
OBJECT_DELETE()	Delete an existing object
OBJECT_GETARG()	Get argument value
OBJECT_GETARGC ()	Get the number of arguments
OBJECT_GETDATA()	Get object's data area
OBJECT_GETOBJECT()	Get an object
OBJECT_GETPARAMETER()	Get parameter value
OBJECT_GETPROPERTY()	Get property value
OBJECT_GETTYPE()	Get parameter type
OBJECT_GETVALUE()	Get value to set property to
OBJECT_NEW()	Create a new object
OBJECT_RETERROR()	Return error
OBJECT_RETPROPERTY()	Return property value
OBJECT_RETRESULT()	Return the value of a result
OBJECT_SETARG()	Set argument value
OBJECT_SETDATA()	Set object's data area
OBJECT_SETPROPERTY()	Set property value

## OVERVIEW OF LEVEL 1

---

Level 1 functions are used for passing parameters between the 'C' functions and Recital. When creating a 'C' function with the API the parameters being passed should not be defined in the function name like a normal 'C' function. Instead the parameters passed are referenced with the API \_par functions. Each parameter passed is referenced by specifying a number representing the ordinal position in the parameter list with the API \_par function call.

For example, the following 'C' function is passed 2 parameters from Recital. The data type of each parameter is checked first with the \_parinfo() function, then the value of the parameters are passed to a char with the \_parc() function. Lastly the char value is then returned to Recital.

```
#include "dbapi.h"
```

```
dbapi_test_function()
{
    char string1[9];
    char string2[9];

    if (_parinfo(1) == API_CTYPE) {
        strcpy( string1, _parc(1));
    }
    if (_parinfo(2) == API_CTYPE) {
        strcpy( string2, _parc(2));
    }

    _retc( string1 );
}
```

Values are returned to Recital using the API \_ret functions. Recital memory variables and array elements can also be updated from within the 'C' function to store the results, see level 3.

## **\_parc()**

---

### **PURPOSE**

Pass a pointer to a Recital character string

### **SYNONYM**

api\_par\_c()

### **SYNOPSIS**

#include "dbapi.h"

```
char    *_parc(order [,index])
```

<input parameters>

```
int      order;  /* Placement in actual parameter list    */
int      index;  /* Array element index                                    */
```

<output parameters>

none

### **DESCRIPTION**

The \_parc() function is used for getting a character parameter from Recital. The order specifies the actual placement in the parameter list. If the parameter passed is a Recital array, then the optional index value is used to specify the required element number.

NOTE: There is only one pointer on the stack for \_parc(). The 'C' strcpy() function must be used to copy the result to a variable, you cannot simply pass the pointer.

### **EXAMPLE**

The following example returns the character string passed in the first parameter.

#### **Example Recital program:**

```
m_string="RECITAL"
m_string=string( m_string )
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_string()
{
    char string[9];

    if ( _parinfo(1) == API_CTYPE ) {
        strcpy( string, _parc(1));
    } else {
        strcpy( string, "");
    }

    _retc( string );
}
```

**SEE ALSO**

`_parcrlen()`, `_parcsiz()`, `_parinfo()`, `_retc()`, `_retcrlen()`, `ISCHAR()`, `CHAR_ALLTRIM()`, `CHAR_LOWER()`, `CHAR_LPAD()`, `CHAR_LTRIM()`, `CHAR_RPAD()`, `CHAR_RTRIM()`, `CHAR_STR()`, `CHAR_UPPER()`, `DATE_STOD()`

## **\_parclen()**

---

### **PURPOSE**

Return the length of a character string

### **SYNONYM**

api\_par\_clen()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _parclen(order [,index])
```

<input parameters>

```
int      order;          /* Placement in actual parameter list    */
int      index;          /* Array element index                      */
```

<output parameters>

none

### **DESCRIPTION**

The \_parclen() function returns the length of a character string passed as a parameter. It will return the length of a string without counting the null terminator. The order specifies the actual placement in the parameter list. If the parameter passed is a Recital array, then the optional index value is used to specify the required element number.

### **EXAMPLE**

The following example will return the length of the character string passed in the first parameter.

#### **Example Recital program:**

```
m_string="RECITAL"
m_len=string( m_string )
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_string_length()
{
    int      stringlen;

    if ( _parinfo(1) == API_CTYPE ) {
        stringlen = _parclen(1);
    } else {
        stringlen=0;
    }

    _retni( stringlen );
}
```

**SEE ALSO**

`_parc()`, `_parcsiz()`, `_parinfo()`, `_retc()`, `_retcrlen()`, `ISCHAR()`, `CHAR_ALLTRIM()`, `CHAR_LOWER()`, `CHAR_LPAD()`, `CHAR_LTRIM()`, `CHAR_RPAD()`, `CHAR_RTRIM()`, `CHAR_STR()`, `CHAR_UPPER()`, `DATE_STOD()`

## **\_parcsiz()**

---

### **PURPOSE**

Return the size of a character parameter

### **SYNONYM**

api\_par\_csiz()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _parcsiz(order [,index])
```

<input parameters>

```
int      order;          /* Placement in actual parameter list    */
int      index;          /* Array element index                      */
```

<output parameters>

none

### **DESCRIPTION**

The \_parcsiz() function returns the number of bytes in memory allocated for the character string including the NULL terminator. The order specifies the actual placement in the parameter list. If the parameter passed is a Recital array, then the optional index value is used to specify the required element number.

### **EXAMPLE**

The following example returns the maximum string size of the character string passed as the first parameter.

#### **Example Recital program:**

```
m_string="RECITAL"
m_size=string_size( m_string )
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_string_size()
{
    int      stringlen;

    if ( _parinfo(1) == API_CTYPE ) {
        stringlen = _parcsiz(1);
    } else {
        stringlen=0;
    }

    _retni(stringlen);
}
```



**SEE ALSO**

`_parc()`, `_parclen()`, `_parinfo()`, `_retc()`, `_retclen()`, `ISCHAR()`, `CHAR_ALLTRIM()`, `CHAR_LOWER()`, `CHAR_LPAD()`, `CHAR_LTRIM()`, `CHAR_RPAD()`, `CHAR_RTRIM()`, `CHAR_STR()`, `CHAR_UPPER()`, `DATE_STOD()`

## **\_pards()**

---

### **PURPOSE**

Return a pointer to a date string

### **SYNONYM**

api\_par\_ds()

### **SYNOPSIS**

#include "dbapi.h"

```
char    *_pards(order [,index])
```

<input parameters>

```
int      order;           /* Placement in actual parameter list    */
int      index;           /* Array element index                    */
```

<output parameters>

none

### **DESCRIPTION**

The \_pards() function gets a date parameter from Recital and returns a character pointer in the form of "YYYYMMDD". The order specifies the actual placement in the parameter list. If the parameter passed is a Recital array, then the optional index value is used to specify the required element number.

NOTE: there is only one pointer on the stack for \_pards(). The 'C' strcpy() function must be used to copy the result to a variable, you cannot simple pass the pointer.

### **EXAMPLE**

The following example copies the first parameter passed to a string and then returns the result.

#### **Example Recital program:**

```
m_date=date()
m_ansidate =date_ansi( m_date )
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_date_ansi()
{
    char    date[9];

    if (_parinfo(1) == API_DTYPE ) {
        strcpy( date, _pards(1));
    } else {
        strcpy( date, "" );
    }

    _retc( date );
}
```

**SEE ALSO**

`_retds()`, `ISDATE()`, `DATE_STOD()`

## **\_parinfo()**

---

### **PURPOSE**

Check data type of any array elements

### **SYNONYM**

api\_par\_infa()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _parinfo(order [,index])
```

<input parameters>

```
int      order;          /* Placement in the array in the parameter list    */
int      index;          /* Array element index                                */
```

<output parameters>

none

### **DESCRIPTION**

The function returns a value specifying the data type of an array element and is used for parameter checking. Array elements can be of varying data types in Recital and must be checked before using. The order specifies the actual placement in the parameter list. The index value is used to specify the required element number in the specified array. If the index value is 0, then the total number of elements in the array will be returned.

The \_parinfo() function returns a value from the following table:

TYPE	VALUE	DESCRIPTION
API_CTYPE	1	Character
API_NTYPE	2	Numeric
API_LTYPE	4	Logical
API_DTYPE	8	Date

### **EXAMPLE**

The following example returns the number of character elements in the array passed as the first parameter.

#### **Example Recital program:**

```
Declare m_array[2]
m_array[1]="RECITAL"
m_array[2]="7.1"
m_num_char=numchar(m_array)
```

**Example ‘C’ function:**

```
#include "dbapi.h"
```

```
dbapi_numchar()
{
    int    num;
    int    i;

    if (_parinfo(1) == API_ATYPE) {
        for (i = 1; i <= _parinfo(1, 0); ++i) {
            if (_parinfo(1,i) == API_CTYPE) {
            }
        }
    } else {
        num = -1;
    }

    _retni( num );
}
```

**SEE ALSO**

`_parinfo()`, `ALENGTH()`, `ISARRAY()`, `ARRAY_ALEN()`, `ARRAY_DEFINE()`, `ARRAY_LOOKUP()`, `ARRAY_UPDATE()`, `MEMVAR_DEFINE()`, `MEMVAR_LOOKUP()`, `MEMVAR_UPDATE()`

## **\_parinfo()**

---

### **PURPOSE**

Check data type of parameters

### **SYNONYM**

api\_par\_info()

### **SYNOPSIS**

#include "dbapi.h"

int        \_parinfo(order )

<input parameters>

int        order;               /\* Placement in list of parameters \*/

<output parameters>

none

### **DESCRIPTION**

The \_parinfo() function returns the data type of a parameter passed. The order specifies the actual placement in the parameter list. If 0 is specified as the order value then the total number of parameters passed is returned. This function should be used to check the data types of all parameters passed before they are assessed.

The following table contains possible data types that may be passed

<b><u>TYPE</u></b>	<b><u>VALUE</u></b>	<b><u>DESCRIPTION</u></b>
API_UTYPE	0	Undefined
API_CTYPE	1	Character
API_NTTYPE	2	Numeric
API_LTYPE	4	Logical
API_DTYPE	8	Date
API_MTYPE	16	Memo
API_REFPAR	32	Passed by reference
API_ATYPE	512	Array

### **EXAMPLE**

The following returns .T. if the first parameter passed is a logical.

#### **Example Recital program:**

```
m_value=.t.  
m_result=islogical(m_value)  
return
```

**Example ‘C’ function:**

```
#include "dbapi.h"
```

```
dbapi_islogical()
{
    char    result[10];

    if (_parinfo(1) == API_CTYPE) {
        strcpy(result , "Character");
    }
    else if (_parinfo(1) == API_NTTYPE)
        strcpy(result , "Numeric");
    else if (_parinfo(1) == API_LTYPE)
        strcpy(result , "Logical");
    else if (_parinfo(1) == API_DTYPE)
        strcpy(result , "Date");
    else if (_parinfo(1) == API_ATYPE)
        strcpy(result , "Array");
    else
        strcpy(result , "");

    _retc( result );
}
```

**SEE ALSO**

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISARRAY(), ISCHAR(), ISDATE(), ISLOG(), ISMEMO(), ISNUM(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD(), PCOUNT

## **\_parl()**

---

### **PURPOSE**

Convert a Recital logical to an integer

### **SYNONYM**

api\_par\_l()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _parl(order [,index]) )
```

<input parameters>

```
int      order;          /* Placement in actual parameter list      */
int      index;          /* Array element index                          */
```

<output parameters>

none

### **DESCRIPTION**

The function converts a logical parameter passed from Recital to an integer, where .T. = 1 and .F. = 0. The order specifies the actual placement in the parameter list. If the parameter passed is an array, then the optional index value is used to specify the required element number.

### **EXAMPLE**

The following example returns the logical passed in the first parameter in English.

#### **Example Recital program:**

```
m_condition=.t.
m_word =ltow(m_condition)
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_ltow()
{
    char          strbuff[6];

    if ( _parinfo(1) == API_LTYPE ) {
        if ( _parl(1) == 0 ) {
            strcpy( strbuff, "False");
        } else {
            strcpy( strbuff, "True");
        }
    } else {
        strcpy( strbuff, "");
    }

    _retc( strbuff );
}
```



**SEE ALSO**

`_parmi()`, `_retl()`, `_retni()`, `ISLOG()`, `PCOUNT`

## **\_parnd()**

---

### **PURPOSE**

Convert a Recital numeric to a double

### **SYNONYM**

api\_par\_nd()

### **SYNOPSIS**

#include "dbapi.h"

double \_parnd(order [,index])

<input parameters>

int	order;	/* Placement in actual parameter list	*/
int	index;	/* Array element index	*/

<output parameters>

none

### **DESCRIPTION**

The \_parnd() function returns a numeric parameter passed from Recital as a double. The order specifies the actual placement in the parameter list. If the parameter passed is an array, then the optional index value is used to specify the required element number.

### **EXAMPLE**

The following example converts a Recital numeric value passed as the first parameter to a double.

#### **Example Recital program:**

```
m_double=1252
m_double =condouble(m_double)
return
```

#### **Example 'C' function:**

#include "dbapi.h"

```
dbapi_condouble()
{
    double value;

    if ( _parinfo(1) == API_NTTYPE ) {
        value = _parnd(1);
    }else {
        value = 0;
    }

    _retnl( value );
}
```

### **SEE ALSO**

\_parinfo(), \_parinfo(), \_parni(), \_parnl(), retnl(), \_retni(), \_retnl(), ISNUM()

## **\_parni()**

---

### **PURPOSE**

Convert a Recital numeric to an integer

### **SYNONYM**

api\_par\_ni()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _parni(order [,index])
```

<input parameters>

```
int      order;          /* Placement in actual parameter list    */
int      index;          /* Array element index                      */
```

<output parameters>

none

### **DESCRIPTION**

The \_parni() function returns a numeric parameter passed from Recital as an integer. The order specifies the actual placement in the parameter list. If the parameter passed is an array, then the optional index value is used to specify the required element number.

### **EXAMPLE**

The following example converts a Recital numeric value passed as the first parameter to an integer.

#### **Example Recital program:**

```
m_int=1252
m_double =conint(m_int)
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_conint()
{
    int      value;

    if (_parinfo(1) == API_NTTYPE) {
        value = _parni(1);
    }else {
        value = 0;
    }

    _retni( value );
}
```

### **SEE ALSO**

\_parinfo(), \_parinfo(), \_parnd(), \_parnl(), retn(), \_retni(), \_retnl(), ISNUM()

## **\_parnl()**

---

### **PURPOSE**

Convert a Recital numeric to a long

### **SYNONYM**

api\_par\_nl()

### **SYNOPSIS**

#include "dbapi.h"

long     \_parnl(order [,index]))

<input parameters>

int	order;	/* Placement in actual parameter list	*/
int	index;	/* Array element index	*/

<output parameters>

none

### **DESCRIPTION**

The \_parnl() function returns a numeric parameter passed from Recital as a long. The order specifies the actual placement in the parameter list. If the parameter passed is an array, then the optional index value is used to specify the required element number.

### **EXAMPLE**

The following example converts a Recital numeric value passed as the first parameter to a long.

#### **Example Recital program:**

```
m_long=1252
m_long=conlong(m_long)
return
```

#### **Example 'C' function:**

#include "dbapi.h"

```
dbapi_conlong()
{
    long    value;

    if ( _parinfo(1) == API_NTTYPE ) {
        value = _parnl(1);
    }else {
        value = 0;
    }

    _retnl( value );
}
```

### **SEE ALSO**

\_parinfa(), \_parinfo(), \_parnd(), \_parni(), retnd(), \_retni(), \_retnl(), ISNUM()

## **\_paro()**

---

### **PURPOSE**

Return an object pointer to an object

### **SYNONYM**

api\_par\_o()

### **SYNOPSIS**

#include "dbapi.h"

```
char    *_paro(order [,index])
```

<input parameters>

```
int      order;           /* Placement in actual parameter list    */
int      index;           /* Array element index                    */
```

<output parameters>

none

### **DESCRIPTION**

The \_paro() function gets an object parameter from Recital and returns a pointer in the form of an OBJECT. The order specifies the actual placement in the parameter list. If the parameter passed is a Recital array, then the optional index value is used to specify the required element number.

### **EXAMPLE**

The following example checks the first parameter passed is an object and then returns the number of properties.

#### **Example Recital program:**

```
m_exception=NEWOBJECT ('exception')
m_propcnt =object_propcnt( m_exception)
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_object_protcnt()
{
    OBJECT objptr;
    int      cnt;

    if (_parinfo(1) == API_OTYPE ) {
        cnt = OBJECT_PARACNT(_paro(1));
    } else {
        ERROR("Invalid Parameter");
    }

    _retni( cnt );
}
```

**SEE ALSO**

`_reto()`, `ISOBJECT()`, `OBJECT_NEW()`

## **\_parts()**

---

### **PURPOSE**

Return a pointer to a date time string

### **SYNONYM**

api\_par\_ts()

### **SYNOPSIS**

#include "dbapi.h"

```
char    *_parts(order [,index])
```

<input parameters>

```
int      order;           /* Placement in actual parameter list    */
int      index;           /* Array element index                    */
```

<output parameters>

none

### **DESCRIPTION**

The \_parts() function gets a datetime parameter from Recital and returns a character pointer in the format "YYYYMMDDHHMMSS". The order specifies the actual placement in the parameter list. If the parameter passed is a Recital array, then the optional index value is used to specify the required element number.

NOTE: there is only one pointer on the stack for \_parts(). The 'C' strcpy() function must be used to copy the result to a variable, you cannot simple pass the pointer.

### **EXAMPLE**

The following example copies the first parameter passed to a string and then returns the result.

#### **Example Recital program:**

```
m_datetime=datetime()
m_ansidatetime =datetime_ansi( m_datetime )
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_datetime_ansi()
{
    char    datetime[15];

    if (_parinfo(1) == API_TTYPE ) {
        strcpy( datetime, _parts(1));
    } else {
        strcpy( date, "" );
    }

    _retc( datetime );
}
```

**SEE ALSO**

`_retts()`, `ISDATETIME()`, `DATE_TTOS()`, `DATE_STOT()`



## **\_parys()**

---

### **PURPOSE**

Return a pointer to a currency string

### **SYNONYM**

api\_par\_ys()

### **SYNOPSIS**

#include "dbapi.h"

```
char    *_parys(order [,index])
```

<input parameters>

```
int      order;           /* Placement in actual parameter list    */
int      index;           /* Array element index                    */
```

<output parameters>

none

### **DESCRIPTION**

The \_parys() function gets a currency parameter from Recital and returns a character pointer. The order specifies the actual placement in the parameter list. If the parameter passed is a Recital array, then the optional index value is used to specify the required element number.

NOTE: there is only one pointer on the stack for \_parys(). The 'C' strcpy() function must be used to copy the result to a variable, you cannot simple pass the pointer.

### **EXAMPLE**

The following example copies the first parameter passed to a string and then returns the result.

#### **Example Recital program:**

```
m_currency=$23.8327
m_strcurr =currency_ctos( m_currency )
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_currency_ctos()
{
    char    currency[32];

    if (_parinfo(1) == API_YTYPE ) {
        strcpy( currency, _parys(1));
    } else {
        strcpy( date, "" );
    }

    _retc( currency );
}
```

**SEE ALSO**

`_parc()`, `_parclen()`, `_parcsiz()`, `_parinfo()`, `_retc()`, `_retclen()`, `_retys()`, `ISCURRENCY()`, `CURR_YTOS()`, `CURR_STOY()`

## **\_ret()**

---

### **PURPOSE**

Return to Recital

### **SYNONYM**

api\_ret()

### **SYNOPSIS**

#include "dbapi.h"

int       \_ret()

<input parameters>

none

<output parameters>

none

### **DESCRIPTION**

The \_ret() function is used to Return to Recital without returning any value. This allows for the function to be used like a command.

### **EXAMPLE**

The following example returns to Recital without returning any value.

#### **Example Recital program:**

```
exec_command("test")
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_exec_command()  
{  
    ret();  
}
```

### **SEE ALSO**

\_retc(), \_retclen(), \_retcls(), \_retl(), \_retnd(), \_retni(), \_retnl(), \_reto(), \_retts(), \_retys()

## **\_retc()**

---

### **PURPOSE**

Pass a character string to Recital

### **SYNONYM**

api\_ret\_c()

### **SYNOPSIS**

#include "dbapi.h"

int      \_retc(string)

<input parameters>

char    \*string;            /\* Address of a buffer containing a character string    \*/

<output parameters>

none

### **DESCRIPTION**

The \_retc() function is used to pass a character string back to Recital. The maximum length of a string that can be returned to Recital is 8192. If the string is too long, Recital will return an error message.

### **EXAMPLE**

The following example evaluates the first parameter passed and returns the English translation of the logical condition.

#### **Example Recital program:**

```
m_word=ltow(.T.)
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_ltow()
```

```
{
    char    strbuff[6];

    if ( _parinfo(1) == API_LTYPE ) {
        if ( _parl(1) == 0 ) {
            strcpy( strbuff, "False" );
        } else {
            strcpy( strbuff, "True" );
        }
    } else {
        strcpy(strbuff, "");
    }

    _retc( strbuff );
}
```

**SEE ALSO**

`_parc()`, `_parclen()`, `_parcsiz()`, `_parinfo()`, `_retclen()`, `ISCHAR()`, `CHAR_ALLTRIM()`, `CHAR_LOWER()`, `CHAR_LPAD()`, `CHAR_LTRIM()`, `CHAR_RPAD()`, `CHAR_RTRIM()`, `CHAR_STR()`, `CHAR_UPPER()`, `DATE_STOD()`

## **\_retclen()**

---

### **PURPOSE**

Pass a character string to Recital

### **SYNONYM**

api\_ret\_clen()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _retclen(string, len)
```

<input parameters>

```
char      *string;          /* Address of a buffer containing a character string */
int      len;               /* Length of string */
```

<output parameters>

none

### **DESCRIPTION**

The \_retclen() function is used to pass a character pointer back to your application. The second parameter, len, is used to specify the length of the character string returned. The maximum length of a string that can be returned to Recital is 8192. If the string is too long, Recital will return an error message.

### **EXAMPLE**

The following example evaluates the first parameter passed and returns the English translation of the logical condition with a length of 5 characters.

#### **Example Recital program:**

```
m_word=ltow(.T.)
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_ltow()
```

```
{
    char    strbuff[6];

    if (_parinfo(1) == API_LTYPE) {
        if ( _parl(1) == 0 ) {
            strcpy( strbuff, "False" );
        } else {
            strcpy( strbuff, "True" );
        }
    } else {
        strcpy(strbuff, "");
    }

    _retclen(strbuff, 5);
}
```

**SEE ALSO**

`_parc()`, `_parclen()`, `_parcsiz()`, `_parinfo()`, `_retc()`, `ISCHAR()`, `CHAR_ALLTRIM()`, `CHAR_LOWER()`, `CHAR_LPAD()`, `CHAR_LTRIM()`, `CHAR_RPAD()`, `CHAR_RTRIM()`, `CHAR_STR()`, `CHAR_UPPER()`, `DATE_STOD()`

## **\_retds()**

---

### **PURPOSE**

Pass a date string to Recital

### **SYNONYM**

api\_ret\_ds()

### **SYNOPSIS**

#include "dbapi.h"

int        \_retds(string)

<input parameters>

char      \*string;                / \*Address of a buffer containing a character string    \*/

<output parameters>

none

### **DESCRIPTION**

The \_retds() function returns a character string in the format of "YYYYMMDD" to Recital as a Recital date type. The DATE\_DTOS() function can be used to convert a date stored as an unsigned long to a string format.

### **EXAMPLE**

The following example evaluates the first parameter passed to a string and then returns the result.

#### **Example Recital program:**

```
m_date=date()
m_date=date_con(m_date)
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_date_con()
{
    char    date[9];

    if ( _parinfo(1) == API_DTYPE ) {
        strcpy( date, _pards(1));
    } else {
        strcpy( date, "");
    }

    _retds (date );
}
```

### **SEE ALSO**

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_DTOS(), DATE\_STOD()



## **\_retl()**

---

### **PURPOSE**

Return an integer as a Recital logical

### **SYNONYM**

api\_ret\_l()

### **SYNOPSIS**

#include "dbapi.h"

int       \_retl(flag)

<input parameters>

int       \*flag;               /\* Value for logical       \*/

<output parameters>

none

### **DESCRIPTION**

The \_retl() function returns an integer as a logical back to Recital. If flag is 0 then .F. is returned, if flag is 1 then .T. is returned.

### **EXAMPLE**

The following returns .T. if the first parameter passed is a logical.

#### **Example Recital program:**

```
m_value=.T.  
m_result=islogical(m_value)
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_islogical()  
{  
    int       result;  
  
    if (_parinfo == API_LTYPE) {  
        result = 1;  
    } else {  
        result = 0;  
    }  
  
    _retl( result );  
}
```

### **SEE ALSO**

\_parmi(), \_parl(), \_retni(), ISLOG(), PCOUNT

## **\_retn()**

---

### **PURPOSE**

Return a double as a Recital numeric

### **SYNONYM**

api\_ret\_nd()

### **SYNOPSIS**

#include "dbapi.h"

int       \_retn(n)

<input parameters>

double   n;                   /\*numeric expression       \*/

<output parameters>

none

### **DESCRIPTION**

The \_retn() function returns a double as a numeric value back to Recital.

### **EXAMPLE**

The following example returns a value stored as a double to a Recital numeric memory variable.

#### **Example Recital program:**

```
m_value=dtan(1234)
```

```
return.
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_dtan()
```

```
{
    double   value;

    if (_parinfo(1) == API_NTTYPE) {
        value = _parnd(1);
    } else {
        value = 0;
    }

    _retn(value);
}
```

### **SEE ALSO**

\_parinfo(), \_parinfo(), parnd(), \_parni(), \_parl(), \_retni(), retnl(), ISNUM()

## **\_retni()**

---

### **PURPOSE**

Return an integer as a Recital numeric

### **SYNONYM**

api\_ret\_ni()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _retni(n)
```

<input parameters>

```
int      n;                /* Numeric value          */
```

<output parameters>

none

### **DESCRIPTION**

The \_retni() function returns an integer as a numeric value back to Recital.

### **EXAMPLE**

The following example returns the length of a character string passed as the first parameter as an integer.

#### **Example Recital program:**

```
m_len=getlen("Recital")
return.
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_getlen()
{
    int      value;

    if ( _parinfo(1) == API_NTTYPE ) {
        value = _parni(1);
    } else {
        value = 0;
    }

    _retni( value );
}
```

### **SEE ALSO**

\_parinfa(), \_parinfo(), parnd(), \_parni(), \_parl(), \_retni(), ISNUM()

## **\_retnl()**

---

### **PURPOSE**

Return a long as a Recital numeric

### **SYNONYM**

api\_ret\_nl()

### **SYNOPSIS**

#include "dbapi.h"

int        \_retnl(n)

<input parameters>

int        n;                    / \*Numeric value                \*/

<output parameters>

none

### **DESCRIPTION**

The function returns a long as a numeric value back to Recital.

### **EXAMPLE**

The following example returns a value store as a long to a Recital numeric memory variable.

#### **Example Recital program:**

```
m_value=dtol(1234)
```

```
return.
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_dtol()
```

```
{
    long     value;

    if ( _parinfo(1) == API_NTTYPE ) {
        value = _parnl(1);
    } else {
        value = 0;
    }

    _retnl( value );
}
```

### **SEE ALSO**

\_parinfo(), \_parinfo(), parnd(), \_parni(), \_parl(), \_retnl(), retni(), ISNUM()

## **\_reto()**

---

### **PURPOSE**

Return an object to Recital as a Recital OBJECT type

### **SYNONYM**

api\_ret\_o()

### **SYNOPSIS**

#include "dbapi.h"

```
int      _reto(OBJECT)
```

<input parameters>

```
char      OBJECT;           /*Address of an object   */
```

<output parameters>

none

### **DESCRIPTION**

The \_reto() function returns an object to Recital as a Recital OBJECT type.

### **EXAMPLE**

The following example evaluates the first parameter passed to a string and then returns the result.

#### **Example Recital program:**

```
m_name="NewObj"
m_obj=obj_excp(m_name)
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_obj_excp ()
{
    OBJECT      objptr;
    char        name[33];

    if ( _parinfo(1) == API_CTYPE ) {
        strncpy( name, _parcs(1));
        name[32] = '\0';
    } else {
        ERROR("Invalid Parameter.");
    }

    objptr = OBJECT_NEW(name, "exception", NULL);
    _reto (objptr );
}
```

### **SEE ALSO**

\_paro(), ISOBJECT()

## **\_retts()**

---

### **PURPOSE**

Pass a datetime string to Recital

### **SYNONYM**

api\_ret\_ts()

### **SYNOPSIS**

#include "dbapi.h"

int        \_retts(string)

<input parameters>

char       \*string;                / \*Address of a buffer containing a character string    \*/

<output parameters>

none

### **DESCRIPTION**

The \_retts() function returns a character string in the format "YYYYMMDDHHMMSS" to Recital as a Recital datetime type. The DATE\_TTOS() function can be used to convert a datetime stored in a DATETIME structure to a string format.

### **EXAMPLE**

The following example evaluates the first parameter passed to a string and then returns the result.

#### **Example Recital program:**

```
m_datetime=datetime()
m_datetime=datetime_con(m_datetime)
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"

dbapi_datetime_con()
{
    char     datetime[15];

    if ( _parinfo(1) == API_TTYPE ) {
        strcpy( datetime, _parts(1));
    } else {
        strcpy( datetime, "");
    }

    _retts (datetime );
}
```

### **SEE ALSO**

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISDATETIME(), DATE\_TTOS(), DATE\_STOT()

## **\_retys()**

---

### **PURPOSE**

Pass a currency string to Recital

### **SYNONYM**

api\_ret\_ys()

### **SYNOPSIS**

#include "dbapi.h"

int      \_retys(string)

<input parameters>

char    \*string;                / \*Address of a buffer containing a character string    \*/

<output parameters>

none

### **DESCRIPTION**

The \_retys() function returns a character string in the format "99999999999999.9999" to Recital as a Recital currency type. The CURR\_YTOS() function can be used to convert a currency value stored in a CURRENCY structure to a string format.

### **EXAMPLE**

The following example evaluates the first parameter passed to a string and then returns the result.

#### **Example Recital program:**

```
m_curr=$2873.89
m_curr=curr_con(m_curr)
return
```

#### **Example 'C' function:**

```
#include "dbapi.h"
```

```
dbapi_curr_con()
{
    char    curr[21];

    if ( _parinfo(1) == API_YTYPE ) {
        strcpy( curr, _parys(1));
    } else {
        strcpy( curr, "");
    }

    _retys (curr );
}
```

### **SEE ALSO**

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_parys(), \_retc(), \_retclen(), ISCURRENCY(), CURR\_YTOS(), CURR\_STOY()

## ALENGTH()

---

### PURPOSE

Return the number of array elements

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

```
int      ALENGTH(order)
```

<input parameters>

```
int      order;          /*Placement in actual parameter list      */
```

<output parameters>

none

### DESCRIPTION

The function returns the number of elements in the array. The order specifies the actual placement in the parameter list.

### EXAMPLE

The following example returns the number of elements in the array passed as the first parameter.

#### Example Recital program:

```
Declare m_values[20]
m_anumber=anum(m_value)
return.
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
dbapi_anum()
{
    int      numarray

    if (_parinfo(1) == API_ATYPE) {
        numarray = ALENGTH(1);
    } else {
        numarray = -1;
    }

    _retni(numarray);
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), ISARRAY(), ARRAY\_ALLEN(), ARRAY\_DEFINE(), ARRAY\_LOOKUP(),  
ARRAY\_UPDATE(), MEMVAR\_DEFINE(), MEMVAR\_LOOKUP(), MEMVAR\_UPDATE()



## ISARRAY()

---

### PURPOSE

Is the parameter an array

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      ISARRAY(order)
```

<input parameters>

```
int      order;          /* Placement in actual parameter list          */
```

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is an array and returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is an array, otherwise .F. is returned.

#### Example Recital program:

```
declare m_values[10]
m_isarray=check_array(m_values)
return
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
check_array()
{
    retl(ISARRAY(1));
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), ALENGTH(), ARRAY\_ALEN(), ARRAY\_DEFINE(), ARRAY\_LOOKUP(),  
ARRAY\_UPDATE(), MEMVAR\_DEFINE(), MEMVAR\_LOOKUP, MEMVAR\_UPDATE()

## ISCHAR()

---

### PURPOSE

Is the parameter a character string

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

int ISCHAR(order)

<input parameters>

int order; /\* Placement in actual parameter list \*/

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is a character string. It returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is a character string, or .F. otherwise.

#### Example Recital program:

```
m_values = "five"
m_ischar=check_char(m_values)
return
```

#### Example 'C' function:

```
#include "dbapi.h"

check_char()
{
    _retl(ISCHAR(1));
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_parinfo(), \_retc(), \_retclen(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD()

## ISCURRENCY()

---

### PURPOSE

Is the parameter a currency data type

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

int       ISCURRENCY(order)

<input parameters>

int       order;               /\* Placement in actual parameter list               \*/

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is a currency data type. It returns 1 if true, or 0 if false. The order specifies the actual placement in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is a currency value, or .F. otherwise.

#### Example Recital program:

```
m_values = $5
m_iscurr=check_curr(m_values)
return
```

#### Example 'C' function:

```
#include "dbapi.h"

check_curr()
{
    _retl(ISCURRENCY(1));
}
```

### SEE ALSO

\_parys(), \_retys(), CURR\_YTOS(), CURR\_STOY()

## ISDATE()

---

### PURPOSE

Is the parameter a date

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      ISDATE(order)
```

<input parameters>

```
int      order;          /* Placement in actual parameter list          */
```

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is a Recital date. It returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is a Recital date, or .F. otherwise.

#### Example Recital program:

```
m_values = date()
m_isdater=check_date(m_values)
return
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
check_date()
{
    _retl( ISDATE(1));
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_YEAR()

## ISDATETIME()

---

### PURPOSE

Is the parameter a datetime

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

int ISDATETIME(order)

<input parameters>

int order; /\* Placement in actual parameter list \*/

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is a Recital datetime. It returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is a Recital datetime, or .F. otherwise.

#### Example Recital program:

```
m_values = datetime()
m_isdatetime=check_datetime(m_values)
return
```

#### Example 'C' function:

```
#include "dbapi.h"

check_datetime()
{
    _retl( ISDATETIME(1));
}
```

### SEE ALSO

\_parts(), \_retts(), DATE\_DATETIME(), DATE\_STOT(), DATE\_TTOS()

## ISLOG()

---

### PURPOSE

Is the parameter a logical

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      ISLOG(order)
```

<input parameters>

```
int      order;          /* Placement in actual parameter list          */
```

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is logical. It returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is logical, or .F. otherwise.

#### Example Recital program:

```
m_values = deleted()
m_islog=check_log(m_values)
return
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
check_log()
{
    _retl(ISLOG(1));
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_parl(), \_parni(), \_retl(), \_retni(), PCOUNT

## ISMEMO()

---

### PURPOSE

Is the parameter a memo

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      ISMEMO(order)
```

<input parameters>

```
int      order;          /* Placement in actual parameter list          */
```

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is a memo field. It returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is a memo field or .F. otherwise.

#### Example Recital program:

```
m_ismemo =check_memo(accounts->notes)
return
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
check_memo()
{
    _retl(ISMEMO(1));
}
```

### SEE ALSO

\_parinfo(), PCOUNT

## ISNUM()

---

### PURPOSE

Is the parameter a numeric

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

int ISNUM(order)

<input parameters>

int order; /\* Placement in actual parameter list \*/

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is a numeric value. It returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is a numeric value, or .F. otherwise.

#### Example Recital program:

```
m_value=534
m_isnum=check_num(m_value)
return
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
check_num()
{
    _retl( ISNUM());
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_parnd(), \_parni(), parnl(), \_retn(), retni(), retl(), PCOUNT



## ISOBJECT()

---

### PURPOSE

Is the parameter an object

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      ISOBJECT(order)
```

<input parameters>

```
int      order;          /* Placement in actual parameter list          */
```

<output parameters>

none

### DESCRIPTION

The function checks if the specified parameter is a Recital OBJECT. It returns 1 if true, or 0 if false. The order specifies the actual placements in the parameter list.

### EXAMPLE

The following example returns .T. if the first parameter passed is a Recital OBJECT, or .F. otherwise.

#### Example Recital program:

```
m_values = NEWOBJECT('MyClass')
m_isobj=check_obj(m_values)
return
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
check_obj()
{
    _retl(ISOBJECT(1));
}
```

### SEE ALSO

\_paro(), \_reto()

## PCOUNT

---

### PURPOSE

The number of parameters passed

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      PCOUNT
```

```
<input parameters>
```

```
none
```

```
<output parameters>
```

```
none
```

### DESCRIPTION

The function returns the number of parameters passed.

### EXAMPLE

The following example returns the number of parameters passed to the 'C' function.

#### Example Recital program:

```
m_value=534
m_num_par=numpar(m_value)
return
```

#### Example 'C' function:

```
#include "dbapi.h"

dbapi_num_par()
{
    _retl(PCOUNT);
}
```

### SEE ALSO

\_parinfo(), \_parinfo()

## COMMAND()

---

### PURPOSE

Execute a Recital command

### SYNONYM

api\_command()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      COMMAND(command)
```

<input parameters>

```
char      *command;      /* Address of a buffer containing a valid Recital command */
```

<output parameters>

none

### DESCRIPTION

The COMMAND() function will execute the specified Recital command from within your 'C' function. This function returns 0 if successful, otherwise a Recital error number.

### EXAMPLE

The following example will open a Recital database called accounts.

#### Example Recital program:

```
m_result=open("use accounts")
return
```

#### Example 'C' function:

```
#include "dbapi.h"
```

```
dbapi_open()
{
    if (_parinfo(1) == APICTYPE) {
        _retni(COMMAND(_parc(1)));
    } else {
        _retni(-1);
    }
}
```

### SEE ALSO

EXPRESSION()

## ERROR()

---

### PURPOSE

Used to define and signal a user-defined error

### SYNONYM

api\_error()

### SYNOPSIS

#include "dbapi.h"

```
void    ERROR(number, message)
```

<input parameters>

```
int      number;          /* Error number          */
char     *message;        /* Address of a buffer containing error message */
```

<output parameters>

none

### DESCRIPTION

The ERROR() function is used to define and signal a user-defined error. When the ERROR() function is called, a run-time error occurs and the current error handler will be called. The current error handler is an active ON ERROR <command>, a TRY...CATCH block, or the default Recital error handler which creates an error.mem file.

### EXAMPLE

The following example calls the error handler if the wrong number of parameters has been passed.

#### Example Recital program:

```
if pcount() <> 3
    error 10000, "Invalid parameter count"
endif
```

#### Example 'C' function:

```
#include "dbapi.h"

dbapi_checkparams()
{
    if (PCOUNT != 3) {
        ERROR(10000, "Invalid parameter count");
    }
}
```

### SEE ALSO

COMMAND()

## EXPRESSION()

---

### PURPOSE

Evaluate a Recital expression

### SYNONYM

api\_expression()

### SYNOPSIS

#include "dbapi.h"

int        EXPRESSION(result, exp)

<input parameters>

char    \*exp;                                /\* Address of a buffer containing a valid Recital command    \*

<output parameters>

API\_EXPRESSION result;                    /\* Expression result     \*/

### DESCRIPTION

The FUNCTION() function evaluates the specified Recital expression and returns the result. This function returns 0 if successful, otherwise a Recital error number.

The API Expression result structure is defined as follows:

```
struct API_EXPRESSION{
    int        errno;                        /* RECITAL error() number        */
    char       type;                        /* data type of result           */
    int        width;                       /* width of result                */
    int        decimal;                     /* decimal places                */
    char       *character;                  /* CHARACTER result              */
    double     number;                      /* NUMERIC result                */
    char       logical;                     /* LOGICAL result                */
    unsigned long date;                     /* Recital DATE result           */
    DATETIME   datetime;                   /* Recital DATETIME result       */
    CURRENCY   currency;                   /* Recital CURRENCY result       */
}
```

The data type result will be any one of the following:

RETURN VALUE	DESCRIPTION
C	character or memo data type
N	numeric data type
D	date data type
L	logical data type
T	datetime data type
Y	currency data type

**EXAMPLE**

The following example evaluates the specified expression and returns the result.

**Example Recital program:**

```
use accounts
m_result=evaluate("ord_value - paid_value")
return
```

**Example 'C' function:**

```
#include "dbapi.h"

dbapi_evaluate()
{
    struct    API_EXPRESSION result;

    if (_parinfo(1) == API_CTYPE) {
        EXPRESSION(&result, _parc(1));

        if (result.errno != 0) {
            _retni(-1);
        } else {
            _retni(result.number);
        }
    } else {
        _retni(-1);
    }
}
```

**SEE ALSO**

COMMAND()

## ARRAY\_ALEN()

---

### PURPOSE

Return the number of array elements

### SYNONYM

api\_array\_alen()

### SYNOPSIS

#include "dbapi.h"

int       ARRAY\_ALEN(string)

<input parameters>

char     \*string;           /\* Address of a buffer containing the name of a Recital array \*/

<output parameters>

none

### DESCRIPTION

The ARRAY\_ALEN() function will return the number of elements in the specified Recital array.

NOTE: Recital array elements start at 1 and not 0 as in 'C'.

### EXAMPLE

The following example returns the number of elements in the given array.

#### Example Recital program:

```
declare value[20]
declare new_value[get_alen("value")]
return
```

#### Example 'C' function:

```
#include "dbapi.h"

dbapi_array_alen ()
{
    int       numarray;

    if (_parinfo(1) == API_ATYPE ) {
        numarray = ARRAY_ALEN(_parc(1));
    } else {
        numarray = 0;
    }

    _retni(numarray);
}
```

### SEE ALSO

\_parinfo(), ALENGTH(), ISARRAY(), ARRAY\_DEFINE(), ARRAY\_LOOKUP(), ARRAY\_UPDATE()

## ARRAY\_DEFINE()

---

### PURPOSE

Define a Recital array

### SYNONYM

api\_array\_define()

### SYNOPSIS

#include "dbapi.h"

```
int    ARRAY_DEFINE(arrayname, level, dim1, dim2)
```

<input parameters>

```
char   arrayname;    /* Address of a buffer containing the name of a Recital array */
int    level;        /* Declaration level of array */
int    dim1;         /* Dimension value 1 */
int    dim2;         /* Dimension value 2 */
```

<output parameters>

none

### DESCRIPTION

The ARRAY\_DEFINE() function will define a two dimensional Recital array with number of elements specified. 'dim1' represents the number of rows and 'dim2' represents the number of columns. If 'dim2' is zero then a one dimensional array is defined. Two dimensional arrays can be accessed as one dimensional using a single element number.

NOTE: Recital array elements start at 1 and not 0 as in 'C'.

The declaration level specifies how the array is declared and can be one of the following:

VALUE	DESCRIPTION
API_PRIVATE	Private to the Recital calling procedure
API_PUBLIC	Public to all Recital higher level procedures

### EXAMPLE

The following example creates a public Recital two dimensional array called value.

```
#include "dbapi.h"
```

```
dbapi_array_define()
```

```
{
    int    dim1;
    int    dim2;

    dim1 = 10;
    dim2 = 20;

    ARRAY_DEFINE("value", API_PUBLIC, dim1, dim2);

    _retl(-1);
}
```



**SEE ALSO**

`_parinfo()`, `ALENGTH()`, `ISARRAY()`, `ARRAY_ALEN()`, `ARRAY_LOOKUP()`, `ARRAY_UPDATE()`

## ARRAY\_LOOKUP()

---

### PURPOSE

Look up value in a Recital array

### SYNONYM

api\_array\_lookup()

### SYNOPSIS

#include "dbapi.h"

```
struct API_MEMVAR  ARRAY_LOOKUP(arrayname, element)
```

<input parameters>

```
char    *arrayname;    /* Address of a buffer containing the name of a Recital array */
int     element        /* Number of element */
```

<output parameters>

none

### DESCRIPTION

The ARRAY\_DEFINE() function returns the data type, value, display width and number of decimal places of the specified element number in the Recital array.

NOTE: Recital array elements start at 1 and not 0 as in 'C'.

A NULL value is returned if the array has not been previously been defined or the specified array element is out of range, otherwise the results are stored in the following API\_MEMVAR data structure.

```
union  API_UNION {
    char    *info_character;    /* char memory variable */
    char    info_logical;      /* logical memory variable */
    double  info_number;       /* numeric memory variable */
    unsigned long info_date;    /* date memory variable */
    DATETIME info_datetime;    /* datetime memory variable */
    CURRENCY info_currency;    /* currency memory variable */
};
```

```
struct  API_MEMVAR {
    char    type;              /* data type of memory variable */
    union  API_UNION value;    /* value of the memory variable */
    int     width;             /* display width */
    int     decimal;           /* display # of decimal places */
};
```

The API\_MEMVAR type specifies both the data type and which API\_UNION value will be used to return the value of the array element.

The API\_MEMVAR width returns the length of the array element. A decimal place may be returned for a number, but will be 0 for other data types.

An array element can be defined as any one of the following.

TYPE	OUTPUT	WIDTH	DESCRIPTION
C	info_character	1 – 8192	address of a buffer containing a character string
D	info_date	8	unsigned long representing a Recital Date
L	info_logical	1	character of either 'T' for true or 'F' for false
N	info_number	1 – 16	a value stored in a double
T	info_datetime	sizeof(DATETIME)	RCT_DATETIME_DEF structure
Y	info_currency	sizeof(CURRENCY)	RCT_CURRENCY_DEF structure

### EXAMPLE

The following example looks up the array element and returns the result.

```
#include <stdio.h>
#include "dbapi.h"

dbapi_array_lookup()
{
    struct API_MEMVAR    result

    if (_parinfo(1) != API_CTYPE || _parinfo(2) !=API_NTTYPE) {
        _retc("")
    }

    result = ARRAY_LOOKUP(_parc(1), _parni(2));

    switch (result->type)
    {
        case 'C':
            retc( result->value.infocharacter );
        case 'D':
            _retc( DATE_DTOS(result->value.info_date));
        case 'N':
            CHAR_STR(buffer,
                result->value.info_number,
                result->width,
                result->decimal);
            _retc(buffer);
        case 'L':
            _retc((result->value.info_logical=='T') ? "True" : "False");
        case 'T':
            _retc( DATE_TTOS(result->value.info_datetime));
        case 'Y':
            _retc( CURR_YTOS(result->value.info_currency));
    }
}
```

### SEE ALSO

`_parinfo()`, `ALENGTH()`, `ISARRAY()`, `ARRAY_ALEN()`, `ARRAY_DEFINE()`, `ARRAY_UPDATE()`

## ARRAY\_UPDATE()

---

### PURPOSE

Update the value in a Recital array

### SYNONYM

api\_array\_update()

### SYNOPSIS

#include "dbapi.h"

```
int      ARRAY_UPDATE(arrayname, element, type, width, decimal, string, number, logical, date,
                    datetime, currency)
```

<input parameters>

char	*arrayname;	/* Address of a buffer containing the name of a Recital array	*/
int	element	/* Element number	*/
char	type;	/* Type of field	*/
int	width;	/* Display width	*/
int	decimal	/* Display # decimals	*/
char	string;	/* Address of a buffer containing a character string	*/
double	*number	/* Number	*/
char	logical;	/* Logical	*/
unsigned long	date;	/* Recital date	*/
DATETIME	datetime;	/* Recital datetime	*/
CURRENCY	currency;	/* Recital currency	*/

<output parameters>

none

### DESCRIPTION

The ARRAY\_UPDATE() function will update the specified element in the Recital array. The name and element number are used to specify the array element to update

Recital will only recognize the first ten characters on a array name as being unique. The array names are not case sensitive and must have been previously defined before updating, or the ARRAY\_UPDATE() function will return a value of -1.

NOTE: Recital array elements start at 1 and not 0 as in 'C'.

Array elements can be updated with data of a different type to their current value. The variable type parameter specifies both the data type and which input parameter value will be used to update the memory variable.

The width parameter is used to specify length of the array element. A decimal place can be specified for a number, but should be 0 for other data types.

A array element can be defined as any one of the following.

TYPE	OUTPUT	WIDTH	DESCRIPTION
C	info_character	1 – 8192	address of a buffer containing a character string
D	info_date	8	unsigned long representing a Recital Date
L	info_logical	1	character of either 'T' for true or 'F' for false
N	info_number	1 – 16	a value stored in a double
T	info_datetime	sizeof(DATETIME)	RCT_DATETIME_DEF structure
Y	info_currency	sizeof(CURRENCY)	RCT_CURRENCY_DEF structure

#### EXAMPLE

The following example will define a six element array and update each array element with the specified values.

```
#include "dbapi.h"
```

```
dbapi_array_update()
```

```
{
    int          rc;
    double       numeric;
    char         string[8];
    char         logical;
    unsigned long date;
    DATETIME     datetime;
    CURRENCY     currency;

    strcpy (string, "Recital");
    date = DATE_CTOD (DATE_DATE());
    logical = 'T';
    numeric = 23.5;
    memcpy (&datetime, DATE_STOT (DATE_DATETIME()), sizeof (DATETIME));
    memcpy (&currency, CURR_STOY ("364.984"), sizeof (CURRENCY));

    COMMAND ("release all");
    ARRAY_DEFINE ("memvar", API_PUBLIC, 6,1);

    rc = ARRAY_UPDATE ("memvar", 1, 'C', 7, 0,
                      string, numeric,
                      logical, date,
                      datetime, currency);
    rc = ARRAY_UPDATE ("memvar", 2, 'N', 10, 2,
                      string, numeric,
                      logical, date,
                      datetime, currency);
    rc = ARRAY_UPDATE ("memvar", 3, 'L', 1, 0,
                      string, numeric,
                      logical, date,
                      datetime, currency);
    rc = ARRAY_UPDATE ("memvar", 4, 'D', 8, 0,
                      string, numeric,
                      logical, date,
                      datetime, currency);
    rc = ARRAY_UPDATE ("memvar", 5, 'T', sizeof (DATETIME), 0,
                      string, numeric,
                      logical, date,
                      datetime, currency);
}
```

```

rc = ARRAY_UPDATE("memvar",6, 'Y',sizeof(CURRENCY),4,
                  string, numeric,
                  logical, date,
                  datetime, currency);

    _retl( (rc ==0) ? 1 : 0 );
}

```

**SEE ALSO**

\_parinfo(), ALENGTH(), ISARRAY(), ARRAY\_ALEN(), ARRAY\_DEFINE(), ARRAY\_LOOKUP()

## BLOB\_READ()

---

### PURPOSE

Read a blob

### SYNONYM

api\_blob\_read()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      BLOB_READ(fldname, blob, maxsize)
```

<input parameters>

```
char      *fldname;      /* Address of a buffer containing a blob field name */
int       maxsize;       /* Maximum size of the blob */
```

<output parameters>

```
unsigned char *blob;     /* Address of a buffer containing the blob data */
```

### DESCRIPTION

The BLOB\_READ() function will read a blob from the memo file for the fieldname specified in the current record.

### EXAMPLE

The following example reads a blob into the buffer from the specified blob field.

```
#include "dbapi.h"
```

```
dbapi_blob_read()
{
    int      blobsize;
    char     blobuf[8000];

    if (_parinfo(1) == API_CTYPE) {
        blobsize = BLOB_SIZE(_parc(1));
        BLOB_READ(_parc(1), blobuf, blobsize);
    }
}
```

### SEE ALSO

BLOB\_RECLAIM(), BLOB\_SIZE(), BLOB\_UPDATE(), BLOB\_WRITE()

## BLOB\_RECLAIM()

---

### PURPOSE

Reclaim the space occupied in the memo file

### SYNONYM

api\_blob\_reclaim()

### SYNOPSIS

#include "dbapi.h"

```
int      BLOB_RECLAIM(fldname)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing a blob field name */
```

<output parameters>

none

### DESCRIPTION

The BLOB\_RECLAIM() function reclaims the space occupied by a blob in the memo file for the specified field in the current record.

### EXAMPLE

The following example deletes the blob on the current record for the blob field specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_blob_reclaim()
{
    int      rc;

    if (_parinfo(1) == API_CTYPE) {
        rc = BLOB_RECLAIM(_parc(1));
    } else {
        rc = -1;
    }

    _retni( rc )
}
```

### SEE ALSO

BLOB\_READ(), BLOB\_SIZE(), BLOB\_UPDATE(), BLOB\_WRITE()



## BLOB\_SIZE()

---

### PURPOSE

Return the size of a blob

### SYNONYM

api\_blob\_size()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      BLOB_SIZE(fldname)
```

<input parameters>

```
char      *fldname;      /* Address of a buffer containing a blob field name */
```

<output parameters>

none

### DESCRIPTION

The BLOB\_SIZE() function will return the number of bytes the blob occupies in the memo file for the specified field in the current record.

It will return - 1 if the blob field is empty on the current record.

### EXAMPLE

The following example will return the size of a blob field specified as the first parameter for the current record of the database.

```
#include "dbapi.h"
```

```
dbapi_blob_size()
{
    int      rc;

    if (_parinfo(1) == API_CTYPE) {
        rc = BLOB_SIZE( _parc(1) );
    } else {
        rc = -1;
    }

    _retni( rc );
}
```

### SEE ALSO

BLOB\_READ(), BLOB\_RECLAIM(), BLOB\_UPDATE(), BLOB\_WRITE()

## BLOB\_UPDATE()

---

### PURPOSE

Update a blob

### SYNONYM

api\_blob\_update()

### SYNOPSIS

#include "dbapi.h"

```
int      BLOB_UPDATE(fldname, blob, size)
```

<input parameters>

char	*fldname;	/* Address of a buffer containing a blob field name	*/
unsigned char	*blob;	/* Address of a buffer containing the blob data	*/
int	size;	/* Blob size	*/

<output parameters>

none

### DESCRIPTION

The BLOB\_SIZE() function updates a blob field from the specified buffer and overwrites the space used by the blob in the current record.

### EXAMPLE

The following example updates the blob field specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_blob_update()
```

```
{
    int      result;

    if (_parinfo(1) == API_CTYPE && _parinfo(2) == API_CTYPE) {
        if (!DBF_ISEXCLUSIVE()) DBF_LOCKR(DBF_RECNO());
        result = BLOB_UPDATE(_parc(1), (unsigned char) _parc(2), _parcsiz(2) );
        if (result) result = DBF_UPDATE();
        if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(DBF_RECNO());
    } else {
        result = -1;
    }

    _retl( (result) ? 1 : 0 );
}
```

### SEE ALSO

BLOB\_READ(), BLOB\_RECLAIM(), BLOB\_SIZE(), BLOB\_WRITE()

## BLOB\_WRITE()

---

### PURPOSE

Write a blob

### SYNONYM

api\_blob\_write()

### SYNOPSIS

#include "dbapi.h"

```
int      BLOB_WRITE(fldname, blob, size)
```

<input parameters>

char	*fldname;	/* Address of a buffer containing a blob field name	*/
unsigned char	blob;	/* Address of a buffer containing blob data	*/
int	size;	/* Blob size	*/

<output parameters>

none

### DESCRIPTION

The BLOB\_WRITE() function writes a blob field from the specified buffer into a memo file, and updates the current record buffer.

### EXAMPLE

The following example updates the blob field specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_blob_write()
```

```
{
    int      result;

    if (_parinfo(1) == API_CTYPE && _parinfo(2) == API_CTYPE) {
        if (!DBF_ISEXCLUSIVE()) DBF_LOCKR(DBF_RECNO());
        result = BLOB_WRITE( _parc(1), (unsigned char) _parc(2), _parcsiz(2) );
        if (result) result = DBF_UPDATE();
        if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(DBF_RECNO());
    } else {
        result = -1;
    }

    _retl( (result) ? 1 : 0 );
}
```

### SEE ALSO

BLOB\_READ(), BLOB\_RECLAIM(), BLOB\_SIZE(), BLOB\_UPDATE()

## CHAR\_ALLTRIM()

---

### PURPOSE

Remove leading and trailing blanks from a string

### SYNONYM

api\_char\_alltrim()

### SYNOPSIS

#include "dbapi.h"

char CHAR\_ALLTRIM(string)

<input parameters>

char \*string; /\* Address of a buffer containing character string \*/

<output parameters>

none

### DESCRIPTION

The CHAR\_ALLTRIM() function removes all leading and trailing blanks from the specified character string.

### EXAMPLE

The following example trims the leading and trailing blanks from the first parameter passed.

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

```
#include "dbapi.h"
```

```
dbapi_char_alltrim()
```

```
{
    char    string[1025]

    if (_parinfo(1) == API_CTYPE) {
        strcpy(string, CHAR_ALLTRIM(_parc(1) ));
    } else {
        strcpy(string, "");
    }

    _retc( string );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD()

## CHAR\_LOWER()

---

### PURPOSE

Convert character string to lower case

### SYNONYM

api\_char\_lower()

### SYNOPSIS

#include "dbapi.h"

char CHAR\_LOWER(string)

<input parameters>

char \*string; /\* address of a buffer containing character string \*/

<output parameters>

none

### DESCRIPTION

The CHAR\_LOWER() function converts the specified character string to lower case.

### EXAMPLE

The following example converts the first parameter passed to lower case.

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

```
#include "dbapi.h"
```

```
dbapi_char_lower()
```

```
{
    char    string[1025]

    if (_parinfo(1) == API_CTYPE) {
        strcpy(string, CHAR_LOWER( _parc(1) ));
    } else {
        strcpy(string, "");
    }

    _retc( string );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD()

## CHAR\_LPAD()

---

### PURPOSE

Pad out a character string to the defined length from the left

### SYNONYM

api\_char\_lpad()

### SYNOPSIS

```
#include "dbapi.h"
```

```
char    CHAR_LPAD(buffer, string, len, padch)
```

<input parameters>

```
char    *string;          /* Address of a buffer containing character string */
int      len;              /* Total length of the returned string */
char     padch;           /* Character to pad with */
```

<output parameters>

```
char    *buffer;          /*Address of the buffer where the result is returned */
```

### DESCRIPTION

The CHAR\_LPAD() function right justifies a character string and pads out the left of the string, to the total length specified with the specified pad character.

### EXAMPLE

The following example pads the left of the first parameter passed to the total length specified by the second parameter with the character passed in the third parameter.

```
#include "dbapi.h"
```

```
dbapi_char_lpad()
{
    char    buffer[1025]

    if (_parinfo(1) == API_CTYPE &&
        _parinfo(2) == API_NTTYPE &&
        _parinfo(3) == API_CTYPE) {

        CHAR_LPAD( buffer, _parc(1), _parni(2), _parc(3) );
    } else {
        strcpy(buffer,"");
    }

    _retc( buffer );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD()

## CHAR\_LTRIM()

---

### PURPOSE

Remove leading blank spaces from a string

### SYNONYM

api\_char\_ltrim()

### SYNOPSIS

#include "dbapi.h"

```
char    CHAR_LTRIM(string,)
```

<input parameters>

```
char    *string;          /* Address of a buffer containing character string */
```

<output parameters>

none

### DESCRIPTION

The CHAR\_LTRIM() function removes leading blank spaces from the specified character string.

### EXAMPLE

The following example trims all leading blank spaces from the first parameter passed..

```
#include "dbapi.h"
```

```
dbapi_char_ltrim()
{
    char    string[1025]

    if ( _parinfo(1) == API_CTYPE) {
        strcpy(string, CHAR_LTRIM( _parc(1) ));
    } else {
        strcpy(string, "");
    }

    _retc( string );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD()

## CHAR\_RPAD()

---

### PURPOSE

Pad out a character string to the defined length from the right

### SYNONYM

api\_char\_rpad()

### SYNOPSIS

#include "dbapi.h"

```
char    CHAR_RPAD(buffer, string, len, padch)
```

<input parameters>

```
char    *string;          /* Address of a buffer containing character string    */
int      len;              /* Total length of the returned string                  */
char     padch;           /* Character to pad with                                */
```

<output parameters>

```
char    *buffer;          /*Address of the buffer where the result is returned    */
```

### DESCRIPTION

The CHAR\_RPAD() function left justifies a character string and pads out the right of the string, to the total length specified with the specified pad character.

### EXAMPLE

The following example pads the right of the first parameter passed to the total length specified by the second parameter with the character passed in the third parameter.

```
#include "dbapi.h"
```

```
dbapi_char_rpad()
{
    char    buffer[1025]

    if (_parinfo(1) == API_CTYPE &&
        _parinfo(2) == API_NTTYPE &&
        _parinfo(3) == API_CTYPE) {

        CHAR_LPAD( buffer, _parc(1), _parni(2), _parc(3) );
    } else {
        strcpy(buffer,"");
    }

    _retc( buffer );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD()



## CHAR\_RTRIM()

---

### PURPOSE

Remove leading blank spaces from a string

### SYNONYM

api\_char\_rtrim()

### SYNOPSIS

#include "dbapi.h"

```
char    *CHAR_RTRIM(string,)
```

<input parameters>

```
char    *string;           /* Address of a buffer containing character string    */
```

<output parameters>

none

### DESCRIPTION

The CHAR\_RTRIM() function remove trailing blank spaces from the specified character string.

### EXAMPLE

The following example trims all leading blank spaces from the first parameter passed..

```
#include "dbapi.h"
```

```
dbapi_char_rtrim()
{
    char    string[1025]

    if ( _parinfo(1) == API_CTYPE ) {
        strcpy(string, CHAR_RTRIM( _parc(1) ));
    } else {
        strcpy(string, "");
    }

    _retc( string );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_RPAD(), CHAR\_LTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_STOD()

## CHAR\_STR()

---

### PURPOSE

Convert a number to a character string

### SYNONYM

api\_char\_str()

### SYNOPSIS

#include "dbapi.h"

```
char    *CHAR_STR(buffer, number, width, decimals)
```

<input parameters>

double	number;	/* Number to convert	*/
int	width;	/* Total width of the returned	*/
int	decimals;	/* Number of decimal places	*/

<output parameters>

char	buffer;	/* Address of the buffer where the result is returned	*/
------	---------	---	----

### DESCRIPTION

The CHAR\_STR() function converts the numeric value to right justified character string for the specified width and rounded to the number of decimal places.

### EXAMPLE

The following example converts a numeric parameter passed from Recital to a 10 character string pointer with the decimal places rounded to 2.

```
#include "dbapi.h"
```

```
dbapi_char_str()
{
    char    buffer[1025]

    if ( _parinfo(1) == API_NTTYPE) {
        CHAR_STR( buffer, _parnd(1), 10, 2);
    } else {
        strcpy(buffer,"");
    }

    _retc( buffer );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_UPPER(), DATE\_STOD()

## CHAR\_UPPER()

---

### PURPOSE

Convert character string to upper case

### SYNONYM

api\_char\_upper()

### SYNOPSIS

#include "dbapi.h"

```
char    *CHAR_UPPER(string)
```

<input parameters>

```
char    string;          /* Address of a buffer containing character string */
```

<output parameters>

none

### DESCRIPTION

The CHAR\_UPPER() function converts the specified character string to upper case.

### EXAMPLE

The following example converts the parameter passed to upper case.

```
#include "dbapi.h"
```

```
dbapi_char_upper()
{
    char    string[1025];

    if ( _parinfo(1) == API_CTYPE ) {
        strcpy(string, CHAR_UPPER( _parc(1) ));
    } else {
        strcpy(string, "");
    }

    _retc( string );
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), DATE\_STOD()

## **CURR\_STOY()**

---

### **PURPOSE**

Return a character string as a CURRENCY data type

### **SYNONYM**

api\_curr\_stoy()

### **SYNOPSIS**

#include "dbapi.h"

CURRENCY     CURR\_STOY(string)

<input parameters>

char     string;                   /\* Address of a buffer containing a valid currency string     \*/

<output parameters>

none

### **DESCRIPTION**

The CURR\_STOY() function will return the specified currency value in Recital CURRENCY storage format.

The character expression must be in the format "99999999999999.9999".

### **EXAMPLE**

The following example converts the string "0202.99" to an currency value, then updates the memory variable specified in the first parameter as a Recital currency type.

```
#include "dbapi.h"
```

```
dbapi_curr_stoy()
```

```
{
    CURRENCY               curr;
    struct API_MEMVAR       *tmpbuf;

    curr = CURR_STOY( "0202.99" );

    if (_parinfo(1) == API_CTYPE) {
        MEMVAR_UPDATE(_parc(1), 'T', sizeof(CURRENCY), 0,
            tmpbuf->value.info_character,
            tmpbuf->value.info_number,
            tmpbuf->value.info_logical,
            tmpbuf->value.info_date ,
            tmpbuf->value.info_datetime,
            curr );
    }
}
```

### **SEE ALSO**

\_parys(), \_retys(), ISCURRENCY(), CURR\_YTOS()

## **CURR\_YTOS()**

---

### **PURPOSE**

Return a currency value as a character string

### **SYNONYM**

api\_curr\_ytos()

### **SYNOPSIS**

#include "dbapi.h"

```
char    *CURR_YTOS(currency)
```

<input parameters>

```
CURRENCY    currency;                /* Recital currency          */
```

<output parameters>

none

### **DESCRIPTION**

CURR\_YTOS() function will return the given currency value as a character.

### **EXAMPLE**

The following example converts a currency value to a character pointer.

```
#include "dbapi.h"
```

```
dbapi_curr_ytos()
```

```
{
    char          curr[21];

    if (_parinfo(1) != API_YTYPE) {
        _retc("");
    }

    strcpy(curr, CURR_YTOS(_pards(1) );

    _retc( curr );
}
```

### **SEE ALSO**

\_parys(), \_retys(), ISCURRENCY(), CURR\_STOY()

## DATE\_AMPM()

---

### PURPOSE

Return the time based on a 12 hour clock

### SYNONYM

api\_date\_ampm()

### SYNOPSIS

```
#include "dbapi.h"
```

```
char    *DATE_AMPM()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DATE\_AMPM() function returns the current time based on a 12 hour clock as a character string followed by "am" or " pm".

### EXAMPLE

The following example stores the current time in ampm format to the character pointer current\_time.

```
#include "dbapi.h"
```

```
dbapi_date_ampm()
{
    char    *current_time;

    current_time = DATE_AMPM();

    retc( current_time );
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_CDOW()

---

### PURPOSE

Return character day of the week

### SYNONYM

api\_date\_cdown()

### SYNOPSIS

#include "dbapi.h"

char \*DATE\_CDOW(date)

<input parameters>

unsigned long date; /\* Recital date \*/

<output parameters>

none

### DESCRIPTION

The DATE\_CDOW() function will return the name of the day of the week from the specified date as a character string. The name is returned as a proper noun.

### EXAMPLE

The following example returns "Weekend" if Saturday or Sunday is returned, otherwise "Workday" is returned.

```
#include "dbapi.h"
```

```
dbapi_date_cdown()
```

```
{
    char      day_type[8];
    char      *day_week;
    unsigned long date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }
    date = DATE_STOD(_pards(1));
    day_week = DATE_CDOW(date);
    if (strcmp(day_week, "Sunday") == 0 || strcmp(day_week, "Saturday") == 0)
        strcpy(day_type, "Weekend");
    else
        strcpy(day_type, "Workday");

    _retc(day_type);
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_AMPM(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_CMONTH()

---

### PURPOSE

Return character name of the month

### SYNONYM

api\_date\_month()

### SYNOPSIS

#include "dbapi.h"

```
char    *DATE_CMONTH(date)
```

<input parameters>

```
unsigned long    date;          /* Recital date          */
```

<output parameters>

none

### DESCRIPTION

The DATE\_CMONTH() function will return the name of the month from the specified date as a character pointer. The name is returned as a proper noun.

### EXAMPLE

The following example stores the name of the month from today's date to the character pointer "month".

```
#include "dbapi.h"
```

```
dbapi_date_month()
{
    char            *month;
    unsigned long    date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD( _pards(1) );
    month = DATE_CMONTH(date);

    if (!strcmp(month, "December") )
    {
        /* commands      */
    }

    _retc(month);
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()



## DATE\_CTOD()

---

### PURPOSE

Return the date as a character string

### SYNONYM

api\_date\_ctod()

### SYNOPSIS

#include "dbapi.h"

unsigned long     DATE\_CTOD(string)

<input parameters>

char     string;                   /\*address of a buffer containing a valid date string     \*/

<output parameters>

none

### DESCRIPTION

The DATE\_CTOD() function will return the specified date as an unsigned long which is the Recital date storage format.

The character expression must be in the form month/day/year if you have issued the Recital SET DATE AMERICAN command, or day/month/year if you have entered the Recital SET DATE BRITISH command, or must be correctly formatted if any other data type has been specified with the Recital SET DATE command. The format must also match the current SET CENTURY setting, four digit years if SET CENTURY is ON, two digit years if SET CENTURY is OFF.

If an invalid date string is specified then a value of zero will be returned.

### EXAMPLE

The following example converts the date "02/02/99" to an unsigned long, then updates the memory variable specified in the first parameter as a Recital date type.

```
#include "dbapi.h"
```

```
dbapi_date_ctod()
```

```
{
    unsigned long             date;
    struct API_MEMVAR        *tmpbuf;

    date = DATE_CTOD( "02/02/99" );

    if (_parinfo(1) == API_CTYPE) {
        MEMVAR_UPDATE(_parc(1), 'D', 2, 0,
            tmpbuf->value.info_character,
            tmpbuf->value.info_number,
            tmpbuf->value.info_logical,
            date );
    }
}
```

**SEE ALSO**

`_parinfo()`, `_parinfo()`, `_pards()`, `_retds()`, `ISDATE()`, `DATE_AMPM()`, `DATE_CDOW()`,  
`DATE_CMONTH()`, `DATE_DATE()`, `DATE_DAY()`, `DATE_DOW()`, `DATE_DTOC()`, `DATE_DTOS()`,  
`DATE_MONTH()`, `DATE_STOD()`, `DATE_TIME()`, `DATE_YEAR()`

## DATE\_DATE()

---

### PURPOSE

Return the current system date

### SYNONYM

api\_date\_date()

### SYNOPSIS

#include "dbapi.h"

char \*DATE\_DATE()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DATE\_DATE() function will return the current system date as a character string. The date format returned is specified with the Recital SET DATE and SET CENTURY commands.

### EXAMPLE

The following example stores the name of the month from today's date to the character pointer "month".

```
#include "dbapi.h"
```

```
dbapi_date_date()
{
    char *month;
    unsigned long date;

    date = DATE_CTOD(DATE_DATE() );
    month = DATE_CMONTH( date

    if (!strcmp(month, "December") )
    {
        /* commands */
    }

    _retc(month);
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_DATETIME()

---

### PURPOSE

Return the current system date and time

### SYNONYM

api\_date\_datetime()

### SYNOPSIS

#include "dbapi.h"

char \*DATE\_DATETIME()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DATE\_DATETIME() function will return the current system date and time as a character string. The date and time format returned is specified with the Recital SET DATE, SET CENTURY, SET MARK, SET HOURS and SET SECONDS commands.

### EXAMPLE

The following example returns the current system date and time as a string.

```
#include "dbapi.h"
```

```
dbapi_date_datetime()
{
    _retc(DATE_DATETIME());
}
```

### SEE ALSO

\_parts(), \_retts(), ISDATETIME(), DATE\_STOT(), DATE\_TTOS()

## DATE\_DAY()

---

### PURPOSE

Return the day of the month

### SYNONYM

api\_date\_date()

### SYNOPSIS

#include "dbapi.h"

```
int    DATE_DAY(date)
```

<input parameters>

```
unsigned long    date;          /* Recital date          */
```

<output parameters>

none

### DESCRIPTION

The DATE\_DAY() function will return the day of the month from the specified date as an integer.

### EXAMPLE

The following example converts today's date to integer value representing the day of the month.

```
#include "dbapi.h"
```

```
dbapi_date_day()
```

```
{
    int          day;
    unsigned long date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD(_pards(1));
    day = DATE_DAY( date );

    if ( day == 1 ) {
        _retc( "First day of the Month");
    }
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retc(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(),  
DATE\_CMONTH(), DATE\_CTOD(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(),  
DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_DOW()

---

### PURPOSE

Return the day of the week

### SYNONYM

api\_date\_day()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int    DATE_DOW(date)
```

<input parameters>

```
unsigned long    date;          /* Recital date          */
```

<output parameters>

none

### DESCRIPTION

The DATE\_DOW() function will return the number representing the day of the week from the specified date as an integer. Starting with Sunday as day 1, and ending with Saturday as day 7.

### EXAMPLE

The following example converts today's date to an integer value representing the day of the month.

```
#include "dbapi.h"
```

```
dbapi_date_dow()
{
    int            day;
    unsigned long  date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD(_pards(1));
    day = DATE_DOW( date );

    if ( day == 1) {
        _retc( "Today is Sunday");
    }
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_DTOC()

---

### PURPOSE

Return a date as a character string

### SYNONYM

api\_date\_dtoc()

### SYNOPSIS

#include "dbapi.h"

```
char    *DATE_DTOC(date)
```

<input parameters>

```
unsigned long    date           /* Recital date           */
```

<output parameters>

none

### DESCRIPTION

The DATE\_DTOC() function will return the specified date as a character string in the format specified by the Recital commands SET DATE and SET CENTURY.

### EXAMPLE

The following example converts today's date to character pointer.

```
#include "dbapi.h"
```

```
dbapi_date_dtoc()
{
    char    *day;
    unsigned long    date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD(_pards(1));
    day = DATE_DTOC( date );

    _retc( day );
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_DTOS()

---

### PURPOSE

Return date as a string

### SYNONYM

api\_date\_dtos()

### SYNOPSIS

#include "dbapi.h"

```
char    *DATE_DTOS(date)
```

<input parameters>

```
unsigned long    date           /* Recital date           */
```

<output parameters>

none

### DESCRIPTION

The DATE\_DTOS() function will return the specified date as a character string in the format "YYYYMMDD".

### EXAMPLE

The following example converts today's date to character pointer.

```
#include "dbapi.h"
```

```
dbapi_date_dtos()
{
    char    *day;
    unsigned long    date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD(_pards(1));
    day = DATE_DTOS( date );

    _retc( day );
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()



## DATE\_MONTH()

---

### PURPOSE

Return the month of the year

### SYNONYM

api\_date\_month()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DATE_MONTH(date)
```

<input parameters>

```
unsigned long    date          /* Recital date          */
```

<output parameters>

none

### DESCRIPTION

The DATE\_MONTH() function will return the numeric month of the year from the specified date as an integer.

### EXAMPLE

The following example converts today's date to an integer value representing the month of the year.

```
#include "dbapi.h"
```

```
dbapi_date_month()
{
    int          month;
    unsigned long date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD(_pards(1));
    day = DATE_MONTH( date );

    if ( month == 12) {
        /* commands      */
    }
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retc(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_STOD()

---

### PURPOSE

Return a character string as a date

### SYNONYM

api\_date\_stod()

### SYNOPSIS

```
#include "dbapi.h"
```

```
unsigned long    DATE_STOD(string)
```

<input parameters>

```
char    string;          /* Address of a buffer containing character string    */
```

<output parameters>

none

### DESCRIPTION

The DATE\_STOD() function will return the specified string in the format "YYYYMMDD" as an unsigned long, the Recital date format. This function can be used to convert data values for updating Recital date memory variables.

### EXAMPLE

The following example converts today's date specified in the character pointer date to an unsigned long.

```
#include "dbapi.h"
```

```
dbapi_date_stod()
{
    int            month;
    unsigned long  date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD(_pards(1));
    day = DATE_DAY( date );

    if ( month == 12) {
        /* commands    */
    }
}
```

### SEE ALSO

\_parc(), \_parclen(), \_parcsiz(), \_parinfo(), \_retc(), \_retclen(), ISCHAR(), CHAR\_ALLTRIM(), CHAR\_LOWER(), CHAR\_LPAD(), CHAR\_LTRIM(), CHAR\_RPAD(), CHAR\_RTRIM(), CHAR\_STR(), CHAR\_UPPER(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_STOD(), DATE\_TIME(), DATE\_YEAR()

## DATE\_STOT()

---

### PURPOSE

Return a character string as a datetime

### SYNONYM

api\_date\_stot()

### SYNOPSIS

#include "dbapi.h"

DATETIME     DATE\_STOT(string)

<input parameters>

char     string;                   /\* Address of a buffer containing character string     \*/

<output parameters>

none

### DESCRIPTION

The DATE\_STOT() function will return will return the specified string in the format "YYYYMMDDHHMMSS" as an unsigned long, the Recital date format. This function can be used to convert data values for updating Recital date memory variables.

### EXAMPLE

The following example converts today's date specified in the character pointer date to a DATETIME.

```
#include "dbapi.h"
```

```
dbapi_date_stot()
{
    DATETIME     datetime;

    if (_parinfo(1) != API_TTYPE) {
        _retc("");
    }

    datetime = DATE_STOT(_pards(1) );
}
```

### SEE ALSO

\_parts(), \_retts(), ISDATETIME(), DATE\_DATETIME(), DATE\_TTOS()

## DATE\_TIME()

---

### PURPOSE

Return the current time

### SYNONYM

api\_date\_time()

### SYNOPSIS

#include "dbapi.h"

char \*DATE\_TIME

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DATE\_TIME() function will return the current system in the 24 hour format HH:MM:SS.

### EXAMPLE

The following example stores the current system into the character pointer "time".

```
#include "dbapi.h"
```

```
dbapi_date_time()
```

```
{
```

```
    char *time()
```

```
    time = DATE_TIME();
```

```
    _retc( time );
```

```
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retds(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_YEAR()

## DATE\_TTOS()

---

### PURPOSE

Return datetime as a string

### SYNONYM

api\_date\_ttos()

### SYNOPSIS

#include "dbapi.h"

char \*DATE\_TTOS(date)

<input parameters>

DATETIME datetime; /\* Recital datetime \*/

<output parameters>

none

### DESCRIPTION

The DATE\_TTOS() function will return the specified datetime as a character string in the format "YYYYMMDDHHMMSS".

### EXAMPLE

The following example converts today's date and time to character pointer.

#include "dbapi.h"

```
dbapi_date_ttos()
{
    char datetime[23];

    if (_parinfo(1) != API_TTYPE) {
        _retc("");
    }

    strcpy( DATE_TTOS(_parts(1) );

    _retc( datetime );
}
```

### SEE ALSO

\_parts(), \_retts(), ISDATETIME(), DATE\_DATETIME(), DATE\_STOT()

## DATE\_YEAR()

---

### PURPOSE

Return the year from a date

### SYNONYM

api\_date\_year()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DATE_YEAR(date)
```

<input parameters>

```
unsigned long   date;           /* Recital date          */
```

<output parameters>

none

### DESCRIPTION

The DATE\_YEAR() function will return the numeric year value from the specified date as an integer.

### EXAMPLE

The following example converts today's date to an integer value representing the year.

```
#include "dbapi.h"
```

```
dbapi_date_year()
{
    int          year;
    unsigned long date;

    if (_parinfo(1) != API_DTYPE) {
        _retc("");
    }

    date = DATE_STOD(_pards(1));
    year = DATE_YEAR( date );

    if (year == 1992) {
        /* commands */
    }
}
```

### SEE ALSO

\_parinfo(), \_parinfo(), \_pards(), \_retc(), ISDATE(), DATE\_AMPM(), DATE\_CDOW(), DATE\_CMONTH(), DATE\_CTOD(), DATE\_DATE(), DATE\_DAY(), DATE\_DOW(), DATE\_DTOC(), DATE\_DTOS(), DATE\_MONTH(), DATE\_STOD(), DATE\_TIME()

## DBF\_ALIAS()

---

### PURPOSE

Return alias name

### SYNONYM

api\_dbf\_alias()

### SYNOPSIS

#include "dbapi.h"

char \*DBF\_ALIAS()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_ALIAS() function will return the alias name of the currently selected workarea as an upper case character string, or a NULL if no database is in use.

Recital provides you with 20 workareas by default, however, up to 100 may be set with the DB\_MAXWKA symbol.

### EXAMPLE

The following example returns the alias name of the current workarea, or executes the command specified in the first parameter if no database is open.

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

```
#include "dbapi.h"
```

```
dbapi_dbf_alias()
```

```
{
    char *alias;

    alias = DBF_ALIAS();

    if ( alias == NULL ) {
        COMMAND(_parc(1) );
        _retc("");
    } else {
        _retc( alias );
    }
}
```

### SEE ALSO

DBF\_DBF(), DBF\_FMT(), DBF\_INDEXKEY(), DBF\_INDEXORDER(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_NDX(), DBF\_SELECT(), DBF\_USED()

## DBF\_APPEND()

---

### PURPOSE

Append a record

### SYNONYM

api\_dbf\_append()

### SYNOPSIS

#include "dbapi.h"

```
int      *DBF_APPEND(blank)
```

<input parameters>

```
int      blank;          /*blank record  */
```

<output parameters>

none

### DESCRIPTION

The DBF\_APPEND() function will append a record into the currently selected database. The current record buffer will be appended unless blank is defined as 1.

The DBF\_APPEND() function will return < 0 if an error occurs during the append operation.

Recital performs automatic file locking on shared databases for the append. If the shared database is locked by another user, then the append will wait until the database is unlocked. The append will not be affected by records locked by other users.

### EXAMPLE

The following example will append in a record specified in the first parameter passed and return the result.

```
#include "dbapi.h"
```

```
dbapi_dbf_append()
{
    int      rc;

    if (_parinfo(1) != API_NTTYPE) {
        rc = DBF_APPEND( _parni(1) );
    } else {
        rc = -1
    }

    _retni( rc );
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_GATHER(), DBF\_READ(), DBF\_RECBUFFER(), DBF\_SELECT(), DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()



## DBF\_DBF()

---

### PURPOSE

Return the database name

### SYNONYM

api\_dbf\_dbf()

### SYNOPSIS

```
#include "dbapi.h"
```

```
char    *DBF_DBF()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_DBF() function will return the name of the currently selected database as character string, or a NULL if none is active.

Recital provides you with 20 workareas by default, however, up to 100 may be set with the DB\_MAXWKA symbol.

The database name is returned in lower case including the file extension.

### EXAMPLE

The following example returns the name of the database in the current workarea.

```
#include "dbapi.h"
```

```
dbapi_dbf_dbf()
{
    char    *dbfname()

    dbfname = DBF_DBF();

    _retc( dbfname );
}
```

### SEE ALSO

DBF\_ALIAS(), DBF\_FMT(), DBF\_INDEXKEY(), DBF\_INDEXORDER(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_NDX(), DBF\_SELECT(), DBF\_USED(), FIELD\_COUNT(), FIELD\_LOOKUP()

## DBF\_DELETE()

---

### PURPOSE

Return the database name

### SYNONYM

api\_dbf\_delete()

### SYNOPSIS

#include "dbapi.h"

```
int      DBF_DELETE()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_DELETE() function will delete the current record in the active database. On shared databases the record should be locked first before it is deleted. If you attempt to delete a record that is locked by another user, then Recital will display an error message.

Deleted records remain in the database until a PACK is performed on the database. The Recital command SET DELETED defines if the record will be displayed

### EXAMPLE

The following example locks, deletes and then unlocks the current record.

```
#include "dbapi.h"
```

```
dbapi_dbf_delete()
{
    int      rc;
    int      recnum;

    if ( DBF_ISEXCLUSIVE() ) {
        _rc = DBF_DELETE();
    } else {
        recnum = DBF_RECNO();
        DBF_LOCKR( recnum );
        rc = DBF_DELETE();
        DBF_UNLOCKR( recnum );
    }

    _retl( rc );
}
```

### SEE ALSO

DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_GOTO(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_RECNO(), DBF\_SKIP(), DBF\_UNLOCKF(), DBF\_UNLOCKR()

## DBF\_DELETED()

---

### PURPOSE

Is record deleted

### SYNONYM

api\_dbf\_isdeleted()

### SYNOPSIS

#include "dbapi.h"

int DBF\_DELETED()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_DELETED() function will return the current record is flagged for deletion, 0 if it is not and -1 if there is no database open in the currently selected workarea.

Deleted records remain in the database until a PACK is performed on the database. The Recital command SET DELETED determines whether or not the record will be displayed.

### EXAMPLE

The following example returns .T. if the current record is marked for deletion.

```
#include "dbapi.h"
```

```
dbapi_dbf_deleted()
{
    int    isdeleted;

    isdeleted = DBF_DELETED();

    _retl( isdeleted );
}
```

### SEE ALSO

DBF\_DELETE(), DBF\_FETCH(), DBF\_FILTER(), DBF\_GOTO(), DBF\_ISEXCLUSIVE(),  
DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_RECNO(),  
DBF\_SKIP(), DBF\_UNLOCKF(), DBF\_UNLOCKR()

## DBF\_FETCH()

---

### PURPOSE

Fetch a record from the current database

### SYNONYM

api\_dbf\_fetch()

### SYNOPSIS

#include "dbapi.h"

int DBF\_FETCH(inflag, forcond, whilecond, lock, position\_indexes, direction)

#### <input parameters>

int	*inflag;	/* Address of a buffer containing initialize flag	*/
char	*forcond;	/* Address of a buffer containing a valid for condition	*/
char	*whilecond;	/* Address of a buffer containing a valid while condition	*/
int	lock;	/* Lock record for update	*/
int	position_indexes;	/* Reposition the indexes	*/
int	direction;	/* Direction of search	*/

#### <output parameters>

none

### DESCRIPTION

The DBF\_FETCH() function will a record from the currently selected database.

If the contents initialize flag is set to 0 then the current record will be reread, otherwise the next record for the specified conditions will be read.

The fetch function moves the record pointer forwards or backwards in currently selected database. If the value of direction is 1 the record pointer is moved forward or backwards if the value is -1.

If a filter condition is set, the record pointer will skip over any records that do not match the filter condition.

Also a FOR condition can be specified with the DBF\_FETCH() function. The record pointer will search in the direction specified until a record is found that matches the FOR condition or until the EOF or BOF is reached.

If a WHILE condition is specified then the DBF\_FETCH() function will terminate when the specified condition is false.

If the value of lock is specified as 1, then the current record is unlocked and the next record fetched is locked. If the value is 0, then the next record fetched is read without locking but the record buffer is purged if the database is shareable.

If the value of position\_indexes is specified as 1, then all secondary indexes open on the current database are repositioned. This should be done when updates are to be performed.

The DBF\_FETCH() function returns -1 if no more records are available.

**EXAMPLE**

The following example returns the number of records in the database that match the condition “ord\_value.1000”.

```
#include ,stdio.h>
#include “dbapi.h”

dbapi_dbf_fetch()
{
    int      nrecs;
    int      initflag;
    int      result;

    DBF_GOTO(API_TOP);

    nrecs=0;
    result = DBF_FETCH(&initflag,
                      “ord_value.1000”,
                      NULL,
                      0,
                      1,
                      1);

    while (result >=0) {
        ++nrecs;
        result = DBF_FETCH(&initflag,
                          “ord_value.1000”,
                          NULL,
                          0,
                          1,
                          1);
    }

    _retni(nrecs);
}
```

**SEE ALSO**

DBF\_DELETE(), DBF\_DELETED(), DBF\_FILTER(), DBF\_GOTO(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_RECNO(), DBF\_SEEK(), DBF\_SKIP(), DBF\_UNLOCKF(), DBF\_UNLOCKR()

## DBF\_FILTER()

---

### PURPOSE

Return filter condition

### SYNONYM

api\_dbf\_filter()

### SYNOPSIS

```
#include "dbapi.h"
```

```
char    *DBF_FILTER()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_FILTER() function will return the filter condition from the currently active database as character string, or a NULL string if none is specified.

### EXAMPLE

The following example checks for a filter condition and removes it.

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

```
#include "dbapi.h"
```

```
dbapi_dbf_filter()
```

```
{
    char    *condition;

    condition = DBF_FILTER();

    if ( condition != NULL)
        COMMAND( "set filter to" );
}
```

### SEE ALSO

COMMAND(), DBF\_ALIAS(), DBF\_DBF(), DBF\_DELETED(), DBF\_FETCH(), DBF\_FMT(),  
DBF\_INDEXKEY(), DBF\_INDEXORDER(), DBF\_NDX(), DBF\_SELECT(), DBF\_USED(),  
FIELD\_COUNT(), FIELD\_LOOKUP()

## DBF\_FMT()

---

### PURPOSE

Return the format file name

### SYNONYM

api\_dbf\_fmt()

### SYNOPSIS

#include "dbapi.h"

char \*DBF\_FMT()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_FMT() function will return the name of the currently active screen format file as a character string, or a NULL string if none is active. The DBF\_FMT() function returns the format file name in lower case including the file extension.

### EXAMPLE

The following example checks for a screen format file and removes it.

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

```
#include "dbapi.h"
```

```
dbapi_dbf_fmt()
```

```
{
```

```
    char *format;
```

```
    format = DBF_FMT();
```

```
    if (format != NULL)
```

```
        COMMAND( "set format to" );
```

```
}
```

### SEE ALSO

COMMAND(), DBF\_ALIAS(), DBF\_DBF(), DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_INDEXKEY(), DBF\_INDEXORDER(), DBF\_NDX(), DBF\_SELECT(), DBF\_USED(), FIELD\_COUNT(), FIELD\_LOOKUP()

## DBF\_GATHER()

---

### PURPOSE

Gather an array into a record

### SYNONYM

api\_dbf\_gather()

### SYNOPSIS

#include "dbapi.h"

```
char    *DBF_GATHER(fldbuf)
```

<input parameters>

```
char    fldbuf[ ];          /* Pointer to an array containing values to update from */
```

<output parameters>

none

### DESCRIPTION

The DBF\_GATHER() function will update the current record with the contents of the specified array. The number of array elements must match the number of fields in the database to be updated.

Recital converts the character array elements automatically to the data type of each field in the record.

Memo and Blob fields are not updated with the DBF\_GATHER() function, however an array element must still be defined for these fields.

### EXAMPLE

The following example updates the second record in the "example.dbf" database from the specified array. See Appendix A for the data structure of "example.dbf".

```
#include "dbapi.h"
```

```
dbapi_dbf_gather()
```

```
{
    int    rc;
    int    element_no;
    char    *fldbuf[12];
    char    blob[5];
    char    byte[3];
    char    character[21];
    char    date[9];
    char    dfloat[11];
    char    integer[5];
    char    logical[2];
    char    number[11];
    char    memo[5];
    char    real[11];
    char    word[11];
    char    zoned[11];

    strcpy(blob,"blob");
    strcpy(byte,"2");
    strcpy(character, "Record number 2");
}
```



```

strcpy(date, "19990112");
strcpy(dfloat, "2.22");
strcpy(integer, "2");
strcpy(logical, "T");
strcpy(number, "2.22");
strcpy(memo, "memo");
strcpy(real, "2.22");
strcpy(word, "2.22");
strcpy(zoned, "2.22");

fldbuf[0] = blob;
fldbuf[1] = byte;
fldbuf[2] = character;
fldbuf[3] = date;
fldbuf[4] = dfloat;
fldbuf[5] = integer;
fldbuf[6] = logical;
fldbuf[7] = number;
fldbuf[8] = memo;
fldbuf[9] = real;
fldbuf[10] = word;
fldbuf[11] = zoned;

rc = COMMAND("use example");
if (rc == 0) {
    DBF_GOTO(2);

    if (!DBF_ISEXCLUSIVE()) DBF_LOCKR(DBF_RECNO());

    rc = DBF_GATHER(fldbuf);
    rc = DBF_UPDATE();

    if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(DBF_RECNO());

    COMMAND("use");
}

_retni(rc);
}

```

#### **SEE ALSO**

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_READ(), DBF\_RECBUFFER(), DBF\_SCATTER(), DBF\_SELECT(), DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE, MEMO\_WRITE()

## DBF\_GOTO()

---

### PURPOSE

Go to a record number

### SYNONYM

api\_dbf\_goto()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_GOTO(recnum)
```

<input parameters>

```
int      recnum;          /* Record number          */
```

<output parameters>

none

### DESCRIPTION

The DBF\_GOTO() function is used to position the record pointer to the specified record in the active database and then read the record into currently selected workarea. If the active database is indexed, then once the record pointer is positioned, and the record read into currently selected workarea, the indexes are repositioned automatically. If SET RELATION TO is in effect for the active database, then any associated databases are related after the record pointer has been positioned. It is worth noting that the GOTO always goes to the relative record in the database, and not the relative record according to the index order. You can position to records which are marked for deletion, even if SET DELETED ON is in effect.

The following variables may be used instead of a record number.

VARIABLE	DESCRIPTION
API_BOTTOM	Goto the last record in the database
API_TOP	Goto the first record in the database

### EXAMPLE

The following example goes to the last record in the database.

```
#include "dbapi.h"
```

```
dbapi_dbf_goto()
{
    int      rc;

    rc = DBF_GOTO (DBF_RECCOUNT());

    _retl( rc );
}
```

### SEE ALSO

DBF\_FETCH(), DBF\_ISBOF(), DBF\_ISEOF(), DBF\_LOCKR(), DBF\_RECCOUNT(), DBF\_RECNO(), DBF\_SEEK(), DBF\_SELECT(), DBF\_SKIP(), DBF\_UNLOCKR()

## DBF\_INDEXKEY()

---

### PURPOSE

Return the index key expression

### SYNONYM

api\_dbf\_indexkey()

### SYNOPSIS

```
#include "dbapi.h"
```

```
char    *DBF_INDEXKEY(order)
```

<input parameters>

```
int      order;          /* Index order          */
```

<output parameters>

none

### DESCRIPTION

The DBF\_INDEXKEY() function will return the index key expression from the currently selected database for the specified index order as a character string. If no index is active then a NULL string will be returned.

The character string is returned in lower case.

### EXAMPLE

The following example returns the index key expression for the master index.

```
#include "dbapi.h"
```

```
dbapi_dbf_indexkey()
{
    char    *indexkey;

    indexkey = DBF_INDEXKEY( DBF_INDEXORDER());

    _retc( indexkey );
}
```

### SEE ALSO

COMMAND(), DBF\_ALIAS(), DBF\_DBF(), DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_FMT(), DBF\_INDEXORDER(), DBF\_NDX(), DBF\_SEEK(), DBF\_SELECT(), DBF\_SKIP(), FIELD\_COUNT(), FIELD\_LOOKUP()

## DBF\_INDEXORDER()

---

### PURPOSE

Return current master index number

### SYNONYM

api\_dbf\_indexorder()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_INDEXORDER()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_INDEXORDER() function will return the master index from the currently selected database as an integer, if no index is active then -1 will be returned

A value from 1 to 20 will can be returned depending on the current master index. If the index is set to natural order, a value of 0 will be returned.

### EXAMPLE

The following example returns the master index for the current database.

```
#include "dbapi.h"
```

```
dbapi_dbf_indexorder()
{
    int      master;

    master = DBF_INDEXORDER();

    _retni( master );
}
```

### SEE ALSO

COMMAND(), DBF\_ALIAS(), DBF\_DBF(), DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_FMT(), DBF\_INDEXKEY(), DBF\_NDX(), DBF\_SEEK(), DBF\_SELECT(), DBF\_SKIP(), FIELD\_COUNT(), FIELD\_LOOKUP()

## DBF\_ISBOF()

---

### PURPOSE

Is record pointer at beginning of file.

### SYNONYM

api\_dbf\_isbof()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_ISBOF()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_ISBOF() function will return 1 if the record pointer is positioned past the first logical record of the currently selected database. If not, 0 is returned or -1 if there is an error.

### EXAMPLE

The following example returns the current position of the record pointer.

```
#include "dbapi.h"
```

```
dbapi_dbf_isbof()
{
    if (DBF_ISBOF()) {
        _retc( "BOF" );
    } else if (DBF_ISEOF()) {
        _retc( "EOF" );
    } else {
        _retc( "INF" );
    }
}
```

### SEE ALSO

DBF\_FETCH(), DBF\_GOTO(), DBF\_ISEOF(), DBF\_LOCKR(), DBF\_RECCOUNT(), DBF\_RECNO(), DBF\_SEEK(), DBF\_SELECT(), DBF\_SKIP(), DBF\_UNLOCKR()

## DBF\_ISEOF()

---

### PURPOSE

Is record pointer at end of file.

### SYNONYM

api\_dbf\_iseof()

### SYNOPSIS

#include "dbapi.h"

int DBF\_ISEOF()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_ISEOF() function will return 1 if the record pointer is positioned past the last logical record of the currently selected database. If not, 0 is returned or -1 if there is an error.

### EXAMPLE

The following example returns the current position of the record pointer.

```
#include "dbapi.h"
```

```
dbapi_dbf_iseof()
{
    if (DBF_ISBOF()) {
        _retc("BOF");
    } else if (DBF_ISEOF()) {
        _retc("EOF");
    } else {
        _retc("INF");
    }
}
```

### SEE ALSO

DBF\_FETCH(), DBF\_GOTO(), DBF\_ISBOF(), DBF\_LOCKR(), DBF\_RECCOUNT(), DBF\_RECNO(), DBF\_SEEK(), DBF\_SELECT(), DBF\_SKIP(), DBF\_UNLOCKR()

## DBF\_ISEXCLUSIVE()

---

### PURPOSE

Is database opened exclusively

### SYNONYM

api\_dbf\_isexclusive()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_ISEXCLUSIVE()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_ISEXCLUSIVE() function will return 1 if the currently active database is opened for exclusive use, 0 if it is opened shared and -1 if there is no database open in the current work area.

The Recital command SET EXCLUSIVE or the EXCLUSIVE key word used with the Recital command USE, control how the database is opened.

### EXAMPLE

The following example returns .T. if the database is opened for shared access.

```
#include "dbapi.h"
```

```
dbapi_dbf_isexclusive()
{
    int      result;

    result = DBF_ISEXCLUSIVE();

    _retl ( ( result ) ? 0 : 1 );
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISREADONLY(),  
DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKF(), DBF\_UNLOCKR(),  
DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()

## DBF\_ISREADONLY()

---

### PURPOSE

Is database readonly

### SYNONYM

api\_dbf\_isreadonly()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_ISREADONLY()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_ISREADONLY() function will return 1 if the currently selected database has readonly access, 0 if it can be updated and -1 if there is no database on the currently selected workarea.

Whenever Recital opens a database file, it examines the file protection mask defined by the operating system. If the user does not have write access to the file, then the file is opened readonly. The Applications Data Dictionary also defines readonly access controls by user and group IDs in the security option. A database can also be opened readonly with the NOUPDATE key word with the Recital command USE.

### EXAMPLE

The following example returns .T. if the database in use has readonly access.

```
#include "dbapi.h"
```

```
dbapi_dbf_isreadonly()
{
    int      readonly;

    readonly = DBF_ISREADONLY();

    _retl( readonly );
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKF(), DBF\_UNLOCKR(), DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()



## DBF\_LOCKF()

---

### PURPOSE

Lock a database

### SYNONYM

api\_dbf\_lockf()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_LOCKF()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_LOCKF() function will lock the database in the current workarea. A value of -1 will be returned if there is no database open in the current workarea.

The file lock will be issued on the shared database when no other user has a file or record lock current.

The database will still be opened shared for readonly access for other users however they will not be able to update any records until you release the file lock.

### EXAMPLE

The following example locks the current database.

```
#include "dbapi.h"
```

```
dbapi_dbf_lockf()
{
    int      locked;

    if (DBF_ISEXCLUSIVE() == 0 ) {
        locked = DBF_LOCKF();
    } else {
        locked = -1;
    }

    _retl( (locked >= 0) ? 1 : 0 );
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKF(), DBF\_UNLOCKR(), DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()

## DBF\_LOCKR()

---

### PURPOSE

Lock a record

### SYNONYM

api\_dbf\_lockr()

### SYNOPSIS

#include "dbapi.h"

```
int      DBF_LOCKR(recnum)
```

<input parameters>

```
long     recnum;          /* Record number          */
```

<output parameters>

none

### DESCRIPTION

The DBF\_LOCKR() function will lock the specified record. A value of -1 will be returned if there is no database open in the current workarea or the specified record number to lock is out of range.

The record lock will be issued when no other user has a file lock or the specified record locked.

The database will still be opened for shared read/write access for other users however they will not be able to update the record you have locked until you release the record lock.

### EXAMPLE

The following example locks the current record.

```
#include "dbapi.h"
```

```
dbapi_dbf_lockr()
{
    int      locked;

    if (DBF_ISEXCLUSIVE() == 0 ) {
        locked = DBF_LOCKR(DBF_RECNO());
    } else {
        locked = -1;
    }

    _retl( (locked) ? 1 : 0 );
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_RECALL(), DBF\_RECNO(), DBF\_UNLOCKF(), DBF\_UNLOCKR(), DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()

## DBF\_NDX()

---

### PURPOSE

Return index name

### SYNONYM

api\_dbf\_ndx()

### SYNOPSIS

```
#include "dbapi.h"
```

```
char    *DBF_NDX(order)
```

<input parameters>

```
long    order;          /* Index order          */
```

<output parameters>

none

### DESCRIPTION

The DBF\_NDX() function will return the index name from the currently selected database for the specified index order as a character string, if no index is active then a NULL string will be returned.

The character string is returned in lower case including the file extension

### EXAMPLE

The following example returns the index name for the specified order.

```
#include "dbapi.h"
```

```
dbapi_dbf_ndx()
{
    char    *indexname;

    if (_parinfo(1) == API_NTTYPE ) {
        indexname = DBF_NDX( _parni(1) );
    } else {
        strcpy(indexname, "");
    }

    _retc( indexname );
}
```

### SEE ALSO

COMMAND(), DBF\_ALIAS(), DBF\_DBF(), DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_FMT(), DBF\_INDEXKEY(), DBF\_INDEXORDER(), DBF\_SEEK(), DBF\_SELECT(), DBF\_SKIP(), DBF\_USED(), FIELD\_COUNT(), FIELD\_LOOKUP()

## DBF\_READ()

---

### PURPOSE

Read a record

### SYNONYM

api\_dbf\_read()

### SYNOPSIS

#include "dbapi.h"

```
int      DBF_READ(recnum, readlock, position_indexes)
```

<input parameters>

long	recnum;	/* Record number	*/
int	readlock;	/* Lock record to read	*/
int	position_indexes;	/* Reposition indexes	*/

<output parameters>

none

### DESCRIPTION

The DBF\_READ() function will read the current record from the database into the record buffer of the selected workarea.

A value of -1 will be returned if no database is open in the current workarea or the record pointer is positioned either past the end or the beginning of the file. If the required record is locked by another user and 'readlock' is not zero, the DBF\_READ() function will wait until the record is available.

The record number to be read must be specified in the first parameter.

A readlock can be specified with a value of 1 and if the database is opened for shared access, the current record is unlocked and the specified record is read and then locked for update.

You can specify if the indexes opened on the database should be read and repositioned by specifying a value of 1 for position\_indexes. Recital will also check the consistency of the indexes and return an error if one is detected. If the database is opened for shared access and a readlock was specified then the indexes are also locked.

If the database is related, the related records are read with the same parameters passed to the DBF\_READ() function. If the related database is related to another database then this related record will also be read, and so on until all the relationships are satisfied.

If the database is cached into memory then the DBF\_READ() function will check the cache first.

Any calculated fields defined in the Applications Data Dictionary are recalculated first before the record is placed into the buffer.

### EXAMPLE

The following example reads the current record into the record buffer and locks the record and repositions the index.

```
#include "dbapi.h"
```

```
dbapi_dbf_read()
```

```
{  
    int      rc;  
  
    rc = DBF_READ(DBF_RECNO(), 1, 1);  
  
    _retni( rc );  
}
```

**SEE ALSO**

BLOB\_READ(), DBF\_APPEND(), DBF\_GATHER(), DBF\_RECBUFFER(), DBF\_RECNO(),  
DBF\_RECSIZE(), DBF\_SCATTER(), DBF\_UPDATE(), FIELD\_COUNT(), FIELD\_LOOKUP(),  
FIELD\_VALUE(), MEMO\_READ()

## DBF\_RECALL()

---

### PURPOSE

Recall a deleted record

### SYNONYM

api\_dbf\_recall()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_RECALL()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_RECALL() function will return 1 if the current record can be recalled, and 0 if not. A -1 will be returned if no database is open in the current workarea.

On shared databases the record should be locked first before it is recalled. If you attempt to recall a record that is locked by another user, Recall will display an error message.

### EXAMPLE

The following example locks, recalls and then unlocks the current record.

```
#include "dbapi.h"
```

```
dbapi_dbf_recall()
{
    int      rc;
    long     recnum;

    if (DBF_ISEXCLUSIVE()) {
        rc = DBF_RECALL();
    } else {
        recnum = DBF_RECNO();
        DBF_LOCKR( recnum );
        rc = DBF_RECALL();
        DBF_UNLOCKR(recnum);
    }

    _retl( rc );
}
```

### SEE ALSO

DBF\_DELETE(), DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_GOTO(),  
DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECNO(),  
DBF\_SKIP(), DBF\_UNLOCKF(), DBF\_UNLOCKR()

## **DBF\_RECBUFFER()**

---

### **PURPOSE**

Return the current record buffer

### **SYNONYM**

api\_dbf\_recbuffer()

### **SYNOPSIS**

#include "dbapi.h"

char     \*DBF\_RECBUFFER()

<input parameters>

none

<output parameters>

none

### **DESCRIPTION**

The DBF\_RECBUFFER() function will return the current record buffer as a character pointer.

### **EXAMPLE**

The following example returns the current record buffer.

```
#include "dbapi.h"
```

```
dbapi_dbf_recordbuffer()
{
    char     *buffer;

    buffer = DBF_RECBUFFER();

    _retc( buffer );
}
```

### **SEE ALSO**

BLOB\_READ(), DBF\_APPEND(), DBF\_GATHER(), DBF\_READ(), DBF\_RECNO(), DBF\_RECSIZE(), DBF\_SCATTER(), DBF\_UPDATE(), FIELD\_COUNT(), FIELD\_LOOKUP(), FIELD\_VALUE(), MEMO\_READ()

## **DBF\_RECCOUNT()**

---

### **PURPOSE**

Return the number of records

### **SYNONYM**

api\_dbf\_reccount()

### **SYNOPSIS**

#include "dbapi.h"

```
int      DBF_RECCOUNT()
```

<input parameters>

none

<output parameters>

none

### **DESCRIPTION**

The DBF\_RECCOUNT() function will return the number of records in the currently selected database.

### **EXAMPLE**

The following example goes to the last record in the database.

```
#include "dbapi.h"
```

```
dbapi_dbf_reccount()
```

```
{
```

```
    int      rc;
```

```
    rc = DBF_GOTO( DBF_RECCOUNT());
```

```
}
```

### **SEE ALSO**

DBF\_GOTO(), DBF\_ISBOF(), DBF\_ISEOF(), DBF\_RECNO(), DBF\_SEEK(), DBF\_SKIP(),  
FIELD\_COUNT(), MEMO\_MLCOUNT()



## DBF\_RECNO()

---

### PURPOSE

Return the current record number

### SYNONYM

api\_dbf\_recno()

### SYNOPSIS

#include "dbapi.h"

int DBF\_RECNO()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_RECNO() function will return the current record number from the currently selected database.

### EXAMPLE

The following example returns the current record number

```
#include "dbapi.h"
```

```
dbapi_dbf_recno()
{
    int    recno;

    recno = DBF_RECNO();

    _retl( recno );
}
```

### SEE ALSO

DBF\_GOTO(), DBF\_ISBOF(), DBF\_ISEOF(), DBF\_RECCOUNT(), DBF\_SEEK(), DBF\_SKIP(),  
FIELD\_COUNT(), MEMO\_MLCOUNT()

## **DBF\_RECSIZE()**

---

### **PURPOSE**

Return the record size

### **SYNONYM**

api\_dbf\_recsize()

### **SYNOPSIS**

#include "dbapi.h"

```
int      DBF_RECSIZE()
```

<input parameters>

none

<output parameters>

none

### **DESCRIPTION**

The DBF\_RECSIZE() function will return the length of the record size in the currently selected database. The size will be the sum of all the field storage sizes plus 1 character for the deletion marker in the record.

### **EXAMPLE**

The following example returns the record size.

```
#include "dbapi.h"
```

```
dbapi_dbf_recsize()
{
    int      size;

    size = DBF_RECSIZE();

    _retni( size );
}
```

### **SEE ALSO**

DBF\_GOTO(), DBF\_ISBOF(), DBF\_ISEOF(), DBF\_RECCOUNT(), DBF\_RECNO(), DBF\_SEEK(), DBF\_SKIP(), FIELD\_COUNT(), MEMO\_MLCOUNT()

## DBF\_SCATTER()

---

### PURPOSE

Copy record buffer to an array

### SYNONYM

api\_dbf\_scatter()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_SCATTER(fldbbuf)
```

<input parameters>

none

<output parameters>

```
char      *fldbuf[];          /* A pointer to an array to hold the value of each field      */
```

### DESCRIPTION

The DBF\_SCATTER() function will scatter the contents of the current record into the specified array pointer.

The array elements to store the field values must be defined as characters strings as Recital will convert all the field data types to character strings. The contents of MEMO and BLOB fields are not copied into the array elements, instead the word "MEMO" is returned.

### EXAMPLE

The following example scatters the contents of the first record in the "example" database to the array pointer "fldbuf". The array value specified by the first parameter passed to the "dbapi\_dbf\_scatter()" function is returned. See Appendix A for the data structure of the "example.dbf" database.

```
#include "dbapi.h"
```

```
dbapi_dbf_scatter()
```

```
{
    int      rc;
    int      element_no;
    char      *fldbuf[12];
    char      blob[5];
    char      byte[3];
    char      character[21];
    char      date[9];
    char      dfloat[11];
    char      integer[6];
    char      logical[2];
    char      number[11];
    char      memo[5];
    char      real[11];
    char      word[11];
    char      zoned[11];

    if (_parinfo(1) != API_NTTYPE) _retc("");
```

```

fldbuf[0] = blob;
fldbuf[1] = byte;
fldbuf[2] = character;
fldbuf[3] = date;
fldbuf[4] = dfloat;
fldbuf[5] = integer;
fldbuf[6] = logical;
fldbuf[7] = number;
fldbuf[8] = memo;
fldbuf[9] = real;
fldbuf[10] = word;
fldbuf[11] = zoned;

element_no = _parni(1)-1;

rc = COMMAND("use example");
if ( rc == 0) {
    if ((element_no+1) > FIELD_COUNT() || element_no < 0) {
        COMMAND("use");
        _retc("Array value out of bounds.");
    }

    rc = DBF_SCATTER(fldbuf);
    COMMAND("use");
} else {
    strcpy(fldbuf[element_no], "");
}

_retc(fldbuf[ element_no]);
}

```

#### **SEE ALSO**

BLOB\_READ(), DBF\_APPEND(), DBF\_GATHER(), DBF\_READ(), DBF\_RECBUFFER(),  
 DBF\_SELECT(), DBF\_UPDATE(), FIELD\_COUNT(), FIELD\_UPDATE(), MEMO\_MLINE(),  
 MEMO\_READ()

## DBF\_SEEK()

---

### PURPOSE

Seek a key

### SYNONYM

api\_dbf\_seek()

### SYNOPSIS

#include "dbapi.h"

```
int      DBF_SEEK(key, readlock)
```

<input parameters>

```
char      *key;                /* Index key to seek          */
int       readlock;            /* Specify a read lock        */
```

<output parameters>

none

### DESCRIPTION

The DBF\_SEEK() function will seek the specified key in the master index on the current database. The maximum length of any index key is 200 bytes.

If the value of readlock is specified as 1, then the current record is unlocked and the 'seeked' record is locked. If the value is 0, then the 'seeked' record is read without locking but the record buffer is purged if the database is shareable.

A value of 1 is returned if the key is found in the index, 0 if it is not.

### EXAMPLE

The following example seeks the key specified in the first parameter passed and does not perform a readlock.

```
#include "dbapi.h"
```

```
dbapi_dbf_seek()
```

```
{
    int      rc;

    if (_parinfo(1) == API_CTYPE) {
        rc = DBF_SEEK(_parc(1), 0);
    } else {
        rc = -1
    }

    _retni( rc );
}
```

### SEE ALSO

DBF\_DELETE(), DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_GOTO(), DBF\_ISBOF(), DBF\_ISEOF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_RECNO(), DBF\_SKIP(), DBF\_UNLOCKR()

## DBF\_SELECT()

---

### PURPOSE

Select a workarea

### SYNONYM

api\_dbf\_select()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_SELECT(workarea)
```

<input parameters>

```
int      workarea;          /* Workarea to select      */
```

<output parameters>

none

### DESCRIPTION

The DBF\_SELECT() function will select the specified workarea.

Recital provides you with 20 workareas by default, however, up to 100 may be set with the DB\_MAXWKA symbol.

Each workarea contains the context for the database which has been opened in that workarea, including: the current record pointer, the current record, the current record buffer, the format file, the index files, the filter condition, and the relationships to other workareas.

### EXAMPLE

The following example selects the workarea specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_dbf_select()
```

```
{
    int      rc;

    if (_parinfo(1) == API_NTTYPE) {
        rc = DBF_SELECT(_parni(1));
    } else {
        rc = -1;
    }

    _retni( rc );
}
```

### SEE ALSO

DBF\_ALIAS(), DBF\_DBF(), DBF\_USED()

## DBF\_SKIP()

---

### PURPOSE

Skip records

### SYNONYM

api\_dbf\_skip()

### SYNOPSIS

#include "dbapi.h"

```
int      DBF_SKIP(amount)
```

<input parameters>

```
int      amount;                /* Number of records    */
```

<output parameters>

none

### DESCRIPTION

The DBF\_SKIP() function moves the record pointer forwards or backwards in the currently selected database. The amount may be a positive or negative value.

If the currently selected database is indexed, then the DBF\_SKIP() function follows the order of the master index. The RECALL SET ORDER TO command can be used to select which of the open index files should be master.

If the record pointer is currently positioned on the first record of the database, and DBF\_SKIP(-1) is specified, then the DBF\_ISBOF() function will return 1. If the record pointer is currently positioned on the last record of the database and DBF\_SKIP(1) is specified, then the DBF\_ISEOF() function will return 1.

The record pointer will also be moved on any related databases.

### EXAMPLE

The following example will skip the number of records specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_dbf_skip()
```

```
{
    int      rc;

    if (_parinfo(1) == API_NTTYPE) {
        rc = DBF_SKIP(_parni(1));
    } else {
        rc = -1;
    }

    _retni( rc );
}
```

### SEE ALSO

DBF\_DELETED(), DBF\_FETCH(), DBF\_FILTER(), DBF\_GOTO(), DBF\_ISBOF(), DBF\_ISEOF(), DBF\_LOCKR(), DBF\_RECNO(), DBF\_SEEK(), DBF\_UNLOCKR()

## DBF\_UNLOCKF()

---

### PURPOSE

Unlock a file

### SYNONYM

api\_dbf\_unlockf()

### SYNOPSIS

#include "dbapi.h"

int DBF\_UNLOCKF()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_UNLOCKF() function will unlock the currently selected database.

### EXAMPLE

The following example unlocks the current database.

```
#include "dbapi.h"
```

```
dbapi_dbf_unlock()
{
    int rc;

    rc = DBF_UNLOCKF();

    _retl( rc );
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKR(), DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()



## DBF\_UNLOCKR()

---

### PURPOSE

Unlock a record

### SYNONYM

api\_dbf\_unlockr()

### SYNOPSIS

#include "dbapi.h"

```
int      DBF_UNLOCKR()
```

<input parameters>

```
long     recnum;                /* Record number          */
```

<output parameters>

none

### DESCRIPTION

The DBF\_UNLOCKR() function will unlock the specified record in the currently selected database.

### EXAMPLE

The following example locks then unlocks the current record.

```
#include "dbapi.h"
```

```
dbapi_dbf_unlockr()
{
    int      recno;

    recno = DBF_RECNO();

    if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(recno);

    _retni( recno );
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(),  
DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKF(),  
DBF\_UPDATE(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()

## DBF\_UPDATE()

---

### PURPOSE

Flush current record buffer to disk

### SYNONYM

api\_dbf\_update()

### SYNOPSIS

#include "dbapi.h"

int DBF\_UPDATE()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_UPDATE() function will flush the current record buffer to disk. If the database is opened for shared access, the record must be locked first.

All active indexes on the database will be updated and the record pointer will be repositioned.

The Update trigger will be called if one has been specified for the database. The update operation will then only be completed successfully once the Recital procedure has been executed and .T. is returned.

Before or after image journaling will also be updated if either is active.

The record buffer is updated from the BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_GATHER(), DBF\_READ(), FIELD\_UPDATE(), MEMO\_UPDATE() and MEMO\_WRITE() functions. The DBF\_RECBUFFER() function will return the contents of the current record buffer.

### EXAMPLE

The following example locks the current record then flushes the record buffer to disk.

```
#include "dbapi.h"
```

```
dbapi_dbf_update()
{
    int rc;

    if (DBF_ISEXCLUSIVE()) {
        rc = DBF_UPDATE();
    } else {
        DBF_LOCKR(DBF_RECNO());
        rc = DBF_UPDATE();
        DBF_UNLOCKR(DBF_RECNO());
    }

    _retni( rc );
}
```

**SEE ALSO**

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(),  
DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKF(),  
DBF\_LOCKR(), FIELD\_UPDATE(), MEMO\_UPDATE(), MEMO\_WRITE()

## DBF\_USED()

---

### PURPOSE

Is a database in use

### SYNONYM

api\_dbf\_used()

### SYNOPSIS

```
#include "dbapi.h"
```

```
int      DBF_USED()
```

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DBF\_USED() function will return 1 if a database is in use in the current workarea, or 0 if no database is opened.

### EXAMPLE

The following example checks if a database is opened in the current workarea.

```
#include "dbapi.h"
```

```
dbapi_dbf_used()
{
    int      result;

    result = DBF_USED();

    _retl( result );
}
```

### SEE ALSO

COMMAND(), DBF\_ALIAS(), DBF\_DBF(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_SELECT

## FIELD\_COUNT()

---

### PURPOSE

Return the number of fields

### SYNONYM

api\_field\_count()

### SYNOPSIS

#include "dbapi.h"

int FIELD\_COUNT()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The FIELD\_COUNT() function will return the number of fields in the currently selected database.

The Recital command SET FIELDS TO will be reflected in the value returned by the FIELD\_COUNT() function.

### EXAMPLE

The following example returns the number of fields in the current database.

```
#include "dbapi.h"
```

```
dbapi_field_count()
```

```
{
```

```
    int    fldnum;
```

```
    fldnum = FIELD_COUNT();
```

```
    _retni( fldnum );
```

```
}
```

### SEE ALSO

BLOB\_SIZE(), DBF\_RECCOUNT(), DBF\_RECSize(), FIELD\_LOOKUP(), FIELD\_NAME(),  
FIELD\_NAME(), FIELD\_UPDATE(), FIELD\_VALUE(), MEMO\_MLCOUNT(), MEMO\_SIZE()

## FIELD\_LOOKUP()

---

### PURPOSE

Return field information

### SYNONYM

api\_field\_lookup()

### SYNOPSIS

```
#include "dbapi.h"
```

```
struct API_FIELD      FIELD_LOOKUP(fldname, fldinfo)
```

<input parameters>

```
char                  *fldname;          /* Address of a buffer containing a field name      */
```

<output parameters>

```
struct API_FIELD      *fldinfo;          /* Address of a buffer containing field information  */
```

### DESCRIPTION

The FIELD\_LOOKUP() function will return information for the specified field in the currently selected database.

Field protections defined in the Applications Data Dictionary are checked first and if the field is hidden a NULL value is returned.

The field information is stored in the API\_FIELD data structure defined below.

```
Struct  API_FIELD {
    int    fieldno;          /* Field number          */
    char   type;             /* Datatype of field     */
    int    size;             /* Storage size in the record */
    int    width;            /* Display width         */
    int    decimal;          /* Decimal places for numerics */
    char   value;            /* Pointer to data in current record */
};
```

The data type returned for the field is one of the following:

FIELD DATA TYPE	API DATA TYPE
Blob	M
Byte	B
Character	C
Currency	Y
Date	D
DateTime	T
Float	F
Integer	I
Logical	L
Numeric	N
Memo	M
Packed	P *
Quad	Q *

Real	R
Short	S
Zoned	Z
Vaxdate	V *

\* OpenVMS only

The storage size of a field in the record can be different to the display width of the field.

### EXAMPLE

The following example looks up to the width of the specified field in the current database.

```
#include <stdio.h>
#include "dbapi.h"

dbapi_field_lookup()
{
    struct API_FIELD    *fldvalue;
    struct API_FIELD    fldinfo;
    int                 width;

    if (_parinfo(1) == API_CTYPE) {
        fldvalue = FIELD_LOOKUP(_parc(1), &fldinfo);
        if (fldvalue == NULL) width = -1;
        else width = fldvalue->width;
    } else width = -1;

    _retni( width );
}
```

### SEE ALSO

BLOB\_READ(), BLOB\_RECLAIM(), BLOB\_SIZE(), BLOB\_UPDATE(), BLOB\_WRITE(),  
 DBF\_FETCH(), DBF\_GATHER(), DBF\_RECBUFFER(), DBF\_SCATTER(), DBF\_SEEK(),  
 FIELD\_COUNT(), FIELD\_NAME(), FIELD\_UPDATE(), FIELD\_VALUE(), MEMO\_MLINE(),  
 MEMO\_MLCOUNT(), MEMO\_READ(), MEMO\_RECLAIM(), MEMO\_SIZE(), MEMO\_UPDATE(),  
 MEMO\_WRITE()

## FIELD\_NAME()

---

### PURPOSE

Return a field name

### SYNONYM

api\_field\_name()

### SYNOPSIS

#include "dbapi.h"

```
char    *FIELD_NAME(fldno)
```

<input parameters>

```
int      fldno;          /* A field number          */
```

<output parameters>

none

### DESCRIPTION

The FIELD\_NAME() function will return the name of the field specified by the field number.

The field name is returned as an upper case character string, unless the field number is out of range in which case a NULL value is returned.

### EXAMPLE

The following example will return the field name for the specified number.

```
#include "dbapi.h"
```

```
dbapi_field_name()
```

```
{
    int      fldnum;
    char     *fldname;

    if (_parinfo(1) == API_NTTYPE) {
        fldnum = _parni(1);
        if (fldnum > FIELD_COUNT() || fldnum <= 0) {
            _retc("Field number out of range.");
        } else {
            fldname = FIELD_NAME(_parni(1));
        }
    } else {
        strcpy(fldname, "");
    }

    _retc( fldname );
}
```

### SEE ALSO

BLOB\_READ(), BLOB\_RECLAIM(), BLOB\_SIZE(), BLOB\_UPDATE(), BLOB\_WRITE(),  
DBF\_FETCH(), DBF\_GATHER(), DBF\_RECBUFFER(), DBF\_SCATTER(), DBF\_SEEK(),  
FIELD\_COUNT(), FIELD\_LOOKUP(), FIELD\_UPDATE(), FIELD\_VALUE(), MEMO\_MLINE(),



MEMO\_MLCOUNT(), MEMO\_READ(), MEMO\_RECLAIM(), MEMO\_SIZE(), MEMO\_UPDATE(),  
MEMO\_WRITE()

## FIELD\_UPDATE()

---

### PURPOSE

Update a field in the record buffer

### SYNONYM

api\_field\_update()

### SYNOPSIS

#include "dbapi.h"

```
int      *FIELD_UPDATE(name, type, string, number, logical, date)
```

<input parameters>

char	*name	/* Address of a buffer containing a field name	*/
char	type;	/* Type of field	*/
char	*string;	/* Address of a buffer containing a character string	*/
double	*number;	/* number	*/
char	logical;	/* T for True and F for False	*/
unsigned long	date;	/* Recital date	*/
DATETIME	datetime;	/* Recital datetime	*/
CURRENCY	currency;	/* Recital currency	*/

<output parameters>

none

### DESCRIPTION

The FIELD\_UPDATE() function will update a specified field in the currently selected database.

If any validation rules have been defined in the Applications Data Dictionary they will be checked first and the field will only be updated if the checks pass. A field can not be updated if it has been defined as calculated in the Applications Data Dictionary.

### EXAMPLE

The following example replaces all the fields in the example database with the specified expressions. See Appendix A for the data structure of "example.dbf".

```
#include "dbapi.h"
```

```
dbapi_field_update()
```

```
{
    int      result;
    double   num;
    char     char[16];
    char     log;
    unsigned long date;
    DATETIME datetime;
    CURRENCY currency;

    strcpy(char, "Record Number 3");
    num = 3;
    log = 'T';
    date = DATE_CTOD(DATE_DATE());
    datetime = DATE_STOT(DATE_DATETIME());
```

```

currency = CURR_STOY("7363.93");

if (!DBF_ISEXCLUSIVE()) DBF_LOCKR(DBF_RECNO());

rc=FIELD_UPDATE("blob", 'M', "", &num, log, date, datetime, currency);
rc=FIELD_UPDATE("byte", 'N', char, &num, log, date, datetime, currency);
rc=FIELD_UPDATE("char", 'C', char, &num, log, date, datetime, currency);
rc=FIELD_UPDATE("date", 'D', char, &num, log, date, datetime, currency);
rc=FIELD_UPDATE("float", 'N', char, &num, log, date, datetime, currency);
rc=FIELD_UPDATE("int", 'N', char, &num, log, date, datetime, currency);
rc=FIELD_UPDATE("logical", 'L', char, &num, log, date, datetime, currency);
rc=FIELD_UPDATE("number", 'N', char, &num, log, date, datetime, currency);
rc=FIELD_UPDATE("memo", 'M', "", num, log, date, datetime, currency);
rc=FIELD_UPDATE("real", 'N', char, num, log, date, datetime, currency);
rc=FIELD_UPDATE("word", 'N', char, num, log, date, datetime, currency);
rc=FIELD_UPDATE("zoned", 'N', char, num, log, date, datetime, currency);

result = DBF_UPDATE();

if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(DBF_RECNO());

_retni( result );
}

```

#### **SEE ALSO**

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_GATHER(), DBF\_READ(), DBF\_RECBUFFER(),  
 DBF\_UPDATE(), FIELD\_COUNT(), FIELD\_LOOKUP(), FIELD\_NAME(), FIELD\_VALUE(),  
 MEMO\_UPDATE(), MEMO\_WRITE()

## FIELD\_VALUE()

---

### PURPOSE

Return a field value

### SYNONYM

api\_field\_value()

### SYNOPSIS

#include "dbapi.h"

```
int FIELD_VALUE(fldname, value)
```

<input parameters>

```
char *fldname; /* Address of a buffer containing a field name */
```

<output parameters>

```
char *value; /* Address of a buffer to return the field value */
```

### DESCRIPTION

The FIELD\_VALUE() function will update the specified buffer with the value in the specified field from the currently selected database.

All database field types are automatically converted by Recital to character strings. The values from Memo and Blob fields are not copied into the buffer, instead the word "MEMO" is returned.

### EXAMPLE

The following example gets the value of all the fields from the example database and returns the specified field by the number of the parameter passed to the function. See appendix A for the data structure of "example.dbf".

```
#include "dbapi.h"
```

```
dbapi_field_value()
```

```
{
    int rc;
    int element_no;
    char *fldbuf[12];
    char blob[5];
    char byte[3];
    char character[21];
    char date[9];
    char dfloat[11];
    char integer[5];
    char logical[2];
    char number[11];
    char memo[5];
    char real[11];
    char word[11];
    char zoned[11];

    if (_parinfo(1) != API_NTTYPE) _retc("");

    fldbuf[0] = blob;
```

```

fldbuf[1] = byte;
fldbuf[2] = character;
fldbuf[3] = date;
fldbuf[4] = dfloat;
fldbuf[5] = integer;
fldbuf[6] = logical;
fldbuf[7] = number;
fldbuf[8] = memo;
fldbuf[9] = real;
fldbuf[10] = word;
fldbuf[11] = zoned;

element_no = _parni(1) - 1;

rc = COMMAND("use example");
if (rc == 0 ) {
    if (element_no > FIELD_COUNT() || element_no < 0) {
        COMMAND("use");
        _retc("Field number out of bounds.");
    }
    rc=FIELD_VALUE ("blob",blob);
    rc=FIELD_VALUE ("byte",byte);
    rc=FIELD_VALUE ("char",character);
    rc=FIELD_VALUE ("date",date);
    rc=FIELD_VALUE ("float",float);
    rc=FIELD_VALUE ("int",int);
    rc=FIELD_VALUE ("number",number);
    rc=FIELD_VALUE ("memo",memo);
    rc=FIELD_VALUE ("real",real);
    rc=FIELD_VALUE ("word",word);
    rc=FIELD_VALUE ("zoned",zoned);
    COMMAND("use");

} else {
    _retc( "");
}

_retc(fldbuf[element_no]);
}

```

#### SEE ALSO

BLOB\_READ(), DBF\_FETCH(), DBF\_READ(), DBF\_RECBUFFER(), DBF\_SCATTER()  
 FIELD\_COUNT(), FIELD\_LOOKUP(), FIELD\_NAME(), FIELD\_UPDATE(), MEMO\_MLINE(),  
 MEMO\_READ()

## MEMO\_MLCOUNT()

---

### PURPOSE

Return the number of lines in a memo

### SYNONYM

api\_memo\_mlcount()

### SYNOPSIS

#include "dbapi.h"

```
int      MEMO_MLCOUNT(fldname)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing memo field name */
```

<output parameters>

none

### DESCRIPTION

This function will return the number of lines in the specified memo field for the currently selected database.

The line length of the memo field is defined with the Recital command SET MEMOWIDTH TO. The MEMOWIDTH is also affected by the setting of the Recital command MEMOFORMAT, if ON lines are word wrapped.

NOTE: MEMOFORMAT should be either be left turned on or off. You should not toggle between settings as the formatting in the memo will be lost.

### EXAMPLE

The following example returns the number of lines in the memo field specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_memo_mlcount()
{
    int      numlines;

    if (_parinfo(1) == API_CTYPE ) {
        numlines = MEMO_MLCOUNT(_parc(1));
    } else {
        numlines = 0;
    }

    _retni ( numlines );
}
```

### SEE ALSO

BLOB\_SIZE(), DBF\_RECCOUNT(), FIELD\_COUNT(), MEMO\_MLINE(), MEMO\_READ(), MEMO\_RECLAIM(), MEMO\_SIZE(), MEMO\_UPDATE(), MEMO\_WRITE()

## MEMO\_MLINE()

---

### PURPOSE

Extract a line of text from a memo

### SYNONYM

api\_memo\_mline()

### SYNOPSIS

#include "dbapi.h"

```
int      *MEMO_MLINE(fldname, line, lineno)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing memo field name */
int      lineno;             /* Line number */
```

<output parameters>

```
char      *line;             /* Address of a buffer to return the memo line */
```

### DESCRIPTION

The MEMO\_MLINE() function extracts a line of text from the specified memo field from the currently selected database.

The line number to extract is specified by lineno and starts at 1 and has a maximum of MEMO\_MLCOUNT()

The line length of the memo field is defined with the Recital command SET MEMOWIDTH TO. The MEMOWIDTH is also affected by the setting of the Recital command MEMOFORMAT, if ON lines are word wrapped.

NOTE: MEMOFORMAT should be either be left turned on or off. You should not toggle between settings as the formatting in the memo will be lost.

### EXAMPLE

The following example will extract each line from a memo field.

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

```
#include "dbapi.h"
```

```
dbapi_memo_mline()
```

```
{
    int      i;
    int      numlines;
    char      memoline[80];
    char      *fieldname;

    if (_parinfo(1) == API_CTYPE ) _retc("");

    strcpy(fieldname, _parc(1));
    numlines = MEMO_MLCOUNT(fieldname);

    for (i=0; i<=numlines; ++i) {
        MEMO_MLINE(fieldname, i, memoline);
```

```
        /* other commands */  
    }  
  
    _retni( numlines );  
}
```

**SEE ALSO**

BLOB\_READ(), DBF\_FETCH(), DBF\_READ(), DBF\_RECBUFFER(), DBF\_SCATTER(),  
FIELD\_VALUE(), MEMO\_MLCOUNT(), MEMO\_READ(), MEMO\_RECLAIM(), MEMO\_SIZE(),  
MEMO\_UPDATE(), MEMO\_WRITE()



## MEMO\_READ()

---

### PURPOSE

Read a memo field

### SYNONYM

api\_memo\_read()

### SYNOPSIS

#include "dbapi.h"

```
int      MEMO_READ(fldname, memobuf, maxsize)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing the name of a field memo */
int       maxsize;           /* Maximum size of the memo */
```

<output parameters>

```
char      *memobuf;          /* Address of a buffer where the memo is stored */
```

### DESCRIPTION

The MEMO\_READ() function will update a buffer with the contents from the specified memo field on the current record. The number of bytes read from the memo is specified by maxsize, a Recital string can store up to 8000 bytes.

A value of -1 is returned if the field is not a MEMO, otherwise the number of bytes read is returned.

### EXAMPLE

The following example returns the contents of the memo field specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_memo_read()
```

```
{
    unsigned char    memobuf[8000];
    int              memosize;
    int              result;

    if (_parinfo(1) == API_CTYPE ) {
        memosize = MEMO_SIZE( _parc(1) );
        if (memosize >= 0) {
            result = MEMO_READ( _parc(1), memobuf, memosize);
        } else {
            _retc("Not a memo field.");
        }
    } else {
        strcpy(memobuf, "");
        memosize=0;
    }

    _retclen(memobuf, memosize);
}
```

**SEE ALSO**

BLOB\_READ(), DBF\_FETCH(), DBF\_READ(), DBF\_RECBUFFER(), DBF\_SCATTER(),  
FIELD\_VALUE(), MEMO\_MLCOUNT(), MEMO\_MLINE(), MEMO\_RECLAIM(), MEMO\_SIZE(),  
MEMO\_UPDATE(), MEMO\_WRITE()

## MEMO\_RECLAIM()

---

### PURPOSE

Reclaim space in a memo file

### SYNONYM

api\_memo\_reclaim()

### SYNOPSIS

#include "dbapi.h"

```
int      MEMO_RECLAIM(fldname)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing the name of a memo field */
```

<output parameters>

none

### DESCRIPTION

The MEMO\_RECLAIM() function will free up space occupied by the memo field specified in the current record in the database. The space occupied by the memo will be reused for the next update of the memo file. All data in the specified memo field will be deleted. A value of -1 is returned if the field is not a MEMO, otherwise a value >0 will be returned for success.

### EXAMPLE

The following deletes the contents of the memo field specified in the first parameter passed for the current record.

```
#include "dbapi.h"
```

```
dbapi_memo_reclaim()
```

```
{
    int      rc;
    if (_parinfo(1) == API_CTYPE ) {

        if (!DBF_ISEXCLUSIVE()) DBF_LOCKR(DBF_RECNO())

        rc = MEMO_RECLAIM(_parc(1));
        if (rc) rc = DBF_UPDATE();

        if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(DBF_RECNO());

    } else {
        rc = -1;
    }

    _retni( rc );
}
```

### SEE ALSO

DBF\_DELETE(), DBF\_RECALL(), MEMO\_MLCOUNT(), MEMO\_MLINE(), MEMO\_READ(), MEMO\_SIZE(), MEMO\_UPDATE(), MEMO\_WRITE()

## MEMO\_SIZE()

---

### PURPOSE

Return the size of a memo field

### SYNONYM

api\_memo\_size()

### SYNOPSIS

#include "dbapi.h"

```
int      MEMO_SIZE(fldname)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing the name of a memo field */
```

<output parameters>

none

### DESCRIPTION

The MEMO\_SIZE() function will return the number of bytes that the specified memo field is occupying in the memo file. A value of -1 will be returned if the field is not a MEMO.

### EXAMPLE

The following example returns the number of bytes in the memo field specified in the first parameter passed.

```
#include "dbapi.h"
```

```
dbapi_memo_size()
{
    int      memosize;

    if (_parinfo(1) == API_CTYPE ) {
        memosize = MEMO_SIZE(_parc(1));
    } else {
        memosize = -1;
    }

    _retni(memosize);
}
```

### SEE ALSO

BLOB\_SIZE(), DBF\_RECCOUNT(), DBF\_RECSize(), MEMO\_MLCOUNT(), MEMO\_READ(), MEMO\_RECLAIM(), MEMO\_UPDATE(), MEMO\_WRITE()

## MEMO\_UPDATE()

---

### PURPOSE

Update a memo field

### SYNONYM

api\_memo\_update()

### SYNOPSIS

#include "dbapi.h"

```
int      MEMO_UPDATE(fldname, memo)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing the name of a memo field */
char      *memo;             /* Address of a buffer containing the memo */
```

<output parameters>

none

### DESCRIPTION

The MEMO\_UPDATE() function will update the specified memo field with the contents of the memo buffer. The space occupied by the memo in the memo file is overwritten with the specified memo buffer. The MEMO\_UPDATE() function is the same as using both the MEMO\_RECLAIM() and MEMO\_WRITE() function together.

A value of -1 will be returned if the field is not a MEMO.

### EXAMPLE

The following example deletes the current contents of the specified memo and updates it with the value specified in the second parameter

```
#include "dbapi.h"
```

```
dbapi_memo_update()
{
    int      result;

    if (_parinfo(1) == API_CTYPE && _parinfo(2) == API_CTYPE) {
        if (!DBF_ISEXCLUSIVE()) DBF_LOCKR(DBF_RECNO());

        result = MEMO_UPDATE(_parc(1), _parc(2));
        if (result) result = DBF_UPDATE();

        if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(DBF_RECNO());
    } else {
        result = -1;
    }

    _retni(result);
}
```

**SEE ALSO**

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(),  
DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKF(),  
DBF\_UNLOCKR(), FIELD\_UPDATE(), MEMO\_MLCOUNT(), MEMO\_MLINE(), MEMO\_READ(),  
MEMO\_SIZE(), MEMO\_WRITE()

## MEMO\_WRITE()

---

### PURPOSE

Write a string to a memo field

### SYNONYM

api\_memo\_write()

### SYNOPSIS

#include "dbapi.h"

```
int      MEMO_WRITE(fldname, memo)
```

<input parameters>

```
char      *fldname;          /* Address of a buffer containing the name of a memo field */
char      *memo;             /* Address of a buffer containing the memo */
```

<output parameters>

none

### DESCRIPTION

The MEMO\_WRITE() function will update the specified memo field with the contents of the memo buffer. The space occupied by the memo in the memo file is not overwritten with the specified memo buffer. A value of -1 will be returned if the field is not a MEMO.

### EXAMPLE

The following example updates the current contents of the specified memo with the value specified in the second parameter passed. It locks the record and then updates the record.

```
#include "dbapi.h"
```

```
dbapi_memo_write()
{
    int      result;

    if (_parinfo(1) == API_CTYPE && _parinfo(2) == API_CTYPE) {

        if (!DBF_ISEXCLUSIVE()) DBF_LOCKR(DBF_RECNO());
        result = MEMO_WRITE(_parc(1), _parc(2));
        if (result) result = DBF_UPDATE();
        if (!DBF_ISEXCLUSIVE()) DBF_UNLOCKR(DBF_RECNO());

    } else {
        result = -1;
    }

    _retni(result);
}
```

### SEE ALSO

BLOB\_UPDATE(), BLOB\_WRITE(), DBF\_APPEND(), DBF\_DELETE(), DBF\_ISEXCLUSIVE(), DBF\_ISREADONLY(), DBF\_LOCKF(), DBF\_LOCKR(), DBF\_RECALL(), DBF\_UNLOCKF(), DBF\_UNLOCKR(), FIELD\_UPDATE(), MEMO\_MLCOUNT(), MEMO\_MLINE(), MEMO\_READ(), MEMO\_SIZE(), MEMO\_UPDATE()

## MEMVAR\_DEFINE()

---

### PURPOSE

Define a Recital memory variable

### SYNONYM

api\_memvar\_define()

### SYNOPSIS

#include "dbapi.h"

```
struct API_MEMVAR MEMVAR_DEFINE(name, level)
```

<input parameters>

```
char    *name;          /* Address of a buffer containing the name of memory variable */
int      level;          /* Private or public declaration */
```

<output parameters>

none

### DESCRIPTION

The MEMVAR\_DEFINE() function will define a Recital memory as .F..

The level of declaration must be specified from one of the following.

VALUE	DESCRIPTION
API_PRIVATE	Private to the Recital calling procedure
API_PUBLIC	Public to all Recital higher level procedures

The following API\_MEMVAR structure is used to store the results.

```
union  API_UNION {
    char    *info_character;    /* Char memory variable */
    char    info_logical;      /* Logical memory variable */
    double   info_number;      /* Numeric memory variable */
    unsigned long info_date;    /* Date memory variable */
    DATETIME info_datetime;    /* Datetime memory variable */
    CURRENCY info_currency;    /* Currency memory variable */
};
```

```
struct  API_MEMVAR {
    char    type;              /* Data type of memory variable */
    union    API_UNION value;  /* Value of the memory variable */
    int     width;             /* Display width */
    int     decimal;           /* Display # of decimal places */
};
```

The API\_MEMVAR type specifies both the data type and which API\_UNION value will be used to return the value of the memory variable.

The API\_MEMVAR width returns the length of the memory variable. A decimal place may be returned for a number, but will be 0 for other data types.



A memory variable can be defined as any one of the following.

TYPE	OUTPUT	WIDTH	DESCRIPTION
C	*info_character	1-8192	Address of a buffer containing a character string
D	info_date	8	Unsigned long representing a recital date
L	info_logical	1	Character of either 'T' for true or 'F' for false
N	info_number	1-16	A value stored in a double
T	info_datetime	sizeof(DATETIME)	RCT_DATETIME_DEF structure
Y	info_currency	sizeof(CURRENCY)	RCT_CURRENCY_DEF structure

#### EXAMPLE

The following example defines the six memory variables and then stores the specified values to them.

```
#include "dbapi.h"
```

```
dbapi_memvar_define()
```

```
{
    int          rc;
    double       numeric;
    char         string[8];
    char         logical;
    unsigned long date;
    DATETIME     datetime;
    CURRENCY     currency;

    strcpy(string, "Recital");
    date = DATE_CTOD(DATE_DATE());
    logical = 'T';
    numeric = 23.5;
    datetime = DATE_STOT(DATE_DATETIME());
    currency = CURR_STOY("6363.939");

    COMMAND("release all");
    MEMVAR_DEFINE("character", API_PUBLIC);
    MEMVAR_DEFINE("number", API_PUBLIC);
    MEMVAR_DEFINE("logical", API_PUBLIC);
    MEMVAR_DEFINE("date", API_PUBLIC);
    MEMVAR_DEFINE("datetime", API_PUBLIC);
    MEMVAR_DEFINE("currency", API_PUBLIC);

    rc = MEMVAR_UPDATE( "character", 'C', 7, 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "number", 'N', 10, 2,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "logical", 'L', 1, 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "date", 'D', 8, 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "datetime", 'T', sizeof(DATETIME), 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "currency", 'Y', sizeof(CURRENCY), 0,
                        string, &numeric, logical, date, datetime, currency);

    _retni(rc >= 0) ? 1 : 0);
}
```

**SEE ALSO**

ARRAY\_DEFINE(), MEMVAR\_LOOKUP(), MEMO\_UPDATE()

## MEMVAR\_LOOKUP()

---

### PURPOSE

Lookup a Recital memory variable

### SYNONYM

api\_memvar\_lookup()

### SYNOPSIS

#include "dbapi.h"

struct API\_MEMVAR MEMVAR\_LOOKUP(name)

<input parameters>

char \*name; /\* Address of a buffer containing the name of memory variable \*/

<output parameters>

none

### DESCRIPTION

The MEMVAR\_LOOKUP() function will lookup the specified Recital memory variable and return its data type and value into the specified API\_MEMVAR data structure.

A NULL value is returned if the memory variable has not previously been defined, otherwise the results are stored in the following API\_MEMVAR data structure:

```
union  API_UNION {
    char      *info_character;    /* Char memory variable      */
    char      info_logical;      /* Logical memory variable    */
    double     info_number;      /* Numeric memory variable    */
    unsigned long info_date;     /* Date memory variable       */
    DATETIME   info_datetime;    /* Datetime memory variable   */
    CURRENCY   info_currency;    /* Currency memory variable   */
};

struct  API_MEMVAR {
    char      type;              /* Data type of memory variable */
    union     API_UNION value;   /* Value of the memory variable */
    int       width;            /* Display width                */
    int       decimal;          /* Display # of decimal places  */
};
```

The API\_MEMVAR type specifies both the data type and which API\_UNION value will be used to return the value of the memory variable.

The API\_MEMVAR width returns the length of the memory variable. A decimal place may be returned for a number, but will be 0 for other data types.

A memory variable can be defined as any one of the following.

TYPE	OUTPUT	WIDTH	DESCRIPTION
C	*info_character	1-8192	Address of a buffer containing a character string
D	info_date	8	Unsigned long representing a recital date
L	info_logical	1	Character of either 'T' for true or 'F' for false
N	info_number	1-16	A value stored in a double
T	info_datetime	sizeof(DATETIME)	RCT_DATETIME_DEF structure
Y	info_currency	sizeof(CURRENCY)	RCT_CURRENCY_DEF structure

#### EXAMPLE

The following example looks up the memory variable value specified in the first parameter passed and then returns the value as a character string.

```
#include "dbapi.h"
```

```
dbapi_memvar_lookup()
{
    struct API_MEMVAR    *result;
    char                  buffer[1025];

    if (_parinfo(1) != API_CTYPE) _retc("");

    result = MEMVAR_LOOKUP(_parc(1));

    switch ( result->type )
    {
        case 'C':
            _retc( result->value.info_character );
        case 'D':
            _retc( DATE_DTOS(result->value.info_date));
        case 'N':
            CHAR_STR( buffer,
                      result->value.info_number,
                      result->width,
                      result->decimal);
            _retc( buffer );
        case 'L':
            _retc((result->value.info_logical=='T') ? "True" : "False" );
    }

    _retc("");
}
```

#### SEE ALSO

ARRAY\_LOOKUP(), MEMVAR\_DEFINE(), MEMVAR\_UPDATE()

## MEMVAR\_UPDATE()

---

### PURPOSE

Update a Recital memory variable

### SYNONYM

api\_memvar\_update()

### SYNOPSIS

#include "dbapi.h"

```
int      MEMVAR_UPDATE(name, type, width, decimal, string, number, logical, date)
```

<input parameters>

char	*name;	/* Address of a buffer containing the memory variable name	*/
char	type;	/* Memory variable type	*/
int	width;	/* Memory variable width	*/
int	decimal;	/* Number of decimal places	*/
char	string;	/* Address of a buffer containing character string	*/
double	*number;	/* Numeric value	*/
char	logical;	/* Logical value	*/
unsigned long	date;	/* Recital date	*/
DATETIME	datetime;	/* Recital datetime	*/
CURRENCY	currency;	/* Recital currency	*/

<output parameters>

none

### DESCRIPTION

The MEMVAR\_UPDATE() function will update a specified Recital memory variable.

Recital will only recognize the first 10 characters on a memory name as being unique. The memory variable names are not case sensitive and must have been previously defined before updating, or the MEMVAR\_UPDATE() function will return a value of -1.

Memory variables can be updated with different data types than they are currently defined. The variable type parameter specifies both the data type and which input parameter value will be used to update the memory variable.

The width parameter is used to specify length of the memory variable. A decimal place can be specified for a number, but should be 0 for other data types.

A memory variable can be defined as any one of the following.

TYPE	OUTPUT	WIDTH	DESCRIPTION
C	string	1-8192	Address of a buffer containing a character string
D	date	8	Unsigned long representing a recital date
L	logical	1	Character of either 'T' for true or 'F' for false
N	number	1-16	A value stored in a double
T	info_datetime	sizeof(DATETIME)	RCT_DATETIME_DEF structure
Y	info_currency	sizeof(CURRENCY)	RCT_CURRENCY_DEF structure

## EXAMPLE

The following example defines six memory variables and then stores the specified values to each variable.

```
#include "dbapi.h"
```

```
dbapi_store()
{
    int rc;
    double numeric;
    char string[8];
    char logical;
    unsigned long date;
    DATETIME datetime;
    CURRENCY currency;

    strcpy(string, "Recital");
    date = DATE_CTOD(DATE_DATE());
    logical = 'T';
    numeric = 23.5;
    datetime = DATE_STOT(DATE_DATETIME());
    currency = CURR_STOY("6474.99");

    COMMAND("release all");
    MEMVAR_DEFINE( "character", API_PRIVATE );
    MEMVAR_DEFINE( "number", API_PRIVATE );
    MEMVAR_DEFINE( "logical", API_PRIVATE );
    MEMVAR_DEFINE( "date", API_PRIVATE );
    MEMVAR_DEFINE( "datetime", API_PRIVATE );
    MEMVAR_DEFINE( "currency", API_PRIVATE );

    rc = MEMVAR_UPDATE( "character", 'C', 7, 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "number", 'N', 10, 2,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "logical", 'L', 1, 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "date", 'D', 8, 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "datetime", 'T', sizeof(DATETIME), 0,
                        string, &numeric, logical, date, datetime, currency);
    rc = MEMVAR_UPDATE( "currency", 'Y', sizeof(CURRENCY), 0,
                        string, &numeric, logical, date, datetime, currency);
}
```

## SEE ALSO

ARRAY\_UPDATE(), MEMVAR\_DEFINE(), MEMVAR\_LOOKUP()

## OVERVIEW OF LEVEL 4

---

### Macros:

Level 4 DEFINE\_XXX macros are used for defining classes and their associated methods that can be instantiated from the Recital 4GL. The DISPATCH\_XXX macros are used to dispatch method calls in an OBJARG format to the specified methods. The OBJECT\_XXX macros are used to access and return values passed to and from the object.

### OBJARG:

The OBJARG format is stored in a string in the format “X:ASCII” where ‘X’ can be any one of the following:

TYPE	DESCRIPTION
E	Error
C	Character
N	Numeric
L	Logical
D	Date
T	Datetime
Y	Currency
O	Object

### OBJPTR:

This macro is the pointer to an instantiated object defined by the class.

### Object Data Structures:

Public properties and methods need to be defined a structure for introspection called public\_properties and public\_methods. The structures are in an OBJARG format except for the last item which must be defined as an empty string. The string part of the OBJARG is defined as “Name;Type;Description”, for example:

```
static  char    *public_properties[] = {
    "C:CHARVALUE;C;Character property value",
    "C:NUMVALUE;N;Numeric property value",
    "C:LOGVALUE;L;Logical property value",
    "C:DATEVALUE;D;Date property value",
    "C:TIMEVALUE;T;DateTime property value",
    "C:CURRENVALUE;Y;Currency property value",
    "C:OBJVALUE;O;Object property value", ""
};

static  char    *public_methods[] = {
    "C:DEFINE;L;Define settings in exception",
    ""
};
```

### Return codes:

The macros OBJECT\_ERROR and OBJECT\_SUCCESS are defined to be used to return the result of a method call.

## DEFINE\_CLASS()

---

### PURPOSE

A macro to define a class

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

DEFINE\_CLASS(classname)

<input parameters>

constant            classname;            /\* Class name       \*/

<output parameters>

none

### DESCRIPTION

The DEFINE\_CLASS() macro is used to define a class that can be accessed from the Recital 4GL. This class needs to be added to the API\_SHARED\_FUNCTION\_TABLE data structure which is loaded dynamically when Recital is started.

### EXAMPLE

The following example defines a class called clsMyClass and defines some dispatch methods.

#### Example Recital program:

```
// The constructor is called when the object is created
test = newobject("myclass")
? type("test")
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
static struct API_SHARED_FUNCTION_TABLE api_function_table[2] = {
    {"myclass", "clsMyClass", API_CLASS},
    {NULL, NULL, -1}
};
```

```
DEFINE_CLASS(clsMyClass)
```

```
{
    /* Dispatch factory methods and return */
    DISPATCH_FACTORY();

    /* Dispatch constructor and return */
    DISPATCH_METHOD(clsMyClass, Constructor);

    /* Dispatch destructor and return */
    DISPATCH_METHOD(clsMyClass, Destructor);
}
```



**SEE ALSO**

DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(),  
OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## DEFINE\_METHOD()

---

### PURPOSE

A macro to define a method

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

DEFINE\_METHOD(classname, methodname)

<input parameters>

```
constant      classname;      /* Class name that contains the method      */
constant      methodname;     /* Method name to define                      */
```

<output parameters>

none

### DESCRIPTION

The DEFINE\_METHOD() macro is used to define a method for class that can be accessed from the Recital 4GL.

### EXAMPLE

The following example defines a method called Destructor for the class clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// The destructor is called when the object is released
release test
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
DEFINE_METHOD(clsMyClass, Destructor)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();

    if (objectData != NULL) {
        if (objectData->prop_objvalue != NULL) OBJECT_DELETE(objectData->prop_objvalue);
        free(objectData);
        objectData = NULL;
    }
    return(0);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),

OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERROR(), OBJECT\_RETPROPERTY(),  
OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## DEFINE\_PROPERTYGET()

---

### PURPOSE

A macro to define a method to get the value of a property

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
DEFINE_PROPERTYGET(classname, propertyname)
```

<input parameters>

```
constant      classname;      /* Class name that contains the property */
constant      propertyname;    /* Property name to define */
```

<output parameters>

none

### DESCRIPTION

The DEFINE\_PROPERTYGET() macro is used to define a method to get the value of a property from a class that can be accessed from the Recital 4GL.

### EXAMPLE

The following example defines a method to get the value of the property called NumValue for the class clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Display the value of the property NumValue
? "numvalue", test.numvalue
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define get property method */
```

```
DEFINE_PROPERTYGET(clsMyClass, NumValue)
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();

    if (objectData == NULL) return(-1);

    OBJECT_RETPROPERTY('N', objectData->prop_numvalue);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYSET(), DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETEERROR(),

OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(),  
OBJECT\_SETPROPERTY()

## DEFINE\_PROPERTYSET()

---

### PURPOSE

A macro to define a method to set the value of a property

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
DEFINE_PROPERTYSET(classname, propertyname)
```

<input parameters>

```
constant      classname;      /* Class name that contains the property */
constant      propertyname;    /* Property name to define */
```

<output parameters>

none

### DESCRIPTION

The DEFINE\_PROPERTYSET() macro is used to define a method to set the value of a property from a class that can be accessed from the Recital 4GL.

### EXAMPLE

The following example defines a method to set the value of the property called NumValue for the class clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Set the value of the property NumValue
test.numvalue = test.numvalue * 5
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define set property method */
```

```
DEFINE_PROPERTYSET(clsMyClass, ObjValue)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    OBJECT      objvalue;
    int         rc = OBJECT_ERROR;

    if (OBJECT_GETTYPE() == 'O') {
        objvalue = OBJECT_GETOBJECT();
        objectData->prop_objvalue = OBJECT_ASSIGN(objvalue, objectData->object_name);
        rc = OBJECT_SUCCESS;
    }

    return(rc);
}
```

**SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DISPATCH\_FACTORY(),  
DISPATCH\_METHOD(), DISPATCH\_PROPGET(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(),  
OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(),  
OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(),  
OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(),  
OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(),  
OBJECT\_SETPROPERTY()

## DISPATCH\_FACTORY()

---

### PURPOSE

A macro to dispatch factory methods passed to the object

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

DISPATCH\_FACTORY()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The DISPATCH\_FACTORY() macro is used to dispatch factory methods passed to the object. This macro should always be included at the top of each defined class. After a factory method is dispatched by this macro, execution is returned to Recital. The Recital engine uses these factory methods for introspection of the class.

### EXAMPLE

The following example adds a DISPATCH\_FACTORY() method to a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Introspect the object and display its properties and methods
display memory like test
```

#### Example in 'C' object:

```
#include "dbapi.h"

DEFINE_CLASS(clsMyClass)
{
    /* Dispatch factory methods and return */
    DISPATCH_FACTORY();

    /* Dispatch constructor and return */
    DISPATCH_METHOD(clsMyClass, Constructor);

    /* Dispatch destructor and return */
    DISPATCH_METHOD(clsMyClass, Destructor);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(),



OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(),  
OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_REERROR(),  
OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(),  
OBJECT\_SETPROPERTY()

## DISPATCH\_METHOD()

---

### PURPOSE

A macro to dispatch a method passed to the object

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
DISPATCH_METHOD(classname, methodname)
```

<input parameters>

```
constant      classname;      /* class name that contains the method */
constant      methodname;     /* method name to dispatch */
```

<output parameters>

none

### DESCRIPTION

The DISPATCH\_METHOD() macro is used to dispatch a method passed to the object. This macro should always be placed after the DISPATCH\_FACTORY() macro in each defined class. After a method is dispatched by this macro, execution is returned to Recital.

### EXAMPLE

The following example adds two DISPATCH\_METHOD() macros for the constructor and destructor methods to a class called clsMyClass.

#### Example Recital program:

```
// The constructor is called when the object is created
test = newobject("myclass")
? type("test")
// The destructor is called when the object is released
release test
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
DEFINE_CLASS(clsMyClass)
{
    /* Dispatch factory methods and return */
    DISPATCH_FACTORY();

    /* Dispatch constructor and return */
    DISPATCH_METHOD(clsMyClass, Constructor);

    /* Dispatch destructor and return */
    DISPATCH_METHOD(clsMyClass, Destructor);
}
```

**SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
DISPATCH\_FACTORY(), DISPATCH\_PROPGET(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(),  
OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(),  
OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(),  
OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_REERROR(),  
OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(),  
OBJECT\_SETPROPERTY()

## DISPATCH\_PROPGET()

---

### PURPOSE

A macro to dispatch a get property method passed to the object

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
DISPATCH_PROPGET(classname, propertyname)
```

<input parameters>

```
constant      classname;      /* class name that contains the property */
constant      propertyname;    /* property name to dispatch get method */
```

<output parameters>

none

### DESCRIPTION

The DISPATCH\_PROPGET() macro is used to dispatch a method that gets the value of a specified property passed to the object. After a method is dispatched by this macro, execution is returned to Recital.

### EXAMPLE

The following example adds a DISPATCH\_PROPGET() macro to get the value of the property NumValue in a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Display the value of the property NumValue
? "numvalue", test.numvalue
```

#### Example in 'C' object:

```
#include "dbapi.h"

DEFINE_CLASS(clsMyClass)
{
    /* Dispatch factory methods and return */
    DISPATCH_FACTORY();

    /* Dispatch SET or GET PROPERTY */
    /* method for property NumValue */
    /* then return. */
    DISPATCH_PROPSET(clsMyClass, NumValue);
    DISPATCH_PROPGET(clsMyClass, NumValue);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(), DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(),

OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(),  
OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_REERROR(),  
OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(),  
OBJECT\_SETPROPERTY()

## DISPATCH\_PROPSET()

---

### PURPOSE

A macro to dispatch a set property method passed to the object

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
DISPATCH_PROPSET(classname, propertyname)
```

<input parameters>

```
constant      classname;      /* Class name that contains the property */
constant      propertyname;    /* Property name to dispatch set method */
```

<output parameters>

none

### DESCRIPTION

The DISPATCH\_PROPSET() macro is used to dispatch a method that sets the value of a specified property passed to the object. After a method is dispatched by this macro, execution is returned to Recital.

### EXAMPLE

The following example adds a DISPATCH\_PROPSET() macro to set the value of the property NumValue in a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Set the value of the property NumValue
test.numvalue = test.numvalue * 5
```

#### Example in 'C' object:

```
#include "dbapi.h"

DEFINE_CLASS(clsMyClass)
{
    /* Dispatch factory methods and return */
    DISPATCH_FACTORY();

    /* Dispatch SET or GET PROPERTY */
    /* method for property NumValue */
    /* then return. */
    DISPATCH_PROPSET(clsMyClass, NumValue);
    DISPATCH_PROPGET(clsMyClass, NumValue);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(), DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(),

OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(),  
OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_REERROR(),  
OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(),  
OBJECT\_SETPROPERTY()

## OBJECT\_ASSIGN()

---

### PURPOSE

Assign a name to an instantiated object

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

OBJECT\_ASSIGN(objptr, objname)

<input parameters>

OBJECT	objptr;	/* Pointer to the object to assign name to	*/
char	*objname;	/* Name to assign to the object	*/

<output parameters>

none

### DESCRIPTION

The OBJECT\_ASSIGN() macro is used to assign a name to an instantiated object. You must assign a name to an object when it is passed as a value to set a property.

### EXAMPLE

The following example instantiates an object from a system class EXCEPTION and assigns it to the ObjValue property in a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Instantiate an EXCEPTION object and assign it to the property ObjValue
test.objvalue=newobject("exception")
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* define set property method */
```

```
DEFINE_PROPERTYSET(clsMyClass, ObjValue)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    OBJECT      objvalue;
    int         rc = OBJECT_ERROR;

    if (OBJECT_GETTYPE() == 'O') {
        objvalue = OBJECT_GETOBJECT();
        objectData->prop_objvalue = OBJECT_ASSIGN(objvalue, objectData->object_name);
        rc = OBJECT_SUCCESS;
    }

    return(rc);
}
```



**SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
DISPATCH\_PROPSET(), OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(),  
OBJECT\_GETDATA(), OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(),  
OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(),  
OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(),  
OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_DELETE()

---

### PURPOSE

Deletes an instantiated object

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_DELETE(objptr)
```

<input parameters>

```
OBJECT      objptr;          /* Pointer to the object to be deleted      */
```

<output parameters>

none

### DESCRIPTION

The OBJECT\_DELETE() macro is used to delete an instantiated object. The object must have been first instantiated with OBJECT\_NEW() or assigned with OBJECT\_ASSIGN().

### EXAMPLE

The following example deletes an object that was assigned to the ObjValue property from the destructor method for the class clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// The destructor is called when the object is released
release test
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
DEFINE_METHOD(clsMyClass, Destructor)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();

    if (objectData != NULL) {
        if (objectData->prop_objvalue != NULL) OBJECT_DELETE(objectData->prop_objvalue);
        free(objectData);
        objectData = NULL;
    }
    return(0);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(), DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(),

OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(),  
OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(),  
OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_GETARG()

---

### PURPOSE

Return an OBJARG format string as an API\_EXPRESSION

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_GETARG(buffer, result)
```

<input parameters>

```
OBJARG          buffer;          /* Address of a buffer containing an OBJARG      */
```

<output parameters>

```
API_EXPRESSION  *result;         /* Pointer to an API_EXPRESSION to return result */
```

### DESCRIPTION

The OBJECT\_GETARG() macro is used to convert an OBJARG format string to a API\_EXPRESSION. Objects are not converted with this macro.

### EXAMPLE

In the following example OBJECT\_GETARG() is used to convert an OBJARG string returned from an introspection of a property in the ObjValue object to an API\_EXPRESSION.

#### Example Recital program:

```
test = newobject("myclass")
// Executes method 'define' and updates two properties
? test.define("Error Message", -1000)
? "test.objvalue.errorno",test.objvalue.errorno
? "test.objvalue.message",test.objvalue.message
```

#### Example in 'C' object:

```
#include "dbapi.h"

/* Define define method */
DEFINE_METHOD(clsMyClass, Define)
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    struct   API_EXPRESSION result;
    char    buffer[512];
    int     rc;

    /* Check the object class */
    OBJECT_GETPROPERTY(objectData->prop_objvalue, "class", buffer);
    rc = OBJECT_GETARG(buffer, &result);
    if (result.erno == 0 && result.type == 'C' &&
        strcmp(result.character, "Exception") == 0) {
        switch (OBJECT_GETARGC()) {
            case 2:
```

```

        rc = OBJECT_GETPARAMETER(2, &result);
        if (result.errno == 0 && result.type == 'N') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "errno", buffer);
        }

    case 1:
        rc = OBJECT_GETPARAMETER(1, &result);
        if (result.errno == 0 && result.type == 'C') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "message", buffer);
        }
        break;
    }
}

result.type = 'L';
result.logical = (rc == 0 ? 'T' : 'F');
OBJECT_RETRESULT(&result);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARGC(),  
 OBJECT\_GETDATA(), OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(),  
 OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(),  
 OBJECT\_RETEROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(),  
 OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_GETARGC()

---

### PURPOSE

Return the number of arguments passed

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

OBJECT\_GETARGC()

<input parameters>

None

<output parameters>

None

### DESCRIPTION

The OBJECT\_GETARGC() macro is used to return the number of arguments passed to a method.

### EXAMPLE

In the following example OBJECT\_GETARGC() will return 2, as two parameters were passed when executing the 'define' method from the Recital 4GL.

#### Example Recital program:

```
test = newobject("myclass")
// Executes method 'define' and updates two properties
? test.define("Error Message", -1000)
? "test.objvalue.errorno",test.objvalue.errorno
? "test.objvalue.message",test.objvalue.message
```

#### Example in 'C' object:

```
#include "dbapi.h"

/* Define define method */
DEFINE_METHOD(clsMyClass, Define)
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    struct API_EXPRESSION result;
    char    buffer[512];
    int     rc;

    /* Check the object class */
    OBJECT_GETPROPERTY(objectData->prop_objvalue, "class", buffer);
    rc = OBJECT_GETARG(buffer, &result);
    if (result.erno == 0 && result.type == 'C' &&
        strcmp(result.character, "Exception") == 0) {
        switch (OBJECT_GETARGC()) {
            case 2:
                rc = OBJECT_GETPARAMETER(2, &result);
```

```

        if (result.errno == 0 && result.type == 'N') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "errno", buffer);
        }

    case 1:
        rc = OBJECT_GETPARAMETER(1, &result);
        if (result.errno == 0 && result.type == 'C') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "message", buffer);
        }
        break;
    }
}

result.type = 'L';
result.logical = (rc == 0 ? 'T' : 'F');
OBJECT_RETRESULT(&result);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETDATA(), OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(),  
 OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(),  
 OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(),  
 OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_GETDATA()

---

### PURPOSE

Return a pointer to a user defined data area

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

OBJECT\_GETDATA()

<input parameters>

None

<output parameters>

none

### DESCRIPTION

The OBJECT\_GETDATA() macro is used to return a pointer to a user defined data area in an instantiated object. This function returns the data area from the method's associated object.

### EXAMPLE

The following example gets the data area from an object instantiated from the class clsMyClass. The object is passed with the DISPATCH\_PROPERTYGET() macro.

#### Example Recital program:

```
test = newobject("myclass")
? test.numvalue
```

#### Example in 'C' object:

```
#include "dbapi.h"

/* Define get property method */
DEFINE_PROPERTYGET(clsMyClass, NumValue)
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();

    if (objectData == NULL) return(-1);

    OBJECT_RETPROPERTY('N', objectData->prop_numvalue);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(), DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()



## OBJECT\_GETOBJECT()

---

### PURPOSE

Get an object passed to a method

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

OBJECT\_GETOBJECT()

<input parameters>

none

<output parameters>

none

### DESCRIPTION

The OBJECT\_GETOBJECT() macro is used to return an object pointer to an object passed to a method.

### EXAMPLE

The following example gets a pointer to the instantiated object passed to the set property method of the ObjValue property in a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Instantiate an EXCEPTION object and assign it to the property ObjValue
test.objvalue=newobject("exception")
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* define set property method */
```

```
DEFINE_PROPERTYSET(clsMyClass, ObjValue)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    OBJECT      objvalue;
    int         rc = OBJECT_ERROR;

    if (OBJECT_GETTYPE() == 'O') {
        objvalue = OBJECT_GETOBJECT();
        objectData->prop_objvalue = OBJECT_ASSIGN(objvalue, objectData->object_name);
        rc = OBJECT_SUCCESS;
    }

    return(rc);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(), DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),

DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETPARAMETER(),  
OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(),  
OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(),  
OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_GETPARAMETER()

---

### PURPOSE

Return an API\_EXPRESSION from a parameter

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_GETPARAMETER(argnum, result)
```

<input parameters>

```
int                argnum;                /* Number of parameter to get          */
```

<output parameters>

```
API_EXPRESSION     *result;               /* Pointer to a API_EXPRESSION to return result */
```

### DESCRIPTION

The OBJECT\_GETPARAMETER() macro is used to convert the specified parameter to an API\_EXPRESSION. Parameters start a one. Objects are not converted with this macro, see OBJECT\_GETOBJECT() for more information.

### EXAMPLE

In the following example OBJECT\_GETPARAMETER() is used to get the two parameters passed to the 'define' method as an API\_EXPRESSION.

#### Example Recital program:

```
test = newobject("myclass")
// executes method 'define' and updates two properties
? test.define("Error Message", -1000)
? "test.objvalue.errorno",test.objvalue.errorno
? "test.objvalue.message",test.objvalue.message
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define define method */
```

```
DEFINE_METHOD(clsMyClass, Define)
```

```
{
```

```
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
```

```
    struct    API_EXPRESSION result;
```

```
    char     buffer[512];
```

```
    int      rc;
```

```
    /* Check the object class */
```

```
    OBJECT_GETPROPERTY(objectData->prop_objvalue, "class", buffer);
```

```
    rc = OBJECT_GETARG(buffer, &result);
```

```
    if (result.errno == 0 && result.type == 'C' &&
```

```
        strcmp(result.character, "Exception") == 0) {
```

```
        switch (OBJECT_GETARGC()) {
```

```

case 2:
    rc = OBJECT_GETPARAMETER(2, &result);
    if (result.errno == 0 && result.type == 'N') {
        OBJECT_SETARG(buffer, &result);
        rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "errno", buffer);
    }

case 1:
    rc = OBJECT_GETPARAMETER(1, &result);
    if (result.errno == 0 && result.type == 'C') {
        OBJECT_SETARG(buffer, &result);
        rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "message", buffer);
    }
    break;
}
}

result.type = 'L';
result.logical = (rc == 0 ? 'T' : 'F');
OBJECT_RETRESULT(&result);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(), OBJECT\_GETPROPERTY(),  
 OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(),  
 OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(),  
 OBJECT\_SETPROPERTY()

## OBJECT\_GETPROPERTY()

---

### PURPOSE

Return an OBJARG format string from a property

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_GETPROPERTY(objptr, propertyname, buffer)
```

<input parameters>

OBJECT	objptr;	/* Pointer to the object to introspect	*
char	*propertyname;	/* property name to introspect	*/

<output parameters>

OBJARG	buffer;	/* pointer to an OBJARG buffer to return result	*/
--------	---------	---	----

### DESCRIPTION

The OBJECT\_GETPROPERTY() macro is used to introspect the specified object and return the property value in an OBJARG format string.

### EXAMPLE

In the following example OBJECT\_GETPROPERTY() is used to return the value of the property "class" in an OBJARG string format returned from introspecting the object ObjValue.

#### Example Recital program:

```
test = newobject("myclass")
// Executes method 'define' and updates two properties
? test.define("Error Message", -1000)
? "test.objvalue.errorno",test.objvalue.errorno
? "test.objvalue.message",test.objvalue.message
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define define method */
```

```
DEFINE_METHOD(clsMyClass, Define)
```

```
{
```

```
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
```

```
    struct    API_EXPRESSION result;
```

```
    char    buffer[512];
```

```
    int      rc;
```

```
    /* Check the object class */
```

```
    OBJECT_GETPROPERTY(objectData->prop_objvalue, "class", buffer);
```

```
    rc = OBJECT_GETARG(buffer, &result);
```

```
    if (result.errno == 0 && result.type == 'C' &&
```

```
        strcmp(result.character, "Exception") == 0) {
```

```
        switch (OBJECT_GETARGC()) {
```

```

case 2:
    rc = OBJECT_GETPARAMETER(2, &result);
    if (result.errno == 0 && result.type == 'N') {
        OBJECT_SETARG(buffer, &result);
        rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "errno", buffer);
    }

case 1:
    rc = OBJECT_GETPARAMETER(1, &result);
    if (result.errno == 0 && result.type == 'C') {
        OBJECT_SETARG(buffer, &result);
        rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "message", buffer);
    }
    break;
}

result.type = 'L';
result.logical = (rc == 0 ? 'T' : 'F');
OBJECT_RETRESULT(&result);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
 OBJECT\_GETPARAMETER(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(),  
 OBJECT\_RETERORR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(),  
 OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_GETTYPE()

---

### PURPOSE

Get the data type of a value passed to a method

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

OBJECT\_GETTYPE()

<input parameters>

None

<output parameters>

None

### DESCRIPTION

The OBJECT\_GETTYPE() macro is used to return the data type of a value passed to a set property method.

### EXAMPLE

The following example gets the data type of the value passed to the set property method of the ObjValue property in a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
// Instantiate an EXCEPTION object and assign it to the property ObjValue
test.objvalue=newobject("exception")
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* define set property method */
```

```
DEFINE_PROPERTYSET(clsMyClass, ObjValue)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    OBJECT      objvalue;
    int         rc = OBJECT_ERROR;

    if (OBJECT_GETTYPE() == 'O') {
        objvalue = OBJECT_GETOBJECT();
        objectData->prop_objvalue = OBJECT_ASSIGN(objvalue, objectData->object_name);
        rc = OBJECT_SUCCESS;
    }

    return(rc);
}
```

**SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETVALUE(),  
OBJECT\_NEW(), OBJECT\_REERROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(),  
OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()



## OBJECT\_GETVALUE()

---

### PURPOSE

Return an API\_EXPRESSION from a property

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_GETVALUE(result)
```

<input parameters>

none

<output parameters>

```
API_EXPRESSION      *result;          /* Pointer to an API_EXPRESSION to return result */
```

### DESCRIPTION

The OBJECT\_GETVALUE() macro is used to return the value passed to a set property method to an API\_EXPRESSION. Objects are not converted with this macro, see OBJECT\_GETOBJECT() for more information.

### EXAMPLE

In the following example OBJECT\_GETVALUE() is used to return the value passed to the set property method of the NumValue property as an API\_EXPRESSION.

#### Example Recital program:

```
test = newobject("myclass")
// Set the value of the property NumValue
test.numvalue = test.numvalue * 5
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define set property method */
```

```
DEFINE_PROPERTYSET(clsMyClass, NumValue)
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    struct API_EXPRESSION result;
    int rc = OBJECT_ERROR;

    OBJECT_GETVALUE(&result);
    if (result.erno == 0 && result.type == 'N') {
        objectData->prop_numvalue = result.number;
        rc = OBJECT_SUCCESS;
    }

    return(rc);
}
```

**SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_NEW(),  
OBJECT\_RETEROR(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(), OBJECT\_SETARG(),  
OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_NEW()

---

### PURPOSE

Instantiate an object

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

OBJECT\_NEW(objname, classname, parameters)

<input parameters>

char	*objname;	/* name to assign to the object	*/
char	*classname;	/* class name to instantiate object from	*/
char	*parameters;	/* comma separated list of parameters	*/

<output parameters>

none

### DESCRIPTION

The OBJECT\_NEW() macro is used to instantiate an object . You must assign a name to an object when it is being instantiated and may pass optional parameters.

### EXAMPLE

The following example instantiates an object from a system class EXCEPTION and assigns it to the ObjValue property from the constructor in a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define constructor method */
```

```
DEFINE_METHOD(clsMyClass, Constructor)
```

```
{
```

```
    struct example_data *objectDataArea;
```

```
    /* Allocate memory for objects objectData area */
```

```
    objectDataArea = (struct example_data *) malloc(sizeof(struct example_data));
```

```
    if (objectDataArea == NULL) return(-1);
```

```
    /* Assign the default property values */
```

```
    strcpy(objectDataArea->prop_charvalue, "Test API object");
```

```
    objectDataArea->prop_numvalue = 15.2827;
```

```
    objectDataArea->prop_logvalue = 'F';
```

```
    strcpy(objectDataArea->prop_datevalue, DATE_DATE());
```

```
    strcpy(objectDataArea->prop_timevalue, DATE_DATETIME());
```

```
    strcpy(objectDataArea->prop_currvalue, "15.2827");
```

```
    strcpy(objectDataArea->object_name, "APIobject");
```

```
    objectDataArea->prop_objvalue = OBJECT_NEW(objectDataArea->object_name,
```

```

    "exception",
    NULL);

    /* Set the object objectData area */
    OBJECT_SETDATA((char *)objectDataArea);

    return(0);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
 OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
 OBJECT\_GETVALUE(), OBJECT\_REERROR(), OBJECT\_RETPROPERTY(),  
 OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_RETERROR()

---

### PURPOSE

Return an error

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_RETERROR(message)
```

<input parameters>

```
char                *message;        /* Error message to return */
```

<output parameters>

None

### DESCRIPTION

The OBJECT\_RETERROR() macro is used to return an error from a user defined class in an OBJARG format.

### EXAMPLE

The following example will call OBJECT\_RETERROR() as the property passed is unknown.

#### Example Recital program:

```
test = newobject("myclass")
// A error will be returned as the property is unknown
? test.unknown_property
```

#### Example in 'C' object:

```
#include "dbapi.h"

DEFINE_CLASS(clsMyClass)
{
    /* Dispatch factory methods and return */
    DISPATCH_FACTORY();

    /* Dispatch constructor and return */
    DISPATCH_METHOD(clsMyClass, Constructor);

    /* Dispatch destructor and return */
    DISPATCH_METHOD(clsMyClass, Destructor);

    /* If message not found return error */
    OBJECT_RETERROR("Unknown message type");
}
```

### SEE ALSO

DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),

DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETPROPERTY(), OBJECT\_RETRESULT(),  
OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_RETPROPERTY()

---

### PURPOSE

Return an OBJARG format string

### SYNONYM

None

### SYNOPSIS

#include "dbapi.h"

OBJECT\_RETPROPERTY(type, value)

<input parameters>

char	type;	/* Data type to return	*/
void	value;	/* Value to return	*/

<output parameters>

none

### DESCRIPTION

The OBJECT\_RETPROPERTY() macro is used to return the specified value in an OBJARG format from get property method.

### EXAMPLE

The following example returns the numeric value of the property NumValue as a numeric type.

#### Example Recital program:

```
test = newobject("myclass")
? test.numvalue
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define get property method */
```

```
DEFINE_PROPERTYGET(clsMyClass, NumValue)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();

    if (objectData == NULL) return(-1);

    OBJECT_RETPROPERTY('N', objectData->prop_numvalue);
}
```

### SEE ALSO

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(), DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(), DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(), OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(), OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(), OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(), OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_RETRESULT()

---

### PURPOSE

Return an API\_EXPRESSION from a method

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_RETRESULT(result)
```

<input parameters>

```
API_EXPRESSION          *result; /* Pointer to the result expression to return */
```

<output parameters>

none

### DESCRIPTION

The OBJECT\_RETRESULT() macro is used to return a value specified in an API\_EXPRESSION in an OBJARG formatted string.

### EXAMPLE

In the following example OBJECT\_RETRESULT() is used to return the success status of the call to the 'define' method.

#### Example Recital program:

```
test = newobject("myclass")
// Executes method 'define' and updates two properties
? test.define("Error Message", -1000)
? "test.objvalue.errorno",test.objvalue.errorno
? "test.objvalue.message",test.objvalue.message
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define define method */
```

```
DEFINE_METHOD(clsMyClass, Define)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    struct    API_EXPRESSION result;
    char      buffer[512];
    int       rc;

    /* Check the object class */
    OBJECT_GETPROPERTY(objectData->prop_objvalue, "class", buffer);
    rc = OBJECT_GETARG(buffer, &result);
    if (result.erno == 0 && result.type == 'C' &&
        strcmp(result.character, "Exception") == 0) {
        switch (OBJECT_GETARGC()) {
            case 2:
```



```

        rc = OBJECT_GETPARAMETER(2, &result);
        if (result.errno == 0 && result.type == 'N') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "errno", buffer);
        }

    case 1:
        rc = OBJECT_GETPARAMETER(1, &result);
        if (result.errno == 0 && result.type == 'C') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "message", buffer);
        }
        break;
    }
}

result.type = 'L';
result.logical = (rc == 0 ? 'T' : 'F');
OBJECT_RETRESULT(&result);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
 OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
 OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(),  
 OBJECT\_SETARG(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_SETARG()

---

### PURPOSE

Return an API\_EXPRESSION as an OBJARG formatted string.

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_SETARG(buffer, result)
```

<input parameters>

```
API_EXPRESSION      *result;          /* pointer to an API_EXPRESSION      */
```

<output parameters>

```
OBJARG              buffer            /* Address of a buffer containing an OBJARG */
```

### DESCRIPTION

The OBJECT\_SETARG() macro is used to convert an API\_EXPRESSION to an OBJARG formatted string.

### EXAMPLE

In the following example OBJECT\_SETARG () is used to convert an API\_EXPRESSION return from a parameter to an OBJARG string.

#### Example Recital program:

```
test = newobject("myclass")
// Executes method 'define' and updates two properties
? test.define("Error Message", -1000)
? "test.objvalue.errorno",test.objvalue.errorno
? "test.objvalue.message",test.objvalue.message
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define define method */
```

```
DEFINE_METHOD(clsMyClass, Define)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    struct    API_EXPRESSION result;
    char      buffer[512];
    int       rc;

    /* Check the object class */
    OBJECT_GETPROPERTY(objectData->prop_objvalue, "class", buffer);
    rc = OBJECT_GETARG(buffer, &result);
    if (result.erno == 0 && result.type == 'C' &&
        strcmp(result.character, "Exception") == 0) {
        switch (OBJECT_GETARGC()) {
            case 2:
```

```

        rc = OBJECT_GETPARAMETER(2, &result);
        if (result.errno == 0 && result.type == 'N') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "errno", buffer);
        }

    case 1:
        rc = OBJECT_GETPARAMETER(1, &result);
        if (result.errno == 0 && result.type == 'C') {
            OBJECT_SETARG(buffer, &result);
            rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "message", buffer);
        }
        break;
    }
}

result.type = 'L';
result.logical = (rc == 0 ? 'T' : 'F');
OBJECT_RETRESULT(&result);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
 OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
 OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(),  
 OBJECT\_RETRESULT(), OBJECT\_SETDATA(), OBJECT\_SETPROPERTY()

## OBJECT\_SETDATA()

---

### PURPOSE

Set user defined data

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_SETDATA(userdata)
```

<input parameters>

```
void          *userdata;          /* Pointer to a user defined data area      */
```

<output parameters>

none

### DESCRIPTION

The OBJECT\_SETDATA() macro is used assign a user defined data area to an instantiated object. This data area can point to any type of data that can be returned with the OBJECT\_GETDATA() macro for the life of the instantiated object.

### EXAMPLE

The following example sets the data area from the constructor in a class called clsMyClass.

#### Example Recital program:

```
test = newobject("myclass")
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define constructor method */
```

```
DEFINE_METHOD(clsMyClass, Constructor)
```

```
{
```

```
    struct example_data *objectDataArea;
```

```
    /* Allocate memory for objects objectData area */
```

```
    objectDataArea = (struct example_data *) malloc(sizeof(struct example_data));
```

```
    if (objectDataArea == NULL) return(-1);
```

```
    /* Assign the default property values */
```

```
    strcpy(objectDataArea->prop_charvalue, "Test API object");
```

```
    objectDataArea->prop_numvalue = 15.2827;
```

```
    objectDataArea->prop_logvalue = 'F';
```

```
    strcpy(objectDataArea->prop_datevalue, DATE_DATE());
```

```
    strcpy(objectDataArea->prop_timevalue, DATE_DATETIME());
```

```
    strcpy(objectDataArea->prop_currvalue, "15.2827");
```

```
    strcpy(objectDataArea->object_name, "APIobject");
```

```
    objectDataArea->prop_objvalue = OBJECT_NEW(objectDataArea->object_name,  
        "exception",
```

```

        NULL);

    /* Set the object objectData area */
    OBJECT_SETDATA((char *)objectDataArea);

    return(0);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
 OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
 OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERROR(), OBJECT\_RETPROPERTY(),  
 OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETPROPERTY()

## OBJECT\_SETPROPERTY()

---

### PURPOSE

Set a property from an OBJARG formatted string

### SYNONYM

None

### SYNOPSIS

```
#include "dbapi.h"
```

```
OBJECT_SETPROPERTY(objptr, propertyname, buffer)
```

<input parameters>

OBJECT	objptr;	/* Pointer to the object	*
char	*propertyname;	/* property name to set	*/

<output parameters>

OBJARG	buffer;	/* pointer to an OBJARG buffer	*/
--------	---------	--------------------------------	----

### DESCRIPTION

The OBJECT\_SETPROPERTY() macro is used to set the value of a property from an OBJARG formatted string.

### EXAMPLE

In the following example OBJECT\_SETPROPERTY () is used to set the value of the property "class" from an OBJARG string format returned from a parameter passed to the method.

#### Example Recital program:

```
test = newobject("myclass")
// Executes method 'define' and updates two properties
? test.define("Error Message", -1000)
? "test.objvalue.errorno",test.objvalue.errorno
? "test.objvalue.message",test.objvalue.message
```

#### Example in 'C' object:

```
#include "dbapi.h"
```

```
/* Define define method */
```

```
DEFINE_METHOD(clsMyClass, Define)
```

```
{
    struct example_data *objectData = (struct example_data *)OBJECT_GETDATA();
    struct    API_EXPRESSION result;
    char      buffer[512];
    int       rc;

    /* Check the object class */
    OBJECT_GETPROPERTY(objectData->prop_objvalue, "class", buffer);
    rc = OBJECT_GETARG(buffer, &result);
    if (result.errno == 0 && result.type == 'C' &&
        strcmp(result.character, "Exception") == 0) {
        switch (OBJECT_GETARGC()) {
```

```

case 2:
    rc = OBJECT_GETPARAMETER(2, &result);
    if (result.errno == 0 && result.type == 'N') {
        OBJECT_SETARG(buffer, &result);
        rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "errno", buffer);
    }

case 1:
    rc = OBJECT_GETPARAMETER(1, &result);
    if (result.errno == 0 && result.type == 'C') {
        OBJECT_SETARG(buffer, &result);
        rc = OBJECT_SETPROPERTY(objectData->prop_objvalue, "message", buffer);
    }
    break;
}

result.type = 'L';
result.logical = (rc == 0 ? 'T' : 'F');
OBJECT_RETRESULT(&result);
}

```

#### **SEE ALSO**

DEFINE\_CLASS(), DEFINE\_METHOD(), DEFINE\_PROPERTYGET(), DEFINE\_PROPERTYSET(),  
 DISPATCH\_FACTORY(), DISPATCH\_METHOD(), DISPATCH\_PROPGET(),  
 DISPATCH\_PROPSET(), OBJECT\_ASSIGN(), OBJECT\_DELETE(), OBJECT\_GETARG(),  
 OBJECT\_GETARGC(), OBJECT\_GETDATA(), OBJECT\_GETOBJECT(),  
 OBJECT\_GETPARAMETER(), OBJECT\_GETPROPERTY(), OBJECT\_GETTYPE(),  
 OBJECT\_GETVALUE(), OBJECT\_NEW(), OBJECT\_RETERERROR(), OBJECT\_RETPROPERTY(),  
 OBJECT\_RETRESULT(), OBJECT\_SETARG(), OBJECT\_SETDATA()

## APPENDIX A

---

The following data structure is for the “example.dbf” database file which is used in some of the examples.

FIELD	FIELD NAME	TYPE	WIDTH	DEC
1	BLOB	Memo	4	
2	BYTE	Byte	3	
3	CHAR	Character	20	
4	DATE	Date	8	
5	FLOAT	Float	10	2
6	INT	Integer	5	
7	LOGICAL	Logical	1	
8	NUMBER	Number	10	2
9	MEMO	Memo	4	
10	REAL	Real	10	2
11	WORD	Short	10	2
12	ZONED	Zoned	10	2